DEVELOPMENT OF FREE AND OPEN SOURCE SOFTWARE FOR FLOW MAPPING INTEGRATED TO GEOGRAPHIC INFORMATION SYSTEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

NAİM CEM GÜLLÜOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
GEODETIC AND GEOGRAPHIC INFORMATION TECHNOLOGIES

SEPTEMBER 2014

Approval of the thesis:

DEVELOPMENT OF FREE AND OPEN SOURCE SOFTWARE FOR FLOW MAPPING INTEGRATED TO GEOGRAPHIC INFORMATION SYSTEMS

submitted by **NAİM CEM GÜLLÜOĞLU** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Geodetic and Geographic Information Technologies Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Ahmet Coşar
Head of Department, **Geodetic and Geographic Information Tech.**

Prof. Dr. Oğuz Işık
Supervisor, **City and Regional Planning Department, METU**

Prof. Dr. Nurkan Karahanoğlu
Co-Supervisor, **Geological Engineering Department, METU**

Examining Committee Members:

Prof. Dr. Ali Türel
City and Regional Planning Dept., METU

Prof. Dr. Oğuz Işık
City and Regional Planning Dept., METU

Assoc. Prof. Dr. Nurünnisa Usul
Civil Engineering Dept., METU

Assoc. Prof. Dr. Ela Babalık Sutcliffe
City and Regional Planning Dept., METU

Prof. Dr. Can Ayday
Institute of Earth and Space Sciences, Anadolu University

Date: September 2, 2014

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name	:	Naim Cem Güllüoğlu
Signature	:	

ABSTRACT

DEVELOPMENT OF FREE AND OPEN SOURCE SOFTWARE FOR FLOW MAPPING INTEGRATED TO GEOGRAPHIC INFORMATION SYSTEMS

Güllüoğlu, Naim Cem
Ph.D., Department of Geodetic and Geographic Information Technologies
Supervisor : Prof. Dr. Oğuz Işık
Co-Supervisor : Prof. Dr. Nurkan Karahanoğlu

September 2014, 219 pages

Mapping of spatial interaction data is an ongoing challenge for cartographers. In many Geographic Information Systems (GIS) software there is no off-the-shelf functionality for processing and visualizing spatial interactions or geographical flows. Considering the development efforts that have been made in the last few decades to discover the potential of GIS almost in every aspect, handling and visualization of spatial interaction data under GIS remain underutilized.

The main objective of this study is to develop a general purpose free and open source software for flow mapping that is fully integrated to a desktop GIS application. Identified as the most fundamental form of geographical flows, the scope of this study focuses on exploration and visualization of interactions taking place between geographic locations where the actual flow routes are unknown or negligible. The flow mapping software, FlowMapper, is designed as a plugin to the popular, free and open source Geographic Information Systems software Quantum GIS (QGIS). Development environment tools utilized in this study consists of

Python programming language, PyQGIS Python bindings for QGIS API, PyQt Python bindings for Qt framework, Qt Designer tool and OGR Simple Feature Library. Designed as a fully menu driven and user friendly plugin, users of FlowMapper are capable of generating flow maps easily by supplying node coordinates and interaction matrix. Besides, flow related attributes such as net, gross magnitude calculations are automatically performed and flow gaining, flow losing nodes are automatically identified. In order to increase cartographic quality, advanced symbology options and flow filtering capabilities are also offered in FlowMapper as spatially non-distorting visual clutter reduction techniques. Capabilities of developed plugin are successfully tested with different scenarios and by using several flow datasets consisting of four to two hundred nodes.

Comprising of more than 6.500 lines of code, FlowMapper plugin received more than ten thousand downloads during two years of development period. Besides, plugin website received visitors from more than eighty countries. These indicators prove the need for integration of flow mapping tools to popular, open source desktop GIS applications. The main contribution of this study is the free and open source, general purpose flow mapping application FlowMapper which is integrated to QGIS in plugin form that aids exploration of spatial interaction data and creation of flows maps with symbology and filtering options.

Keywords: Flow Mapping, Spatial Interaction Data, Geographic Information Systems (GIS), Quantum GIS (QGIS)

ÖZ

AKIM HARİTALAMASI İÇİN COĞRAFİ BİLGİ SİSTEMLERİNE ENTEGRE ÖZGÜR VE AÇIK KAYNAK KODLU YAZILIM GELİŞTİRİLMESİ

Güllüoğlu, Naim Cem
Doktora, Jeodezi ve Coğrafi Bilgi Teknolojileri Bölümü
Tez Yöneticisi : Prof. Dr. Oğuz Işık
Ortak Tez Yöneticisi : Prof. Dr. Nurkan Karahanoğlu

Eylül 2014, 219 sayfa

Mekansal etkileşim verisinin haritalanması kartograflar için süregelen bir uğraştır. Çoğu Coğrafi Bilgi Sistemleri (CBS) yazılımında mekansal etkileşimleri veya coğrafi akımları işlemek ve görselleştirmek için kullanıma hazır fonksiyonlar bulunmamaktadır. Geçmiş on yıllarda CBS'nin her alandaki potansiyelini keşfetmek için harcanan çabalar göz önüne alındığında mekansal etkileşim verisinin CBS altında işlenmesi ve görselleştirilmesi yetersiz seviyede kalmıştır.

Bu çalışmanın başlıca amacı masaüstü bir CBS uygulamasıyla tümüyle bütünleşmiş, genel maksatlı, özgür ve açık kaynak kodlu bir akım haritalama yazılımı geliştirilmesidir. Bu çalışmanın kapsamı coğrafi akımların en temel formu olarak tanımlanan coğrafi lokasyonlar arasında gerçekleşen, fiili akım güzergahının bilinmediği veya ihmal edilebilir olduğu etkileşimlerin incelenmesi ve görselleştirilmesi üzerine odaklanmıştır. FlowMapper akım haritalama yazılımı popüler, özgür ve açık kaynak kodlu Coğrafi Bilgi Sistemleri yazılımı Quantum GIS'a (QGIS) bir eklenti olarak tasarlanmıştır. Bu çalışmada yararlanılan geliştirme ortamı araçları Python programlama dili, QGIS uygulama programlama arayüzü için PyQGIS Python bağlantıları, Qt geliştirme ortamı için Python bağlantıları, Qt tasarım aracı ve OGR kütüphanesinden oluşmaktadır. Tümüyle menü yönlendirmeli ve kullanıcı dostu bir eklenti olarak tasarlanan FlowMapper'ın kullanıcıları nod koordinatlarını ve etkileşim matrisini sağlayarak akım haritalarını kolayla üretebilmektedirler. Ayrıca net, brüt akım büyüklüğü gibi hesaplamalar otomatik olarak gerçekleştirilmekte ve akım kazanan, akım kaybeden nodlar otomatik olarak tespit edilmektedir. Kartografik gösterim kalitesini arttırmak için gelişmiş semboloji seçenekleri ve akım filtreleme yetenekleri mekansal bozulmaya neden olmayan görsel karmaşa azaltma teknikleri olarak FlowMapper içerisinde sunulmuştur. Geliştirilen eklentinin yetenekleri farklı senaryolar ve dörtten iki yüze kadar nod içeren çeşitli akım veri setleri ile başarılı şekilde test edilmiştir.

FlowMapper eklentisi 6.500 satırdan fazla koddan oluşmaktadır ve geliştirildiği iki yıllık dönem içerisinde on binden fazla indirilmiştir. Ayrıca eklentinin web sitesi seksenden fazla ülkeden ziyaretçi almıştır. Bu göstergeler akım haritalama araçlarının popüler, açık kaynak kodlu masaüstü CBS uygulamalarına entegrasyonuna olan ihtiyacı da kanıtlamıştır. Bu çalışmanın temel katkısı, QGIS'a eklenti formunda bütünleştirilmiş, mekansal etkileşim verisinin analizine ve akım haritalarının oluşturulmasına sunduğu gösterim ve filtreleme seçenekleriyle yardımcı, genel maksatlı, özgür ve açık kaynak kodlu FlowMapper uygulamasıdır.

Anahtar Kelimeler: Akım Haritalaması, Mekansal Etkileşim Verisi, Coğrafi Bilgi Sistemleri (CBS), Quantum GIS (QGIS)

To my wife

and

my parents

ACKNOWLEDGEMENTS

First and foremost I would like to express my deepest gratitude to my supervisor Prof. Dr. Oğuz Işık for his invaluable contributions, continuous support and encouragement throughout my PhD studies. His guidance and immense knowledge helped me to represent my research interests in the best possible way in my thesis study. I am also very grateful to my co-supervisor Prof Dr. Nurkan Karahanoğlu for his interest, advices and insightful discussions during the preparation of this thesis.

I would like to thank to my thesis supervising committee members Assoc. Prof. Dr. Nurünnisa Usul and Assoc. Prof. Dr. Ela Babalık Sutcliffe for their contributions and suggestions. I will never forget their warm and positive attitude during this research. I am very grateful to my PhD examining committee members Prof. Dr. Ali Türel and Prof. Dr. Can Ayday for their time, valuable comments and insightful reviews.

I owe Prof. Dr. Waldo Tobler a debt of gratitude for his pioneering studies in the literature that enlighten the importance of flow mapping. I also would like to thank to Alan Glennon and Michael Frank Goodchild for their inspiring studies focusing on handling and visualization of spatial interaction data under GIS. Besides, I would like to give special thanks to Alan Glennon for his open source development effort "Flowpy" script which inspired me for the development of FlowMapper plugin for QGIS.

Most important thanks go to tens of users who made valuable contributions by giving feedbacks to FlowMapper plugin via emails. Their feedbacks and suggestions helped me to improve the plugin for the better. Besides that, I am especially grateful to more than ten thousand anonymous users of FlowMapper

who made the plugin this much popular. In addition, I appreciate the work of all members in QGIS community for their collaborative efforts in development of such great GIS software, Quantum GIS.

I especially want to thank to my friends Ali Özgün Ok and Aslı Özdarıcı Ok for encouraging me to pursue a PhD. Their friendship and support during the hard times will always be remembered.

I am also grateful to my friends Burak Gürhan, Nilay Aygüney Berke, Mustafa Özgür Berke, Recep Özdoğan, Caner Şahin and Gizem Altınkaya Kurtulmuş for their understanding, support, motivation and close friendship which mean a lot to me. I am very lucky to have such amazing workmates as Ayhan Kavşut, Dursun Yıldırım Bayar, Saadet Aslı Bozkurt and Hatice Güneş from the office. I appreciate their friendship, help and kindness during the writing of the thesis.

I would like to express my sincere gratitude to my aunt L. Funda Şenol Cantek, my grandfather Mustafa Naim Şenol and my late grandmother Nermin Şenol. I am blessed to have them and thanks for everything that you have made from the day I was born. I also want to thank to my mother-in-law Aysel Urgancı for her motivation and support during the preparation of this thesis.

Lastly, I would like to thank to my parents Nurşen Güllüoğlu and Gürol Güllüoğlu for all their love and encouragement throughout my life. Their endless support and absolute confidence in me always keep me going. And most of all I thank to my wife İlksen Urgancı Güllüoğlu, for all the love and happiness we have shared during time spent and that will be spent together.

TABLE OF CONTENTS

CHAPTERS

APPENDICES

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

AML	Arc Macro Scripting Language
ANSI	American National Standards Institute
API	Application Programming Interface
dmg	Extension of Apple disk image file format
ESRI	Environmental Systems Research Institute
EULA	End User License Agreement
FlowMapper	QGIS plugin written in Python for flow mapping
FSF	Free Software Foundation
GDAL	Geospatial Data Abstraction Library
GIS	Geographic Information Systems
GIS-T	Geographic Information Systems for Transportation
GNU	Acronym for GNU's Not Unix
GPL	General Public License
GUI	Graphical User Interface
IDE	Integrated Development Environment
IDLE	Integrated Development Environment for Python
OGR	Simple Features Library
OS	Operating System
OSGeo	Open Source Geospatial Foundation
OSGeo4W	OSGeo Binary Distribution (Installer) for Microsoft Windows
OSI	Open Source Initiative Organization
png	Extension for Portable Network Graphics file format
POSIX	Portable Operating System Interface
py	Extension of Python code file in human readable format
pyc	Extension for Python byte code file in binary format
PyQGIS	Python Bindings for QGIS API
PyQt	Python Bindings for Qt

QGIS	Quantum GIS
Qt	Cross Platform Application and GUI Framework
shp	Extension for ESRI shapefile format
SOM	Self Organizing Map or Self Organizing Feature Map
VBA	Visual Basic for Applications
XML	Extendable Markup Language

CHAPTER 1

INTRODUCTION

"Geographical movement is critically important. This is because much change in the world is due to geographical movement. Movement of people, ideas, money, energy or material"

Waldo R. Tobler

Advances emerged in transportation infrastructure during 20th century and numerous developments in telecommunication technology witnessed in the last few decades have transformed the world to a highly dynamic global space more than ever. This dynamic and unbounded space notion was discussed by Castells (1989) as "Space of Flows" and can be characterized not only with the physical movements of people or material but also with the flows of energy, money, ideas and information as well. Therefore, representing geographic movements between locations, mapping flows and spatially dynamic events have critical importance in perceiving today's highly mobilized world.

Typically flow mapping corresponds to the act of visualizing movement patterns or interactions taking place between geographical locations. Node-to-node flows represent the most fundamental and common form of spatial interactions where the actual flow routes between locations are unknown or negligible (Tobler, 1976). Based on this assumption, the core elements needed to construct a flow map can be given as follows: (i) nodes as the origins and destinations of geographic movements, (ii) links as lines or curves for displaying flows between nodes and (iii) magnitude attributes to hold the amount of flows between nodes. Although this appears to be a simple cartographic task, lucid and effective representation of spatial interactions tables on maps has been a lengthy ongoing challenge for cartographers (Tufte, 2007) that involve both computational and cartographic issues.

The earliest noticeable examples of flow maps were prepared in the second half of 19th century by Charles Joseph Minard, a French civil engineer and a pioneer in thematic cartography (Robinson, 1982). Approximately a century after Minard's inspiring studies, in 1959 first examples of computer based flow mapping were produced by the Chicago Area Transportation Study, currently known as Chicago Metropolitan Agency for Planning (CMAP), to support decision making on highway route selection (Tobler, 1987). In the late 1960s, Kern and Ruston (1969) made the next significant contribution to computer based flow mapping by linking origins and destinations with the aid of a dedicated computer program. Computer based flow mapping techniques continued to develop during the 1970s (Tufte, 2007) and in the 1980s Tobler drew attention to the importance of geographical flows with his several studies Tobler (1976, 1981, 1987) by focusing on the visualization of spatial interaction data. Soon after Tobler drew attention to spatial interactions and flow phenomena; in his pioneering publication "The Informational City", Castells (1989) introduced the "Space of Flows" concept. In contrast to geographically bounded spaces, the "Space of Flows" concept pertains to human actions and interactions circulating dynamically among organizational nodes (Stalder, 2003).

Considering extensive capabilities of today's computers for processing large datasets and the proven potential of GIS in handling spatial data offers great opportunities for extracting interesting patterns and revealing valuable information from spatial interaction data. However, considering the development efforts that have been made in the last few decades to reveal the full potential of GIS almost in every domain; it is surprising to see that flow mapping remained relatively less developed under GIS (Rae, 2009). For example, either commercial (e.g. ArcGIS, MapInfo) or open source (e.g. GRASS, QGIS, gvSIG, uDIG), there exists neither functionality for visualizing spatial interactions nor a data model for storing inter-nodal flows in any of these popular desktop GIS software. In other words, currently none of the popular applications offers out of the box functionality that enables users to perform GIS-aware analyses on spatial interaction data. Thus, with the development of a flow mapping software that is fully coupled to GIS, this gap could partially be filled.

1.1. Identifying Problems in Flow Mapping

GIS provides substantial tools for analyzing and visualizing spatially static datasets; yet handling of spatially dynamic events still poses challenges for cartographers (Tufte, 2007). These challenges can be evaluated from two main perspectives: (i) problems emerging due to nature of flow phenomena and data intensive structure of spatial interaction tables and (ii) problems emerging from the absence of special flow mapping tools integrated to off-the-shelf GIS software.

Ever since the earliest noticeable examples of flow maps, which were produced by Charles Joseph Minard in the second half of 19th century (Robinson, 1982), general cartographic techniques utilized in visualization of flow data have not changed a lot (Boyandin et al., 2010). As mentioned by Tobler (1981), first origin and destination locations are mapped and then these locations are linked by lines. Besides, proportional line widths, graduated colors and arrow heads are commonly used to give additional information regarding the magnitude and direction of flows. Although this appears to be a simple cartographic task, in flow mapping researchers or users usually have to deal with datasets which are many times bigger than the total number of pixels available on computer display.

Flow data are usually stored in square matrix form. This matrix or interaction table stores the magnitude of flow lines between origin and destination nodes. As the number of flow nodes increases linear, the number of corresponding flow lines increase in a quadratic way. For example, a flow matrix involving 10 nodes will only result 100 inter-nodal flows while a flow matrix involving 100 nodes will result 10.000 flow lines to be rendered. This poses one of the most common challenges in flow mapping known as visual clutter problem which is caused by overlapping flow features (e.g. crossing of flow lines, overlapping of adjacent flow nodes) while displaying large interaction matrices. In addition to cartographic concerns, manual computation of flow related attributes, such as net or gross flow magnitudes, quickly becomes unfeasible as the number of nodes increases.

As a solution to the calculation of flow related attributes problem, utilization of computer processing power has already been discussed by Tobler (1987). Tobler suggests utilization of computerized techniques for datasets involving especially more than ten flow nodes. As a way out to the visual clutter problem, where much of the ongoing research efforts in flow mapping are focusing on (Phan et al., 2005; Guo, 2009; Boyandin et al., 2010), utilization of several visual clutter reduction techniques (Ellis and Dix, 2007) emerge as a solution. However at this point, absence of GIS integrated, user friendly and general purpose flow mapping tools emerges as another challenge. In fact, this problem is so severe that unless some third party tools or add-ons are used, it is not possible to automatically map few pairs of flows via interconnecting lines and get the flow related attributes calculated under GIS. One of the reasons for the absence of integrated flow mapping tools to popular commercial desktop GIS applications may be due to weak demand of market for such domain specific tools. In other words, the number of potential users demanding to pay for flow mapping tools may not be commercially enough or profitable to attract companies to initiate development. Based on this assumption, free and open source development methodology should be considered as a way of developing and distributing software in specific research domains such as flow mapping. In Chapter 3, free and open source software concept and development environment are reviewed in detail.

In general, potential value of a flow map increases proportional to the complexity of the flow dataset (Tobler, 1981). Thus, spatial interaction matrices having smaller sizes are usually considered to be geographically uninteresting since visualization of few interactions is not a difficult problem (Tobler, 1981). On the contrary, as the spatial interaction matrix gets larger, the number of flow lines exhibit quadratic increase. In other words, as the number of movements increase, more flow lines are added onto map and cartographic representation gets more difficult due to edge crossings (Pieke and Krüger, 2007). This usually requires implementation of special techniques to avoid or reduce visual clutter in flow maps. Another problem associated with a large spatial interaction matrix is that visual interpretation of such a large dataset without mapping is usually not convenient. Thus, to reveal any hidden spatial pattern and to extract valuable information, utilization of flow mapping techniques under GIS offer great potential. With this motivation almost three decades ago, Tobler (1987) suggested utilization of computer power for mathematical operations, statistical filtering and visualizations especially for datasets including more than ten nodes

According to Tobler (1981), another difficulty in flow mapping stems from the first law of geography: “Near places interact more than distant places.” Based on this assumption many interactions often take place between adjacent locations where there is little room to render flow lines. As a result of this, both flow lines and flow nodes suffer from visual clutter due to edge crossings and overlapping nodes. Similar to the methods utilized for avoiding edge crossing (e.g. flow bundling, hierarchical routing), there are lossy or spatially distorting (e.g. node clustering, point displacement) and non-distorting (e.g. node filtering, proportional symbology) techniques for avoiding node overlaps which are reviewed in Chapter 2 during literature survey.

In addition to the core components of a basic flow dataset which are locations as flow nodes, interactions as flow lines and flow magnitude as attribute data; flow data may involve multiple attributes. For example; internal migration data of a country may also include age groups, income levels and occupations of migrants in addition to the amount of migrants. This leads to high dimensionality in flow data which needs utilization of data mining and multivariate statistical techniques for extracting useful information. For example, Guo (2009) applied self organizing maps (SOM) technique for multivariate analysis and visualization of large spatial interaction data in order to decrease the number of origin destination pairs by clustering them based on multiple attributes. As an advanced topic, analysis and visualization of multivariate flow data is a more challenging area since it involves similar spatial concerns faced in univariate data but together with handling of multiple attributes. Yet, the scope of this thesis only focuses on mapping of univariate spatial interactions between discrete locations.

It is evident that mapping of large interaction datasets often poses visual clutter problem that hides interesting patterns in data and hinders valuable information. Besides, mapping of spatial interaction data remained underutilized in GIS. Thus, development of a specially designed and fully GIS integrated flow mapping tool which offers several visual clutter reduction techniques together with some cartographic representation options could fill this gap and pave the way for transforming spatial interaction data to spatial information and knowledge.

1.2. Aim and Contributions of the Study

The main objective of this study is to develop free and open source software for flow mapping that is fully integrated to a desktop GIS application. By tightly coupling this tool with a popular GIS application, such as in plugin form, users will also have the chance of benefiting from all existing capabilities of this GIS application without needing to recode them in the plugin.

The most noticeable contribution of this study is the flow mapping software named FlowMapper. Rather than a standalone application, FlowMapper is designed as a plugin to the popular, multi platform, free and open source desktop GIS application QGIS. By this way, FlowMapper can benefit from the agile development environment offered by QGIS platform. Motivation for open source development and reasons for selecting QGIS as the core GIS component that the FlowMapper is built on are further discussed and rationalized in Chapter 3.

Tobler (1976) identified the most fundamental and common form of spatial interactions as flows taking place between discrete locations where flow routes are unknown or negligible. Based on this insight, with the intention of developing a general purpose, multi platform flow mapping tool, FlowMapper plugin is developed to handle and visualize this type of spatial interactions under QGIS. Other types of flow scenarios are explained in Chapter 2 during literature survey. Yet, FlowMapper plugin targets automatic calculation of flow related attributes (e.g. length, net magnitude, gross magnitude etc.) and visualization of inter-nodal interactions between discrete locations while accounting cartographic concerns (e.g. symbol size, line width, color etc.).

One of the most common challenges in flow mapping can be identified as visual clutter problem which is caused by overlapping flow features (e.g. crossing of flow lines, overlapping of adjacent flow nodes) while displaying large interaction matrices. Thus, implementation of several visual clutter reduction techniques is highly desired. Yet, there are many visual clutter reduction techniques in the literature (Ellis and Dix, 2007) and implementation of all these techniques in FlowMapper is almost impossible. Since FlowMapper is intended to be a general purpose visualization tool that tries to keep the essence and spatial arrangement of flow data, only spatially non-distorting visual clutter reduction techniques (e.g. filtering and advanced symbology techniques) are considered during development rather than other techniques involving rearrangement of flow data (e.g. node clustering, hierarchical edge routing etc.). Prior to development, details regarding these techniques are reviewed in Chapter 2.

Another promise of this study is to develop a user friendly application. Since many GIS users are used to perform operations via menu driven and GUI based software, it is highly probable that potential users of FlowMapper will also be seeking for this experience. Thus, FlowMapper is intended to be designed as a fully interactive and GUI based plugin which is very critical for the utilization and acceptance level of FlowMapper.

As an integral part of the objective of this study, FlowMapper should not require any licensing fee and it should be distributed without any restriction under the GNU GPL v2 license. In other words, FlowMapper should be free and source code must be open. So it could reach more potential users who are seeking for spatial interaction mapping tools under GIS. Besides, it is highly desirable that the desktop GIS software on which the FlowMapper is to be built should also be provided at no cost. By this way, whole setup could be established without paying any license fee. In this context, free and open source development emerges as a promising method that encourages distribution of source code and code reuse. As a contribution of this kind of development, source code of FlowMapper could be used by other developers for further development of plugin. Besides, some parts of the source code of plugin could be examined and reused by researchers in derivative works.

The aim of this study is defined as development of a flow mapping software that is fully integrated to GIS. To fulfill this aim, several objectives are defined. These can be listed as follows: (i) The flow mapping software, FlowMapper, should be designed in plugin form on top of a desktop GIS application for seamless GIS integration, (ii) both FlowMapper and the desktop GIS application should be supplied free of charge in order to reach more people, (iii) Source code of FlowMapper should be open in order to encourage researchers for further development or code reuse, (iv) FlowMapper should be fully GUI based and should not require complex installation, (v) FlowMapper should be designed as a general purpose spatial interaction mapping tool to aid visualization of flows and calculation of flow attributes, (vi) FlowMapper should include several tools for reducing visual clutter problem in flow maps without distorting the essence of flow data. In Chapter 5, structural and functional requirements of FlowMapper plugin will be with the intention of achieving these objectives.

It is also appropriate to mention what is not aimed in this study. Neither development of a standalone, complete GIS application nor development of a flow mapping application that implements most current and complex visual clutter reduction algorithms are aimed in this study. In contrast, FlowMapper is intended to be a general purpose, user friendly tool to answer generic needs of GIS users.

The most prominent contribution of this study is the FlowMapper software that is fully integrated to QGIS in plugin form. Besides with the release of FlowMapper, one more plugin is added to the QGIS repository which also extends the capabilities of QGIS. As a plugin to one of the most popular desktop GIS platform, FlowMapper intends to pave the way from spatial interaction data to spatial knowledge and to aid people in understanding today's highly mobilized world.

1.3. Scope and Methodology of the Study

With the motivation of developing a general purpose flow mapping application integrated to GIS, the scope of this study mainly involves exploration and visualization of spatial interaction data pertaining to the scenario described by Glennon and Goodchild (2004) as flows taking place between discrete locations where the actual flow routes are unknown or negligible (e.g. human migration, telephone calls, imported and exported goods etc.). Nearly four decades ago, this scenario was mentioned as the most fundamental form of flow mapping by Tobler (1976) and generally involves flow nodes and lines as geographic features and magnitude of flows as attribute data. While keeping focus on this type of flows and intending to produce cartographically appealing flow maps, the scope of this study also involves implementation of visual clutter reduction techniques such as filtering and advanced symbology options since visual clutter is one of the most common problems of flow mapping especially when working with large datasets. Considering that FlowMapper is designed as a user friendly tool that intends to respond general purpose flow mapping needs of GIS users; implementation of only spatially non-distorting clutter reduction techniques (e.g. subsetting by a given criteria, graduated color rendering, proportional symbol size adjustment etc.) are considered in FlowMapper and other algorithms such as node clustering or flow line bundling, routing which imply either spatial distortion or rearrangement of flow data are excluded from the requirements list of FlowMapper.

Methodology of this study can be evaluated from two perspectives. While the first one involves general methodology and organization of this study, the second one pertains to the software development methodology of FlowMapper plugin. Under this heading general methodology and organization of this study is explained. Development methodology of the plugin is at the beginning of Chapter 5 because it depends on the findings from the literature survey and the review of free and open source software concept together with the review of open source desktop GIS applications.

The aim of this study is set as development of a free and open source flow mapping tool. This software should be designed as a general purpose GUI based tool and must be fully integrated to a popular desktop GIS application. Besides, it should include several techniques for exploring flow data and reducing visual clutter problem. To fulfill all these requirements, following questions should be answered first: What is flow mapping? What does free and open source software mean and how it is developed? What are the current challenges in flow mapping? Which visual clutter reduction techniques are listed in the literature? What are the characteristics of existing flow mapping tools? Which desktop GIS applications exist on the free and open source market? Which GIS application should be preferred as the base platform to built FlowMapper on? Which software environment is needed to develop FlowMapper?

In order to answer all the questions above and to properly evaluate functional requirements of FlowMapper prior to initiating coding, a detailed literature survey is undertaken in Chapter 2 together with a review regarding free and open source software concept in Chapter 3. Besides existing open source desktop GIS applications are reviewed in Chapter 3 and upon determining the suitable platform for building FlowMapper on, development environment required for the plugin is discussed in Chapter 4. Development methodology of the plugin is given at the beginning of Chapter 5 and then the architecture of FlowMapper is explained in detail. In Chapter 6, capabilities of the plugin are explored by using several test datasets and generated flow maps are given. Organization of this study is presented with the flow chart given in Figure 1.1.

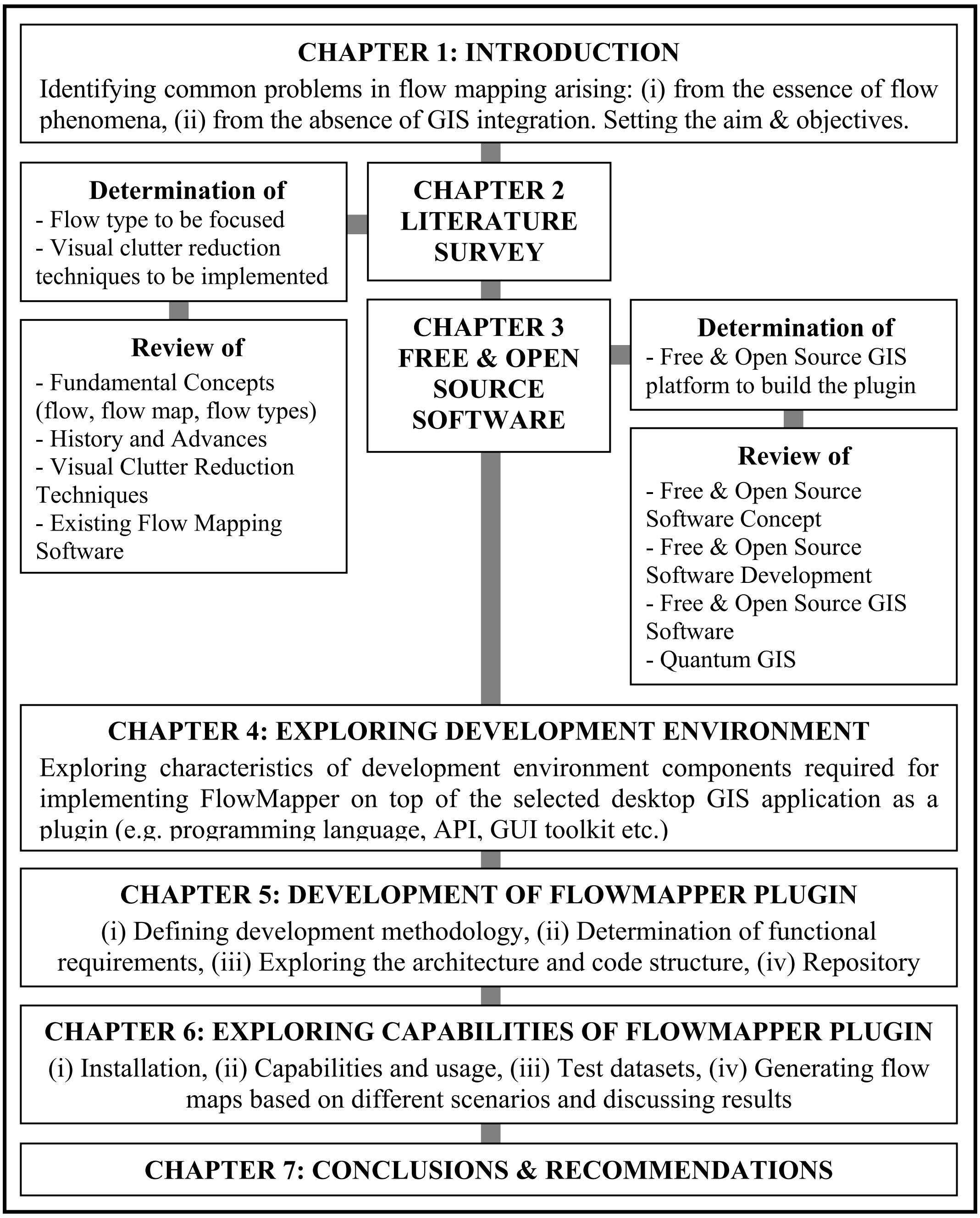

CHAPTER 1: INTRODUCTION
Identifying common problems in flow mapping arising: (i) from the essence of flow phenomena, (ii) from the absence of GIS integration. Setting the aim & objectives.
Determination of
- Flow type to be focused
- Visual clutter reduction techniques to be implemented
CHAPTER 2 LITERATURE SURVEY
Review of
- Fundamental Concepts (flow, flow map, flow types)
- History and Advances
- Visual Clutter Reduction Techniques
- Existing Flow Mapping Software
CHAPTER 3 FREE & OPEN SOURCE SOFTWARE
Determination of
- Free & Open Source GIS platform to build the plugin
Review of
- Free & Open Source Software Concept
- Free & Open Source Software Development
- Free & Open Source GIS Software
- Quantum GIS
CHAPTER 4: EXPLORING DEVELOPMENT ENVIRONMENT
Exploring characteristics of development environment components required for implementing FlowMapper on top of the selected desktop GIS application as a plugin (e.g. programming language, API, GUI toolkit etc.)
CHAPTER 5: DEVELOPMENT OF FLOWMAPPER PLUGIN
(i) Defining development methodology, (ii) Determination of functional requirements, (iii) Exploring the architecture and code structure, (iv) Repository
CHAPTER 6: EXPLORING CAPABILITIES OF FLOWMAPPER PLUGIN
(i) Installation, (ii) Capabilities and usage, (iii) Test datasets, (iv) Generating flow maps based on different scenarios and discussing results
CHAPTER 7: CONCLUSIONS & RECOMMENDATIONS

Figure 1.1. Organization of the Study

Development of a computer program is usually triggered upon identification of the software requirements. In this study, unless detailed literature survey on flow mapping is carried and open source software concepts are understood; neither functional nor development environment requirements can be determined properly. Thus, as presented on the upper section of the flowchart (Figure 1.1), introduction chapter is followed by a literature survey and review of open source software concept.

In Chapter 2, fundamental concepts of flow mapping are explained and a brief history is presented in conjunction with the advances in flow mapping. Besides, several visual clutter reduction techniques cited in the literature and existing flow mapping software are discussed. With the guidance of this survey, type of flow scenario to be focused is determined (e.g. inter-nodal flows taking place over uncertain paths). Besides, visual clutter reduction techniques to be implemented in FlowMapper are identified (e.g. non-distorting methods focalizing on the appearance of data rather than the methods dealing with spatial rearrangement of data).

In Chapter3, issues regarding semantics of free and open source software, types of open source software licensing and community based open source development issues are surveyed. Besides, open source desktop GIS applications are reviewed in detail to determine the most suitable open source desktop GIS platform that the FlowMapper is going to be built on.

In Chapter 4, upon determining the QGIS as the core GIS component, characteristics of other essential development environment components are explored. This involves a review of Python programming language, QGIS development API, Qt framework and open source OGR library.

At the beginning of Chapter 5, development methodology of FlowMapper is presented with respect to the requirements of plugin. Then, architecture of the plugin is explored with respect to QGIS plugin structure, coding, modules and dependencies, GUI structure, functions and features implemented. Last section of Chapter 5 includes a review of plugin release history. Repository download statistics and plugin website visits are presented at the end of Chapter 5.

Chapter 6 mainly involves demonstration of the capabilities of FlowMapper. Installation methods, input data structure and user interface of FlowMapper are explained. Then, several flow maps are generated by using the test datasets to demonstrate different capabilities of plugin. Besides, cartographic quality of these flow maps is discussed. Following this chapter, conclusions of this study are presented together with recommendations for further development of FlowMapper. Besides two appendices, one of which contains the source code of the module that creates flow features and the other which portraits inner structure and interaction of the plugin with QGIS, are given at the end of this study.

CHAPTER 2

LITERATURE SURVEY

In this chapter, the aim is to give better understanding of the history of flow mapping and ongoing challenges for displaying spatial interaction data. In the first part, fundamental concepts are explained and brief history is given in conjunction with the advances in flow mapping. Then a taxonomy study, presenting different approaches aiming to reduce the amount of visual clutter encountered in flow maps, is given. Besides, some of the methods referred to in this taxonomy are discussed and exemplified by means of several selected studies from the literature. At the end of this chapter, review of existing flow mapping software is presented in order to reach a better understanding of the maturity and functional capacity of flow mapping applications.

2.1. Fundamental Concepts

From a general perspective, a flow is defined as a steady and continuous movement of something or somebody in one direction (Oxford University Press, 2012). In computer science; flow is perceived as data transfer and implies flows of bits through an information system between processes (Bruza and Weide, 1993). Besides, in computer networks, it corresponds to flows of data packets over a topology between source and destination nodes. In transportation management, which studies the interactions between vehicles, people and infrastructure, flow is perceived as traffic flow and characterized by density and velocity (Gülgeç, 1998). It is clear that flow has many other meanings in different domains such as hydrology, chemistry, telecommunication etc.

Within the scope of this study, flow is considered as cumulative movements of people, animals, goods, money and information within a period of time. In GIS, a flow corresponds to the geographical movement of a body and embodies at least three fundamental components in its simplest form; (i) origin, (ii) destination and (iii) magnitude which are very identical with a vector.

Tobler, who made the earliest contributions to computer aided mapping of spatial interaction data in the late 70s (Tobler, 1976), makes the definition of flow mapping as the act of visualizing movement patterns between geographic locations. More than three decades after Tobler described flow phenomena, in the literature there are still very close definitions to Tobler's regarding flow mapping and flow maps. For example; Rae (2009) defines flow mapping as a special cartographic technique that includes mapping and visualizing movements from origin to destination points in geographic space. Similarly, Pieke and Krüger (2007) define flow maps as visualizations of interconnected links indicating movements of people or objects between geographic locations within a certain period of time.

With reference to definitions given above, it can be inferred that following components are emphasized as the core elements to construct a flow map: (i) nodes as the source and destination locations of geographic movements (e.g. points, centroids of polygons); (ii) edges as links either with or without arrow heads for displaying flow directions or flows between sources and destinations (e.g. lines, polylines, arcs with fixed or varying widths, graduated colors) and (iii) magnitude which indicates the amount of cumulative flow between nodes (e.g. attribute(s) data stored in flow matrix or cube).

In their study regarding the methods for visualizing spatial interaction data, Andrienko et al. (2008) criticize Tobler's early approach (Tobler, 1976) to flow mapping with the missing time component. As a result of this, flow mapping techniques initially introduced by Tobler (1976) are inconvenient for displaying any temporal change, trend that exists in flow data cube or among matrices. Both Andrienko et al. (2008) and Boyandin et al. (2010) suggest cartographic production techniques such as animated flow map rendering or utilization of flow maps in series, named as small-multiples, in order to add the notion of time to flow mapping for demonstrating any possible temporal trend within data.

Glennon and Goodchild (2004) characterize three diverse use cases, scenarios for flow maps in their study which aims to develop a generic GIS database template for flow data. Based on the characteristics of flow phenomena, following scenarios can be depicted on flow maps: (i) node-to-node flows, (ii) flows along networks or through known routes and (iii) situations where node-to-node and network flows occur in proximity. Flow maps showing node-to-node flows represent patterns of geographical movements via straight lines, arrows or curves between discrete nodes by using origin, destination and magnitude data available in flow matrix. Since node-to-node flows represent the most fundamental form of spatial interactions between known locations where the actual flow route is unknown or negligible (Tobler, 1976), within the context of this study, visualization and analysis of discrete, inter-nodal flow data under GIS is primarily kept in focus. For the second scenario, flow maps focus on the magnitudes of flows taking place along network segments. For the last scenario, where node-to-node and network flows occur in proximity; both origin – destination nodes and the magnitude of flow along network edges are equally considered. Respectively, these three flow scenarios can be exemplified with the following cases: (i) human migration, (ii) trips with known routes and (iii) drainage networks, flow accumulation (Figure 2.1).

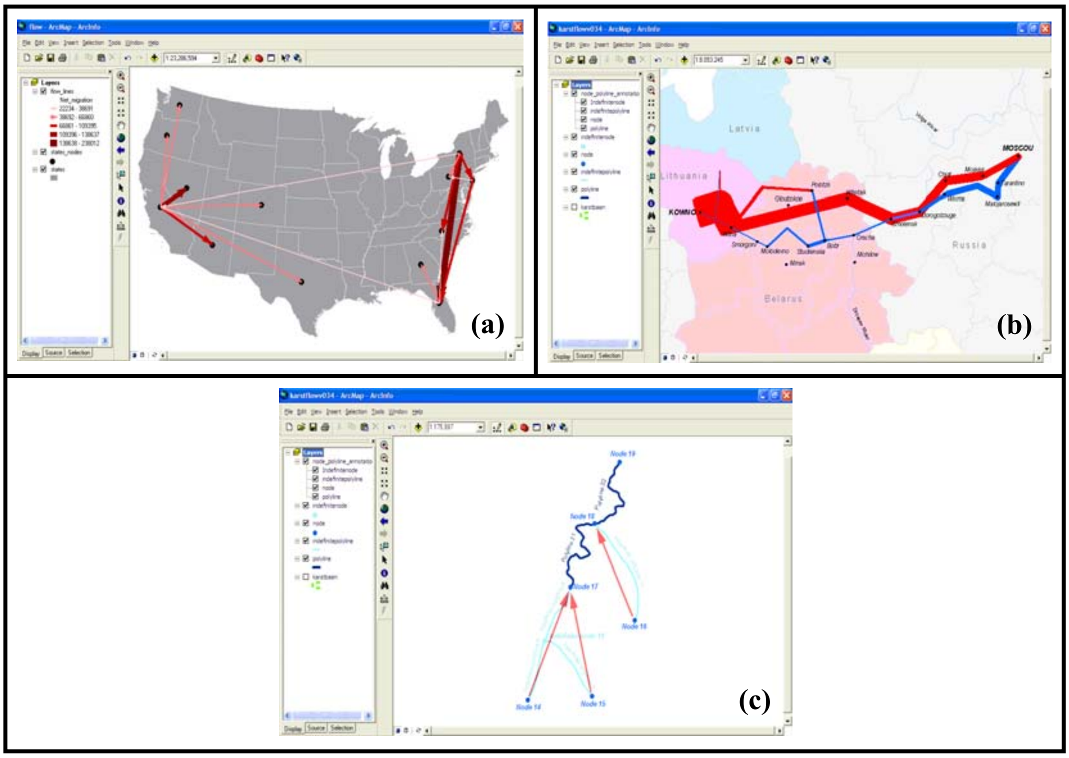


Figure 2.1. Flow Maps based on Diverse Flow Scenarios: (a) Largest migrations in The US observed during 1995 – 2000 shown as node-to-node flows, (b) Napoleon's march on Moscow illustrated along a route via successive flow lines with decreasing magnitude, (c) Subsurface flow routes in a karst watershed which follow both network and node-to-node paths are depicted as a hybrid approach (ArcGIS Flow Data Model Tools plugin implementations are adapted from Glennon and Goodchild, 2004)

In addition to Glennon's and Goodchild's (2004) approach, in the literature there are other studies categorizing flow maps from other aspects. For example, in their study regarding the visual analytical methods for movement data, Andrienko et al. (2008) categorize flow maps as (i) maps displaying individual's movement behaviors and (ii) maps showing cumulative movements of multiple entities during a certain time period.

Maps in the first category demonstrate the temporal movement behavior of individuals or individual events and space time cube visualization technique, introduced by Hagerstrand at the end of sixties (Hagerstrand 1970 in Hedley et al., 1999), is a common cartographic way for displaying them in the research area of time geography (Kraak, 2003). In its basic appearance, space time cube visualization can be described with the following components: (i) geography on the base plane of the cube to point out locations along x and y axis; (ii) z axis, along which locations scattered on the base plane are extruded with respect to time and duration of visit, movement; (iii) 3D polylines either showing directions of movements or displaying actual routes by interconnecting scattered, transit nodes extruded along z axis. Flow map displaying Napoleon's March on Moscow, previously given in Figure 2.1 (b), is adapted to space time cube representation by extruding the locations on the transit route along z axis proportional with the time (Figure 2.2).

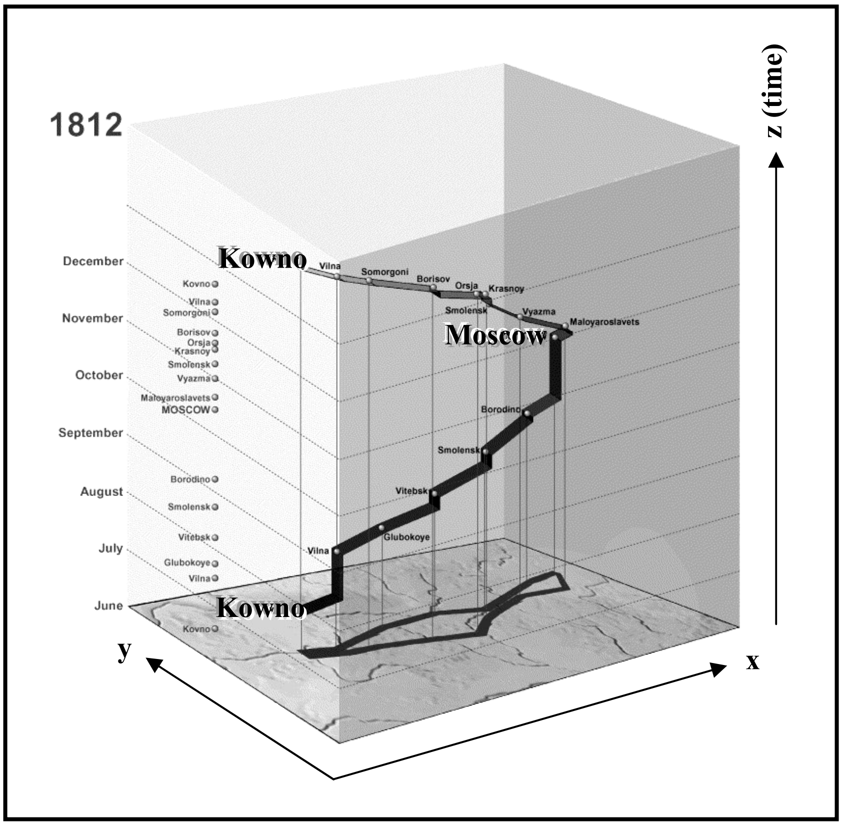


Figure 2.2. Geo-visualization of Napoleon's 1812 Campaign into Russia in Space Time Cube Notation (adapted from Kraak, 2009)

Studies analyzing temporal movement behaviors of individual events and discussing techniques for aggregation of spatiotemporal data in space time cubes can also be found in the literature: Mountain, 2005; Kraak, 2003; Dykes and Mountain, 2003, 2002; Mountain and Raper 2001.

Maps in the second category, previously presented in Figure 2.1(a), visualize cumulative flow data which is analogous to a collection of long time data series derived by the aggregation of individual movements. Flow mapping techniques discussed in Tobler's early studies (Tobler, 1976, 1981, 1985, 1987) and his successors (Boyandin et al., 2010; Rae, 2009; Guo, 2009; Pieke, 2007; Phan et al., 2005) focus on geo-visualization of mass, cumulative flows and discuss visual clutter reduction techniques such as sampling, filtering, node clustering, edge bundling, edge routing etc.

Rather than keeping focus on temporal movements of individuals, which is especially studied in time-geography (Kraak, 2003), scope of this study strictly focuses on mass, cumulative flow data and geo-visualization of this data under GIS with a user friendly, GUI based flow mapping plugin.

2.2. History and Advances in Flow Mapping

The earliest noticeable examples of flow maps were produced in the second half of 19th century by Charles Joseph Minard, a French civil engineer and a pioneer in thematic cartography and statistical graphics (Robinson, 1982). Phan et al. (2005) refer to Minard's hand-drawn flow maps and identify his success with the utilization of following techniques: (i) intelligent distortion of positions, (ii) bundling of edges sharing similar destinations and (iii) intelligent edge routing. Examples of Minard's flow maps selected from the literature are given in Figure 2.3.

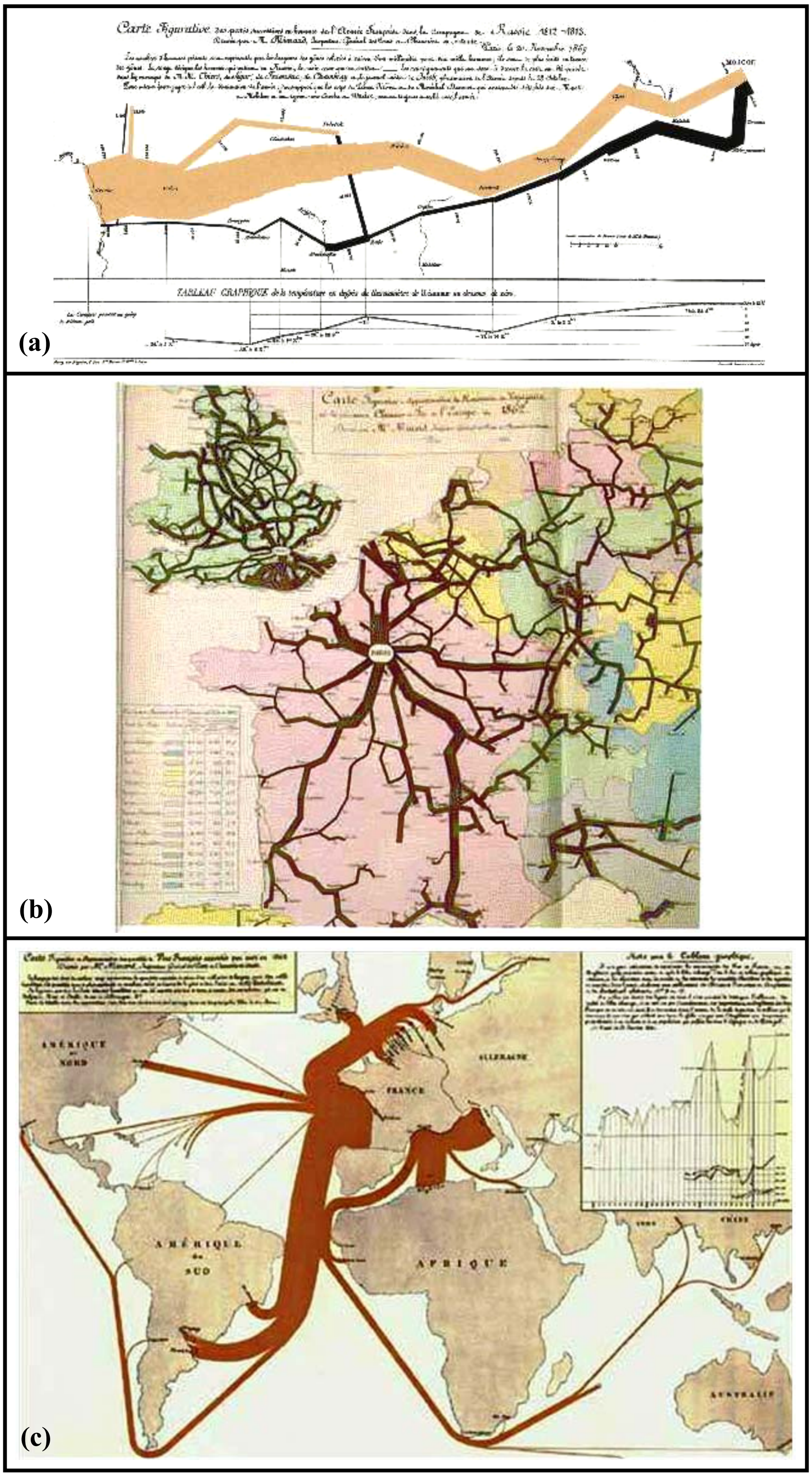


Figure 2.3. Minard's Original Flow Maps Prepared in the 1860s: (a) Napoleon's 1812 campaign into Russia (adapted from Tufte, 2007), (b) Movement of travelers on the major railroads of Europe (adapted from Friendly, 2002), (c) French wine exports in 1864 (adapted from Tufte, 2007)

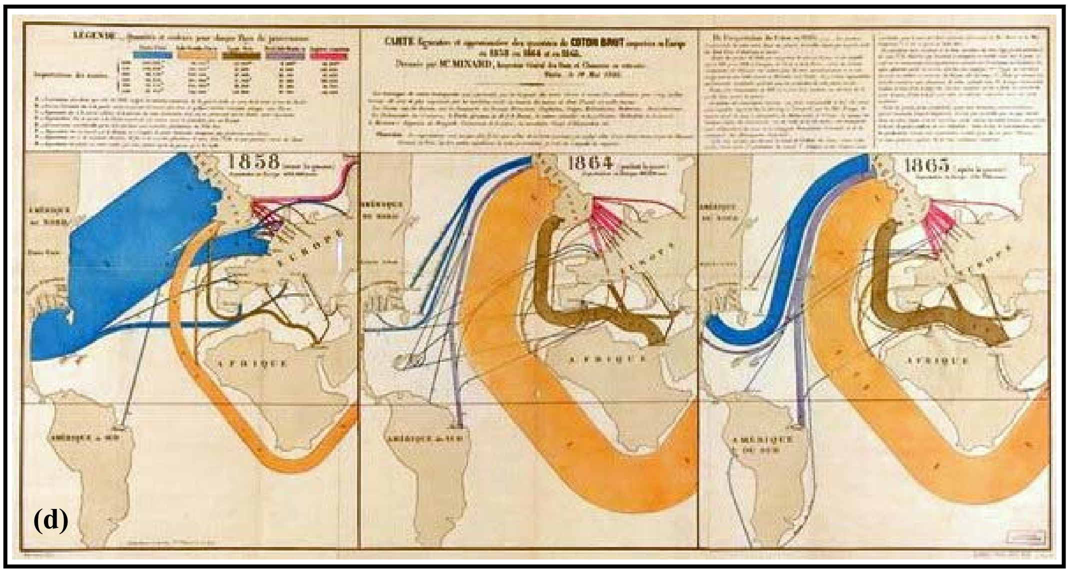
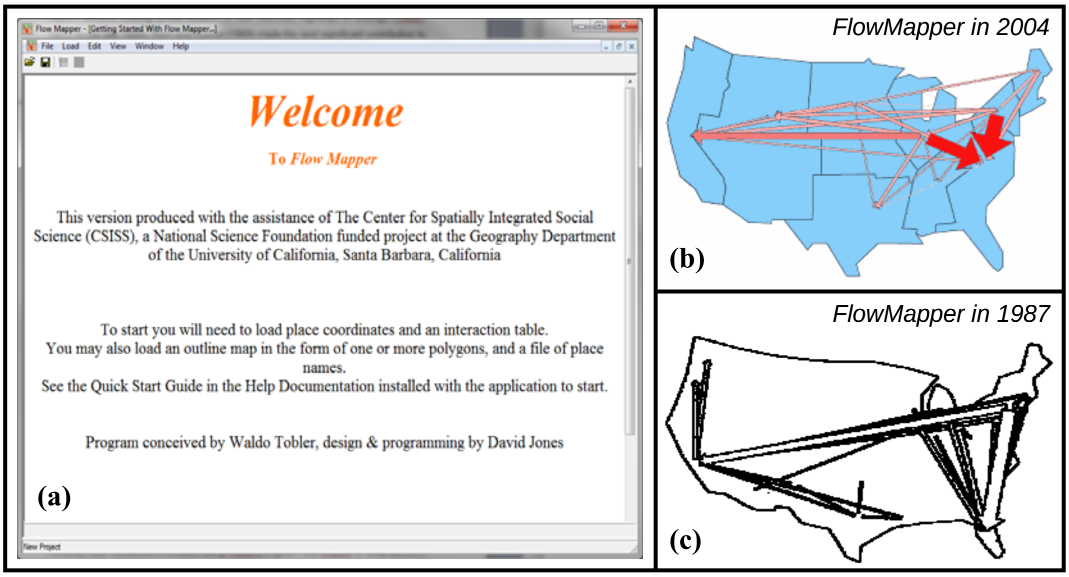


Figure 2.3. Minard's Original Flow Maps Prepared in the 1860s (cont.): (d) Europe raw cotton imports in 1858, 1864 and 1865 (adapted from Börner and Hardy, 2008)

Approximately a century after Minard's pioneering studies, in 1959 first examples of computer based flow mapping were produced by the Chicago Area Transportation Study, currently named as Chicago Metropolitan Agency for Planning (CMAP), in order to support decision making on the location of new interstate highways in Chicago (Tobler, 1987). In the late 1960s, Kern and Ruston (1969) made the next significant contribution to computer-based flow mapping by plotting single lines on a map with the aid of a dedicated computer program.

Computer based techniques for mapping geographic flows continued developing in the late 1970s (Tufte, 2007) and during the 1980s Tobler advanced this subject with his two studies: (i) "A Model of Geographic Movement" (Tobler, 1981) and (ii) "Experiments in Migration Mapping by Computer" (Tobler, 1987). In 1987, Waldo Tobler made the most prevailing contribution by developing the Flow Mapper software by which discrete, node-to-node flows could be mapped. Software attracted much attention later in 2003, FlowMapper was ported, recoded to operate under Microsoft Windows with the support of Center for Spatially Integrated Social Science (Tobler, 2003; CSISS, 2004) (Figure 2.4).


Flow Mapper - [Getting Started With Flow Mapper...]
File Load Edit View Window Help
New Project
(a)
FlowMapper in 2004
(b)
FlowMapper in 1987
(c)

Welcome

To *Flow Mapper*

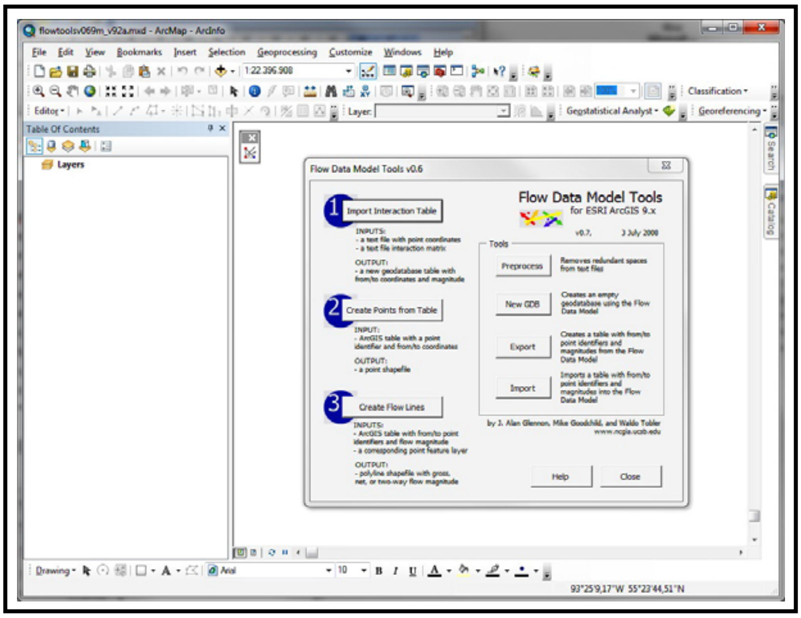
This version produced with the assistance of The Center for Spatially Integrated Social Science (CSISS), a National Science Foundation funded project at the Geography Department of the University of California, Santa Barbara, California

To start you will need to load place coordinates and an interaction table.
You may also load an outline map in the form of one or more polygons, and a file of place names.
See the Quick Start Guide in the Help Documentation installed with the application to start.

Program conceived by Waldo Tobler, design & programming by David Jones

Figure 2.4. (a) Tobler's FlowMapper Software Adapted to Run under Microsoft Windows and (b) Its Sample Output (CSISS, 2004) vs. (c) Output of Original FlowMapper (Tobler, 1987)

Glennon and Goodchild (2004) adapted node-to-node flow mapping capabilities of Tobler's FlowMapper to ESRI ArcGIS 9 platform by developing a series of VBA macros (Figure 2.5). Having attracted attention of many users and researchers (Rae, 2009); this tool realized integration of fundamental flow mapping techniques into an off-the-shelf, proprietary desktop GIS package rather than standalone applications. Besides, Glennon and Goodchild (2004) designed several flow data models for handling interaction data under GIS regarding different types of flow scenarios which are given as: (i) node-to-node flows where actual route is unknown (e.g. human migration), (ii) flows taking place along networks or through routes with varying magnitude (e.g. railroad passenger trips) and (iii) situations where node-to-node and network flows occur in proximity (e.g. watersheds).


flowtoolsv069m_v92a.mxd - ArcMap - ArcInfo
File Edit View Bookmarks Insert Selection Geoprocessing Customize Windows Help
1:22.396.908
Classification
Editor
Layer:
Geostatistical Analyst
Georeferencing
Table Of Contents
Layers
Search
Catalog
Drawing
Arial
10
93°25'9,17"W 55°23'44,51"N

Flow Data Model Tools v0.6

Flow Data Model Tools
for ESRI ArcGIS 9.x
v0.7, 3 July 2008

1 Import Interaction Table
INPUTS:
- a text file with point coordinates
- a text file interaction matrix
OUTPUT:
- a new geodatabase table with from/to coordinates and magnitude

2 Create Points from Table
INPUT:
- ArcGIS table with a point identifier and from/to coordinates
OUTPUT:
- a point shapefile

3 Create Flow Lines
INPUTS:
- ArcGIS table with from/to point identifiers and flow magnitude
- a corresponding point feature layer
OUTPUT:
- polyline shapefile with gross, net, or two-way flow magnitude

Tools
Preprocess — Removes redundant spaces from text files
New GDB — Creates an empty geodatabase using the Flow Data Model
Export — Creates a table with from/to point identifiers and magnitudes from the Flow Data Model
Import — Imports a table with from/to point identifiers and magnitudes into the Flow Data Model

by J. Alan Glennon, Mike Goodchild, and Waldo Tobler
www.ncgia.ucsb.edu

Help Close

Figure 2.5. Flow Data Model Tools for ArcGIS 9.x

However, visualizations of spatial interaction data created either using Tobler's original FlowMapper or its adaptations potentially suffer from overlapping nodes and edge crossings especially for cases where flow data matrix includes more than ten nodes since software lack implementations of proper clutter reduction techniques. Although none of these were implemented in his FlowMapper software; Tobler (1987) discussed some techniques to prevent from visual clutter. These techniques can be listed as follows: (i) Filtering: applying filters on flow tables based on some descriptive statistics to summarize dense data, (ii) Clustering: spatial aggregation of adjacent nodes with respect to predefined clustering tolerance and (iii) Bundling and Routing: merging of individual flows into larger streams those sharing same general directions. Besides, to improve cartographic quality, Tobler (1987) suggested rendering of small arrows on top of larger ones so that flow lines having less magnitudes would not be suppressed by major flows. The other minor enhancement discussed by Tobler (1987) was the utilization of curved flow bands or arrows with trajectories "through the air" above a map shown in perspective, which was previously discussed by Thornthwaite (1934 in Tobler, 1987).

Motivation behind all these techniques discussed by Tobler in 1987 to reduce high dimensionality of spatial interaction data and to avoid visual clutter in flow maps, still attract attention of many researches. For instance, Boyandin et al. (2010), Holten and Wijk (2009), Cui and Zhou (2008), Phan et al. (2005) focused bundling and routing of flow lines in their studies (Figure 2.6). Thomson and Lavin (1996) discussed utilization of animation in migration maps and Wood et al. (2009), Xiao and Chun (2009) tried to build completely new visualizations such as flow trees and kriskograms.

SOM is a type of neural network based clustering technique utilized for reducing high dimensionality in data while preserving topological properties of the input surface. In VIS-STAMP software, Guo et al. (2006) offered utilization of SOM technique for multivariate clustering of flows. In his further study focusing on flow mapping and multivariate visualization of large spatial interaction data, Guo (2009) employed hierarchical clustering, data mining techniques and SOM in migration data under VIS-STAMP (Figure 2.6). However, limited cartographic functionality offered in VIS-STAMP is criticized by Adrienko et al. (2008).

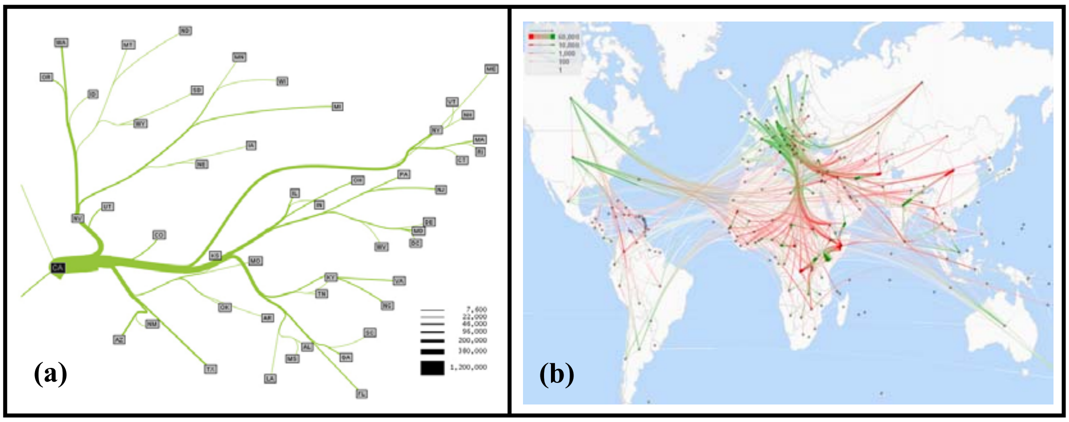


Figure 2.6. Methods for Reducing Visual Clutter: (a) Flow Map Layout: Migration from California between 1995 – 2000 using edge routing and layout adjustment (Phan et al., 2005), (b) JFlowMap: Refugee flows between countries in 2008 with bundled flow lines (Boyandin et al., 2010)

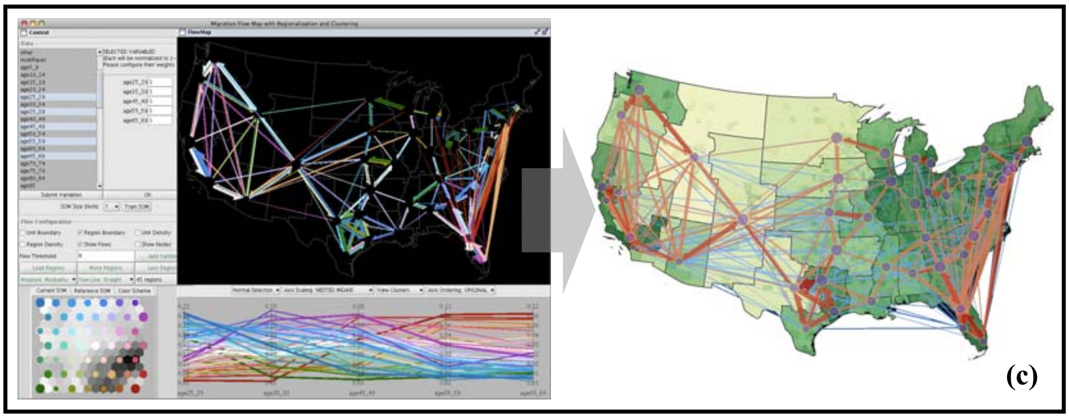


Figure 2.6. Methods for Reducing Visual Clutter (cont.): (c) VIS-STAMP: SOM visualization (bottom left), parallel coordinate plot (bottom center) and output migration map between counties between 1995 – 2000 with hierarchical regions (right) (Guo, 2009)

In addition to these studies, other advances in analysis and mapping of spatial interaction data are presented below.

In 1990, as a result of the collaboration between Utrecht University and Gadjah Mada University in Indonesia, a stand alone application known as Flowmap was developed (Geertman et al., 2003). Flowmap was initially released as a package to operate under Microsoft DOS; however current releases can operate under Microsoft Windows platform and have been successfully used in some studies (e.g. de Jong and van Eck Ritsema, 1996; van Eck Ritsema and de Jong, 1999). Despite its compatibility with proprietary GIS file formats (e.g. ESRI shapefile, MapInfo mif/mid) it is not yet widely used (Boyandin et al., 2010) which may be arising from its standalone design. In its official website, developers of Flowmap state that "Flowmap is not a general purpose GIS. In fact, its spatial analysis tools and functionalities are rather basic. It is specifically designed to be used in combination with a Database Management System (DBMS) and a mapping system and/or general purpose GIS." (Flowmap, 2013) (Figure 2.7)

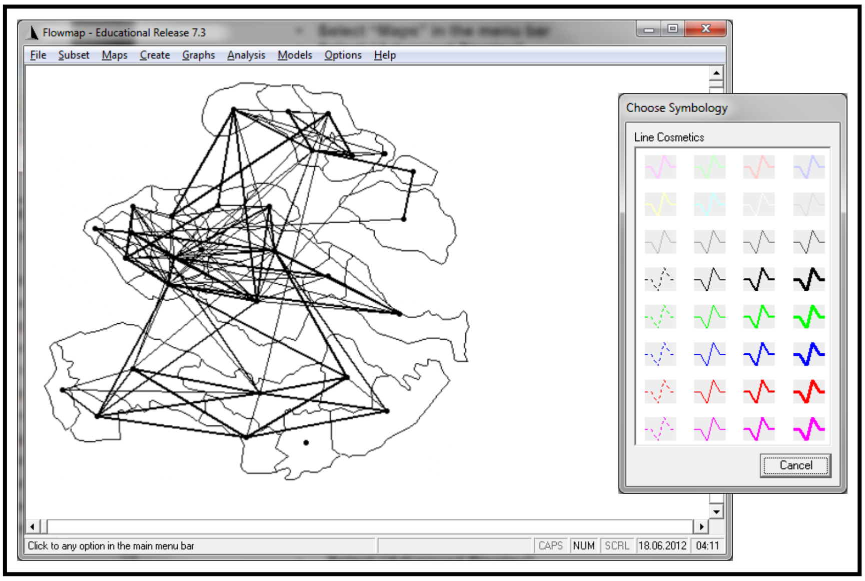
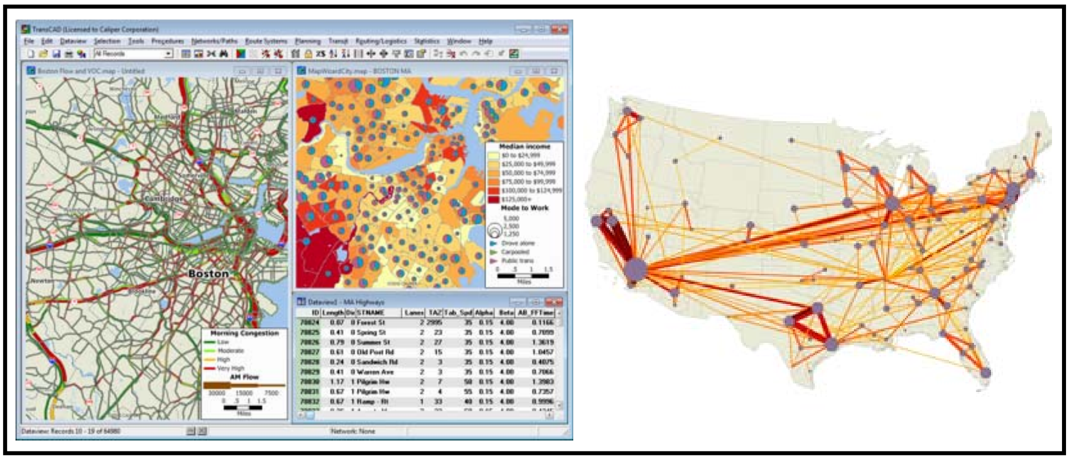

Flowmap - Educational Release 7.3
File Subset Maps Create Graphs Analysis Models Options Help
Choose Symbology
Line Cosmetics
Cancel
Click to any option in the main menu bar
CAPS
NUM
SCRL
18.06.2012
04:11

Figure 2.7. Sample Flow Map Created in Flowmap v7.3 between Discrete Nodes with Straight Line Segments

Other applications allowing for some forms of flow mapping can be given as follows: (i) MapInfo by means of MapBasic program “Cre8Line” to draw lines between discrete nodes; (ii) O’Malley’s (1998) Desire Line Maker extension for ArcView 3.x which can produce flow maps from either a combination of shapefile and data table or a data table with origin and destination coordinate information; (iii) ET GeoWizards extension for ArcGIS 9.x for node-to-node flows; (iv) Caliper’s GIS based transportation planning software TransCAD for drawing transport desire lines between trip generation zones or nodes (Figure 2.8). However, it should be noticed that except from TransCAD neither of these software offers any specific built-in tools for reducing visual complexity that arises especially when working with dense datasets.



ID	Length	Dir	STNAME	Lanes	TAZ	Tab_Spd	Alpha	Beta	AB_FFTime
70824	0.07	0	Forest St	2	2995	35	0.15	4.00	0.1166
70825	0.41	0	Spring St	2	23	35	0.15	4.00	0.7059
70826	0.79	0	Summer St	2	27	35	0.15	4.00	1.3619
70827	0.61	0	Old Post Rd	2	15	35	0.15	4.00	1.0457
70828	0.24	0	Sandwich Rd	2	3	35	0.15	4.00	0.4075
70829	0.41	0	Warren Ave	2	3	35	0.15	4.00	0.7066
70830	1.17	1	Pilgrim Hw	2	7	50	0.15	4.00	1.3983
70831	0.67	1	Pilgrim Hw	2	4	55	0.15	4.00	0.7357
70832	0.67	1	Ramp - Rt	1	33	40	0.15	4.00	0.9996

Figure 2.8. Trip Desire Lines between Origin and Destination Locations Created by Using TransCAD v5

Considering the development efforts that have been made in the last few decades to discover the full potential of GIS in every aspect, handling and visualization of spatial interaction data under GIS remain underutilized, which this study intends to contribute. Thus, its potential should be developed if the path from spatial interaction data to spatial information and knowledge is to be made clear (Longley et al., 2005).

2.3. Review and Comparison of Visual Clutter Reduction Techniques

In this part, review of visual clutter reduction techniques selected from the literature is given together with a taxonomy study presenting the benefits and shortcomings of these techniques.

In their study reviewing the developments in multidimensional, multivariate visualizations, Wong and Bergeron (1997) reveal visual clutter problem by stating “scientists have to deal with data that is many thousand times bigger than the number of pixels on display”. Situation is the same for flow mapping; amount of spatial interaction data could be very large while visual displays are relatively small. Thus, visual clutter reduction techniques must be used to ensure preparation of clear, understandable visualizations from which users can obtain knowledge. Otherwise, plotting too much data on too small an area will result in visual clutter which diminishes the usefulness of visualization. Besides, problem is much severe if the users of spatial interaction data focus on explorative visualization where the potential information and knowledge locked within the mass of data is unknown (Ellis and Dix, 2007).

Below, brief review of selected studies discussing clutter reduction techniques are presented in chronological order.

In the early 90s, Leung and Apperley (1994) presented a taxonomy study on distortion-oriented visualizations. Although visual clutter problem is not directly discussed in this study, in the conclusion they state that “non-distortion techniques, such as information suppression, should be investigated further since they are potentially powerful”. A decade after Tobler (1987) identified visual clutter problem arising in flow maps mainly due to large spatial interaction tables, Keim (1997) discussed sampling, querying, segmentation and aggregation as techniques for reducing the amount of data to be displayed. Card et al. (1999) proposed user interaction into the process of data visualization, mapping and mentioned techniques as zooming, magic lens, context focusing and dynamic querying which are latter discussed as clutter reduction techniques in the taxonomy study of Ellis and Dix (2007). Comprehensive reviews of data clustering algorithms are performed by Jain et al. (1999) and Murtagh (2000). Although these studies did not directly keep focus on visual clutter reduction they presented methods for preprocessing mass data into manageable sets.

In his taxonomy, discussing strategies for multidimensional data visualization, Ward (2002) states following placement principles: (i) position derived from the data (e.g. node coordinates), (ii) degree of overlaps allowed (e.g. tolerance for edge crossings of flow lines), (iii) display utilization (e.g. screen resolution, allocation) and (iv) spatial displacement for improving visibility (e.g. topological distortion of base map). In his study, Ward (2002) also recommended user control over these placement principles since they are a tradeoff between efficient display use, amount of occlusion and distortion.

Ellis and Dix (2007) divided clutter reduction techniques into three groups in their taxonomy study principally keeping focus on 2D visualizations (Table 2.1).

Table 2.1. Visual Clutter Reduction Techniques (adapted from Ellis and Dix, 2007)

	Clutter Reduction Techniques
Appearance	Sampling
	Filtering
	Change Point Size
	Change Opacity
	Clustering
Spatial Distortion	Point / Line Displacement
	Topological Distortion
Temporal	Animation

Under the appearance group (Table 2.1), Ellis and Dix (2007) list the techniques that affect the look of data. Changing point size and opacity are self-explanatory techniques. Clustering results in different representations such as bundle of individual lines or group of points, thus this technique is also listed under appearance group. Filtering and sampling techniques are also included into the appearance group since they have a direct influence on appearance; such as items may disappear as a direct result of filtering. Ellis and Dix (2007) explain the difference between sampling and filtering as sampling is the random selection of a subset from the whole dataset whereas filtering is the selection of a subset from the whole dataset that satisfies a given criteria. Thus, if the user needs to focus on a specific set of data or has an insight of what might be interesting then filtering is ideal; otherwise sampling provides a way to explore the data without preconceptions.

Spatial distortion techniques focus on displacement of points, lines and distortion of base plane. Point, line displacement technique adjusts the position of each data item. Topological distortion stretches the background, base map either non-uniformly (e.g. fisheye) or uniformly (e.g. zoom) by also including data items plotted on it. Point, line displacement techniques do not alter the base map or background plane, but points, lines may be plotted in slightly different positions than their actual coordinates. For the case of topological distortion, since this technique distorts the base map by including the map items (e.g. different map projections), features are relatively in their right positions, but space has changed.

Animation, which can be explained as visualization of map series in a successive order, is mentioned in the literature (Bruijin and Spence, 2000) in relation to clutter reduction; thus Ellis and Dix (2007) include this technique under temporal category.

In their study, evaluating clutter reduction techniques for information visualization, Ellis and Dix (2007) define eight major benefits from the aspect of user and system to compare these techniques. These criteria include; (i) avoiding overlaps, (ii) retaining spatial information, (iii) local applicability, (iv) scalability, (v) adjustability, (vi) capability to show point/line attributes, (vii) capability to discriminate points/lines and (viii) capability to show overlap density. In the next paragraph, possible benefits of each criterion are discussed briefly.

Avoiding overlaps directly reduces clutter thereby gives the ability of identifying underlying patterns and avoids the loss of information by giving more display space to features. For any type of geospatial data, retaining spatial information, coordinate pairs defining positions of points is significant. However, the accuracy to which users can measure the absolute position of a point is argued by Ellis and Dix (2007) and relative positions are considered to be more important than the absolute positions of features especially when searching for patterns in geospatial data. Local applicability stands for implementation of any clutter reduction algorithm limited to a specific region or regions of the display. Benefits of localization include; capability of reducing clutter in localized regions in order to reveal information underneath, allowing users to investigate items in detail whilst keeping the general context, avoiding possible information loss in low density areas when reducing clutter in high density areas. Scalability is a desirable property in any information system, thus clutter reduction techniques that are capable of handling large datasets are desirable. As Ellis and Dix (2007) state, most of the visualizations are interactive and allow user control up to some degree. Adjustability criterion is related with the ability of adjusting parameters of the system that have an influence on the degree of visual clutter. Possible benefits of adjustability are the ability of adjusting sampling rate to desired level in order to see patterns (Dix and Ellis, 2002), interactive adjustment helps users to understand cluster distribution (Chen and Liu, 2003). Capability of showing point/line attributes is beneficial when displaying multivariate data; with techniques providing this capability user has the possibility of mapping flow nodes and flow lines with color, size and opacity based on desired attribute. Capability to discriminate individual points or lines in a crowded display helps to differentiate overlapping points (Brodbeck et al., 1997) and reduce apparent clutter by making each cluster of lines distinct (Fua et al., 1999). Ellis and Dix (2007) suggested that if over plotting is present in display, users should be aware of this; otherwise they may not realize that data may remain hidden from their view. As Wegman (1996) stated, capability of showing overlap density help users to discriminate individual lines such as outliers.

Considering both the visual clutter reduction techniques listed in Table 2.1 and eight criteria explained in the previous paragraph, a comparison of clutter reduction techniques versus plotting criteria is given in Table 2.2. It should be noted that the purpose of this classification is not to prefer any technique over others; on the contrary the aim is to present a systematic comparison in order to create a better understanding about the strengths and weaknesses of different approaches.

Table 2.2. Comparison of Visual Clutter Reduction Techniques (adapted from Ellis and Dix, 2007)

Criteria Clutter Reduction Technique	avoids overlaps	retains spatial information	local applicability	scalability	adjustability	capable to show point/line attributes	capable to discriminate points/lines	capable to show overlap density
sampling	possibly	✓	✓	✓	✓	✓	✗	✗
filtering	possibly	✓	✓	✓	✓	✓	✗	✗
point size	possibly	✓	✓	✗	✓	✓	possibly	✗
opacity	partly	✓	✓	✗*	✓	✗*	✓*	✓*
clustering	possibly	partly	✗*	✓	✓	partly	✓*	possibly
point/line displacement	✓*	✗*	✓	✗	possibly	✓	possibly	✗
topological distortion	possibly	possibly	✓	✗	✓	✓	✗	✗*
animation	✓*	✓	✓*	✓*	✓*	✓	✗	✗*

✓ **satisfies criterion** ✗ **does not satisfy criterion** * **some exception**
possibly : criterion is met in some situation but not in others partly : criterion is partly met in some situations

With reference to Table 2.2, not all clutter reduction techniques avoid overlap. Since points and lines are not displaced in sampling technique, it cannot avoid overlap completely. However, sampling can be used successfully to reveal hidden patterns (Ellis and Dix, 2007). Besides, Ward (2002) suggests that an acceptable amount of overlap can be tolerated and allowing users to adjust sampling rate may be an optimum solution. As sampling, filtering cannot completely avoid overlapping; but it can reveal desired relationships for the chosen data range. Rendering large points may conceal other points underneath those plotted earlier. Thus reducing point sizes to optimum level may be beneficial. However, there are trade-offs between overlap and points size; besides screen space need to be large enough to see the required detail (Derthick et al., 2003). Although change in opacity cannot avoid overlap, it can reveal underlying or partially overlapping points. By means of clustering, the number of data items may be reduced to simplify the plot. So, clustering may be used to avoid overlaps by either representing a group of points by a single point or a group of lines by a single line or band (Ellis and Dix, 2007). Although point, line displacement algorithms are specifically designed to avoid overlaps, they are strictly limited by the number of pixels on the display and human visual perception. Topological distortion involves stretching the base map either uniformly (e.g. zooming) or non-uniformly (e.g. fisheye lens) to give extra display space to the map features. However, as discussed in Phan et al. (2005), it may not be completely possible to avoid overlapping in flow maps unless additional techniques, such as edge routing and flow bundling, are implemented. In animation technique, rapid serial visual presentation (de Bruijn and Spence, 2000) may avoid overlaps by showing a stack of images to user in quick succession.

Sampling, filtering, changes to opacity and point size; all retain spatial information. Clustering, by default, loses individual spatial information; however it keeps aggregated spatial information for the clusters. The higher the number of overlapping points, the greater the distortion required to accommodate these points without overlaps. However, spatial information actually lost does not completely depend on the new displacement since relative positions after displacement are still representative in terms of spatial adjacency which was identified by Ellis and Dix (2007) as a more determining factor than absolute positions. Case is also analogous to topological distortion technique. However since base plane is being stretched, Ellis and Dix (2007) argued that spatial information could be retained as long as sufficient landmarks are preserved on the map.

Clustering is performed on similarity measures and does not necessarily maintain any spatial locality. Thus, performing local operations such as passing a lens over a set of points to examine clusters would be meaningless.

Sampling, filtering and clustering techniques can be considered as scalable since all these techniques deal with large datasets and inherently reduce the number of plotted points. However, reducing the point size is limited to the available resolution of the display and visual perception; thus categorized as not scalable. For opacity, Healey et al. (1995) suggested that it is only useful up to five overlapping items; thus allows very limited scalability. Utilization of topological distortion to very large number of data items would lead to a significant spread of points which results unmanageable distortion from the point of user. De Bruijn and Spence (2000) argued that some animation techniques showing data items in sequence can deal with very large datasets; however the time required to show all data should need to be taken into account.

Adjustability criterion is related to whether the degree of clutter reduction can be adjusted interactively by the user. Ellis and Dix (2007) proposed that sampling rate, dynamic query range, point size, opacity and to some extent cluster size can all be adjusted by the user. However, the amount of spatial displacement depends on the data density. The rate of animation rate can also be adjusted; however this has no effect on the clutter reduction.

All techniques apart from opacity and clustering do not have adverse effect on the utilization of symbology for points or lines (e.g. color, texture). However, reducing the opacity diminishes the significance of points, especially color; besides for clustering symbology is applied based on aggregated values rather than individual features.

As Ellis and Dix (2007) also stated, it is desirable to distinguish between individual points or lines so in a crowded display. A side effect of increasing point size was criticized by Ellis and Dix (2007) as giving additional prominence to outlier data which tend to be in sparse areas of the map. Thus, it may lead a distorted view of the data distribution. Opacity is used to discriminate overlapping lines, especially in association with clustering technique. However, as Healey et al. (1995) stated, opacity is only useful when up to five items overlap. As an implementation of point, line displacement technique; Wong and Bergeron (1997) discussed curving lines with edge lens technique to disambiguate the connected nodes of a graph.

Fekete and Plaisant (2002) discussed that with careful adjustment of opacity, users can gain insight about overlap density. Subsequent to clustering, although aggregated attributes can be shown with proportional symbology (e.g. applying gradual color proportional to the number of items in the cluster), opacity remains the only clutter reduction among others that promotes visual indication of overlap density.

2.4. Review and Comparison of Existing Flow Mapping Software

Under this heading characteristics of several flow mapping applications are explored. This involves the review of several applications especially designed for flow mapping (e.g. Tobler's Flow Mapper, JFlowMap etc.) together with other software which are not solely developed for flow mapping but similar or comparable kind of functionality (e.g. TransCAD, Gephi).

The motivation in performing this review is not the determination of the "best" flow mapping application or choosing one tool over another. In contrast, the motivation is to compare these applications to understand the current situation together with the strengths and weaknesses of existing flow mapping applications for rationalizing "What to be implemented in FlowMapper". Besides, by exploring general characteristics and features of these applications, current challenges and common problems in development of a flow mapping application are tried to be identified. So that, some structural and functional requirements can be determined for FlowMapper plugin.

Although there are only a few applications specifically designed for flow mapping, there are some on the market that offer this kind of functionality even though many of them lack GIS integration. Some of these applications are already introduced while reviewing history and advances in flow mapping. For example; in Figure 2.4 Windows version of Tobler's Flow Mapper (Tobler, 1987; CSISS, 2004), in Figure 2.5 Flow Data Model Tools for ArcGIS 9 (Glennon and Goodchild, 2004), in Figure 2.6 Flow Map Layout (Phan et al., 2005), JFlowMap (Boyandin et al., 2010) and VIS-STAMP (Guo et al., 2006; Guo, 2009), in Figure

2.7 Flowmap (Flowmap, 2013) and in Figure 2.8 TransCAD are presented with their main user interfaces under previous headings. Yet, it is clear that these are not the only available applications on the market in terms of establishing links between nodes; in other words creating inter-nodal flow lines. For example, Gephi is one of such software that is worth mentioning before presenting a side-by-side comparison. Bastian et al. (2009) define Gephi as open source network exploration and manipulation software. Based on the information on its website (The Gephi Consortium, 2014), Gephi offers highly customizable symbology and labeling options for exploring and visualizing networks. Besides, it also offers filtering, node clustering and edge bundling algorithms which are similarly included in JFlowMap (Boyandin et al., 2010). From all these aspects, Gephi may appear as a perfect tool for flow mapping. However, Gephi has no GIS integration; meaning that users cannot use their GIS datasets in Gephi. Yet, until the release of "Export to SHP" plugin in 2013, there was no option for exporting created flow features to any GIS file format in Gephi. Thus, rather than a GIS integrated, specially designed flow mapping tool; Gephi should be considered as a fully featured, multi purpose network exploration and visualization tool that has roots in graph theory. Apart from Gephi, there are other minor tools for creating links between nodes. For example, by using the "XY to Line" tool that is located under ArcGIS ArcToolbox, it is possible to create lines between origin and destination nodes. Similar operation can be performed by using the "Cre8Line" MapBasic program under MapInfo. However, neither of these tools can handle interaction tables and calculate flow related attributes such as net or gross flow magnitudes. Thus, it is not convenient to consider this kind of line creation tools as flow mapping applications. Based on the findings above, both Gephi and other minor tools designed for line generation are excluded from the side-by-side comparison given in Table 2.3. On the contrary, seven different flow mapping applications are listed on this table together with the latest release of FlowMapper plugin to give better comparison against existing software.

Performing an in depth review of a computer program involves detailed examination of each feature and requires long term experience with that software.

Regarding seven different flow mapping applications listed in Table 2.3, this kind of detailed survey should be considered as a separate study which is beyond the scope of this study. Since the intention in performing this review is not the determination of “best” flow mapping application or choosing one tool over another, these applications are examined on the basis of their general characteristics. In other words, to understand the current situation of existing flow mapping software, rather than applying a quantitative methodology, a qualitative comparison is performed with respect to several criteria which can be given as follows; type of application, multi platform support, installation method, runtime requirements, licensing, GIS integration, GUI support, continuity of development and availability of visual clutter reduction techniques.

Table 2.3. Comparison of Existing Flow Mapping Software

Criteria	**Flow Mapper** (Tobler, 1987) (CSISS, 2004)	**TransCAD** (Caliper Corporation, 2014)	**Flowmap** (Flowmap, 2013) (Geertman et al., 2003)	**Flow Data Model Tools for ArcGIS** (Glennon & Goodchild, 2004)	**Flow Map Layout** (Phan et al., 2005)	**VIS-STAMP & GraphRECAP & FlowMap** (Guo et al., 2006; Guo, 2009)	**JFlowMap** (Boyandin et al., 2010)	**FlowMapper for QGIS**
Application Type	Standalone	Standalone	Standalone	Add on as VBA macro	Standalone	Standalone (4)	Standalone	Add on as Python plugin
Supported Platforms	MS. Win.	MS. Win.	MS. Win.	MS. Win.	MS. Win. Linux MacOSX	MS. Win. Linux MacOSX	MS. Win. Linux MacOSX	MS. Win. Linux MacOSX
Installation Method	Binary setup, installer	Binary setup, installer	Binary setup, installer	Available as mxd project file	Manual	Manual	Manual	One click via QGIS plugin manager
Requirements	.NET Framework Runtime & Adobe SVG Viewer	N/A	N/A	ArcGIS v9.x	Java Runtime 5 (JRE or JDK)	Java Runtime 6 (JRE or JDK)	Java Runtime 6 (JRE or JDK)	QGIS v2.x

Table 2.3. Comparison of Existing Flow Mapping Software (cont.)

Criteria	**Flow Mapper** (Tobler, 1987) (CSISS, 2004)	**TransCAD** (Caliper Corporation, 2014)	**Flowmap** (Flowmap, 2013) (Geertman et al., 2003)	**Flow Data Model Tools for ArcGIS** (Glennon & Goodchild, 2004)	**Flow Map Layout** (Phan et al., 2005)	**VIS-STAMP & GraphRECAP & FlowMap** (Guo et al., 2006; Guo, 2009)	**JFlowMap** (Boyandin et al., 2010)	**FlowMapper for QGIS**
Licensing	Freeware	Commercial Std. Lic. 12.000 USD	Edu. edt. Freeware & Pro. edt. Commercial	Open Source	Open Source BSD	Freeware	Open Source Apache Lic. 2.0	Open Source GPL v2
GIS Integration	No	Yes (2) (GIS-T)	Partially (Export, Import GIS formats)	Yes (as ArcGIS macro)	No	No	No	Yes (as QGIS Plugin)
GUI Based	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Initial Release Date	2004 (1)	1985	1990	2004	2005	2006	2009	2012
Current Release	1.1	5.0	7.4.2	0.7	0.1 Alpha	N/A	0.16.6	0.4
Developed with	VB .NET	Proprietary	Unknown	VBA	Java	Java	Java	Python
Development Depreciated	Unknown	No	No	Yes (3)	Unknown	No	Unknown	No
Visual Clutter Reduction Techniques	Limited (Size & Color)	Yes (Advanced Symbology & Filtering)	Yes (Advanced Symbology & Filtering)	No	Yes (Advanced Symbology & Edge routing & Node adjust.)	Yes (Advanced Symbology & Filtering & Node Clustering & SOM)	Yes (Advanced Symbology & Filtering & Flow Bundling & Node Clustering)	Yes (Advanced Symbology & Filtering)
Website	Flow Mapper: www.csiss.org/clearinghouse/FlowMapper TransCAD: www.caliper.com/tcovu.htm Flowmap: http://flowmap.geo.uu.nl/index.php Flow Data Model Tools for ArcGIS: http://dynamicgeography.ou.edu/flow Flow Map Layout: http://graphics.stanford.edu/papers/flow_map_layout VIS-STAMP & FlowMap : http://www.spatialdatamining.org/software JFlowMap: http://ilya.boyandin.me/works/2010/10/01/jflowmap FlowMapper for QGIS: http://plugins.qgis.org/plugins/FlowMapper							

(1) Originally developed by W. Tobler in 1987; further ported to Windows by CSISS in 2004.
(2) TransCAD is a GIS based application designed for transportation planning
(3) ESRI discontinued VBA support in ArcGIS with the release of 10.1.
(4) VIS-STAMP is for multivariate analysis. GraphRECAP & FlowMap is an integrated tool for exploring large flow datasets.

Based on Table 2.3, it can be concluded that excluding the FlowMapper plugin for QGIS and Flow Data Model Tools developed in VBA as an add on to ArcGIS 9, the rest of the applications run as standalone software. However, it should be noted that since VBA support in ArcGIS is discontinued with the release of version 10.1, it is not possible to run Flow Data Model Tools on current releases of ArcGIS. This makes the situation for GIS integration even worse. Except from QGIS FlowMapper plugin, among all these applications listed in Table 2.3, there is neither freeware nor open source software for flow mapping that offers GIS integration or support for common GIS file formats. One of the reasons of this lack of integration may be due to standalone design preference. Excluding TransCAD, which is a GIS based commercial application for transportation planning, unless core GIS capabilities are embedded or extensive import export capabilities are added to support common GIS formats, effective GIS integration cannot be realized in standalone applications. Furthermore, implementation of these capabilities in a standalone application requires additional coding effort.

Standalone design of these flow mapping applications has also some impacts on installation requirements. Based on Table 2.3, three of these applications, namely Flow Map Layout, VIS-STAMP and JFlowMap do not come with any user friendly installer. This means users must perform installation either by manually locating required files or by using command line prompts to install and execute the software. It is clear that this type of installation may be confusing for average users who are accustomed to GUI based setups. Moreover, the situation becomes worse when these applications depend on external libraries for execution or require installation of runtime environment. For example; although JFlowMap is a fully GUI based software that offers several visual clutter reduction techniques, it needs installation of Java Runtime Environment on the system and needs to be manually executed from the command line by pointing to the “jar” file which is not the most user friendly way of running a program. In contrast to JFlowMap, Tobler’s Flow Mapper has a wizard like user friendly setup for Windows. However, to run Tobler’s Flow Mapper on Windows platforms, users must also install .NET Framework Runtime and Adobe SVG Viewer on their system which adds some extra steps to the installation process. On the other hand, FlowMapper plugin for QGIS does not require installation of any additional library which is not already included in QGIS and can be installed from the QGIS plugin manager just by one click.

All applications included in this review are GUI based and offer menu driven structure for performing operations. While all work on Windows platforms, owing to multi platform support of Java language, Flow Map Layout, VIS-STAMP and JFlowMap can also work on Linux and MacOSX environment. Similarly FlowMapper plugin can also work on multiple platforms thanks to cross platform support of Python language and QGIS development environment.

From the perspective of licensing concerns, there are both open source and proprietary applications. Regarding the proprietary applications, they are either offered as freeware software (e.g. Tobler's Flow Mapper, Educational edition of Flowmap) or require non-free commercial license (e.g. TransCAD, Professional edition of Flowmap). On the commercial software side, TransCAD appears as a featured GIS application that also includes tools for making detailed analyses on origin destination matrices and creating flow lines between these locations which are called "desire lines" in analytical transportation planning. However this comes at a price of more than 10.000 USD for the standard version (Caliper Corporation, 2014) which is obviously out of the purchasing limits of most GIS users. On the free and open source software side, there are four applications including QGIS FlowMapper plugin. Based on this finding, it can be inferred that free and open source development is also a preferred method for developing applications in specific research domains such as flow mapping.

One of the most common challenges in flow mapping is visual clutter problem on which most researchers have focused (Phan et al., 2005; Guo, 2009; Boyandin et al., 2010). Under the previous heading, in Table 2.1, several visual clutter reduction techniques are presented from the literature. While some techniques bring about spatial distortion in data, some affect only the appearance of data which can be defined as spatially non-distorting techniques.

Based on Table 2.3, almost all flow mapping applications, except for Tobler's Flow Mapper and Flow Data Model Tools for ArcGIS, offer some kinds of non-distorting techniques such as filtering or customizable symbology for reducing clutter and generating cartographically more appealing maps. Besides, three applications implement node clustering (e.g. Flow Map Layout, VIS-STAMP and JFlowMap), flow line routing (e.g. Flow Map Layout) and flow line bundling (e.g. JFlowMap) algorithms which involve rearrangement of flow data (Ellis and Dix, 2007). Although implementation of these algorithms result appealing flow maps, they imply a tradeoff between keeping the original spatial arrangement of data and avoiding visual clutter.

One of the intentions of performing a literature survey on flow mapping by means of reviewing visual clutter reduction techniques and existing flow mapping software is for properly determining functional requirements of FlowMapper and also rationalizing them. Based on the findings of this chapter, flow scenario to be focused in FlowMapper is determined as node-to-node flows taking place through uncertain routes. With reference to the reviews regarding visual clutter reduction techniques and comparison of existing flow mapping software, non-distorting reduction techniques such as filtering of flow data and implementation of advanced symbology options are determined as the clutter reduction techniques to be implemented in FlowMapper. Besides, several requirements of FlowMapper such as GUI based design, hassle free installation, multi platform support, GIS integration and inclusion of visual clutter reduction techniques are rationalized by reviewing the current status of flow mapping applications.

CHAPTER 3

FREE AND OPEN SOURCE SOFTWARE

In the previous chapter, fundamental concepts of flow mapping are discussed together with a brief history that presents advances in flow mapping. Besides, visual clutter reduction techniques and existing flow mapping software are reviewed for better identifying maturity and capabilities of these applications. Based on the findings of Chapter 3, flow scenario to be focused in FlowMapper is identified as node-to-node flows where actual route is negligible. Besides several requirements to be met in FlowMapper are identified as GIS integration, user friendly GUI and installation, multi platform support and inclusion of visual clutter reduction methods.

The aim of this study involves development of a free and open source flow mapping application that is fully integrated to GIS. Thus, before starting development, fundamentals of open source development should be investigated and the GIS component to integrate the FlowMapper application should be determined.

From the perspective above, at the beginning of this chapter, a general overview is given regarding definitions and concepts in free and open source software. Then, existing free and open source GIS applications are reviewed with respect to their popularity, development environments, technical, economic and marketing potentials. Besides, as a free and open source desktop GIS application, QGIS is reviewed in detail. At the end of this chapter, motivation for free and open source development and reasons for selecting QGIS as the main component to develop the FlowMapper plugin are presented.

3.1. Overview of Free and Open Source Software Concept

Owing to valuable development efforts made by open source developers in the last few decades and anonymous contributions from open source software communities; today free and open source software gamut offers such great diversity that it is almost possible to find more than one free and open source software almost for any user for any purpose. There are such successful, mature open source projects that some of these software are competing with their well known proprietary, commercial alternatives. In Table 3.1, well known commercial products are listed with their popular free and open source alternatives.

Table 3.1. Free and Open Source Software Alternatives to Well Known Commercial Software Products

Category	Commercial & Proprietary Software	Free & Open Source Alternative
Operating System	Microsoft Windows Apple Mac OS X iOS	Linux Distributions such as Ubuntu, openSUSE, Pardus Android
Offfice Suite	Microsoft Office for Windows Microsoft Office for Mac	LibreOffice NeoOffice
Database	Microsoft SQL Server Oracle Database + Spatial	MySQL PostgreSQL + PostGIS
GIS	ESRI ArcGIS Desktop, MapInfo	GRASS, Quantum GIS, uDig, openJUMP
Multimedia	Adobe Photoshop Adobe Illustrator Adobe Premiere Adobe Audition Windows Media Payer	The GIMP Inkscape Avidemux Audacity VLC Media Player
Desktop Publishing	Adobe PDF Reader (Acrobat)	PDFCreator
Web Browser & Mailing Client	Microsoft Internet Explorer Microsoft Outlook	Mozilla Firefox Mozilla Thunderbird
Utilities	CuteFTP WinZip	FileZilla 7Zip

In its most abstract form, free and open source software, sometimes referred to as free / libre open source software, is a computer program that is developed by the collaborative efforts of programmers and both the software and its source code are licensed free of charge which encourages utilization of the software and modifications, improvements to the source code.

It is possible to find similar descriptions on the websites of Free Software Foundation (FSF) and Open Source Initiative (OSI). Free Software Foundation is a nonprofit organization established in 1985 to support the free software movement. According to FSF, free software means software that respects users' freedom and community. Thus, a program can be called as free software if the users have the freedom to run, copy, distribute, study, change and improve the software (FSF, 2013a). Similarly, Open Source Initiative is a nonprofit organization established in 1998 to promote development and distribution of open source software. However, OSI has coined a new term "open source" alternative to the term "free software" due to confusion of the word "free" in English. According to OSI (2013a), free software and open source software are two terms for the same thing; software released under licenses that guarantee a certain, specific set of freedoms. Thus, in general, terms "free software" and "open source software" can be used interchangeably. In this study, as a combination of these two terms, the term "free and open source software" is preferred and used.

Although both FSF(2013a) and OSI (2013a) make quite similar definitions for the free and open source software concept; they disagree when defining the levels of freedom and distribution terms that software must conform. FSF (2013a) prefers an abstract, four point definition for software freedom while OSI (2013b) prefers a more detailed, ten point definition. In practice, either approach server the same goal; promoting open source development and keeping the software free but use different language.

General Public License (GPL), which is the most widely used free software license (OSRC, 2013), originally written for the GNU Project (an acronym for GNU is

Not UNIX) by Richard Stallman who is the founder of Free Software Foundation (FSF), guarantees four essential levels of freedom to end users (FSF, 2013a). These are;

Freedom 0: To run the program, for any purpose,
Freedom 1: To study how the program works,
Freedom 2: To redistribute copies so you can help your neighbor,
Freedom 3: To distribute copies of your modified versions to others.

In addition to have unrestricted access to source code, OSI (2013b) defines the distribution terms that open source software must comply as follows;

1. Free redistribution without any loyalty or fee
2. Providing source code as well as compiled form
3. Allowing modifications and derived works
4. Integrity of the Author's Source Code
5. No Discrimination against Persons or Groups
6. No Discrimination against Fields of Endeavor
7. Distribution of License to all to whom the program is redistributed
8. License Must Not Be Specific to a Product
9. License Must Not Restrict Other Software
10. License Must Be Technology Neutral

In the free and open source software environment, it carries significant importance to distinguish between these two terms; “free” vs. “freeware”. Freeware is software which is available free of charge but it is copyrighted by its developer who retain the rights to modify and distribute it (The Linux Information Project, 2006). Freeware distributions typically lack human readable source code, thus any modifications by contributing users is unfeasible. Freeware software licenses usually include specific expressions in EULA (End User License Agreement) such as “can be freely copied but not sold” or “free for personal usage only” or “prohibition of use by the military” etc. all of which limit redistribution. A quick comparison between freeware and free (open source) software concept is presented in Table 3.2.

Table 3.2. Freeware vs. Free & Open Source Software Concept

	Freeware	Free and Open Source
Description	Software that can be used free of charge	Software that can be used, modified and redistributed without any restrictions.
License & Copyright	Distributed with EULA that is specific to the software. Copyright laws are applicable to all components of the software	Most popular one is GNU GPL License. Usually only the name of the software may be subject to copyright.
Features	All features are free of charge. Otherwise it is called shareware or commercial.	All features are free of charge
Distribution	Can be distributed free of charge or maybe subject to limited distribution for different usage purposes (e.g. commercial, military)	Can be distributed freely for any purpose
Source Code	Proprietary. Typically not available and subject to copyright rights.	Always available to public access free of charge.
Example	Adobe PDF Reader, Adobe Flash Player, Google Talk, Google Earth, Faststone Image Viewer	Mozilla Firefox, GIMP, VLC Media Player, Inkscape, Notepad++, Quantum GIS

In its most basic form, source code is a human readable statement of the intent and mechanism of the program (Darrell, 1991). Source code, as a human readable text, includes data structures and algorithms; thus it provides many clues about how the program works and evolves. Practically, if the source code of software is freely accessible; anyone who has enough programming skills can develop new features by modifying the source code and improve the program. In the free and open source software environment, source code is typically created with the collaborative efforts of community developers. If a developer improves the existing source code, other community developers start using the improved code written by previous developer and throughout the lifecycle of software this cycle loops. For any free and open source software project, providing unrestricted, free access to source code has vital importance for the continuity of the project. There are more than 60 open source licenses approved by OSI (2013c), each of which defines different rights for their users. GNU General Public License (GPL), which is listed as the most commonly used open source software license by OSRC (2013), legally ensures this requirement by keeping the source code open and giving right to anyone to develop without needing to receive any permission from any of the previous developers. However, Berkeley Standard Distribution (BSD) License, which is listed as the fifth most popular open source software license by OSRC (2013), permits development of proprietary derivative works from free and open source software. Vries et al. (2008) comment on BSD license as providing maximum level of freedom in using the source code as well as giving right for development of closed source, proprietary software. However, this type of freedom also brings risk of not opening the source code of derivate works.

Proprietary, closed source software is protected with copyright laws and typically distributed in compiled binary form which is not human readable and can only be interpreted by computers. As well as the copyright laws, binary distribution completely disables access to the source code and prevents any unauthorized modifications to the software. In the proprietary software market, nobody except the commercial developer of the software has authorization to modify the source code and privatization of the proprietary software is guaranteed with copyrights. On the contrary, copyleft concept, pioneered by Richard Stallman, utilizes the principles of copyright law but to keep the software free and open source. Generally, copyleft is a method for making a program free and requiring all modified and extended versions of the program to be free as well (FSF, 2013b). For example, popular GNU General Public License promotes the copyleft concept. However, some other free and open source software licenses such as BSD, MIT and Apache Licenses are not copyleft type licenses.

One of the intentions of this study is to provide the flow mapping software to any GIS user free of charge and keeping the source code accessible for possible contributions in order to strengthen the continuity of software. Having the motivation of developing domain specific software, it is logical to take the advantage of previous development efforts by studying the source code of existing similar software. Through this approach, there is a possibility of reusing the code rather than starting development from scratch. It is clear that this type of development can only be ensured in free and open source environment.

3.2. Development in Free and Open Source Environment

Free and open source software is a result of collaborative development efforts. Thus projects without an active community cannot have a long lifetime. If the project has enough potential, it attracts attention of users and quickly builds an active community. Project with an active community quickly reaches a mature state. However, building an active community requires organizational and management skills as well as development efforts.

Eric Raymond, a computer engineer and a well known open source advocate, offers a model for open source software development known as the bazaar model (Raymond, 1999). Contrary to the traditional model of software development, in which roles are clearly defined; in the bazaar model roles are not clearly defined and development takes place in a decentralized way. According to Robles (2004), software developed using the bazaar model typically displays the following patterns:

> (i) Users should be treated as co-developers: Having free access to source code and assuming that some users hold programming skills each user is a potential co-developer and should be encouraged to submit improvements to the software such as code fixes, bug reports, documentation. It is clear that having more co-developers and an active community increase the rate at which the software evolves since each user machine provides an additional test platform.

(ii) Early releases: To increase the level of user participation and to create an active community, initial version of the software should be released as early as possible.

(iii) Frequent integration: Code changes should be frequently merged into a shared code base. By this approach, additional workload of fixing large number of bugs prior to major releases can be avoided or reduced.

(iv) Several versions: Robles (2004) mentions the need of two different versions in open source software development. First one is the buggier, development version which offers more but experimental features for test users and co-developers. Second one is the stable version with less but tested functionality.

(v) High modularization: Structure of the software should allow parallel development on software components.

(vi) Dynamic decision making structure: A dynamic structure is needed to make strategic decisions depending on user requirements.

Free and open source development philosophy, discussed by Raymond (1999) and Robles (2004) with the bazaar model, is quite similar to agile development. Agile development methodology can be defined with the following characteristics (Warsta and Abrahamsson, 2003):

Incremental: Small releases with rapid cycles
Cooperative: Developers constantly work together with the customer.
Straightforward: Method is easy to learn and to modify
Adaptive: Tolerating last minute changes.

In their study, Warsta and Abrahamsson (2003) discuss how open source software development paradigm complies with the characteristics of agile development.

Although there are arguments on how open source development differs from agile development in philosophical, economic and team structural aspects; Warsta and Abrahamsson (2003) find open source development method rather close to agile software development.

During the development of open source software there are plenty of voluntary tasks to be performed by the community. Preparation of user manuals, tutorials, test data, translation of documents and graphical user interface (GUI), providing technical support via email lists, management of project website and user forum, bug reporting, performing initial tests prior to release, preparing installer packages are tasks those rely on voluntary contributions.

Practically in free and open source projects, similar to agile development which prioritizes working software over comprehensive documentation (Beck et al., 2001), speed of development is beyond the speed of documentation. Thus, contributions from especially advanced, experienced users are expected to prepare valuable resource material for new, novice users such as tutorials, user guides. Besides translations of these documents are usually prepared by voluntary efforts of community members who are native in the target language to which the document is being translated.

Discussion forums and email lists are valuable sources for getting support. They are usually divided into categories according to their scope such as; developers section for discussion of development issues, users section for questions regarding installation and usage of the program, translation section that deals with translation of GUI and documentation, bugs section where users can submit bug reports, community team or project committee section for management of web sites, blogs, mailing lists and to discuss future strategy of the software. By means of discussion forums and email lists members can help each other and keep the community active. After registering on forums or joining emailing lists users can follow development progress and post messages for requesting help. However, there is also no guarantee that all questions or support requests will be answered or met since contribution to forums and emailing lists are totally voluntary. In compliance with the spirit of freedom, in open source projects, registrations on forums and subscriptions to emailing lists are free of charge and open to anyone without any discrimination.

In free and open source software projects, it is a must that source code is accessible from internet to any user free of charge. The term codebase refers to collection of source code used to compile a software (adapted from Janssen, 2013). Typically codebase includes only human readable source code. Codebase is stored in source control repositories and edited by developers. Code repository stores large amounts of source code and utilized in multi-developer projects for committing code changes and managing software versions. According to Ohloh (2013), which is a free, public directory of open source software, Subversion, GIT and CVS are the three most popular version control systems distributed under an open source license. In open source projects, all users have read-only access to repositories however community developers have write permissions as well. Improvements to the source code are made by community developers unless it contradicts the aim and scope of the project. By utilizing version control system, community developers get a chance of working as a team on the same source code simultaneously.

Preparation of an installation package is needed in order to ease the installation process for users. A package usually includes an application installation file known as setup. Since setup files are platform dependent, source code should be compiled separately for each target operating system. For example, popular open source desktop GIS software, Quantum GIS offers different installers for different platforms such as executable (exe) setup file for Windows OS, DMG installer for Mac OS X, DEB installation package for Debian and Ubuntu OS, RPM installation package for Mandriva and Fedora OS, APK installation package for Android Mobile OS (QGIS, 2013b). Similar to other development tasks, preparation of setup files and installation packages in open source projects are performed by voluntary contributions of release or packaging team.

Typically, a community gathering around an open source project is not homogeneous (Cascadoss, 2007a). Not all users are interested to become actively involved in the project; besides not all users in the community have programming skills. Mockus et al. (2000), who examined Apache web server development process, found that about %80 of the source code was written by contributions of nearly 15 core developers. Besides, a group of co-developers, nearly tenfold bigger than the core developers group, were fixing bugs while another group of users, which is nearly tenfold larger than the co-developers group, were reporting bugs in software. In their study, Mockus et al. (2000) estimated nearly half million Apache web server installations. However, total number of users submitting bug reports was about 3.000 which indicates an active participation rate much lower than %1. Based on these findings, community of an open source project exhibits a three layered structure. In the first layer, core developers are surrounded by a larger group of co-developers and testers. In the second layer, there are active users submitting bug reports and providing support on forums and email lists. In the third layer, largest group of users, passive consumers of the software exist. Mockus et al. (2000) identify that although only a small part of the community actively participates in the project, numerically it is still greater than the number typically seen in the development of comparable proprietary software.

In free and open source development, continuity of a project can only be guaranteed with an active community. Otherwise, project will end without reaching enough maturity due to insufficient interest and participation. In order to become a sustainable project, a fair and motivating governance structure should be constructed and shared with the community. Leadership in free and open source projects should be meritocratic. That means, members who make more valuable contributions to project have more to say regarding the future of the project. Ideas are freely discussed, voted between members on forums and emailing lists then strategic decisions are taken by consensus. Project leaders must be responsive to user concerns and ensure consensus. Otherwise, a group of developers may "fork" the project which is briefly copying the source code to start a competing project (Fogel, 2006). Generally, forking causes duplication of work and separation of communities. Thus, unless there is a valid reason, forking should be avoided. A recent and well known forking activity was experienced in the multi platform, popular office suite OpenOffice due to Oracle's (former Sun Microsystems) central role in the project (Paul, 2010). As a result, a group of key contributors to OpenOffice formed a new organization called Document Foundation and forked the project in 2010 with LibreOffice project having the intension of liberating the project from Oracle's control. Fogel (2006) explains the main reason for not being true dictators in free and open source projects with this forking threat and states that "Replicability implies forkability; forkability implies consensus."

In commercial software market, generating revenue directly from the proprietary software by selling license is perceived as the key factor for a successful business model (Cascados, 2007a). However, in the free and open source software environment, there are other business models to get financial support since selling an open source program with a commercial license is not realistic. In Table 3.3, a list of open source business models collected either from the literature or observed in the market is presented with their brief descriptions.

Table 3.3. Free and Open Source Software Business Models (adapted from Cascados, 2007a)

Model Name	Alternative Naming	Description	Example
Dual Licensing	Twin Licensing	Companies offer their software with two types of licenses; free and commercial	MySQL, Nokia (Trolltech) Qt
Support Seller	Product Specialists, Software Provider (GPL)	Companies provide paid support for free and open source software that they developed	Support for Red Hat Linux Distribution
Third Party Support Seller	Third Party Service Provider	Companies provide paid support for free and open source software that are not developed by themselves	Support for PostgreSQL from EnterpriseDB
Platform Providers	Distributor	Companies bundle several free and open source software into a complete platform and guarantee the quality of integrated platform	Red Hat, Novell

Table 3.3. Free and Open Source Software Business Models (cont.) (adapted from Cascados, 2007a)

Model Name	Alternative Naming	Description	Example
Consulting	N/A	Companies provide consultancy for free and open source software based on their knowledge	
Software as a Service	Hosted Strategy	Free and open source software is used to provide access to revenue generating online services	Google
Added Value Providers	Software Provider (non-GPl)	Companies create proprietary software derived from free and open source software	Apple, EnterpriseDB
Accessorizing	N/A	Selling services or physical items related to free and open source software (books, hardware, gadgets)	SourceForge, O'Reilly
Donation	N/A	Users or sponsors may donate money to project on a volunteer basis	SourceForge, Ubuntu, QGIS
Loss Leader	Split OSS / Commercial Products	Free software is used to promote a commercial, proprietary software	Qualcomm's Eudora email client
Widget Frosting	Optimization Strategy	Companies sell high value bundled software and hardware components with low cost free and open source software.	Apple Darwin on MacOSX Server, OracleDB on Linux

In Table 3.3, only pure forms of business models are listed. In practice, companies mix and adapt several business models in order to create their own sustainable models. Among these models, the most popular open source models to create business value are the Support Seller and the Platform Provider Models (Cascados, 2007a).

Voluntary contributions and financial support are two important factors for the continuity of an open source project. In their study, Shapiro and David (2008) tried to identify motivating reasons for developers to contribute to an open source project. In Table 3.4, motives for development and motivation reasons for choosing a project are given respectively from the most motivating to the less.

Table 3.4. Motives for Development and for Choosing a Project (adapted from Shapiro and David, 2008)

Motives for development *	Motives for choosing a project *
We should be free to modify the software we use	Software being developed would be useful to me
As FS user, wanted to give back to community	Technically interesting
Wanted to provide alternatives to proprietary	I launched the project
Way to become better programmer	Important and visible project
Best way for software to be developed	Knew people working on it
Needed modification of existing software	Other reasons
Needed to fix bugs in existing software	(*) Factors are listed from the most motivating (on top row) to the less (on bottom row)
Wanted to interact with like-minded programmers	
Wanted to learn how particular program worked	
Liked challenge of fixing bugs in existing software	
Employer wanted me to collaborate in OS	

3.3. Review of Free and Open Source GIS Software

Free and open source software is becoming increasingly more reliable than it was in the past (Chen et al., 2010). Similarly, existing free and open source GIS software is entering a phase of refinement and enhancement (Ramsey, 2007). As an inevitable result of this phenomenon, free and Open Source GIS software became to provide competing alternatives to proprietary software and now exists at every level of the spatial data infrastructure stack.

Steiniger and Bocher (2009) clarify the reasons of this growing popularity in free and open source GIS software with reference to four indicators. First is the number of projects initiated in the last couple of years. For example, in the last 5 years, nearly 20 new GIS software were added to the list hosted on FreeGIS.org. Second indicator is the increasing financial support by governmental organizations. Third one is listed as the noticeable download rates of free and open source GIS software. For example, during 2012, QGIS installer for Windows was downloaded more than half million times (QGIS, 2013c). Last indicator is given as the increasing number of use cases in free and open source GIS software such as desktop GIS applications, spatial database applications, web GIS toolkits and development libraries which were all identified by Ramsey (2007).

In this part of the study, characteristics of different free and open source desktop GIS software are investigated with the intention of identifying the most mature and popular products which could provide suitable environment for the development of flow mapping plugin. Although all free and open source projects are valuable and include great effort, identification of mature and popular products is potentially important since recognition and utilization level of the plugin will be a derivative of the community size and popularity of selected GIS application. Besides, greater and active community practically means more chance of reaching help during development process and also documentation is more complete in mature projects.

Recent taxonomy and evaluation studies regarding GIS related free and open source software are performed by Cascadoss (2007b), Ramsey (2007), Steiniger and Bocher (2009), Chen et al. (2010). FreeGIS.org, MapTools.org, OpenSourceGIS.org and OSGeo.org are several web resources that list and promote widespread use of GIS related free and open source software. These software lists include all types of GIS applications ranging from less known to very popular mature projects. In September 2008, 339 and in August 2010, 351 open source GIS software were listed in FreeGIS.org (Kepoğlu, 2010) while this number increased to 356 in May 2013. Similary, in May 2007, 238 and in September 2008, 256 open source GIS software were listed in OpenSourceGIS.org (Kepoğlu, 2010) while this number reached up to 350 in May 2013. Based on these indicators, as also mentioned in Steiniger and Bocher's (2009) study, it is clear that open source development has boosted and the number of GIS software projects being initiated shows a growing trend.

Open Source Geospatial Foundation (OSGeo), founded in 2006, is a non-profit organization that provides financial, organizational and legal support to the open source geospatial projects. GIS related free and open source projects those being officially supported by OSGeo and also other geospatial software, announced as incubating projects as of June 2013, are listed in Table 3.5.

Table 3.5. Open Source Geospatial Projects Supported by OSGeo and Projects in Incubation Process (adapted from OSGeo, 2013)

	Supported Projects	Incubating Projects
Web Mapping	deegree, geomajas, GeoMoose, GeoServer, Mapbender, MapBuilder, MapFish, MapGuide Open Source, MapServer, OpenLayers	ZOO-Project, Team Engine
Desktop Applications	GRASS GIS, Quantum GIS	gvSIG, Opticks, Marble
Geospatial Libraries	FDO, GDAL/OGR, GEOS, GeoTools, OSSIM, PostGIS	MetaCRS, rasdaman
Metadata Catalogs	GeoNetwork	pycsw
Outreach Projects	Public Geospatial Data, Education and Curriculum, OSGeo Live	

MapTools.org is another web resource for users and developers interested in open source geospatial applications. As of June 2013, GIS related software either hosted or listed in MapTools.org are given in Table 3.6.

Table 3.6. Open Source Geospatial Projects Hosted or Listed by MapTools.org (adapted from MapTools.org, 2013)

	Project
Packaged Tools and Utilities	MS4W, OS Geo-Live
Web Tools	CartoWeb, Chameleon **(Hosted)**, Dracones, Fusion, GeoEXT, GeoMOOSE, Ka-Map **(Hosted)**, Mapbender, MapFish, MapServer, OpenLayers, OWTChart **(Hosted)**, PHP MapScript **(Hosted)**
Desktop Tools	GRASS, OpenEV, Quantum GIS, uDig
Utilities and Libraries	AVCE00 **(Hosted)**, DGNLib **(Hosted)**, GDAL, GeoTiff, MITAB **(Hosted)**, OGR, Proj4, Shapelib **(Hosted)**

Ramsey (2007) presented an overview of free and open source GIS projects by making taxonomy with respect to the programming language that the project is built on such as C, JAVA, .NET. Besides, Ramsey (2007) added a fourth category to the taxonomy for the web applications which do not fall into a language category. Web applications include various toolkits and web services which provide browser based interface to spatial services (e.g. mapping servers). Applications, libraries, toolkits, frameworks and server based applications in this taxonomy study are listed in Table 3.7.

Table 3.7. Taxonomy of Free and Open Source Geospatial Projects with Respect to Programming Languages (adapted from Ramsey, 2007)

<table>
<tr><th>Category</th><th colspan="2">Shared Libraries</th><th colspan="2">Applications</th></tr>
<tr><td><b>C / C++</b></td><td colspan="2">GDAL/OGR, Proj4, GEOS, Mapnik, FDO</td><td colspan="2">MapGuide Open Source, UMN Mapserver, GRASS, QGIS, OSSIM, TerraLib, GMT, PostGIS</td></tr>
<tr><td><b>JAVA</b></td><td colspan="2">JTS Topology Suite, GeoTools</td><td colspan="2">GeoServer, degree, JUMP, OpenJUMP, Kosmo, gvSIG, OpenMap, uDig</td></tr>
<tr><td><b>.NET / C#</b></td><td colspan="2">NTS, Proj.Net, SharpMap</td><td colspan="2">WorldWind, MapWindow</td></tr>
<tr><td rowspan="2"><b>WEB Projects</b></td><td><b>Toolkits</b></td><td colspan="2"><b>Frameworks</b></td><td><b>Servers</b></td></tr>
<tr><td>MapBuilder, ka-Map, OpenLayers</td><td colspan="2">Mapbender, Cartoweb</td><td>TileCache, FeatureServer</td></tr>
</table>

Steiniger and Bocher (2008) present an overview of free and open source projects strictly focused on developing desktop GIS software. Desktop GIS software is a mapping application which is installed and runs on a personal computer or a workstation and allows users to display, edit, query and analyze geographic data and information linked to locations (ESRI, 2013). Project evaluation criteria referred to Steiniger and Bocher (2008) are presented in Table 3.8.

Table 3.8. Evaluation Criteria for Free and Open Source Desktop GIS Software (adapted from Steiniger and Bocher, 2009)

Criteria	**Explanation & Possible Entries**
Application Focus	Viewing, Editing, Analysis
User Level	Novice (Viewing), Experienced (Editing, Basic Analysis), Expert (Analysis), Research (Scripting, Programming)
Supported OS	MS Windows, Linux, Apple Mac OS X
License	License of core platform. GPL, LGPL etc.
Development Platform	C, C++, C#, JAVA, VB .NET, QT4, Python
Main Data Types	Supported vector & raster formats. SHP, DXF, GeoTIFF, ECW etc.
Features, Functionality	Database Connection Support, Thematic Mapping, Scripting, Coordinate Transformations, Topology, Advanced Editing, Surface Operations and 3D analysis
Supported OGC Standards	WMS, WFS, WCS, SFS, GML, WMC etc.
Development API	Existence of development API

With reference to criteria listed in Table 3.8, review of project characteristics and comparison of functional capability among free and open source desktop GIS applications are respectively given in Table 3.9 and 3.10. Besides, to enable a comparison of the functional capability between open source and a well known commercial product, ArcView 10.1 is added to Table 3.10 as the base product of ESRI ArcGIS Desktop architecture. Both tables are adapted from Steiniger and Bocher (2008) but updated to meet the current versions of software as of May 2013.

Table 3.9. Project Characteristics of Selected Free and Open Source Desktop GIS Software (adapted from Steiniger and Bocher, 2009; Ramsey, 2007)

Software Year Founded Project Website	**Application Focus**	**User Level**	**Supported Operating Systems**	**Development Platform**	**Development by**	**License**
GRASS 1982 grass.osgeo.org	Analysis & Scientific Visualisation, Cartography, Modelling & Simulation	Experienced, Expert, Research	MS-Win, Linux, MacOSX	C, Shell, Tcl/Tk, Python	Research Institutes, Universities, Companies, Volunteers Worldwide	GPL
QGIS 2002 qgis.org	Viewing, Editing, Analysis, GRASS GUI	Novice Experienced, Expert, Research	MS-Win, Linux, MacOSX	C++, Qt4, Python	Universities, Companies, Volunteers Worldwide	GPL
uDig 2004 udig.refractions.net	Viewing, Editing, Analysis,	Novice Experienced, Expert, Research	MS-Win, Linux, MacOSX	JAVA Eclipse RCP	Companies, Organizations, Volunteers	Core Eclipse RCP is EPL
gvSIG 2003 gvsig.org	Viewing, Editing, Analysis,	Novice Experienced, Expert, Research	MS-Win, Linux, MacOSX	JAVA	Companies, Universities, Government	GPL
SAGA 2001 saga-gis.org	Analysis, Modeling, Scientific Visualization	Novice Experienced, Expert, Research	MS-Win, Linux,	MS Visual C+ +	Universities	LGPL (API), GPL
MapWindow 1998 mapwindow.org	Core GIS Functions, Developing Decision Support Systems	Novice Experienced, Expert, Research	MS-Win	C+ +, C#, VB.NET MS Visual Studio .NET	Universities, Companies, Volunteers Worldwide	Mozilla Public License v1.1
OpenJUMP 2002 openjump.org	Viewing, Editing, Analysis,	Novice Experienced, Expert, Research	MS-Win, Linux, MacOSX	JAVA	Volunteers Worldwide	GPL

Table 3.10. Functional Characteristics of Selected Free and Open Source Desktop GIS Software (adapted from Steiniger and Bocher, 2009; functionality chart updated to meet current releases of software)

Functionality			**GRASS v6.4.2**	**QGIS v1.8.0**	**uDig v1.4**	**gvSIG v1.12**	**SAGA v2.0.8**	**Map Window v4.8.6**	**Open Jump v1.6.3**	**Arc View v10.1**
Vector Data	Read	SHP	Y	Y	Y	Y	Y	Y	Y	Y
		GML	Y	Y	Y	Y	-	$Y^{(1)}$	Y	Y
		DXF	Y	Y	-	Y	Y	$Y^{(1)}$	$Y^{(1)}$	Y
	Write	SHP	Y	Y	Y	Y	Y	Y	Y	Y
		GML	Y	Y	Y	Y	-	$Y^{(1)}$	$Y^{(1)}$	$Y^{(4)}$
		DXF	Y	Y	-	Y	-	-	$Y^{(1)}$	Y
Raster Data	Read	JPEG	Y	Y	Y	Y	Y	Y	Y	Y
		ECW	$Y^{(2)}$	Y(Win)	$Y^{(2)}$	Y	-	Y	$Y^{(1)}$	Y
		GeoTIFF	Y	Y	Y	Y	Y	Y	Y	Y
		ArcInfo GRID	Y	Y	Y	Y	Y	Y	-	Y
	Write	JPEG	Y	Y	Y	Y	Y	Y	Y	Y
		ECW	$Y^{(2)}$	$Y^{(2)}$	$Y^{(2)}$	Y	$Y^{(2)}$	$Y^{(2)}$	-	$Y^{(3)}$
		GeoTIFF	Y	Y	Y	Y	Y	Y	-	Y
		ArcInfo GRID	Y	Y	Y	Y	Y	Y	$Y^{(1)}$	Y
Database Access R:Read W:Write		PostGIS	R+W (1) Limited	R+W	R+W	R+W	-	R+W (1)	R+W (1) Limited	R OLE DB
		Oracle	R	R+W (1)	R+W	R+W	-	-	R(P)	R OLE DB
Compliance to OGC Supported Standards			WMS WFS GML WPS (via pyWPS)	WMS WFS(-T) SFS (via PostGIS) GML KML	WMS WFS(-T) SFS GML WPS SLD	WMS WFS(-G) WCS GML KML CSW SLD	Under Dev. WMS, WFS, WCS	$WMS^{(1)}$ $WFS^{(1)}$	WMS $WFS\text{-}T^{(1)}$ SFS GML WPS SLD (dee JUMP)	WMS(-T) $WFS^{(1)}$ WCS $CSW^{(1)}$ SFS GML KML $WMC^{(1)}$
Thematic Mapping (Bar chart, graduated symbols, individual values, pie chart, labels)			Y	Y	- (Labels Only)	Y (Simple)	Y	- (Labels Only)	$Y^{(1)}$	Y
Developer API			Y	Y	Y	-	Y	Y	Y	-
Scripting Functionality			Bash Python Perl	Python	Groovy (1)	Jython	Python Console	C# VB.NET	Beanshell Jyhon (1)	Python Model Builder

Table 3.10. Functional Characteristics of Selected Free and Open Source Desktop GIS Software (cont.) (adapted from Steiniger and Bocher, 2009; functionality chart updated to meet current releases of software)

Functionality	**GRASS v6.4.2**	**QGIS v1.8.0**	**uDig v1.4**	**gvSIG v1.12**	**SAGA v2.0.8**	**Map Window v4.8.6**	**Open Jump v1.6.3**	**Arc View v10.1**
Coordinate Transformations Projections	Y	Y	Y	Y(1)	Y	Y	Y(1)	Y
Creating & Editing Vectors (2D graphics incl. points, lines, area tools, snapping, coordinate input)	Y	Y	Y	Y	Y	Y	Y	Y
GPS Support	Y	Y	Y(1)	-	Y	Y	Y(1)	Y
Topology (Creating link, node, chain and polygon topology)	Y	Y(1)	-	Y	- (TIN Only)	- (TIN Creation API Only)	Y (Limited)	-
Advanced Data Creation & Editing (Creating offset, line generalization, trimming, rotating, intersection)	Y	Y	Y	Y(1)	Y	Y	Y	Y (Except Offset, Line Gen.)
Advanced Thematic Mapping (Thiessen polygon analysis, grid analysis, contour generation)	Y	Y(1)	-	Y(1)	Y	Y	- Only TIN	Y(4)
Creating 3D Views & Terrains (TIN, DTM, shading, surface draping)	Y	Y(1)	-	Y(1)	Y	Y	Y(1) Only TIN	Y(4)
Viewshed & Terrain Analysis (line-of-sight, slope, aspect, gradient)	Y	Y(1)	-	Y(1)	Y	Y	-	Y(4)

(1) Functionality is provided by a plugin.
(2) Erdas ECW SDK, which is needed for ECW read & write support, is not compiled by default in GDAL. Erdas ECW SDK is commercial, proprietary software and users are subject to EULA.
(3) Write to ECW functionality is provided via ERDAS ECW Plugin for ArcGIS Desktop. This plugin is commercial, proprietary software and users are subject to EULA.
(4) ESRI ArcGIS Desktop Extension is needed; such as Spatial Analyst, 3D Analyst.

Based on Table 3.10, as also concluded in Steiniger and Bocher's (2009) study, some free and open source desktop GIS applications, such as GRASS, QGIS and gvSIG, offer more functionality than ArcView 10.1. In order to asses the popularity of each application, free and open source desktop GIS projects referred either on web resources or in the literature are listed in Table 3.11. Parallel to Steiniger and Bocher's (2009) conclusion, based on Table 3.11, GRASS, QGIS, gvSIG and uDig are found as the most cited projects on web resources and in the literature.

Table 3.11. Free and Open Source Desktop GIS Software Cited on Web Resources and in the Literature.

	OSGeo .org (2013)	MapTools .org (2013)	Cascadoss .eu (2007b)	Ramsey (2007)	Steiniger & Bocher (2009)	Count
FMaps			X			1
GRASS	X	X	X	X	X	5
gvSIG	X		X	X	X	4
Jump or OpenJump			X	X	X	3
Kosmo			X	X	X	3
MapWindow			X	X	X	3
OpenEV		X	X			2
OpenMap			X	X		2
OSSIM			X			1
QuantumGIS	X	X	X	X	X	5
SAGA			X		X	2
Thuban			X			1
uDig		X	X	X	X	4

For identifying the most mature free and open source desktop GIS software that offers more functionality than other candidates, a side by side feature comparison chart (Table 3.10) or a list showing how much the software is cited (Table 3.11) may provide an insight but neither can supply enough evidence for decisive selection. Thus different aspects of desktop GIS must be evaluated and graded with respect to a set of objective criteria. Financed by the European Commission under the Sixth Framework Program, such a study was performed by Cascadoss Project for the development of transnational cascade program on open source GIS and RS software for environmental applications (Cascadoss, 2007b).

Cascadoss project evaluates free and open source GIS and RS related software with respect to their (i) marketing potential, (ii) technical potential and (iii) economic potential. Indicators for evaluating the marketing potential are number of users, existing market share, strength of community, level of support and business models that is possible with the software license. Technical potential of any software is usually a derivative of the quality of development. Thus, Cascadoss project refers to the quality model defined by ISO 9126 which is an international standard for the evaluation of software quality. Economic potential of free and open source software can be measured by assessing the total cost of migration from any existing proprietary system and total cost of savings that can be made by choosing open source systems. Multi criteria decision making process is involved in Cascadoss project to determine the marketing, technical and economic potential of each software. Set of evaluation criteria and weight factors involved are listed in Table 3.12.

Table 3.12. Evaluation Criteria and Weighted Scoring Utilized in Cascadoss Project (Cascadoss, 2007b)

Evaluation Criteria for Marketing Potential	**W. Max. Score**	**Evaluation Criteria for Technical Potential**	**W. Max. Score**	**Evaluation Criteria for Economic Potential**	**W. Max. Score**
Max. Sum. 60 Points	Weight Max. Score	Max. Sum 60 Points	Weight Max. Score	Max. Sum 60 Points	Weight Max. Score
Maturity of Project	**15**	**Functionality**	**15**	**Cost of installation**	**24**
Version control	W 5 MS 3	Suitability	W 5 MS 3	Cost of deployment	W 8 MS 3
Mailing list		Accuracy		Cost of labor	
Documentation		Interoperability		**Cost of Migration**	**18**
Testing method		Security		Cost of project migration	W 6 MS 3
Portability		**Reliability**	**9**	Cost of labor that takes part in the operative work of migration	
Strength of Community	**15**	Maturity and product history	W 3 MS 3	Cost of software customization to migrate existing data	
Producer community	W 5 MS 3	Robustness		**Cost of operation**	**18**
User community		Fault Tolerance		Suitability for users	W 6 MS 3
Vendors community		Recoverability		Cost of labor for system administration	
OSS distributor community		Reliability Compliance		Training for the newcomers	
Foundations and non-profit organizations		**Usability**	**9**	Cost of interruption by software weakness	
Governmental agencies		Understandability	W 3 MS 3	Costs of version control	
Overall level of the support		Learn ability			
Market Share	**12**	Documentation			
Popularity of the software	W 4 MS 3	Availability			
		Operability			
Legal / License issues	**9**	Attractiveness			
License of the software	W 3 MS 3	**Efficiency**	**9**		
		Time behavior	W 3 MS 3		
Collaboration with Other Projects	**9**	Resource utilization			
Synergy between OSS products	W 3 MS 3	Runtime			
		Efficiency compliance			
		Maintainability	**9**		
		Analyzability	W 3 MS 3		
		Changeability			
		Stability			
		Portability	**9**		
		Adaptability	W 3 MS 3		
		Coexistence			
		Installability			
		Replaceability			

Cascadoss project (2007b) groups evaluated free and open source software into four headings: (i) desktop GIS and RS applications, (ii) development libraries, (iii) server applications and (iv) environmental applications. However in this thesis, only desktop GIS applications are considered; desktop RS applications, development libraries, server and environmental applications are not taken into consideration. Desktop GIS applications evaluated in Cascadoss project (2007b) are; (i) FMaps, (ii) GRASS, (iii) gvSIG, (iv) JUMP, (v) Kosmo, (vi) OpenEV, (vii) OpenMap, (viii) OSSIM, (ix) QuantumGIS, (x) Saga GIS, (xi) Thuban and (xii) uDig.

As stated earlier, the aim of this thesis is to develop a free and open source flow mapping application that is integrated into GIS. This integration can be effectively achieved on existing desktop GIS platforms which support plugin architecture. With reference to the evaluation performed by Cascadoss project, it is possible to identify suitable open source desktop GIS applications to which the flow mapping plugin can be integrated. With reference to evaluation criteria and weighted scoring principles given in Table 3.12, evaluation results regarding desktop GIS software are given in Table 3.13 with respect to their marketing, technical, economic potentials. Besides, a fourth evaluation column is added in order to present combined potential of each application.

Table 3.13. Marketing, Technical, Economic and Cumulative Potential of Free and Open Source Desktop GIS Applications Evaluated in Cascadoss Project (adapted from Cascadoss, 2007b)

Desktop GIS Application	Version	Marketing potential	Technical Potential	Economic Potential	Cumulative Potential
FMaps	0.0.2	17.4	15.8	28.2	20.5
GRASS	6.2.3	**59.3**	**46.9**	40.1	**48.8**
gvSIG	1.1.2	49.8	**39.8**	37.1	42.2
JUMP	1.2	35.3	25.6	**49.8**	36.9
Kosmo	1.2	29.8	28.6	43.9	34.1
OpenEV	1.8	50.1	27.1	40.7	39.3

Table 3.13. Marketing, Technical, Economic and Cumulative Potential of Free and Open Source Desktop GIS Applications Evaluated in Cascadoss Project (cont.) (adapted from Cascadoss, 2007b)

Desktop GIS Application	Version	Marketing potential	Technical Potential	Economic Potential	Cumulative Potential
OpenMap	4.6.4	48.0	25.4	**47.8**	40.4
OSSIM	1.7.4	50.7	36.7	37.1	41.5
QuantumGIS	0.9.1	**55.6**	**40.8**	**50.6**	**49.0**
Saga GIS	2.0.2	44.9	30.4	36.0	37.1
Thuban	1.2.1	**55.1**	30.2	**47.8**	**44.4**
uDig	1.1-RC 14	42.3	30.6	42.4	38.4
Evaluation is over 60 points. Top three scores on each column are written in bold.					

According to Table 3.13, respectively GRASS, QGIS and Thuban are identified as the first three desktop GIS applications offering the highest marketing potential. In terms of technical potential, as in marketing potential ranking, GRASS and QGIS are identified as the top two desktop GIS applications while gvSIG is listed as the third software. From the point of economic potential, GRASS loses its leading position to QGIS and JUMP is listed as the second software offering the highest economic potential. Collecting the same score, OpenMap and Thuban share the third ranking in terms of economic potential. Cumulative potential indicates equally weighted average of marketing, technical and economic potential. In terms of cumulative potential, respectively QGIS, GRASS, Thuban and gvSIG are listed as the desktop GIS applications sharing top four ranks. This conclusion is quite consistent with the findings of Steiniger and Bocher's (2009) study in which GRASS, QGIS and gvSIG are found to offer more functionality than ArcView 10.1. Besides, with reference to Table 3.11, GRASS, QGIS, gvSIG and uDig are found as the most cited projects on web resources and in the literature. Based on these findings, it can be concluded that GRASS and QGIS are the top two free and open source desktop GIS products; besides Thuban with its great marketing and economic potential and gvSIG with its promising technical potential are other two free and open source desktop GIS products.

3.3.1. QGIS Desktop GIS Application

Quantum GIS, or QGIS in short, is a user friendly multi platform free and open source desktop GIS application that supports vector, raster and database formats (QGIS, 2013a) (Figure 3.1). QGIS is an OSGeo supported project and licensed under the GNU General Public License. This license guarantees the freedom of utilization and redistribution of QGIS for any purpose; besides it also guarantees the availability of human readable source code for further distributions, modified or derived work.

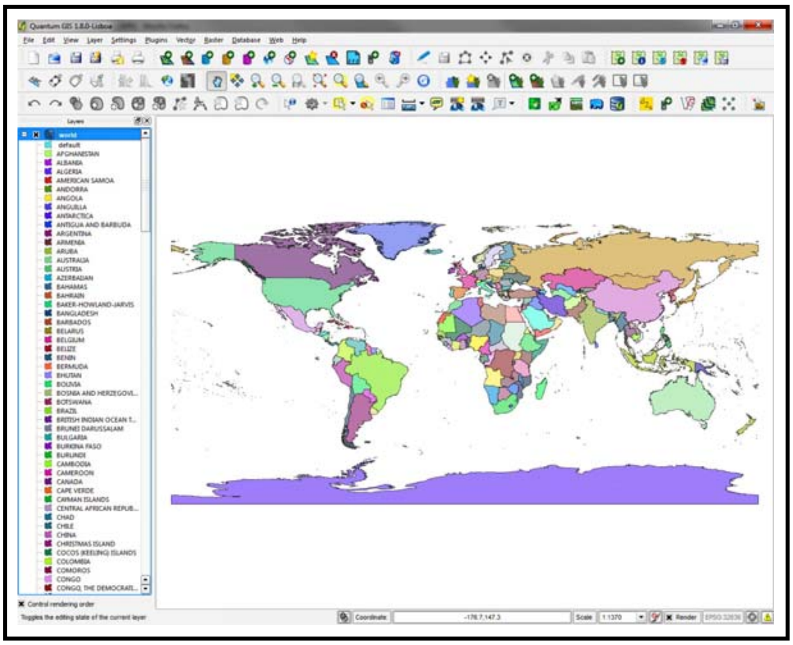


Figure 3.1. QGIS v1.8.0-Lisboa Running on Windows

As a response to the expert oriented design of GRASS, a group of volunteers initiated QGIS project in 2002 (Steiniger and Bocher, 2009). Main intention of the project was to provide a user friendly and fast geographic data viewer for Linux based platforms (Hugentobler, 2008). As the project evolved, utilization of QGIS as a user friendly interface for GRASS emerged as an idea (Steiniger and Bocher, 2009). QGIS increased its development tempo in 2004 by releasing several minor versions with new features and including a tie to GRASS analysis functionality (Ramsey, 2007). As a result of this agile development, QGIS project has already achieved its initial objectives and now community works to extend the functionality beyond data viewing. General project characteristics and major features of QGIS are presented in Table 3.14.

Table 3.14. General Project Characteristics and Major Features of QGIS

General Characteristics	
Project Full Name	Quantum GIS
Variant Name	QGIS
Founded	2002
Type of License	GNU General Public License
Programming Language	C++, QT4 Framework, Pyhton
Supported Platforms	MS Windows, BSD, Apple MacOSX, Linux, Android
GUI Translations	48 languages for v1.8.0
Purpose	QGIS Desktop : General purpose desktop GIS application
	QGIS Browser : Data viewer for network and WMS data
	QGIS Server : WMS 1.3 server configured by QGIS desktop project files
	QGIS Client : Web front-end for web mapping based on OpenLayers and GeoExt
Documentation and Resources	
User Guide	Available in English, German, Russian for v1.8 Available in English, German, Russian, French, Portuguese, Korean for v1.7
Other Documentation	QGIS Coding and Compilation Guide QGIS API Documentation PyQGIS Cookbook
Mailing List	http://qgis.org/en/community/mailing-lists.html
Forum	Current: http://gis.stackexchange.com Depreciated: http://forum.qgis.org
Wiki	http://hub.qgis.org/projects/quantum-gis/wiki
Repository	http://hub.qgis.org/projects/quantum-gis/repository
Plugin Repository	http://plugins.qgis.org/plugins

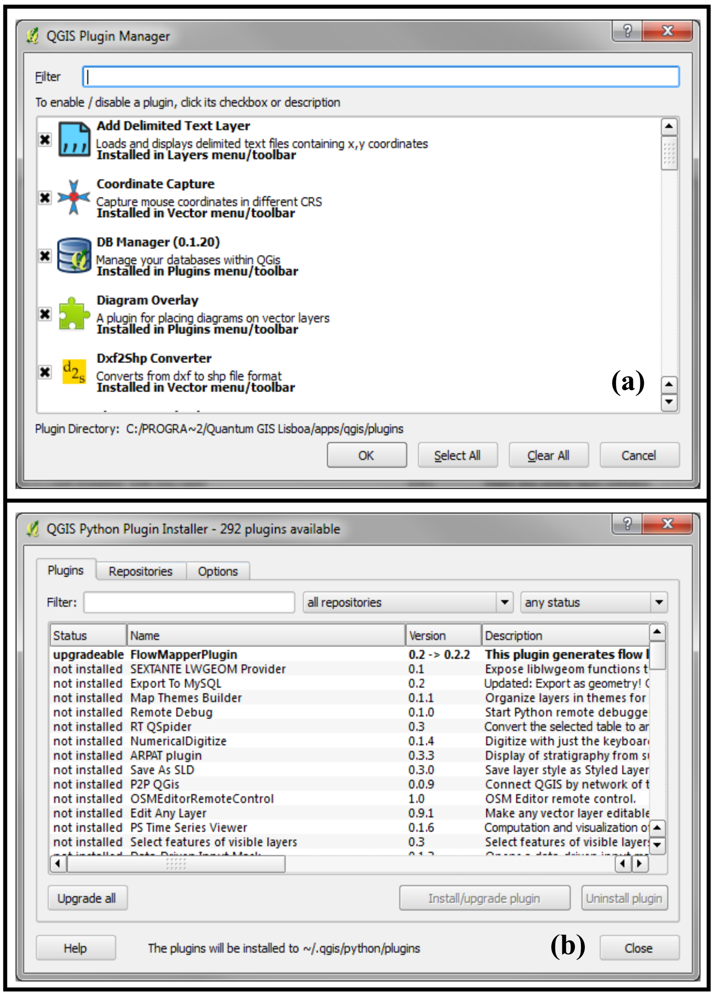
Table 3.14. General Project Characteristics and Major Features of QGIS (cont.)

Functionality	
Support for Common Vector and Raster Formats	Uses OGR/GDAL library to support common vector and raster formats such as; ESRI ShapeFile, ArcInfo Coverages, Spatially Enabled PostGIS Tables, MapInfo MID/MIF and TAB, KML, GML, AutoCad DXF, GRASS Locations ans Mapsets, GeoTiff, ArcGrid, ERDAS IMG etc.
Native Support for Spatial Database	PostGIS, SpatiaLite, MS SQL Spatial
OGC Compliance	WMS, WMS-C, WFS, WFS-T
Exploration and Cartographic Production	On-the-fly reprojection, print composer, overview panel, view/edit/search attributes, labeling, advanced vector and raster symbology, map decorations for cartographic representation
Creating and editing	Digitizing tools for vector features, field and raster calculator, georeferencer plugin, GPS tools to import/export GPX format
Spatial analysis	Map algebra, geoprocessing tools, geometry tools, terrain analysis, grid interpolation, network analysis
Scripting Support	Python scripting
Plugin Support	Plugins with C++ and Python
Review is performed based on v1.8.0 Lisboa released in 21 June 2012	

QGIS is a community driven project. Project management workload is divided between an international team of developers and users who have specific responsibilities other than coding. QGIS project steering committee is comprised of three main teams. These are; (i) release management team, (ii) technical resources team and (iii) community resources team. QGIS releases are managed by release management team which is prepared by packaging and testing teams. Technical resources team recruit and support code maintainers team who ensure consistency and quality of code added into repository. Community resources team is responsible for the organization of GUI translations, user conferences, press material and translations. Although it is difficult to know the exact size of QGIS community, with reference to more than half million QGIS Windows installers downloaded in 2012 (QGIS, 2013c), size of the community is promising. QGIS Project always looks for volunteers and contributions can be improvements to the source code, packaging oriented or non programming help such as documentation and translation.

QGIS is developed using QT cross application framework and C++ programming language. With the release of v0.9, python programming language support was added to QGIS at the end of 2007. Although QGIS desktop application is built primarily for Linux (Ramsey, 2007), QT is available for other platforms such as Windows, OS X, X11, Android and iOS. So QGIS is also built for use in platforms other than Linux. As a multi platform desktop GIS application, QGIS runs on Windows, Linux, BSD, MacOSX and Android platforms. On Windows operating systems QGIS can be installed either by using standalone installer which is recommended for new users or by using OSGEO4W installer which offers more installation options for experienced users. On Linux platforms, prebuilt binary packages exist for many popular Linux distributions such as Debian, Fedora, RedHat, openSUSE, Mandriva, Ubuntu and Slackware. For MacOSX users, Apple dmg disk image and for Android users, apk application package are available for easy installation (QGIS, 2013b).

QGIS can be customized for specific needs. Either by using C++ or Python languages, programmers or experienced users with enough programming knowledge can have access to QGIS libraries via QGIS API (Application Programming Interface). Customizations can be performed in two ways; either a standalone, custom GIS application can be developed by using QGIS libraries or a new plugin can be developed for use with the native GUI of QGIS. As of June 2013, 215 plugins are listed and available for download from the official plugin repository of QGIS which is hosted on "http://plugins.qgis.org/plugins". All these plugins can easily be downloaded, installed and managed by using the "QGIS Plugin Manager" (Figure 3.2). Besides, there are also external developers who develop third party plugins and publish them by creating third party repositories hosted on their web domains.



To enable / disable a plugin, click its checkbox or description

☒ **Add Delimited Text Layer**
Loads and displays delimited text files containing x,y coordinates
Installed in Layers menu/toolbar

☒ **Coordinate Capture**
Capture mouse coordinates in different CRS
Installed in Vector menu/toolbar

☒ **DB Manager (0.1.20)**
Manage your databases within QGis
Installed in Plugins menu/toolbar

☒ **Diagram Overlay**
A plugin for placing diagrams on vector layers
Installed in Plugins menu/toolbar

☒ **Dxf2Shp Converter**
Converts from dxf to shp file format
Installed in Vector menu/toolbar

Plugin Directory: C:/PROGRA~2/Quantum GIS Lisboa/apps/qgis/plugins

Status	Name	Version	Description
upgradeable	**FlowMapperPlugin**	**0.2 -> 0.2.2**	**This plugin generates flow l**
not installed	SEXTANTE LWGEOM Provider	0.1	Expose liblwgeom functions t
not installed	Export To MySQL	0.2	Updated: Export as geometry! (
not installed	Map Themes Builder	0.1.1	Organize layers in themes for
not installed	Remote Debug	0.1.0	Start Python remote debugge
not installed	RT QSpider	0.3	Convert the selected table to ar
not installed	NumericalDigitize	0.1.4	Digitize with just the keyboar
not installed	ARPAT plugin	0.3.3	Display of stratigraphy from s
not installed	Save As SLD	0.3.0	Save layer style as Styled Layer
not installed	P2P QGis	0.0.9	Connect QGIS by network of t
not installed	OSMEditorRemoteControl	1.0	OSM Editor remote control.
not installed	Edit Any Layer	0.9.1	Make any vector layer editable
not installed	PS Time Series Viewer	0.1.6	Computation and visualization o
not installed	Select features of visible layers	0.3	Select features of visible layer

The plugins will be installed to ~/.qgis/python/plugins

Figure 3.2. QGIS v1.8.0-Lisboa (a) Plugin Manager and (b) Plugin Installer

3.4. Motivation for Free and Open Source Development and Reasons for Selecting QGIS

The freedom to have unrestricted access to source code and community based development structure that encourages anonymous contributions are the most important factors promoting productivity in free and open source environment. As an inevitable result of these motivating factors, today free and open source software gamut offers such great diversity. As previously presented in Table 3.1, there are such successful open source projects which can compete with their well known proprietary, commercial alternatives.

Practically, free and open source software is supplied free of charge. Anyone can download it from the internet and start benefiting from it with no restrictions for any purpose. This provides cost reduction compared to commercial software which is subject to payment of license fee. For example, as of 2012 valid in the United States, a bundle which includes annual subscription to the ESRI Developer Network (EDN) and a fixed node single ArcView license is priced for 2.000 USD per year (ESRI, 2012). Besides, this price increases in half to 3.000 USD per year, if bundle is preferred with a fixed node ArcEditor license and doubles to 4.000 USD per year, if fixed node ArcInfo license is preferred. High licensing fees either prevent spread of software or restrict easy access to software since there are limited installations. This may be a discouraging factor for users who are new to GIS. However free and open source software can be installed and used on unlimited number of computers either for personal or for commercial purposes without any restrictions. For example, compared to the EDN annual subscription and ArcView fixed node license bundle which is priced for 2.000 USD per year; QGIS Desktop GIS software is available free of charge with its all plugins. Besides, all developer content such as QGIS API documentation, PyQGIS Cookbook, mailing lists, project's wiki page are all open to public access free of charge.

In commercial software world, unless there is enough market potential to compensate the cost of development efforts, software vendors choose to behave conservative in terms of integrating domain specific tools to their existing architecture since these tools may be rarely used. In other words, commercial software vendors are usually slow to respond in terms of expanding existing functionality of software unless it is profitable. For example, one of the most popular GIS software on the market, ESRI ArcGIS for Desktop is available with three levels of license and more than twenty paid extensions (ESRI, 2014). While most of these extensions are designed for general purpose analyses (e.g. Spatial Analyst, 3D Analyst, Geostatistical Analyst) and require "Basic" type of license (formerly ArcView); solution based specific extensions (e.g. ArcGIS for Maritime, ArcGIS for Aviation, ESRI Roads and Highways) require purchasing of either

“Standard” (formerly ArcEditor) or “Advanced” (formerly ArcInfo) type of license which costs few times more than the base license. Yet, offering more than twenty extensions as one of the most popular desktop GIS software, there is neither an extension nor specific tools in the ArcToolbox designed for flow mapping. Thus, proprietary or commercial GIS products are generally designed to meet general needs of users.

Flow mapping is a special domain where the market is commercially not strong enough to compensate the cost of developing specially designed tools for visualization of spatial interaction data. In software development, source code is a valuable resource not only because it gives developers a chance to analyze how the application works but also it avoids developers from recoding algorithms that are already coded. In free and open source development, since the source code is available from the internet, existing functionality of software can be extended by development efforts of researchers to meet the demand occurring in special domains where development is not profitable enough for commercial vendors. For example, QGIS can be customized to meet specific needs with its plugin support or even standalone GIS applications can be developed either by using C++ or Python programming languages by accessing QGIS libraries via QGIS API. In contrast to free and open source development, in the proprietary development source code is kept confidential and only the software vendor has right to access. Thus, modifications to software by third party developers are either not allowed due to restricting EULA terms or limited to an extent that is determined by the development libraries supplied by the vendor.

Possible advantages of the free and open source development methodology can be listed as follows:

(i) Unrestricted and gratis access to human readable source code and implementation of algorithms;

(ii) Opportunity of studying the source code and modifying the software in order to fix bugs, improve performance and extend functionality;

(iii) Opportunity of redistributing modified or extended versions of software and source code without any restrictions;

(iv) Encouraging voluntary contributions to development process which can be performed either by participating to coding or preparation of documentation and translations;

(v) Open development encouraging voluntary contributions guarantees the continuity of software up to some level since any researcher may continue on development without facing any legal obstacles even if the originator stops development.

In order to identify the most mature free and open source desktop GIS software that offers more potential than others several web resources and research studies are reviewed. While GRASS, QGIS, Thuban and gvSIG are identified as leading GIS applications having more potential than others (Cascadoss 2007b, Steiniger and Bocher 2009); GRASS and QGIS are identified as the top two free and open source desktop GIS application (Cascadoss, 2007b). In their study evaluating GIS software for water resources management Chen et al. (2010) criticize complex structure of GRASS and promotes QGIS due to its adequately powerful structure, functionality and user friendly interface. Besides, with the aid of community developed GRASS plugin for QGIS, all tools of GRASS can be executed from the user friendly interface of QGIS. Owing to its user friendly interface (Chen et al., 2010), great economic potential (Cascadoss, 2007b) and proven popularity with over half million downloads in 2012 (QGIS, 2013c), although GRASS is identified to have more technical potential than QGIS (Cascadoss, 2007b), in this study QGIS is preferred to GRASS as the desktop GIS platform that the flow mapping plugin will be developed.

History of GRASS, an acronym for Geographic Resources Analysis Support System, dates back to 1982 and has a long development history. For the past three decades GRASS has been primarily developed to run on high powered UNIX workstations. Until the release of v5.0, core components of GRASS were developed by the U.S. Army Construction Engineering Research Laboratory

(USA-CERL). In 1997 development efforts for GRASS was transferred to community and now source code, documentation and release are all managed by the GRASS Development Team. Until v6.3, GRASS can be run on Windows platforms with the aid of Cygwin emulator that ports Linux environment to Windows. WinGRASS is an open source project that is maintained by the GRASS Development Team with the intention of developing native Windows version of GRASS which does not need any emulator to run. Latest development of GRASS GIS for Windows Native is v6.4 and released in 2012. However, GRASS Development Team still identifies WinGRASS as an experimental project and notices that it should not be considered as a fully working release of GRASS for Windows since it needs more testing (GRASS, 2013). Thus, to develop a flow mapping plugin integrated into a desktop GIS, it is preferable to choose a more stable platform than GRASS which is not subject to major changes.

Unlike GRASS which has a development history of nearly three decades, development of QGIS dates back only to 2002. With the release of v0.9 at the end of 2007, Python programming language support was added to QGIS. Python is an interpreted programming language that runs directly from the human readable source code and has a shorter and more lucid syntax than C++. Thus, developing with Python is easier. By using Python programming language and Python bindings such as pyQT and pyQGIS, either plugins or standalone GIS applications can be developed by accessing QGIS libraries via QGIS API. Taking the advantage of high level programming language Python and offering an up-to-date structure than GRASS, QGIS offers more stable developing environment than GRASS. Main reasons for choosing QGIS as the desktop GIS platform to develop a plugin can be given as follows:

(i) Popularity of project which promises large and active community;

(ii) User friendly GUI especially compared to GRASS;

(iii) Native multi platform support; Windows, Linux, MacOSX;

(iv) Support for popular raster and vector formats via OGR/GDAL;

(v) Extensible functionality with plugin structure;

(vi) Python programming language support for rapid development;

(vii) Valuable resources such as QGIS user manual, API documentation, PyQGIS Cookbook and mailing lists;

(viii) General Public License (GNU GPLv2) which guarantees unrestricted access to source code either in later releases or in derivative works;

(ix) Support of OSGeo Foundation.

Many GIS users prefer menu driven, user friendly applications since they offer more comfortable working environment and have sharp learning curves than command line executed applications. Practically, desktop GIS users who want to map spatial flows would also prefer similar menu driven interactive applications. However, flow mapping is a special research area where the market is commercially not strong enough to compensate the cost of developing a fully featured GIS application dedicated for flow mapping. Instead, in this study, flow mapping application is preferred to be developed as a plugin to a popular, open source desktop GIS application. By this way, users can both benefit from common GIS functions and utilize flow mapping tools seamlessly in the same application. Besides, by performing this work with the open source development methodology, researchers interested in flow mapping will have access to source code and opportunity of contributing to development which increases the continuity of project.

Up to this point of the study, a literature survey regarding flow mapping and a review regarding free and open source software concept are respectively given in Chapter 2 and 3. Besides, a comparison of visual clutter reduction techniques and existing flow mapping software is given in Chapter 2. A similar comparison is presented in Chapter 3 regarding existing free and open source GIS software together with a review for QGIS. Based the findings from these two chapters; type of flow scenario to be focused and visual clutter reduction techniques to be implemented in FlowMapper are determined. Besides, free and open source desktop GIS platform for building FlowMapper on is selected as QGIS. In the next chapter, software development environment required for building a QGIS plugin is explored.

CHAPTER 4

DEVELOPMENT ENVIRONMENT OF FLOW MAPPING SOFTWARE

In Chapter 3, QGIS was selected as the base desktop GIS component in order to build a flow mapping application in plugin form due to its agile development environment that supports Python programming language and its up-to-date API compared to GRASS. In order to develop a QGIS plugin with Python, at least following tools and libraries are needed: (i) QGIS desktop application, (ii) Python 2.5 or more, (iii) Qt4 framework, (iv) PyQt4 Python bindings, (v) PyQGIS Python bindings. In addition to these development tools, OGR Simple Features Library is utilized to perform operations on vector data files.

In this chapter, main components of the development environment are defined. Respectively (i) Python programming language, (ii) QGIS API and PyQGIS Python bindings, (iii) Qt development framework and its Python bindings PyQT and (iv) Geospatial Data Abstraction Library (GDAL) and Simple Features Library (OGR) are examined.

4.1. Python Programming Language

Python is a dynamic, high level, multi platform, general purpose programming language. Python's design philosophy explicitly emphasizes code readability (PSF, 2013a) and its syntax allows developers to perform operations with fewer lines of code compared to same operations in other well known languages such as C++ or Java. As a dynamic, high level programming language, Python is often admitted as a scripting language; however, owing to its thousands of third party modules, Python is also used in a wide range of non-scripting domains such as scientific and numeric computing, database access, desktop GUI development, network programming, 3D graphics and game development (PSF, 2013b).

Python Software Foundation (PSF, 2013a), which is a non-profit organization that aims to advance open source technology related to Python programming language, lists some of Python's key distinguishing features as follows:

(i) Clear and readable syntax,
(ii) Intuitive object orientation,
(iii) Modularity and hierarchical package support,
(iv) Support for high level dynamic data types,
(v) Extensive standard libraries and plenty of third party modules,
(vi) Capability of being embedded in applications as a scripting language,
(vii) Extensions and modules can be written in other languages (e.g. C and C++ for Python, Java for Jython, .NET languages for IronPython).

Python development efforts include many contributions; however Guido van Rossum is regarded as the principal author of Python programming language (Pilgrim, 2004). The name of the programming language originates from the BBC comedy series named "Monty Python's Flying Circus". Guido van Rossum needed a unique name for the new language that he began implementing and since he enjoyed this comedy series, he gave "Python" name to the language.

Python was conceptualized in the late 1980s and developed in the early 1990s by Guido van Rossum as a successor of the programming language ABC at Stichting Mathematisch Centrum (CWI) in Netherlands. Python 1.2 was the last version released from CWI. Between 1995 and 2000, Guido van Rossum continued his development efforts at the Corporation for National Research Initiatives (CNRI) in Reston, Virginia and released several Python versions. Python 1.6 was the latest of the versions released from CNRI. In 2000, Guido van Rossum and core development team members shifted to BeOpen.com in order to establish BeOpen PythonLabs team and then Python 2.0 was released by BeOpen.com. Just after Guido van Rossum left CNRI, it was understood that ability to use Python with software released under GNU General Public License was very desirable and promising. With this intention, CNRI and Free Software Foundation (FSF) collaborated to make changes on the Python license. In 2001, Python Software Foundation (PSF) was established as a non-profit organization in Delaware, USA to protect, promote and advance Python programming language while facilitating community based development efforts. Starting with Python 2.1, all intellectual property produced on top of this structure is owned by the Python Software Foundation (PSF). Python 2.7 was released in 2013 as the latest and most recent version under Python 2.x series. Python 3.0, which was developed to rectify certain design flaws encountered in Python 2.x series, was released in 2008. However, during development of Python 3.0, modifications required to fix shortcomings could not be implemented while keeping backward compatibility with Python 2.x series. Thus, there is no warranty that a Python 2.x code snippet would run unmodified on Python 3.0. In other respect, Python 2.x development was not depreciated with the release of Python 3.0. Python 2.5, 2.6 and 2.7 versions were released between 2011 and 2013. Besides, many features of Python 3.0 have been back ported to be compatible with Python 2.6 and 2.7 (van Rossum, 2006).

In Table 4.1, Python release history is given with respect to each version's release date and software license characteristics. As it can be inferred from the table, all Python releases are open source. Python license does not have copyleft restrictions as in GNU GPL type licenses; thus, modified versions can be distributed without having to keep the modified source code open. Besides, not all but most Python releases are GNU GPL compatible. This means compatible versions of Python can be combined and released with the source code of the software that is released under GNU GPL. A well-matched example is distribution of Python with QGIS which is licensed under GNU GPL (e.g. QGIS 2.0.1-Dufour is distributed with Python 2.7.5)

Table 4.1. Python Release History and Licensing Characteristics (adapted from PSF, 2013c and PSF, 2013d)

Python Version	Release Year	Owner	Open Source	GNU GPL Compatibility
from 2.1.1 to 3.3.3	2001 - 2013	PSF	Yes	Yes
2.1	2001	PSF	Yes	No
1.6.1	2001	CNRI	Yes	No
2.0	2000	BeOpen.com	Yes	No
1.6	2000	CNRI	Yes	No
1.3 - 1.5.2	1995 - 1999	CNRI	Yes	Yes
0.9.0 - 1.2	1991 - 1995	CWI	Yes	Yes

Python, installed with extensive standard libraries (PSF, 2013e), is also designed to be used as extensible language. Python supports packages and modules which encourage modularity and code reuse for increased productivity. One of Python's striking features is its capability of integrating with other languages as a "glue" (Zelle, 1999). Libraries developed with different languages can be called via Python language (e.g. C and C++ libraries via Python, Java libraries via Jython, C# libraries via IronPython). SIP and SWIG are two common tools for writing Python modules that interface with C and C++ libraries. Riverbank Computing Limited developed SIP in order to enable access to formerly Trolltech's, currently Nokia's Qt cross platform framework via Python. Although SIP was initially developed to create Python bindings for Qt (PyQt) by aggregating Python, Qt and C++; SIP can also be used to create Python bindings for any C and C++ library. For example, QGIS bindings for Python (PyQGIS) are also created by using SIP as in PyQt (QGIS, 2014a). In addition to its clear and understandable syntax, this structure makes Python effective and powerful "glue" language. As inferred from Table 4.1, source code of all Python releases is open, besides all recent Python releases have GNU GPL compatible PSF license. This makes possible to embed Python interpreter into any application to be used as a scripting language in that application (Brown, 2001). By this way Python tools and libraries can be utilized in that relevant application. Apart from these methods, by using third party tools (e.g. Py2exe or Pyinstaller), it is also possible to pack Python code snippets into standalone executable applications.

Owing to Python's design philosophy that promotes clear and readable syntax and its distinctive capabilities discussed above, Python programming language has gained popularity in recent years. TIOBE is a specialized company found in 2000 in assessing the software quality. TIOBE (2014) uses top 25 search engines to calculate the TIOBE Index to asses the popularity of programming languages. In Table 4.2, current popularity ranking of top 12 programming languages are given as of January 2014. Besides, previous popularity rankings of these languages are also listed in Table 4.2 with 5 year intervals. Table 4.3 presents the programming language which shows the highest annual rise in popularity rating.

Table 4.2. Popularity of Programming Languages based on TIOBE Index: Long Term History (adapted from TIOBE, 2014)

Language	**2014**	**2009**	**2004**	**1999**	**1994**	**1989**
C	1	2	2	1	1	1
Java	2	1	1	16	-	-
Objective-C	3	42	48	-	-	-
C++	4	3	3	2	2	4
C#	5	8	9	32	-	-
PHP	6	5	6	-	-	-
(Visual) Basic	7	4	5	3	3	7
Python	**8**	**6**	**11**	**22**	**22**	-
JavaScript	9	9	8	21	-	-
Perl	10	7	4	5	17	23
Lisp	14	18	15	10	7	2
Ada	23	21	16	17	6	3

Table 4.3. Programming Languages Showing Highest Annual Rise in Popularity Rating based on TIOBE Index (adapted from TIOBE, 2014)

Year	Highest Rise
2013	Transact-SQL
2012	Objective-C
2011	Objective-C
2010	**Python**
2009	Go
2008	C
2007	**Python**
2006	Ruby
2005	Java
2004	PHP

With reference to Table 4.2, Python is listed as the 8th most popular programming language according to TIOBE Index results announced on January 2014. Python also displayed a positive trend in the last decade. With reference to Table 4.3, in 2007 and 2010, Python was listed as the language showing the highest annual rise in popularity rating.

Similar to TIOBE Index, LangPop (2013) is a website which collects data about the popularity of programming languages. LangPop presents a normalized comparison chart in their website (LangPop, 2013). This chart is a reflection of listing statistics and returned search results from Github Repositories, Google Files, Ohloh, Craiglist and Google Search. By default, to create the normalized comparison chart rankings, all sites are equally weighted. In Table 4.4, based on the normalized comparison chart, popularity rankings of top twelve programming languages are presented and Python is listed as the sixth popular language (LangPop, 2013).

Table 4.4. Popularity of Programming Languages based on LangPop Normalized Score Chart (adapted from LangPop, 2013)

Rank	Language	Score
1	C	63
2	Java	56
3	PHP	55
4	JavaScript	43
5	C++	35
6	**Python**	**30**
7	Shell	29
8	Ruby	26
9	Objective C	23
10	C#	22
11	Assembly	21
12	SQL	19

As it can be inferred from Tables 4.2, 4.3 and 4.4, Python is one of the most popular languages and its popularity is increasing. Based on this finding, ESRI, as one of the leading geographic information systems companies, replaced its Arc Macro scripting Language (AML) with Python 2.1 in 2004 with the release of ArcGIS 9.0. Besides, with the release of ArcGIS 10.1 in 2012, ESRI discontinued its support to Visual Basic for Applications (VBA) and preferred to replace it with Python 2.7. Similar to ESRI's choice in the commercial GIS market, QGIS community in the free and open source platform, integrated Python scripting support starting from QGIS v0.9 released in 2007.

Python is an interpreted language that can run on many different operating systems. Python is written in C programming language and owing to proven performance of C (Prechelt, 2000), it runs faster than many early scripting languages (Zelle, 1999). It can be argued that computational performance of an interpreted language would be slower than a compiled language like C or C++ typically by a factor of ten. However, considering processing capabilities of today's computers, in practice, this performance constraint will only make difference when working with huge amounts of data or be critical when expecting very reliable performance for real time operations in embedded systems.

Performance of a programming language should be judged not only with reference to its computation speed. Python can be preferred to C, C++, C# or Java due to its clear syntax which is quite close to English and its fast debugging cycle. Python's clear syntax allows performing the same tasks with less code compared to C++, C# and Java (see Table 4.5 below where the codes required for the simple task of printing "Hello, World!" are given). Besides, as an interpreted language, Python codes can be executed without needing compilation to binary. Instead, Python source code is translated into byte code, written into a "pyc" file and executed by Python virtual machine. If the source code is not altered, relevant "py" file is not parsed and "pyc" file is not recompiled. This structure makes Python's debugging cycle faster than compiled languages.

Table 4.5. Examples of Performing “Hello, World!” Statement in Different Programming Languages

Prog. Language	Character Count without Spaces	Code

C++	74	`#include <iostream>`
		
		`int main()`
		`{`
		`    std::cout << "Hello, World!" << std::endl;`
		`    return 0;`
		`}`

C#	88	`using System;`
		`class HelloWorld`
		`{`
		`    static void Main()`
		`    {`
		`    System.Console.WriteLine("Hello, World!");`
		`    }`
		`}`

Java	94	`public class HelloWorld`
		`{`
		`  public static void main(String[] args)`
		`  {`
		`    System.out.println("Hello, World!");`
		`  }`
		`}`

Python 2.x	**19**	`print "Hello, world!"`

Various tools are available for GUI development in Python; such as PyGtk, PyQT and wxPython. All three toolkits are coded with C or C++ and support development on multiple platforms. However, in order to develop GUI with Python by using these toolkits, Python bindings of that toolkit should be installed on the target development platform. By means of these bindings, development libraries supplied with the toolkit can be accessed and implemented by Python. GUI development can be performed either by importing modules from the toolkit and hard coding with Python or by using an interactive graphical design tool to arrange the GUI elements on screen.

For example, QT framework libraries on which the QGIS is also built provides an interactive design tool called QT Designer. For example, in this study, QT Designer is preferred to design form interfaces for QGIS FlowMapper plugin. Then, “ui” files storing designs are transformed into Python “py” files by using “pyuic”, which is an UI compiler for QT included in the PyQT package.

In Chapter 3, it was mentioned that QGIS can be customized for specific needs by developing plugins either by using C++ or Python languages. Design philosophy of Python gives priority to the performance of programmer rather than pure performance of computer. Due to Python’s very clear and easily understandable syntax which endorses its sharp learning curve and due to its high level structure; Python is preferred as the development language of QGIS FlowMapper plugin instead of C++.

4.2. QGIS API and PyQGIS Python Bindings

QGIS not only provides a free and open source desktop GIS environment, but also provides development libraries which can be used to create customized applications. This has been realized with the refactoring of libraries released with QGIS v0.8 in 2007 (Corradini and Racicot, 2011). Starting from QGIS v0.9, Python support is integrated into QGIS. However, QGIS v1.0 or greater is recommended as the basis for development since it provides a stable, consistent API. QGIS API modules and brief explanation of each module are given in Table 4.6.

Table 4.6. QGIS API Modules (adapter from QGIS, 2014b)

QGIS API Modules	Description
QGIS Core Library	Provides all basic GIS functionality.
QGIS GUI Library	Built on top of core library and provides reusable GUI widgets.
QGIS Analysis Library	Built on top of core library and provides ready to use tools for performing spatial analysis both with vector and raster datasets.
Map Composer	QGIS Map Composer provides layout elements (e.g. legend, scalebar) to prepare data for printing.
QGIS Network Analysis Library	Provides high level tool topology building and topological analysis.

PyQGIS refers to Python bindings for QGIS. It can be described as the “Pythonic” application programming interface (API) which wraps QGIS library written in C++ (Corradini and Racicot, 2011). This means developers can write scripts, plugins or applications with “Pythonic” QGIS API without having to learn C++. “Pythonic” QGIS API structure is quite similar to the API in C++.

QGIS is built on top of Qt libraries. PyQt, namely Python bindings for Qt, is developed with the open source SIP tool. For seamless integration with PyQt, PyQGIS, namely “Pythonic” API that wraps QGIS libraries, is also developed with the use of SIP tool. These bindings make possible to develop new GIS applications by using QGIS and Python language. There are several ways of using Python with QGIS; (i) commands can be executed within Python console embedded in QGIS, (ii) plugins can be developed with Python to be used in QGIS, (iii) standalone, custom GIS applications can be developed with Python by utilizing QGIS API.

For the first use case, Python console can be accessed from the interface of QGIS. Python is embedded in QGIS as an interpreter. This interpreter also parses the source code of any Python plugin located under plugins directory at the initialization of QGIS.

Regarding the second use case, capabilities of QGIS can be extended via plugins. QGIS plugins can be developed either by C++ or Python languages. Plugins developed with C++ must be compiled for each platform (e.g. plugin must be compiled into “dll” file for Windows OS). Unlike C++ plugins, plugins developed in Python do not need to be compiled for different platforms. Same Python source code can be distributed for different platforms. Plugin source code is executed by the Python Virtual Machine on-the-fly. Regarding plugin development in QGIS, plugins take advantage of the functionality offered by the libraries licensed under GNU GPL. This means that QGIS plugins must also be licensed under GNU GPL.

From the perspective of third use case, not every possible user needs a fully functional desktop GIS application. In such cases, by means of PyQGIS and PyQt Python bindings, custom standalone applications or widgets within third party applications can be developed to meet specific GIS related needs (e.g. standalone map viewer with basic map controls and query functions).

4.3. Qt Framework and PyQt Python Bindings

Qt is a cross platform application framework commonly used for software development with graphical user interface (GUI) (Wikipedia, 2014a). It has two major versions as Qt 5 (Qt Project, 2013) and Qt 4 on which the QGIS is also built. Although Qt is sometimes classified as a widget tool, it is also possible to use Qt for developing non-GUI based applications (e.g. command line tools) (Wikipedia, 2014a). The idea behind using an application framework such as Qt is speeding up the development cycle and increasing productivity by utilizing the tools available under that framework (e.g. Qt Designer).

Qt framework was originally developed by a Norwegian company Trolltech. Nokia acquired Trolltech in 2008 and until 2011 Qt framework was developed by Nokia. In 2011, Digia Company acquired commercial property of Qt framework and currently it is being developed under Qt Project that involves contributing developers to advance Qt under open governance model.

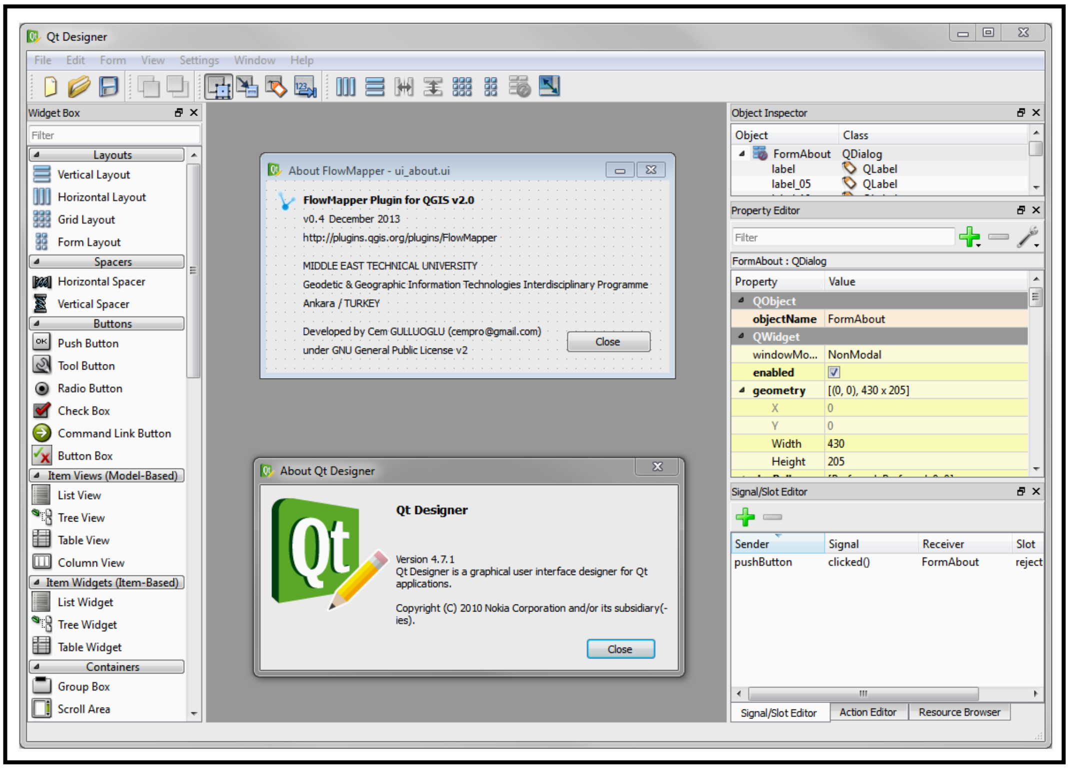
Substantially, Qt uses C++. Yet, it can be used with several programming languages via Qt bindings (e.g. PyQt for Python, Qt Jambi for Java, QtRuby for Ruby). Since Qt is developed as a cross platform framework, it can be used on many platforms such as Windows (Qt for Microsoft), OS X (Qt for Apple OS X), X11 (Qt for X Window System), Embedded Linux (Qt for Embedded Platforms), QNX & BlackBerry 10 (Qt for ONX and QNX based platform BlackBerry 10), Android (Qt for Android) and iOS (Qt for iOS). Besides, Qt has also external ports (e.g. Qt for OpenSolaris and Qt Ubuntu).

Qt framework is available with three different types of licenses: (i) commercial license, (ii) GNU GPL v3 and (iii) GNU LGPL v2.1. Besides, Qt framework is available with three editions (Wikipedia, 2014a): (i) GUI Framework (Commercial, entry level GUI edition without network and database support), (ii) Full Framework (Fully featured commercial edition), (iii) Open Source (Complete open source edition).

PyQt, developed by Riverbank Computing Limited, stands for Python bindings that wrap Qt libraries written in C++ (Riverbank, 2014a). In other words, PyQt links between Qt cross platform application framework and cross platform programming language Python. By means of PyQt bindings one can develop applications with Python by utilizing Qt framework tools without having to know C++. As expected, PyQt runs almost on all platforms on which Qt and Python can be installed.

PyQt incorporates dual licensing model as in Qt. For free and open source development, PyQt is supplied under GNU GPL v2 and GNU GPL v3 (Riverbank, 2014b). However, for proprietary development with Python in Qt framework, commercial licenses for Qt and PyQt should be purchased.

Qt framework includes a GUI design tool known as Qt Designer (Figure 4.1). For Python programmers, by means of the tools bundled in PyQt, it is possible to generate Python code from Qt Designer “ui” files. GUI interactively designed in Qt Designer tool is stored as “ui” file in “xml” structure. By means of “pyuic4” compiler bundled in PyQt, “xml” based “ui” design file is translated into “py” file that stores the Python code. Similar to the procedure for “ui” design files, resource files storing GUI icons are translated into Python code via “pyrcc4” compiler bundled in PyQt. Details regarding implementation of these tools in FlowMapper are explained in Chapter 5.



FlowMapper Plugin for QGIS v2.0
v0.4 December 2013
http://plugins.qgis.org/plugins/FlowMapper

MIDDLE EAST TECHNICAL UNIVERSITY
Geodetic & Geographic Information Technologies Interdisciplinary Programme
Ankara / TURKEY

Developed by Cem GULLUOGLU (cempro@gmail.com)
under GNU General Public License v2

Qt Designer

Version 4.7.1
Qt Designer is a graphical user interface designer for Qt applications.

Copyright (C) 2010 Nokia Corporation and/or its subsidiary(-ies).

Object	Class
FormAbout	QDialog
label	QLabel
label_05	QLabel

Property	Value
QObject	
objectName	FormAbout
QWidget	
windowMo...	NonModal
enabled	☑
geometry	[(0, 0), 430 x 205]
X	0
Y	0
Width	430
Height	205

Sender	Signal	Receiver	Slot
pushButton	clicked()	FormAbout	reject

Figure 4.1. Qt Designer Running on Windows

QGIS is already built on top of Qt libraries. By using PyQt bindings along with PyQGIS, it is possible to develop custom applications in Python for specific GIS related needs by accessing QGIS and Qt libraries. PyQt brings all power and advantages of Qt framework within the clear syntax of Python. In this study, Qt Designer is used to interactively design the GUI of QGIS FlowMapper plugin. By means of the bundled compilers ("pyuic4", "pyrcc4") in PyQt, GUI design and resource files of FlowMapper plugin are transformed into Python code.

4.4. OGR Simple Features Library

OGR used to be an acronym for OpenGIS Simple Features Reference Implementation. However, since OGR is not fully compatible with the OpenGIS Simple Feature specification; the name was changed to Simple Features Library while the acronym OGR stayed same (OSGeo, 2014a).

OGR is a free and open source, cross platform C++ library for reading and writing vector data formats. As a part of the GDAL/OGR library, OGR is packed with GDAL (Geospatial Data Abstraction Library). In theory OGR is separate from GDAL, however both OGR and GDAL reside in the same source tree (OSGeo, 2014a). Similar to OGR, as a cross platform C++ library, GDAL incorporates similar read and write capabilities but on raster data formats. Both GDAL and OGR libraries are open source and distributed with the X11/MIT license. This type of licensing makes it possible to use these libraries even for building proprietary software.

GDAL project was initiated by Frank Warmerdam in 1998 (Warmerdam, 2008). Currently, GDAL/OGR is an OSGeo supported project and development efforts are welcomed under the umbrella of OSGeo (OSGeo, 2014b). GDAL/OGR provides useful command line utilities for data processing and translation. With reference to GDAL/OGR v1.10.1 released in 2013, library supports more than 130 raster formats (GDAL, 2014a) and more than 70 vector formats (GDAL, 2014b) (Table 4.7). GDAL/OGR library provides data access for many open source and commercial software; such as QGIS, GRASS, OpenEV, FME, Google Earth and ArcGIS (OSGeo, 2014b).

Table 4.7. Some Popular Vector and Raster Data Formats Supported by GDAL/OGR (adapted from OSGeo, 2014b)

Vector	Raster
ESRI Shapefile	TIFF, BigTIFF, GeoTIFF
ESRI Coverages	Erdas Imagine
ESRI Personal Geodatabase	PCI Geomatics Database File
MapInfo (including mid/mif and tab)	ESRI Grids
Microstation DGN	ECW
GML	MrSID
PostGIS	JPEG2000
Oracle Spatial	DTED

As a cross platform C++ translator library, GDAL/OGR binary packages are compiled against several platforms such as Windows, Ubuntu, Debian, OpenSUSE and MacOSX. Although GDAL/OGR library is written for C++, following bindings are available for development with other languages such as Perl, Python, Java, C#, Ruby and R. Unlike PyQGIS and PyQt bindings created using SIP, GDAL/OGR bindings are created using SWIG, which is also an open source tool for connecting C or C++ libraries to other languages.

GDAL/OGR is a widely accepted open source library due to its extensive support for many raster and vector data formats. As stated above, GDAL/OGR library provides data access for many open source software. For example QGIS release v2.0.1 is compiled against GDAL/OGR v1.10.0. GDAL/OGR provides ready to use, prebuilt command line utilities for data translation and processing. “Ogr2ogr” is a valuable command line utility included in OGR. It offers conversion of vector data to supported formats while offering extra operations during conversion such as attribute based subsetting, dropping attribute fields, defining output coordinate system. In this study, while OGR library bindings for Python is used to create shapefiles storing flow lines and flow node features; “ogr2ogr” command line utility is utilized for filtering shapefiles based on user defined attribute values and exporting shapefiles to Google Earth “kml” or MapInfo “tab” format.

In this chapter, development environment components required for implementing FlowMapper as a plugin to QGIS are reviewed in detail. These tools can be given as follows: (i) Python programming language, (ii) QGIS API and PyQGIS Python bindings, (iii) Qt development framework and PyQT Python bindings, (iv) GDAL/OGR library. In Chapter 5, details regarding development methodology and architecture of FlowMapper are given together with a review of modules written in Python.

CHAPTER 5

DEVELOPMENT OF FLOWMAPPER PLUGIN FOR QGIS

In previous chapters, fundamental concepts and history of flow mapping were reviewed and ongoing challenges in displaying spatial interaction data were identified from the literature. Besides, free and open source software concept was discussed. Having the motivation for open source development, QGIS was selected as the most promising open source desktop GIS application.

Development of a plugin under QGIS requires some software components which portray the development environment of FlowMapper plugin. These components, such as Python programming language, “Pythonic” API for QGIS and Python bindings for Qt framework, were introduced in Chapter 4.

In this chapter, development of QGIS FlowMapper plugin is explained in detail. At first, development methodology is presented with respect to the scope of the study and requirements of the flow mapping software. Then, architecture of the plugin is explored from the following respects; compliance to QGIS Python plugin structure, coding with Python, modules and dependencies, menu and GUI structure, functions and features implemented. At the end of this chapter, development history is reviewed and a side by side evaluation is presented among major releases. Besides, download and visitor statistics regarding the plugin repository and website are included at the end of this chapter.

5.1. Development Methodology of FlowMapper Plugin

Development methodology of FlowMapper plugin involves determination of software requirements prior to initiating coding. These requirements can be grouped under two headings: (i) requirements regarding development environment of FlowMapper and (ii) requirements regarding functional capabilities or features of FlowMapper. Methodology and stages involved in the development of QGIS FlowMapper plugin is given in Figure 5.1.

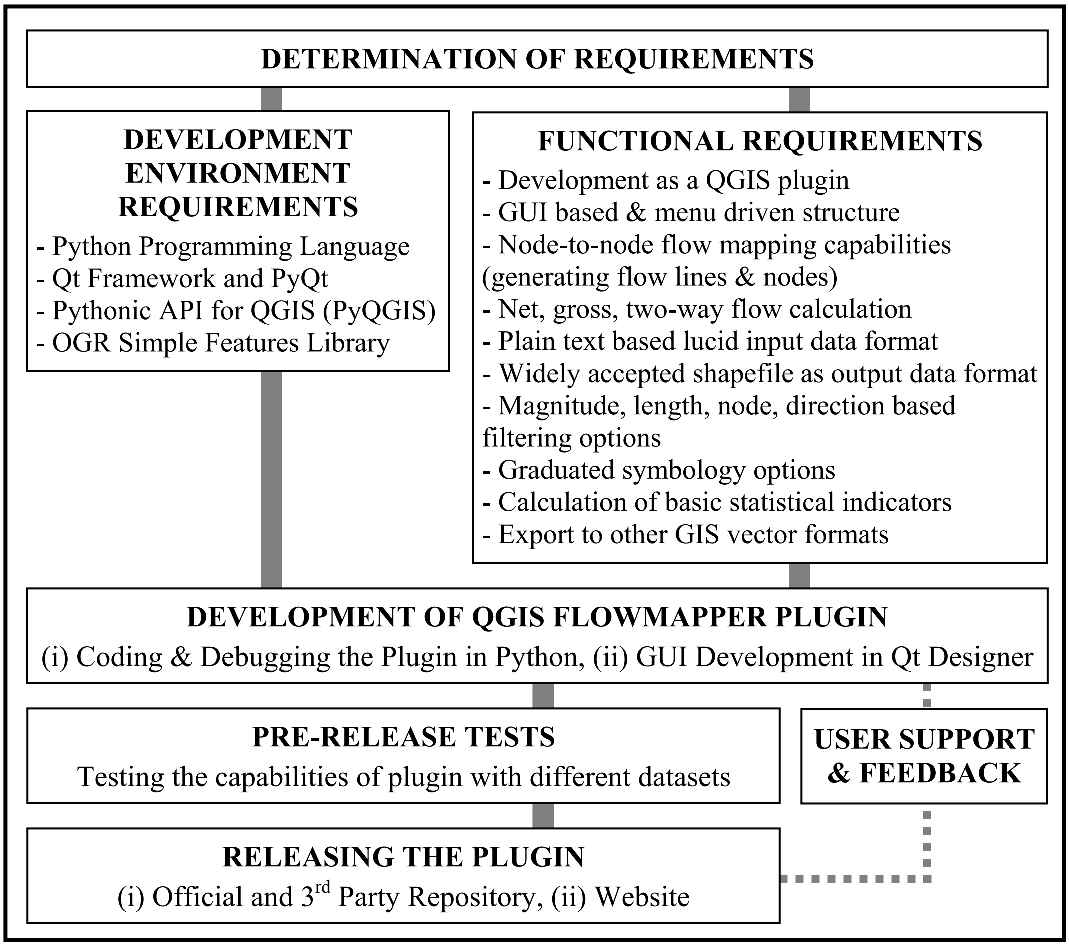

DETERMINATION OF REQUIREMENTS
DEVELOPMENT ENVIRONMENT REQUIREMENTS
- Python Programming Language
- Qt Framework and PyQt
- Pythonic API for QGIS (PyQGIS)
- OGR Simple Features Library
FUNCTIONAL REQUIREMENTS
- Development as a QGIS plugin
- GUI based & menu driven structure
- Node-to-node flow mapping capabilities (generating flow lines & nodes)
- Net, gross, two-way flow calculation
- Plain text based lucid input data format
- Widely accepted shapefile as output data format
- Magnitude, length, node, direction based filtering options
- Graduated symbology options
- Calculation of basic statistical indicators
- Export to other GIS vector formats
DEVELOPMENT OF QGIS FLOWMAPPER PLUGIN
(i) Coding & Debugging the Plugin in Python, (ii) GUI Development in Qt Designer
PRE-RELEASE TESTS
Testing the capabilities of plugin with different datasets
USER SUPPORT & FEEDBACK
RELEASING THE PLUGIN
(i) Official and 3rd Party Repository, (ii) Website

Figure 5.1. Development Methodology of FlowMapper Plugin

Parallel to the general methodology of this study, in order to identify all the requirements given in Figure 5.1, first a detailed literature survey is performed on fundamentals and ongoing challenges of flow mapping. By means of this survey, types of flows and visual clutter reduction techniques to be focused in FlowMapper are identified. Subsequently, free and open source software concept is reviewed together with open source desktop GIS applications. Based on this review, QGIS is selected as the core GIS component of FlowMapper. Afterwards, development environment components required for building FlowMapper as a QGIS plugin are identified. These components can be listed as follows: (i) Python as the programming language together with an IDE such as IDLE and preferably a powerful text editor such as Notepad++, (ii) Qt4 framework, Python bindings for Qt (PyQt), PyQt command line tools "pyrcc4", "pyuic4" and Qt Designer for GUI development, (iii) Python bindings for QGIS API (PyQGIS) and lastly (iv) OGR simple features library and its command line utility "ogr2ogr".

Since flow mapping is a domain where wide range of analysis and various representation techniques exist, functional capabilities of the plugin should be limited to ease development. Based on the findings of detailed literature survey on flow mapping in Chapter 2 and review of open source concept together with open source GIS applications in Chapter 3; functional requirements and structural characteristics of FlowMapper emerged as follows: (i) To ensure seamless GIS integration FlowMapper should be developed as a plugin to QGIS, therefore architecture must comply with the QGIS plugin structure, (ii) Similar to other flow mapping software, FlowMapper should also be designed GUI based and operations should be performed via menu driven structure, (iii) For ease of use input data format should be lucid and plain text based (e.g. tab or space delimited text file) so that it can be easily created and edited on any platform just with a text editor, (iv) Flow lines and nodes should be created in a commonly used GIS based vector data format, such as ESRI shapefile, in order to promote interoperability with other GIS software, (v) FlowMapper should be capable of generating flow lines between nodes via straight links which is identified as the most fundamental form of displaying spatial interactions during literature survey in Chapter 2, (vi)

FlowMapper should be capable of generating flow lines based on user selected interaction scenario such as net, gross or two way flows and attributes such as magnitude, length, origin-destination coordinates should be automatically calculated, (vii) Flow nodes should be generated as point features based on user supplied input test file storing either Cartesian or geographic coordinates, (viii) Attributes for flow nodes, such as node name, magnitude of incoming, outgoing, gross flows taking place at each node and a field for indicating whether the node is gaining or losing flow, should be automatically calculated based on input interaction matrix, (ix) As also discussed by Tobler (1987, 2003), in order to reduce visual complexity and reveal desired patterns in data, plugin should offer filtering capabilities based on magnitude, length, origin-destination node and direction of flows, (x) Cartographic visualization options must be offered for flow lines and nodes (e.g. single symbology mode, arrow heads to depict flow directions, graduated symbology mode to render colors and adjust line thickness or point size automatically based on magnitude), (xi) Users should be able to review basic statistical indicators (e.g. min, max, mean, variance) for flows prior to performing filtering or determining suitable intervals for graduated symbology, (xii) Plugin should be capable of exporting output shapefiles to some popular vector data formats (e.g. MapInfo tab and Google Earth kml).

After determining functional capabilities to be offered in FlowMapper, coding and debugging cycle of the plugin is initiated under QGIS development environment. To code a QGIS plugin with Python, developer needs to be familiar to Qt4 classes and tools as well as the basics of Python. In this study, at the development phase prior to starting coding, more than half a semester was spent to learn Python programming language and to understand QGIS plugin structure and dependencies. The Python tutorial prepared by Cogliati (2005) as a short book was very useful to make a gentle introduction to Python and to gain hands on coding experience. QGIS is developed with Qt framework and Python bindings for Qt (PyQt) is created using SIP which is a tool for writing Python bindings for C/C++ libraries. Similarly, Python bindings for QGIS API (PyQGIS) is also created using SIP tool. According to QGIS coding and compilation guide (QGIS, 2010), PyQt and PyQGIS are listed as the two other components for creating a QGIS Python plugin. After getting familiar with the development environment, five stage agile methodology defined in the PyQGIS developer cookbook (QGIS, 2014a) is applied. These stages are (i) idea, (ii) create files, (iii) write code, (iv) test and (v) publish. Idea stage includes setting the objectives and determination of functional capabilities which are already performed. Second stage includes creation of mandatory files that build the skeleton of a QGIS Python plugin (e.g. metadata.txt, resources.qrc etc.). For this stage, rather than coding these files from scratch, another Python plugin named as “QGIS Plugin Builder” was used to build the initial framework of FlowMapper plugin. Third and fourth stages refer to coding, debugging, GUI development and testing of FlowMapper. GUI development of FlowMapper is mostly done in Qt Designer while for coding and debugging cycle Notepad++ text editor, IDLE IDE, Python console embedded in QGIS, PyQt command line utilities “pyrcc4” and “pyuic4” are used. The last stage of development implies publishing the plugin in QGIS repository to make it available for public.

Coding of FlowMapper started at the beginning of December 2011 and after about four months the initial release v0.1 was published on the QGIS official plugin repository at the end of March 2012. Excluding the automatically compiled “resources.py” file, with more than 6.500 lines of code written in almost two years period, FlowMapper has reached a mature state with four major and several minor releases. All FlowMapper releases and their source codes are freely available for download through the QGIS official plugin repository hosted on “http://plugins.qgis.org/plugins/FlowMapper”. As of August 2014, almost after two and a half year from the first release, FlowMapper has been downloaded more than 12.000 times from the repository. This corresponds more than 10 downloads per day which reveals the demand for integration of flow mapping tools to a popular open source desktop GIS application.

All tools utilized for the development of FlowMapper (e.g. Python v2.6, Qt4, PyQt, PyQGIS, OGR) have both precompiled binary packages for major Linux distributions and installers for Windows OS. However, plugin development platform was setup on Windows 7 due to hands-on experience with Microsoft Windows OS. Since coding started on Windows platform, until release of FlowMapper v0.2.2, plugin was only tested on Windows 7 and XP. Staring from v0.2.2, all releases of FlowMapper offer cross platform support and known to be fully functional under Linux too.

One of the requirements in the development of FlowMapper is keeping the plugin user friendly and fully functional out of the box. To make a user friendly GUI, functions are designed to be accessible via simple menu driven structure that guides the user to relevant form. GUI design is fully performed with Qt Designer tool; however some dynamic form controls were required to be coded fully manually. In order to keep the plugin fully functional out of the box, no third party modules or libraries are used. In other words, FlowMapper only needs a QGIS installation on the system and is not dependent to any other library that is not needed by default QGIS installation. As a result of this structure, even novice QGIS users can start using the plugin just in minutes after installing it via QGIS plugin manager.

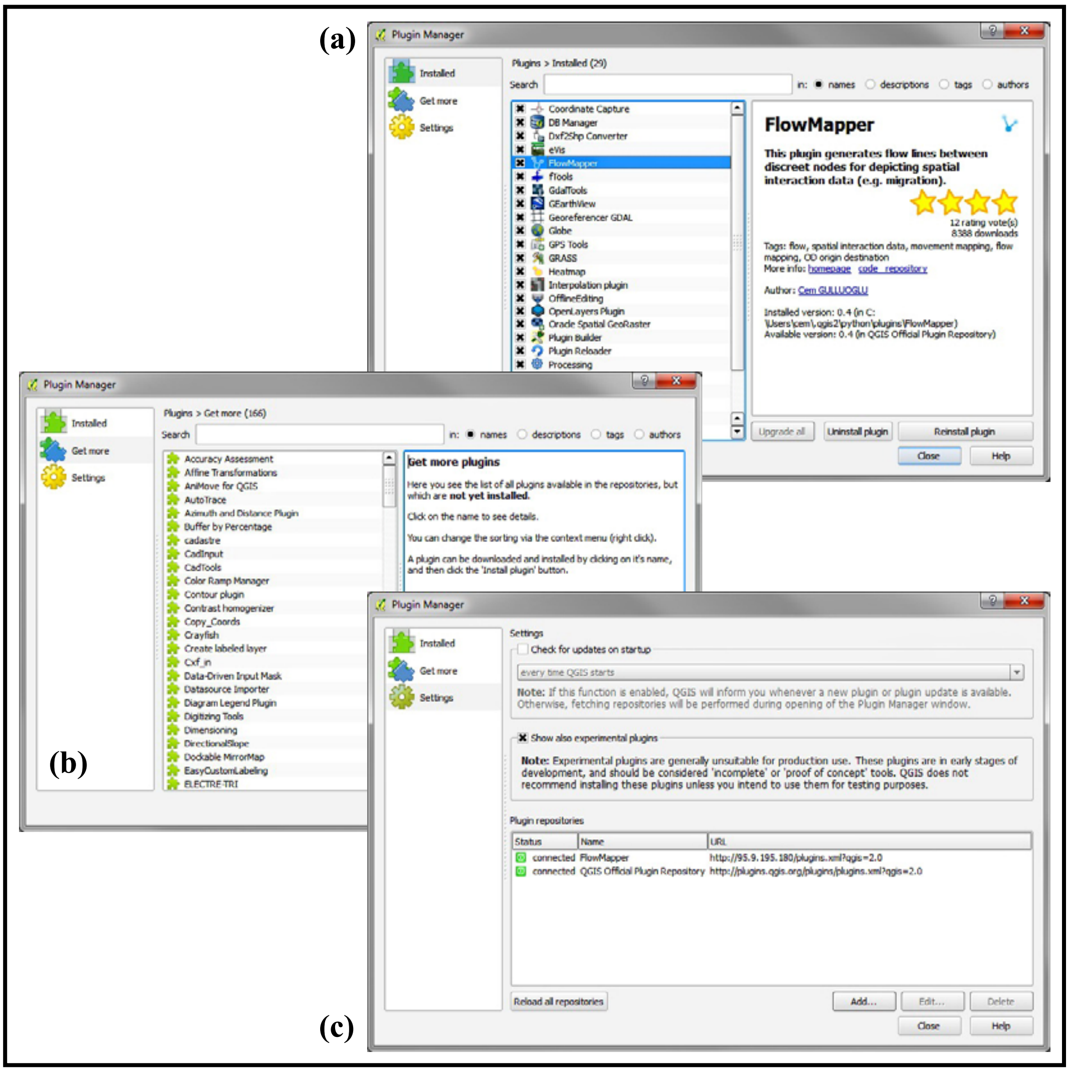
Development methodology of the plugin typically follows free and open source development methodology previously reviewed in Chapter 3. As a QGIS Python plugin, FlowMapper is also licensed with GNU GPL. Hence there is no restriction in terms of usage, redistribution and further development of the plugin as long as the source code is kept open. Inherently free and open source development promotes collaboration among developers and users. Although FlowMapper is coded by just one developer, there were also several contributions to development. For example, a code snippet from Flowpy, which is a Python script originally coded by Glennon (2009), was adapted to be reused in FlowMapper plugin. Besides, StackExchange GIS network was used for asking questions and searching answers about QGIS API related issues. Mails from users were often related with the usage of the plugin with different datasets. Considering these comments, starting from v0.1.1, FlowMapper is supplied with at least two test dataset and a brief documentation as a user manual. There were also few other user requests regarding the functions to be added to next releases of plugin. However, there were no requests from anyone for making active participation to coding.

5.2. Architecture of FlowMapper Plugin for QGIS

Inner structure and source code of FlowMapper are explored in details under this heading. These include explanations about the purpose of files located under the plugin folder, how the plugin works and review of some snippets from the source code. Besides, some details about the tools utilized for coding and GUI development are mentioned. A schema demonstrating the inner structure of the plugin is also given in Appendix A.

5.2.1. Structure of a Python Plugin and FlowMapper

Since FlowMapper is built as a plugin to QGIS, in order to understand the inner structure, it is a good starting point to review the purpose files located under the plugin directory regarding the structure explained in PyQGIS developer cookbook (QGIS, 2014a) and QGIS coding and compilation guide (QGIS, 2010). As stated in the following resources (QGIS, 2010, QGIS, 2014a and QGIS, 2014c), features of QGIS can be extended via plugins either written in C++ or Python. There are two types of QGIS plugins: (i) core and (ii) external (QGIS, 2014c). Core plugins are maintained by the QGIS development team and they are already included in every QGIS distribution. Core plugins are generally written in C++ however there are also some core plugins written in Python. In contrast to core plugins, external plugins are stored in external repositories and maintained by their authors. Currently, all external QGIS plugins are written in Python (QGIS, 2014c). The main outcome of this study, FlowMapper is also an external plugin. Plugin is being stored both under the QGIS official plugin repository "http://plugins.qgis.org/plugins/plugins.xml?qgis=2.0" and under the dedicated repository which is created only for this study “http://95.9.195.180/plugins.xml”. By means of the plugin manager, core and external plugins can be managed, new plugins can be automatically installed and external repositories can be added (Figure 5.2).



FlowMapper

This plugin generates flow lines between discreet nodes for depicting spatial interaction data (e.g. migration).

12 rating vote(s)
8388 downloads

Tags: flow, spatial interaction data, movement mapping, flow mapping, OD origin destination
More info: homepage code repository

Author: Cem GULLUOGLU

Installed version: 0.4 (in C:\Users\cem\.qgis2\python\plugins\FlowMapper)
Available version: 0.4 (in QGIS Official Plugin Repository)

Get more plugins

Here you see the list of all plugins available in the repositories, but which are **not yet installed**.

Click on the name to see details.

You can change the sorting via the context menu (right click).

A plugin can be downloaded and installed by clicking on it's name, and then click the 'Install plugin' button.

Settings

□ Check for updates on startup

every time QGIS starts

Note: If this function is enabled, QGIS will inform you whenever a new plugin or plugin update is available. Otherwise, fetching repositories will be performed during opening of the Plugin Manager window.

☒ Show also experimental plugins

Note: Experimental plugins are generally unsuitable for production use. These plugins are in early stages of development, and should be considered 'incomplete' or 'proof of concept' tools. QGIS does not recommend installing these plugins unless you intend to use them for testing purposes.

Plugin repositories

Status	Name	URL
connected	FlowMapper	http://95.9.195.180/plugins.xml?qgis=2.0
connected	QGIS Official Plugin Repository	http://plugins.qgis.org/plugins/plugins.xml?qgis=2.0

Figure 5.2. QGIS v2.0 Plugin Manager: (a) Manage installed core & external plugins, (b) Download and install more plugins from repositories, (c) Manage official and third party repositories

QGIS Python plugins depend on the functionality of shared libraries “libqgis_core” and “libqgis_gui” (QGIS, 2010). Since both of these libraries are licensed under GNU GPL, any derivative work utilizing these libraries must also be licensed under the GNU GPL. In other words; anyone can develop a QGIS Python plugin and use it for individual needs without being forced to publish it. However, if the plugin is published, source code must also be published under the GNU GPL license. In Chapter 3, this necessity of GNU GPL was explained with the "copyleft" concept. As it should be, FlowMapper plugin is also published under the GNU GPL v2 with its Python "py" files which include the human readable source code.

When a user needs a new external plugin (e.g. FlowMapper), it must be downloaded from the repository and compressed "zip" file contents need to be extracted into a folder that holds the same name with the plugin. Then this folder must be moved to the location where QGIS looks for the plugins. Depending on the operating system (e.g. Windows OS, Linux OS or MacOSX), QGIS scans subdirectories under the following paths and initializes the Python plugins it finds.

For Linux: "plugins" path for QGIS 1.8.0 Lisboa on Xubuntu 12.04.2 LTS

```
./usr/share/qgis/python/plugins/
home/$USERNAME/.qgis/python/plugins/
```

For Mac: "plugins" path for QGIS 1.8.0 Lisboa on Mac OS X Maverics

```
./Applications/QGIS.app/Contents/Resources/python/plugins/
./Users/$USERNAME/.qgis/python/plugins/
```

For Windows: "plugins" path for QGIS 2.2 Valmiera (64bit) on Windows 7

```
C:\Program Files\QGIS Dufour\apps\qgis\python\plugins\
C:\Users\$USERNAME\.qgis2\python\plugins\
```

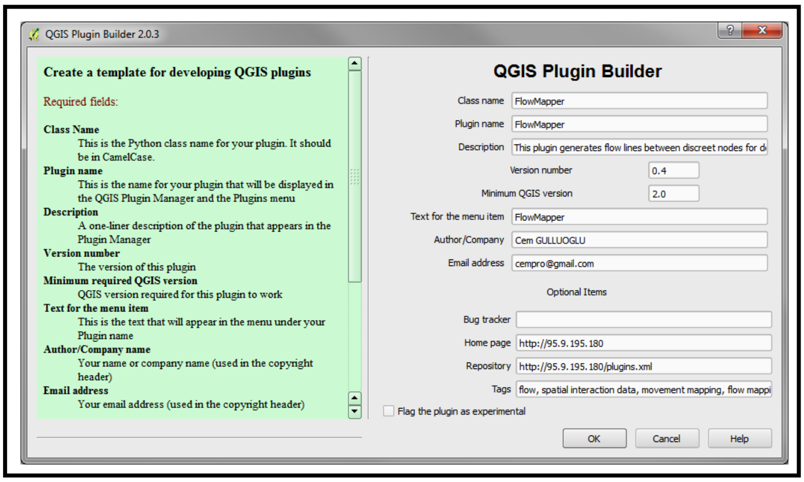
Based on one of the two locations where the plugin resides, plugin will be either loaded for all users or only for the current user indicated with the "$USERNAME" variable. For example, if user wants FlowMapper plugin to be loaded for all users, paths written on the first lines should be used otherwise paths specified on the second lines should be preferred.

QGIS plugin manager (Figure 5.2) provides a user friendly interface for managing both core and external plugins. Instead of manually downloading and extracting the contents into plugins directory, new plugins can be automatically downloaded from repositories and installed by means of the plugin manager shown in Figure 5.2 (b). By default, plugin manager installs external plugins under the user specific path; thus these plugins will be only loaded for the user who has installed them.

Adding a new plugin to QGIS requires downloading contents of the plugin within a compressed “zip” file. For security concerns, there is an approval mechanism for the plugins those uploaded to the official QGIS repository. When a new plugin is uploaded to the official repository, it needs to be reviewed and approved by an administrator; otherwise that plugin is not listed as downloadable. However, there is no such mechanism for third party private repositories maintained by individual developers. This is a security flaw since installing a plugin involves execution of the Python code contained in the plugin folder. So, users should install plugins from third party repositories on their own risk.

Starting from version 0.9, developers can write a QGIS plugin either in C++ or Python language. While C++ is a compiled language that is generally regarded as running fast, Python is an interpreted language which does not require recompilation of whole source code while testing small sections in software. This means, when developing with C++ whole source code must be transformed into a set of computer specific instructions prior to execution. However, when developing in Python, the source code is modified and saved in the same format that is still human readable. Thus, unlike QGIS core plugins those written in C++ and distributed in binary forms (e.g. as “dll” for Windows), Python plugins come with “py” files which store the human readable source code. When QGIS is started, Python “py” source code files (e.g. __init__.py, flowmapper.py) residing under plugins directory are interpreted and automatically compiled into Python “pyc” byte code files (e.g. __init__.pyc, flowmapper.pyc). Once this transformation is completed; these “pyc” files are used to run the plugin until there is any change in the source code stored in “py” files. However, if an error is found during transformation, relevant plugin is automatically disabled by QGIS and a warning message is propagated with an error log for debugging purposes. This log includes some vital information about the source of error such as name of “py” file and line number of erroneous code. This is very valuable because by looking at that information, plugin developer can debug and fix erroneous code. Moreover, initialization cycle of a plugin can be automated by means of another QGIS Python plugin named as “Plugin Reloader” which eases reloading of a plugin without needing to restart QGIS. During development of FlowMapper, this mechanism was found very useful and practical since it made coding and debugging cycle possible almost on any platform by only needing a text editor (e.g. Notepad++) and a QGIS installation.

As previously mentioned under the methodology section, the second stage of five staged development methodology defined in the PyQGIS developer cookbook (QGIS, 2014a) involves creation of mandatory files which will build the skeleton for a plugin. Rather than coding these files from scratch, a Python plugin named “QGIS Plugin Builder” (Sherman et al., 2014) was used to build the skeleton of FlowMapper plugin (Figure 5.3). This provided a working template on which the plugin is further developed regarding the functional requirements that FlowMapper should satisfy.



Create a template for developing QGIS plugins

Required fields:

Class Name
This is the Python class name for your plugin. It should be in CamelCase.
Plugin name
This is the name for your plugin that will be displayed in the QGIS Plugin Manager and the Plugins menu
Description
A one-liner description of the plugin that appears in the Plugin Manager
Version number
The version of this plugin
Minimum required QGIS version
QGIS version required for this plugin to work
Text for the menu item
This is the text that will appear in the menu under your Plugin name
Author/Company name
Your name or company name (used in the copyright header)
Email address
Your email address (used in the copyright header)

QGIS Plugin Builder

Class name: FlowMapper
Plugin name: FlowMapper
Description: This plugin generates flow lines between discreet nodes for d
Version number: 0.4
Minimum QGIS version: 2.0
Text for the menu item: FlowMapper
Author/Company: Cem GULLUOGLU
Email address: cempro@gmail.com

Optional Items

Bug tracker:
Home page: http://95.9.195.180
Repository: http://95.9.195.180/plugins.xml
Tags: flow, spatial interaction data, movement mapping, flow mappi
□ Flag the plugin as experimental

Figure 5.3. Interface of QGIS Plugin Builder

In Table 5.1, directory structure of a typical Pyhon plugin is given together with the directory structure of FlowMapper. Besides, a brief explanation regarding the purpose of each file is also given in the table.

Table 5.1. Directory Structure of a Typical Python Plugin and FlowMapper

<table>
<tr><th>Generic Python Plugin</th><th colspan="3">Purpose of File</th><th>FlowMapper</th></tr>
<tr><td>…\plugins\GenericPlugin</td><td colspan="3">Plugin directory</td><td>…\plugins\FlowMapper</td></tr>
<tr><td>__init__.py</td><td colspan="3">Starting point; QGIS initializes the plugin from __init__.py</td><td>__init__.py</td></tr>
<tr><td>metadata.txt</td><td colspan="3">Contains metadata about the plugin in plain text format *</td><td>metadata.txt</td></tr>
<tr><td>resources.qrc</td><td colspan="3">Contains paths to resources used in forms. “qrc” file is XML based</td><td>resources.qrc</td></tr>
<tr><td>resources.py</td><td colspan="3">Translation of “resources.qrc” file to Python language</td><td>resources.py</td></tr>
<tr><td>plugin.py</td><td colspan="3">Main module of the plugin</td><td>flowmapper.py</td></tr>
<tr><td>N/A</td><td colspan="3">Additional module that creates flow lines, flow nodes and attribute tables</td><td>flowpyv07.py</td></tr>
<tr><td>form.ui</td><td>Main module form</td><td rowspan="9">GUI file created by Qt Designer in XML format</td><td>Main module form</td><td>ui_flowmapper.ui</td></tr>
<tr><td rowspan="8">N/A</td><td rowspan="8">N/A</td><td>About form</td><td>ui_about.ui</td></tr>
<tr><td>Filter by magnitude f.</td><td>ui_form2.ui</td></tr>
<tr><td>Filter by length form</td><td>ui_form3.ui</td></tr>
<tr><td>Symbology for lines form</td><td>ui_form4.ui</td></tr>
<tr><td>Export form</td><td>ui_form5.ui</td></tr>
<tr><td>Filter by node & direction f.</td><td>ui_form6.ui</td></tr>
<tr><td>Symbology for nodes form</td><td>ui_form7.ui</td></tr>
</table>

Table 5.1. Directory Structure of a Typical Python Plugin and FlowMapper (cont.)

Generic Python Plugin	Purpose of File			FlowMapper
form.py	Main module form	Translation of XML based “ui” file to Python language	Main module form	ui_flowmapper.py
N/A	N/A		About form	ui_about.py
			Filter by magnitude f.	ui_form2.py
			Filter by length form	ui_form3.py
			Symbology for lines form	ui_form4.py
			Export form	ui_form5.py
			Filter by node & direction f.	ui_form6.py
			Symbology for nodes f.	ui_form7.py
N/A	N/A	In order to access designer objects & GUI interaction **	Main module form	flowmapperdialog.py
			About form	form_aboutdialog.py
			Filter by magnitude f.	form2dialog.py
			Filter by length form	form3dialog.py
			Symbology for lines form	form4dialog.py
			Export form	form5dialog.py
			Filter by node & direction f.	form6dialog.py
			Symbology for nodes f.	form7dialog.py

* Beginning with QGIS v1.8 “metadata.txt” is the preferred way to supply information about a plugin. Embedding metadata into “__init__.py” will not be supported for QGIS ≥ v2.0.

** A separate Python file is not mandatory but advisable for accessing Qt Designer objects. This provides a middle layer between the main module and “py” files translated from “ui” files.

It can be inferred from Table 5.1 that the structure of FlowMapper is almost identical with the structure of a generic plugin defined in the PyQGIS developer cookbook (QGIS, 2014a) except for the files with “dialog” postfix which are used for accessing Qt Designer objects that exist in form interfaces. These Python files with “dialog” postfix (e.g. flowmapperdialog.py) act as a middle layer between the main body of plugin (e.g. flowmapper.py) and Qt Designer files (e.g. ui_flowmapper.ui) which are translated to Python files (e.g. ui_flowmapper.py) by using "pyuic4" command line tool. To express more technically, "flowmapperdialog.py" module subclasses "QtGui.QDialog" class and wraps "ui_flowmapper.py" file. By default, this is the recommended structure deployed by the QGIS Plugin Builder (Sherman et al., 2014). One advantage of this hierarchy is that since it abstracts the setup of user interface, developer does not have to deal with the user interface setup in the main Python module (flowmapper.py). Thus, as FlowMapper is coded, dialog specific properties such as setting values to form objects that interact with buttons are implemented via Python files having "dialog" postfix (e.g. flowmapperdialog.py, form2dialog.py, form3dialog.py etc.)

Assuming that a QGIS version equal or later than 2.0 is installed on the system; when QGIS is started, QGIS first looks for "__init__.py" and "metadata.txt" files under the FlowMapper folder. Upon locating them, QGIS parses "classFactory(iface)" function from "__init__.py" file and loads "FlowMapper" class from "flowmapper.py". In order to load and unload a plugin, respectively "initGui()" and unload() functions are used. These functions are located under "FlowMapper" class and "initGui()" function uses icons from the resource file "resources.py". By means of "initGui()" function, respectively; (i) an action is created to start plugin configuration, (ii) this action is connected to the "run(self)" method and (iii) by using "addToolBarIcon" and "addPluginToMenu" slots, toolbar icon and menu items are added to QGIS GUI. At the beginning of "flowmapper.py" file, "FlowMapperDialog" class is imported from "flowmapperdialog.py" file for interface interaction. When action connecting the form to the run method is triggered, based on interface design stored in "ui_flowmapper.py" file form is shown to user. This mechanism also applies to rest of the forms. The only difference is that rather than the "run(self)" method, each action created to start configuration is connected to the method which is specific for that form (e.g. "form2(self)", "from3(self)", "about(self)" etc.). This method calls only the related "QtGui.QDialog" classes in order to interact with the user interface and show the form (e.g. “about(self)” method calls “Form_AboutDialog(QtGui.QDialog)” class from “form_aboutdialog.py” file which also calls “Ui_FormAbout(object)” class from “ui_about.py” file.)

Up to this point, folder structure reflecting the architecture of a QGIS Python plugin is examined. Then, details regarding how FlowMapper plugin is initialized by QGIS are given. Besides, procedure for interacting with user interface files, accessing designer objects and making forms visible to users are explained. From this point on, content and purpose of mandatory files building the skeleton of the plugin will be examined in detail before turning focus to GUI development, coding of main module and implementations to meet functional requirements.

Starting point of FlowMapper is the “__init__.py” file which makes it known to QGIS as a Python plugin together with the “metadata.txt” file. As shown in Table 5.2, “classFactory(iface)” (Line 3) method imports “FlowMapper” (Line 5) class from “flowmapper.py” file and returns “iface” (Line 6) object which is the reference for communicating with the QGIS interface in a Python plugin.

Table 5.2. Source Code of “__init__.py” File for FlowMapper

#	**Source Code of __init__.py**
1	`# This script initializes the plugin`
2	`# and makes it known to QGIS.`
3	`def classFactory(iface):`
4	`    # load FlowMapper class from file FlowMapper`
5	`    from flowmapper import FlowMapper`
6	`    return FlowMapper(iface)`

Starting with the release of QGIS v1.8, “metadata.txt” file is the recommended way of providing general information about a plugin. Embedding metadata tags as methods in the “__init__.py” file is obsolete and with the release of QGIS v2.0, “metadata.txt” became the only way that is accepted. This information is used by the QGIS plugin manager and also needed by the QGIS official plugin repository when a plugin is uploaded. As plugin is initialized by QGIS, plugin manager needs to retrieve some mandatory information such as name and description of the plugin, email of the author etc. This is performed through “metadata.txt” file. In Table 5.3, a list of mandatory and optional metadata tags are given. Besides, in Table 5.4, content of “metadata.txt” file for FlowMapper is presented.

Table 5.3. Metadata Tags for QGIS Python Plugins

Metadata Tag	Type	Explanation
name	Mandatory	Name of the plugin in string data type
description	Mandatory	Short description of the plugin in string data type
version	Mandatory	Version of the plugin in dotted notation (e.g. 0.2.3)
author	Mandatory	Name of the developer
email	Mandatory	Email of the developer
qgisMinimumVersion	Mandatory	Min. QGIS version required by the plugin in dotted notation
qgisMaximumVersion	Optional	Max. QGIS version that the plugin works in dotted notation
changelog	Optional	In string data type and can be multiline
experimental	Optional	Indicates that the plugin is experimental (True or False)
deprecated	Optional	Indicates that the plugin is deprecated (True or False)
tags	Optional	Comma separated keywords list for the plugin
homepage	Optional	URL of the plugin’s homepage
repository	Optional	URL of the plugin’s source code repository
tracker	Optional	URL of the plugin’s bug tracker
icon	Optional	Filename or a relative path for the plugin icon
category	Optional	Category of the plugin such as raster, vector, database, web

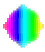
Table 5.4. Content of “metadata.txt” File for FlowMapper

Content of metadata.txt

```
# This file contains metadata for the plugin.

# Mandatory items:
[general]
name=FlowMapper
description=This plugin generates flow lines between
            discrete nodes for depicting spatial
            interaction data (e.g. migration).
version=0.4
author=Cem GULLUOGLU
email=cempro@gmail.com
qgisMinimumVersion=2.0
# End of mandatory metadata

# Optional items:
experimental=False
tags= flow, flow mapping, spatial interaction data
homepage=http://95.9.195.180
repository=http://95.9.195.180/plugins.xml
icon=icon.png
# End of optional metadata
```

Another file listed in Table 5.1 is the “resources.qrc” file which is required for setting the icons used in FlowMapper form interfaces. In Table 5.5 (b), source code of “resources.qrc” for FlowMapper is given. This file is written in XML language and includes the path (Line 2) and name of the icons (Lines 3 – 10). As shown in Table 5.5 (a), eight different icons are designed for FlowMapper in “png” format and each icon is used in one form. After creating icons and editing “resources.qrc” file, this file is translated to Python language by using the PyQt tool “pyrcc4” with the following command line “pyrcc4 –o resources.py resources.qrc”. During development of FlowMapper, this translation is performed each time a new icon is created and added to the “resources.qrc” file. After transformation, as shown in Table 5.5 (c), all icons are stored in hexadecimal notation within the “resources.py” file (Lines 3 – 13).

Table 5.5. FlowMapper Icon Set and Source Code of Resources Files for FlowMapper: (a) FlowMapper Icon Set, (b) XML based “resources.qrc” file, (c) “resources.py” file in Python language

(a) FlowMapper Icon Set							
icon.png	icon2.png	icon3.png	icon4.png	icon5.png	icon6.png	icon7.png	icon8.png
							

(b) Source code of resources.qrc in XML

```xml
<RCC>
    <qresource prefix="/plugins/flowmapper" >
        <file>icon.png</file>
        <file>icon2.png</file>
        <file>icon3.png</file>
        <file>icon4.png</file>
        <file>icon5.png</file>
        <file>icon6.png</file>
        <file>icon7.png</file>
        <file>icon8.png</file>
    </qresource>
</RCC>
```

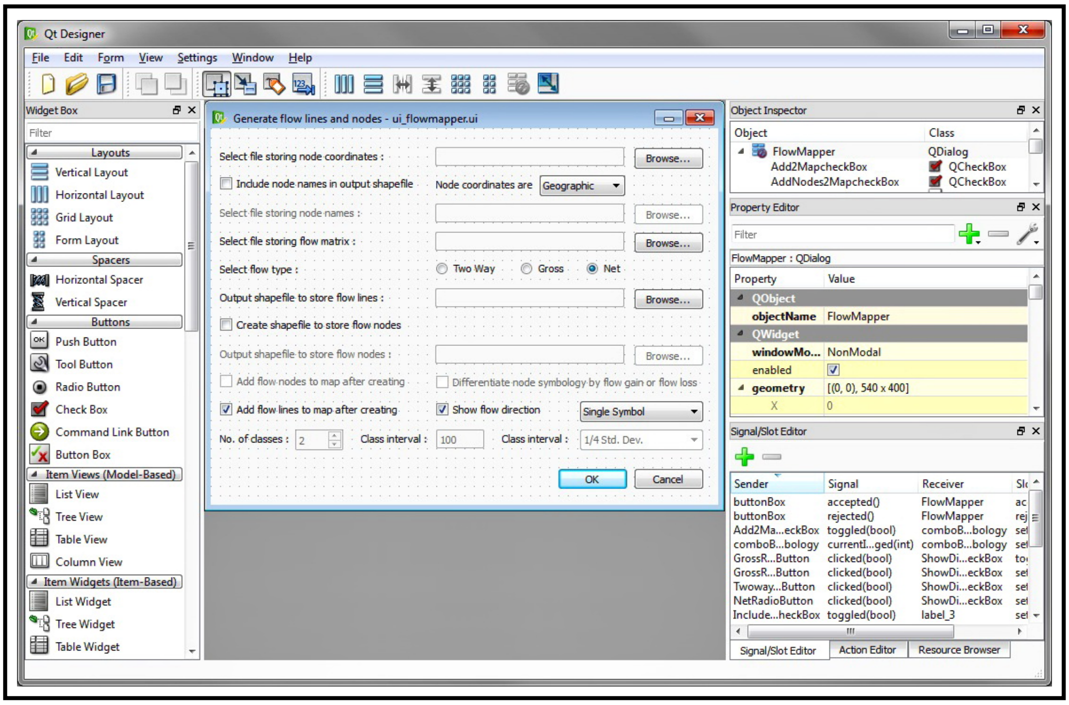
(c) Source code of resources.py in Python

```python
from PyQt4 import QtCore

qt_resource_data = "\
\x00\x00\x0d\x2b\
\x89\
\x50\x4e\x47\x0d\x0a\x1a\x0a\x00\x00\x00\x0d\x49\x48\x44\x52\x00\
\x00\x00\x18\x00\x00\x00\x18\x08\x06\x00\x00\x00\xe0\x77\x3d\xf8\
\x00\x00\x00\x09\x70\x48\x59\x73\x00\x00\x0b\x13\x00\x00\x0b\x13\
...
...
\x00\x00\x00\xbe\x00\x00\x00\x00\x00\x01\x00\x00\x54\x45\
\x00\x00\x00\xd6\x00\x00\x00\x00\x00\x01\x00\x00\x62\x08\
"

def qInitResources():
    QtCore.qRegisterResourceData(0x01, qt_resource_struct, \
    qt_resource_name, qt_resource_data)

def qCleanupResources():
    QtCore.qUnregisterResourceData(0x01, qt_resource_struct, \
    qt_resource_name, qt_resource_data)

qInitResources()
```

5.2.2. GUI Development for FlowMapper

The third and fourth stages of plugin development methodology mentioned in the PyQGIS developer cookbook (QGIS, 2014a) involve coding, debugging cycle and GUI development. Before focusing on the source code of main module “flowmapper.py” and flow generator module “flowpyv07.py”, details regarding GUI development are presented since these modules interact with user interface objects.

One of the requirements of FlowMapper is defined as a user-friendly GUI where operations can be performed via simple menu driven structure that guides user to relevant form interface. GUI design of FlowMapper plugin is fully performed by using Qt Designer which is installed as a part of Qt4 framework. Screenshot of Qt Designer tool and GUI of the form that is used for creating flow lines and nodes are given in Figure 5.4.



Generate flow lines and nodes - ui_flowmapper.ui

Select file storing node coordinates :

□ Include node names in output shapefile   Node coordinates are Geographic

Select file storing node names :

Select file storing flow matrix :

Select flow type : ○ Two Way ○ Gross ● Net

Output shapefile to store flow lines :

□ Create shapefile to store flow nodes

Output shapefile to store flow nodes :

□ Add flow nodes to map after creating   □ Differentiate node symbology by flow gain or flow loss

☑ Add flow lines to map after creating   ☑ Show flow direction   Single Symbol

No. of classes : 2   Class interval : 100   Class interval : 1/4 Std. Dev.

Object	Class
FlowMapper	QDialog
Add2MapcheckBox	QCheckBox
AddNodes2MapcheckBox	QCheckBox

FlowMapper : QDialog

Property	Value
QObject	
objectName	FlowMapper
QWidget	
windowMo...	NonModal
enabled	☑
geometry	[(0, 0), 540 x 400]
X	0

Sender	Signal	Receiver	Sl
buttonBox	accepted()	FlowMapper	ac
buttonBox	rejected()	FlowMapper	rej
Add2Ma...eckBox	toggled(bool)	comboB...bology	se
comboB...bology	currentI...ged(int)	comboB...bology	se
GrossR...Button	clicked(bool)	ShowDi...eckBox	to
GrossR...Button	clicked(bool)	ShowDi...eckBox	se
Twoway...Button	clicked(bool)	ShowDi...eckBox	se
NetRadioButton	clicked(bool)	ShowDi...eckBox	se
Include...heckBox	toggled(bool)	label_3	se

Figure 5.4. GUI Development for FlowMapper with Qt Designer Tool

Qt Designer offers various predefined GUI elements, named as widgets (e.g. push button, combo box, line edit, spin box etc.), which can be easily used in drag and drop manner. Once a form design is completed, its layout is saved as Qt Designer “ui” file. This file is XML based and stores all properties of GUI elements (e.g. form, widgets, signal – slot parameters etc.). Similar to the translation performed on “resources.qrc” file, XML based Qt Designer “ui” file also needs to be translated to Python language. This is performed by running the PyQt tool “pyuic4” with the following command line “pyrcc4 –o ui_flowmapper.py ui_flowmapper.ui”. In Table 5.6, a part of the source code from XML based “ui_flowmapper.ui” file is given together with several lines of Python source code from “ui_flowmapper.py” file. Once translation to Python language is completed, form design and all properties of GUI elements are automatically created under the “Ui_FlowMapper” class as given in Table 5.6 (b) starting from the line no. 7.

Table 5.6. Part of the Source Code for GUI Development: (a) XML based Qt Designer “ui” file

(a) Source code of ui_flowmapper.ui in XML

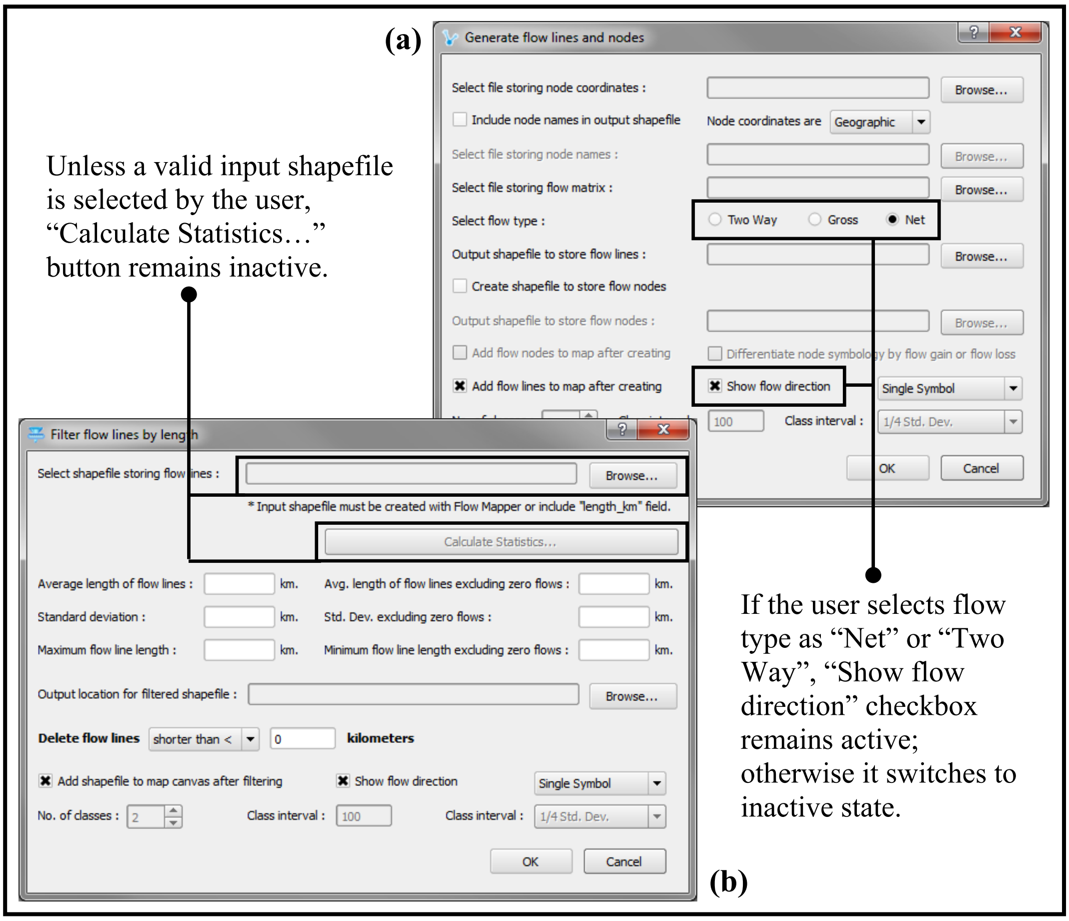
```xml
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>FlowMapper</class>
 <widget class="QDialog" name="FlowMapper">
  <property name="windowModality">
   <enum>Qt::NonModal</enum>
  </property>
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>540</width>
    <height>400</height>
   </rect>
  </property>
...
...
  <property name="windowTitle">
   <string>Generate flow lines and nodes</string>
  </property>
  <property name="windowIcon">
   <iconset resource="resources.qrc">
    <normaloff>:/plugins/flowmapper/icon.png</normaloff>
   :/plugins/flowmapper/icon.png</iconset>
  </property>
 ...
 ...
```

Table 5.6. Part of the Source Code for GUI Development (cont.): (b) Python translated “py” file

#	(b) Source code of ui_flowmapper.py in Python

```
1   from PyQt4 import QtCore, QtGui
2   try:
3       _fromUtf8 = QtCore.QString.fromUtf8
4   except AttributeError:
5       _fromUtf8 = lambda s: s
6
7   class Ui_FlowMapper(object):
8       def setupUi(self, FlowMapper):
9           FlowMapper.setObjectName(_fromUtf8("FlowMapper"))
10          FlowMapper.setWindowModality(QtCore.Qt.NonModal)
11          FlowMapper.resize(540, 400)
12  ...
13  ...
14          icon = QtGui.QIcon()
15          icon.addPixmap(QtGui.QPixmap \
16          (_fromUtf8(":/plugins/flowmapper/icon.png")), \
17          QtGui.QIcon.Normal, QtGui.QIcon.Off)
18          FlowMapper.setWindowIcon(icon)
19  ...
20  ...
21  import resources
```

Form design and properties of widgets are stored in XML based Qt designer “ui” files. For example, in Table 5.6 (a), form dimensions are defined in lines 12 and 13 which reside under geometry property tag (Lines 8 – 15). Similarly, form title is defined in line 19 and properties of form icon are given between lines 21 and 25. Once Python translation is performed, all these properties are expressed in Python syntax. For example, in Table 5.6 (b), form dimensions are set in line 11 and form icon properties are defined between lines 14 and 18. Two sample form interfaces designed for FlowMapper by using Qt Designer are given in Figure 5.5.



Generate flow lines and nodes

Select file storing node coordinates : Browse...

☐ Include node names in output shapefile   Node coordinates are Geographic

Select file storing node names : Browse...

Select file storing flow matrix : Browse...

Select flow type : ○ Two Way ○ Gross ● Net

Output shapefile to store flow lines : Browse...

☐ Create shapefile to store flow nodes

Output shapefile to store flow nodes : Browse...

☐ Add flow nodes to map after creating   ☐ Differentiate node symbology by flow gain or flow loss

☒ Add flow lines to map after creating   ☒ Show flow direction   Single Symbol

100   Class interval : 1/4 Std. Dev.

OK   Cancel

Filter flow lines by length

Select shapefile storing flow lines : Browse...

* Input shapefile must be created with Flow Mapper or include "length_km" field.

Calculate Statistics...

Average length of flow lines : km.   Avg. length of flow lines excluding zero flows : km.

Standard deviation : km.   Std. Dev. excluding zero flows : km.

Maximum flow line length : km.   Minimum flow line length excluding zero flows : km.

Output location for filtered shapefile : Browse...

Delete flow lines shorter than < 0 **kilometers**

☒ Add shapefile to map canvas after filtering   ☒ Show flow direction   Single Symbol

No. of classes : 2   Class interval : 100   Class interval : 1/4 Std. Dev.

OK   Cancel

Unless a valid input shapefile is selected by the user, "Calculate Statistics…" button remains inactive.

If the user selects flow type as "Net" or "Two Way", "Show flow direction" checkbox remains active; otherwise it switches to inactive state.

Figure 5.5. GUI of FlowMapper: (a) "Generate flow lines and nodes" form, (b) "Filter Flow Lines by Length" form

In FlowMapper GUI, properties of some widgets change depending on the state of other widgets which interact with user. For example, on the "Generate flow lines and nodes" form shown on Figure 5.5(a), "enabled" and "checked" properties for "Show flow direction" checkbox is "True" by default. However, if user selects the flow type as "Gross" by checking the radio button, "enabled" and "checked" states of "Show flow direction" checkbox are automatically changed to "False" since gross flows do not imply any direction. Similarly, on the "Filter flow lines by length" form shown on Figure 5.5(b), "enabled" state for "Calculate Statistics…" button is "False" by default; in other words button is grayed out and cannot be clicked unless a shapefile is selected. However, when user clicks "Browse…" button and selects a valid input shapefile, "Calculate Statistics…" button become active and user can populate descriptive statistics upon clicking the button. Among these two examples, there are many other dynamic controls written for each form. These dynamic controls are manually written in Python by using Qt's signal and slot mechanism which can be used to communicate between objects. In Qt, a signal is emitted when an event occurs (e.g. when a user clicks a button). In response to a signal, a slot can be called as a function. While a signal can be connected to a single slot, it is also possible to connect a single signal to many slots or many signals to a single slot. Implementations for dynamically controlling properties of "Show flow direction" checkbox and "Calculate Statistics…" button are given in Table 5.7 with two different code snippets.

Table 5.7. Controlling Properties of GUI Objects: (a) Code snippet from "ui_flowmapper.py" for controlling the properties of "Show flow direction" checkbox via Qt's signal – slot mechanism

(a) Sample code for signal – slot mechanism from ui_flowmapper.py

```
from PyQt4 import QtCore, QtGui
...
...
class Ui_FlowMapper(object):
    def setupUi(self, FlowMapper):
        FlowMapper.setObjectName(_fromUtf8("FlowMapper"))
...
...
        self.retranslateUi(FlowMapper)
...
...
        # Connect click event of “GrossRadioButton”
        # to “uncheck_ShowDirectioncheckBox” method
        QtCore.QObject.connect(self.GrossRadioButton, \
        QtCore.SIGNAL(_fromUtf8("clicked(bool)")), \
        self.uncheck_ShowDirectioncheckBox)
        # Connect click event of “GrossRadioButton”
        # to disable “ShowDirectioncheckBox”
        QtCore.QObject.connect(self.GrossRadioButton, \
        QtCore.SIGNAL(_fromUtf8("clicked(bool)")), \
        self.ShowDirectioncheckBox.setDisabled)
        # Connect click event of “TwowayRadioButton”
        # to enable “ShowDirectioncheckBox”
        QtCore.QObject.connect(self.TwowayRadioButton, \
        QtCore.SIGNAL(_fromUtf8("clicked(bool)")), \
        self.ShowDirectioncheckBox.setEnabled)
        # Connect click event of “NetRadioButton”
        # to enable “ShowDirectioncheckBox”
        QtCore.QObject.connect(self.NetRadioButton, \
        QtCore.SIGNAL(_fromUtf8("clicked(bool)")), \
```

Table 5.7. Controlling Properties of GUI Objects (cont.): (a) Code snippet from “ui_flowmapper.py” for controlling the properties of “Show flow direction” checkbox via Qt’s signal – slot mechanism, (b) Code snippet from “form3dialog.py” to change the state of “Calculate Statistics…” button

(a) Sample code for signal – slot mechanism from ui_flowmapper.py

```
        self.ShowDirectioncheckBox.setEnabled)
...
...
        QtCore.QMetaObject.connectSlotsByName(FlowMapper)
...
...
    def uncheck_ShowDirectioncheckBox(self):
        # Uncheck “ShowDirectioncheckBox”
        self.ShowDirectioncheckBox.setChecked(False)
...
...
import resources
```

(b) Sample code for signal – slot mechanism from form3dialog.py

```
import sys, os
from os.path import isfile

from PyQt4 import QtCore, QtGui
from ui_form3 import Ui_Form
# import flowmapper.py main module
import flowmapper

# Create the dialog for Form3
class Form3Dialog(QtGui.QDialog):
    def __init__(self):
        QtGui.QDialog.__init__(self)
        # Set up the user interface from Designer.
        self.ui = Ui_Form()
        self.ui.setupUi(self)

    # Set input shapefile name and path to be filtered by length
    def SetTextBrowseInputShapeFilterLength(self):
        self.ui.BrowseShapeLineEdit.setText \
        (flowmapper.InputShpFilterLengthName)
        InputShpFilterLengthDirectory = \
        flowmapper.InputShpFilterLengthName
        if len(flowmapper.InputShpFilterLengthName) > 0:
            # Check the path whether it points to a shapefile
            extension = os.path.splitext \
            (str(InputShpFilterLengthDirectory))[1]
            if extension == ".shp":
                # Change state from Disabled to Enabled
                self.ui.CalStatLength.setEnabled(True)
...
...
```

Based on Table 5.7 (a), when user clicks to the “Gross” radio button (Lines 14 – 16), a signal from the “GrossRadioButton” form object is connected to the “uncheck_ShowDirectioncheckBox” method to change the “setChecked” property of “ShowDirectioncheckBox” form object to “False” (Lines 37 – 39). By using another signal – slot mechanism, “Enabled” state of “ShowDirectioncheckBox” form object is changed to “Disabled” (Lines 19 – 21). In contrast, when user clicks on “Two Way” or “Net” radio buttons, state of “ShowDirectioncheckBox” object is set to “Enabled” since these types of flows imply direction (Lines 24 – 26, Lines 29 – 31). In Table 5.7 (b), upon user browses for the input file, path is checked whether it points to a shapefile or not (Lines 25 – 26). If the input is a shapefile (Line 27), “Disabled” state of “CalStatLength” form object is changed to “Enabled” (Line 29). During GUI development of FlowMapper, same logic and identical methodology is applied for the rest of the forms in order to perform form designs and code form controls.

5.2.3. Development of Main Module and Flow Generator Module

Main module of FlowMapper plugin is implemented in the “flowmapper.py” file while flow lines, flow nodes are created and attribute field calculations are performed in the “flowpyv07.py” file. Totally, comprising about more than 3.000 lines of Python code, these two files are at the core of FlowMapper. Yet, equivalent to more than a hundred pages of Python code, it is not convenient to review every line of these files in this section. Instead, implementation of several functions, such as how flow lines and flow nodes are created and graduated symbology schema is applied or how filtering operations are performed upon calculation of basic statistical indicators, are going to be reviewed with respect to code snippets quoted from these files. For example, in Table 5.8, related parts of the source code taken from “flowmapper.py” file are presented in order to explain how flow lines and flow nodes are created in shapefile format and then added into map layout. Besides, since “flowpyv07.py” file is imported at the beginning of "flowmapper.py" file, full source code of "shapefilemaker" function that is implemented in "flowpyv07.py" file is given in Appendix B.

Table 5.8. Partial Content of Main Module: "flowmapper.py"

Code snippets from flowmapper.py

```
# import standard Python modules sys & os
import sys
import os
# from os.path import realpath & isfile functions
from os.path import realpath
from os.path import isfile
# import QGIS core library
from qgis.core import *
# import qgis.gui that brings GUI components e.g. map canvas
import qgis.gui
from qgis.gui import *
# import QtCore and QtGui modules
from PyQt4 import QtCore, QtGui
# QtCore contains non-GUI functionality
from PyQt4.QtCore import *
# QtGui extends QtCore with GUI functionality.
from PyQt4.QtGui import *
#import ogr module from GDAL to work with vectors
import ogr
from osgeo import ogr
# initialize Qt resources from file resources.py
import resources
# include flow generator module
import flowpyv07
# import FlowMapperDialog class to interact with GUI objects
import flowmapperdialog
from flowmapperdialog import FlowMapperDialog
...
...
class FlowMapper:
    def __init__(self, iface):
        # set reference to the QGIS interface
        self.iface = iface
    def initGui(self):
        # create action to initialize plugin configuration
        self.action = Qaction \
        (QIcon(":/plugins/flowmapper/icon.png"), \
        "Generate flow lines and nodes", self.iface.mainWindow())
...
...
        # connect action to the run method
        QObject.connect \
        (self.action, SIGNAL("triggered()"), self.run)
...
...
        # add toolbar button and menu item to QGIS GUI
        self.iface.addToolBarIcon(self.action)
        self.iface.addPluginToMenu("&FlowMapper", self.action)
...
...
```

Table 5.8. Partial Content of Main Module: “flowmapper.py” (cont.)

Code snippets from flowmapper.py

```
    def unload(self):
        # remove toolbar button and menu item from QGIS GUI
        self.iface.removePluginMenu("&FlowMapper",self.action)
        self.iface.removeToolBarIcon(self.action)
...
...
    def OutputShp(self):
        dlg = FlowMapperDialog()
        fd = QtGui.QFileDialog(dlg)
        # define as global variable
        global SaveShpName
        global SaveShpDirectory
        # browse to set shapefile name for output flow lines
        SaveShpName = fd.getSaveFileName \
        (None, 'Shapefile(*.shp)','Type output file name','*.shp')
    def OutputShpNodes(self):
        dlg = FlowMapperDialog()
        fd = QtGui.QFileDialog(dlg)
        # define as global variable
        global SaveShpNameNodes
        global SaveShpDirectoryNodes
        # browse to set shapefile name for output flow nodes
        SaveShpNameNodes = fd.getSaveFileName \
        (None, 'Shapefile(*.shp)','Type output file name','*.shp')
    def InputNodes(self):
        dlg = FlowMapperDialog()
        fd = QtGui.QFileDialog(dlg)
        # define as global variable
        global InputNodesName
        # browse to select input text file storing node coord.
        InputNodesName = fd.getOpenFileName \
        (None, 'Text Files(*.txt)', 'Select txt file', '*.txt')
    def InputNodeNames(self):
        dlg = FlowMapperDialog()
        fd = QtGui.QFileDialog(dlg)
        # define as global variable
        global InputNodeNamesName
        # browse to select input text file storing node names
        InputNodeNamesName = fd.getOpenFileName \
        (None, 'Text Files(*.txt)', 'Select txt file', '*.txt')
    def InputMatrix(self):
        dlg = FlowMapperDialog()
        fd = QtGui.QFileDialog(dlg)
        # define as global variable
        global InputMatrixName
        # browse to select input text file storing flow matrix
        InputMatrixName = fd.getOpenFileName \
        (None, 'Text Files(*.txt)', 'Select txt file', '*.txt')
...
...
    def run(self):
        # create and show “Generate flow lines and nodes” dialog
        dlg = FlowMapperDialog()
        dlg.show()
        # connect to method to browse input or output file names
        # then connect to flowmapperdialog to interact with GUI
        QtCore.QObject.connect(dlg.ui.BrowseShape,QtCore.SIGNAL \
        ("clicked()"), self.OutputShp)
```

Table 5.8. Partial Content of Main Module: “flowmapper.py” (cont.)

Code snippets from flowmapper.py

```
        QtCore.QObject.connect(dlg.ui.BrowseShape,QtCore.SIGNAL \
        ("clicked()"), dlg.SetTextBrowseShape)
        QtCore.QObject.connect(dlg.ui.BrowseNodes,QtCore.SIGNAL \
        ("clicked()"), self.InputNodes)
        QtCore.QObject.connect(dlg.ui.BrowseNodes,QtCore.SIGNAL \
        ("clicked()"), dlg.SetTextBrowseNodes)
        QtCore.QObject.connect(dlg.ui.BrowseNodeNames, \
        QtCore.SIGNAL("clicked()"), self.InputNodeNames)
        QtCore.QObject.connect(dlg.ui.BrowseNodeNames, \
        QtCore.SIGNAL("clicked()"), dlg.SetTextBrowseNodeNames)
        QtCore.QObject.connect(dlg.ui.BrowseShapeNodes, \
        QtCore.SIGNAL("clicked()"), self.OutputShpNodes)
        QtCore.QObject.connect(dlg.ui.BrowseShapeNodes, \
        QtCore.SIGNAL("clicked()"), dlg.SetTextBrowseShapeNodes)
        QtCore.QObject.connect(dlg.ui.BrowseMatrix, \
        QtCore.SIGNAL("clicked()"), self.InputMatrix)
        QtCore.QObject.connect(dlg.ui.BrowseMatrix, \
        QtCore.SIGNAL("clicked()"), dlg.SetTextBrowseMatrix)
        # check if OK button is clicked
        result = dlg.exec_()
        if result == 1:
            global SaveShpName
            global SaveShpNameNodes
            global InputNodeNamesName
            global InputNodesName
            global combotext
            SaveDirectory = SaveShpName
            # get type of coordinates (geographic or cartesian)
            combotext = str(dlg.ui.comboBox.currentText())
            # check if include node names checkbox is checked
            if dlg.ui.IncludeNodeNamescheckBox.isChecked()==False:
                IncludeNodeNames = 0
                InputNodeNamesName = InputNodesName
            else:
                IncludeNodeNames = 1
            # check if create flow nodes checkbox is checked
            if dlg.ui.CreateFlowNodescheckBox.isChecked()==False:
                CreateShpNodes = 0
                SaveShpNameNodes = "NULL"
            else:
                CreateShpNodes = 1
            # set flow type based on user selection
            if dlg.ui.TwowayRadioButton.isChecked():
                FlowType = 1
            elif dlg.ui.GrossRadioButton.isChecked():
                FlowType = 2
            elif dlg.ui.NetRadioButton.isChecked():
                FlowType = 3
            # connect to shapefilemaker function in flow generator
            # module to create flow lines, flow nodes and to
            # calculate attributes based on supplied parameters
            flowpyv07.shapefilemaker(FlowType, CreateShpNodes, \
            IncludeNodeNames, str(SaveDirectory), \
            str(SaveShpName), str(SaveShpNameNodes), \
            str(InputMatrixName), str(InputNodesName), \
            str(InputNodeNamesName), str(combotext))
```

Table 5.8. Partial Content of Main Module: “flowmapper.py” (cont.)

Code snippets from flowmapper.py

```
            # if add to map is not checked, only show success msg.
            if dlg.ui.Add2MapcheckBox.isChecked()==False:
                SuccessMessage = \
                str(SaveShpName) + " created successfully !"
                QMessageBox.information \
                (self.iface.mainWindow(), "info", \
                SuccessMessage, "Close")
            else:
...
...
                # if add to map + show direction boxes are checked
                # and single symbology is selected
                elif dlg.ui.comboBoxSelectSymbology.currentText() \
                =="Single Symbol" and \
                dlg.ui.ShowDirectioncheckBox.isChecked()==True:
                    # set shapefile layer name, path and driver
                    layer = QgsVectorLayer \
                    (str(SaveShpName),str(SaveShpName),'ogr')
                    # create symbology
                    registry = QgsSymbolLayerV2Registry.instance()
                    lineMeta = \
                    registry.symbolLayerMetadata("SimpleLine")
                    markerMeta = \
                    registry.symbolLayerMetadata("MarkerLine")
                    # get layer geometry type
                    symbol = QgsSymbolV2.defaultSymbol \
                    (layer.geometryType())
                    # define symbology properties for flow line
                    lineLayer = lineMeta.createSymbolLayer \
                    ({'width': '0.26', 'color': '255,0,0', \
                    'offset': '0', 'penstyle': 'solid', \
                    'use_custom_dash': '0', 'joinstyle': 'bevel', \
                    'capstyle': 'square'})
                    # define symbology for flow direction marker
                    markerLayer = markerMeta.createSymbolLayer \
                    ({'width': '0.26', 'color': '255,0,0', \
                    'rotate': '1', 'placement': 'centralpoint', \
                    'offset': '0'})
                    subSymbol = markerLayer.subSymbol()
                    # replace default layer with simplemarker
                    subSymbol.deleteSymbolLayer(0)
                    triangle = registry.symbolLayerMetadata \
                    ("SimpleMarker").createSymbolLayer({'name': \
                    'filled_arrowhead', 'color': '255,0,0', \
                    'color_border': '0,0,0', 'offset': '0,0', \
                    'size': '3', 'angle': '0'})
                    subSymbol.appendSymbolLayer(triangle)
                    # replace default layer with custom layers
                    symbol.deleteSymbolLayer(0)
                    symbol.appendSymbolLayer(lineLayer)
                    symbol.appendSymbolLayer(markerLayer)
                    # replace renderer of current layer
                    renderer = QgsSingleSymbolRendererV2(symbol)
                    layer.setRendererV2(renderer)
                    # add layer to map and show success message
                    QgsMapLayerRegistry.instance().addMapLayer \
                    (layer)
                    SuccessMessage = \
                    str(SaveShpName) + " created successfully !"
```

Table 5.8. Partial Content of Main Module: “flowmapper.py” (cont.)

Code snippets from flowmapper.py

```
                    QMessageBox.information \
                    (self.iface.mainWindow(), "info", \
                    SuccessMessage, "Close")
...
...
            # only show success message if add flow nodes to map
            # checkbox is not checked
            if dlg.ui.CreateFlowNodescheckBox.isChecked()==True \
            and dlg.ui.AddNodes2MapcheckBox.isChecked()==False:
                SuccessMessage = \
                str(SaveShpNameNodes) + " created successfully !"
                QMessageBox.information(self.iface.mainWindow(), \
                "info", SuccessMessage, "Close")
...
...
            # determine if user wants to add flow nodes to map
            # and differentiate flow gaining and loosing nodes
            elif dlg.ui.CreateFlowNodescheckBox.isChecked()==True \
            and dlg.ui.AddNodes2MapcheckBox.isChecked()==True \
            and dlg.ui.DifNodeSymbologycheckBox.isChecked()==True:
                # create symbology
                registry=QgsSymbolLayerV2Registry.instance()
                markerMeta=registry.symbolLayerMetadata \
                ("MarkerLine")
                # get layer geometry type
                def validatedDefaultSymbol(geometryType):
                    symbol=QgsSymbolV2.defaultSymbol(geometryType)
                    if symbol is None:
                        if geometryType == QGis.Point:
                            symbol = QgsMarkerSymbolV2()
                        elif geometryType == QGis.Line:
                            symbol =  QgsLineSymbolV2()
                        elif geometryType == QGis.Polygon:
                            symbol = QgsFillSymbolV2()
                    return symbol
                # create default symbology for flow nodes
                def makeSymbologyForRange \
                (layer, min ,max, label ,colour, alpha, size):
                    symbol = validatedDefaultSymbol \
                    (layer.geometryType())
                    symbol.setColor(colour)
                    symbol.setAlpha(alpha)
                    symbol.setSize(size)
                    range = QgsRendererRangeV2 \
                    (min, max, symbol, label)
                    return range
                # set shapefile layer name, path and driver
                vlayer = QgsVectorLayer(str(SaveShpNameNodes), \
                str(SaveShpNameNodes),'ogr')
                # set indicator attribute field
                myTargetField = 'indicator'
                # define class labels for flow nodes
                classlabel1 = \
               "nodes gaining flows (incoming>outgoing)"
                classlabel2 = \
                "nodes losing flows (incoming>outgoing)"
                classlabel3 = "neutral nodes (incoming=outgoing)"
```

Table 5.8. Partial Content of Main Module: “flowmapper.py” (cont.)

#	Code snippets from flowmapper.py
281	# set parameters for three symbology classes
282	myRangeList = []
283	myRangeList.append(makeSymbologyForRange \
284	(vlayer,0.9,2,classlabel1,QColor(0,192,0),1,2))
285	myRangeList.append(makeSymbologyForRange(vlayer, \
286	-2,-0.9,classlabel2,QColor(255,0,0),1,2))
287	myRangeList.append(makeSymbologyForRange(vlayer, \
288	0,0,classlabel3,QColor(128,128,128),1,2))
289	# replace renderer of current layer
290	myRenderer = QgsGraduatedSymbolRendererV2 \
291	(myTargetField, myRangeList)
292	vlayer.setRendererV2(myRenderer)
293	# add layer to map and show success message
294	QgsMapLayerRegistry.instance().addMapLayer(vlayer)
295	message = str(SaveShpNameNodes) + \
296	" created successfully !"
297	QMessageBox.information(self.iface.mainWindow(), \
298	"info" ,message)

Not all content of the source code is given in Table 5.8, however general layout of “flowmapper.py” is organized as follows: Required libraries, classes, modules and functions are imported at the beginning (Lines 1 – 29). All methods and functions in the main module are implemented under the “FlowMapper” class (Lines 30 – 298). This class includes several methods, Python functions and implementation of Qt’s signal – slot mechanism in order to perform such operations: (i) initialization of plugin configuration and menu structure (Lines 31 – 56), (ii) setting global variables (Lines 57 – 100), (iii) calculation of descriptive statistics, (iv) implementation of methods to show forms and run form operations (Lines 101 – 298), (v) calling “shapefilemaker” function from flow generator module with user defined variables (Lines 160 – 164), (vi) executing “ogr2ogr” command line utility to filter flow lines by length or magnitude and (vii) applying symbology to flow lines (Lines 177 – 226) and flow nodes (Lines 241 – 298) based on user preference.

A typical scenario is selected to explain the code snippet quoted from main module “flowmapper.py” (Table 5.8) and to examine the content of flow generator module “flowpyv07.py” (Appendix B, Table B.1). This typical scenario reflects some common operations performed by users for creating and visualizing flow lines. These operations can be given as follows: user (i) selects “Generate flow lines and nodes” sub menu item from the “Plugins” menu under where the FlowMapper is located; (ii) browses to select input files in which node coordinates, node names and interaction matrix are stored; (iii) types output file names to store flow lines and flow nodes; (iv) selects flow type (e.g. net); (v) determines whether to add output files into map layout (e.g. checks add files to map checkbox); (vi) chooses desired symbology (e.g. single symbology); (vii) decides whether to indicate direction of flows and finally (viii) decides whether to differentiate symbology of flow gaining and flow losing nodes. Upon clicking “OK” button, FlowMapper creates flow lines, flow nodes and calculates feature attributes; then adds created files into map window based on user defined preferences.

In order to perform operations defined in the scenario above; at the beginning of “flowmapper.py” (Table 5.8), standard Python modules “sys” and “os”, QGIS core library and GUI module “qgis.core” and “qgis.gui”, PyQt4 core and GUI modules “QtCore” and “QtGui”, “ogr” module from OGR/GDAL, Qt resources “resources.py”, flow generator module “flowpyv07.py” and form dialog classes are imported (Lines 1 – 29). At this point, it should be evoked that all these modules and libraries needed to run FlowMapper are also needed by QGIS and come with the default installation of QGIS. When QGIS starts, FlowMapper plugin is initiated by using “__init.py__” file. This file passes the “iface” object to the main module “flowmapper.py” (Lines 31 – 33). By using the “iface” object, it is possible to interact with the “Pythonic” QGIS API or in other words PyQGIS. By means of the “initGui” method (Lines 34 – 50), plugin configuration is initialized and sub menu structure for FlowMapper is built under “Plugins” menu. When the user clicks “Generate flow lines and nodes” sub menu item, “run” method of the form is triggered through the Qt’s signal – slot mechanism (Lines 42 – 43) and shown to the user (Lines 103 – 104). When the user clicks browse button to select input files; such as the file storing node coordinates, names or interaction matrix; a connection is triggered to each relevant method (e.g. Lines 123 – 124 triggers the “InputMatrix” method to select interaction matrix). By using

“QFileDialog” and “getOpenFileName” method, file name and path is assigned to the global variable “InputMatrixName” (Lines 91 – 98). Just after assigning file name and path to the global variable as string, “SetTextBrowseMatrix” method is called from the “FlowMapperDialog” class in order to populate the textbox content that shows the file path in GUI (Lines 125 – 126). This mechanism is almost similar for all input and output files; however “getSaveFileName” (Line 64) method is used for output files instead of “getOpenFileName” method which is used to locate input files. After setting input and output files, user determines type of flow (e.g. net) that is to be created and calculated (Lines 151 – 156). The user should also determine whether to generate flow nodes (Lines 145 – 149) and to include names of nodes as feature attributes (Lines 139 – 143). Besides, parameter to define the type of input node coordinates (Line 137) and option to show created features on map are set by the user. Proper selection of coordinate type (e.g. decimal degrees or Cartesian) has vital importance for the calculation of flow length. If the user wants to add flow features on map upon creation, it is possible to set several symbology options from the user interface (e.g. single symbology representation or graduated symbology representation such as equal interval or defined interval, indicating directions of net and two way flows, differentiating symbology of nodes by flow gain or loss). After setting all these parameters, when the user clicks to the “OK” button on the form, “shapefilemaker” function is called from the flow generator module “flowpyv07.py” and executed with respect to the parameters assigned to global variables (Lines 160 – 164).

The “shapefilemaker” function that is implemented in the flow generator module “flowpyv07.py” is used to create flow lines and flow nodes in shapefile format (Table B.1 in Appendix B). While most parts of the “shapefilemaker” function are coded from scratch especially for FlowMapper plugin, some parts of it involve code reuse from the “Flowpy” script of Glennon (2009). To create flow features in shapefile format, the “shapefilemaker” function requires (i) a set of coordinate pairs either geographic or Cartesian (Line 21), (ii) a square interaction matrix (Line 20) and (iii) determination of flow type such as net, gross or two way. Besides, some additional parameters such as “IncludeNodeNames” and

“CreateShpNodes” should be passed to the “shapefilemaker” function in order to populate origin and destination node names for flow lines and to trigger creation of flow nodes (Lines 79 – 163) which are optional by default. Flow generator module needs only few modules to run. These are “os”, “sys” and “math” as standard Python modules and “ogr” module from OGR/GDAL library to handle shapefile vector format (Lines 17 – 18). Besides, it is also possible to run “shapefilemaker” function outside of QGIS by manually setting required parameters for the function if a Python version greater than v2.0 and its corresponding bindings for OGR/GDAL are installed on the system.

General layout of flow generator module “flowpyv07.py” is organized as follows: (i) required modules are imported at the beginning (Lines 17 – 18); (ii) paths of input files are assigned to variables (Lines 20 – 22); (iii) input files (node coordinates, node names and interaction matrix) are opened read only (Lines 25 – 31); (iv) total number of nodes is determined (Lines 33 – 36) and entries stored in each input file are appended into a list object as elements (Lines 38 – 77); (v) if “CreateShpNodes” parameters is set to “1”, which means that the user also wants to create flow nodes, first a blank shapefile is created by using the “ogr” driver (Lines 79 – 86); (vi) using the same driver, attribute fields are created (Lines 87 – 105), (vii) after creating empty shapefile with attribute fields, via “ogr” driver, flow nodes are inserted into the shapefile as “wkbPoint” geometry features by accessing the coordinate pairs in the list object “mypoints[ ]” (Lines 107 – 117); (viii) attributes for flow nodes, such as node names and total amount of outgoing/incoming flows from/to that node, are calculated by accessing the list objects “mypointnames[ ]” and “myodmatrix[ ]” (Lines 117 – 163); (ix) similar to the method applied for creating flow nodes, depending on the flow type selected by the user, which is “net” for this scenario (Line 337) , first an empty shapefile is created with attribute fields (Lines 165 – 194) and then flow lines are inserted into the shapefile as “wkbLineString” geometry features by accessing the coordinates of origin and destination nodes stored in the list object “mypoints[ ]” (Lines 348 – 353 and Lines 404 – 409); (x) attributes for flow lines, such as magnitude, names and coordinates of origin – destination nodes, are calculated by accessing the list objects “mypointnames[ ]” and “myodmatrix[ ]” (Lines 354 – 377 and Lines 410 – 433); besides lengths of flow lines are calculated depending on the type of input coordinates (Lines 382 – 400 for geographic coordinates, Lines 379 – 381 for Cartesian coordinates); (xi) after populating contents of attribute fields for each flow line feature, shapefile is closed and “ogr” driver is destroyed (Lines 464 – 466).

With reference to the use case that the code structure of the main module is being examined (Table 5.8), upon creation of flow lines and nodes in shapefile format, these files are automatically added into the QGIS map window with the following configuration: (i) flow lines displaying net flows are rendered with direction arrows in single symbolgy mode; (ii) flow nodes are rendered to differentiate whether the node is gaining or losing flows depending on the cumulative interactions taking place on that node. The first part is initiated in Line 175 and completed in Line 226 by populating a success message upon adding flows to map. The second part is initiated in Line 241 and completed in Line 298 with a similar success message. In order to perform these operations, layer name and path of the vector layer is set by using the “ogr” driver (Lines 181 – 182 for lines, Lines 271 – 272 for nodes). Then desired symbology template is created by using symbol layers (Lines 183 – 188 for lines, Lines 244 – 247 for nodes). In QGIS, using custom symbol layers make possible to overlay marker symbols (e.g. arrows) on lines to indicate flow direction. For flow lines, after defining properties of custom line layer and custom marker layer (Lines 192 – 215), this symbology is rendered in single symbology mode (Lines 216 – 218) and added to map by calling the “addMapLayer” method (Lines 220 – 221). For flow nodes, methodology is similar; however flow nodes are rendered in graduated symbology mode (Lines 289 – 292). Hence, symbology properties and class labels for each graduated interval are separately defined and appended to the range list of graduated renderer (Lines 281 – 288).

Creation of flow lines and flow nodes as well as calculation of attributes for flow features are the core capabilities of FlowMapper plugin which meet the most fundamental requirements expected of a flow mapping software. However, capabilities of FlowMapper plugin are beyond this. Additional capabilities of FlowMapper can be listed as follows: (i) filtering flow lines either by length or magnitude; (ii) filtering flow lines by node and direction to reveal flows either incoming to or outgoing from a node; (iii) calculating basic statistical indicators for flow features (e.g. minimum, maximum., mean, standard deviation, number of features); (iv) rendering of automatically or manually adjusted graduated symbology for flow lines and nodes based on magnitude field; (v) exporting created vector features to Google Earth “kml” and MapInfo “tab” formats.

Code snippet implemented for calculating descriptive statistics is given in Table 5.9. These indicators (Lines 4 – 12) supply preliminary information to the user about the content of dataset before performing any operation on data such as filtering features by magnitude or filtering flows by length or deciding number of classes, intervals for graduated symbology representation. Besides, these indicators are also required by the Python code that builds subclasses for graduated symbology representation (for example, in graduated symbology mode when the user enters the number of equal interval subclasses, min. and max. should be known to calculate the range; then range is divided to the number of subclasses to get the interval.)

Table 5.9. Python Code for Calculating Descriptive Statistics

#	Code snippet for calculating descriptive statistics

```
 1  def CalStatFilter(self):
 2
 3      # create variables to hold descriptive statistics
 4      global CalStatNoOfFeatures # no. of features in shapefile
 5      global CalStatNoOfFeaturesZero # no.of feat.having magnitude>0
 6      global CalStatSumMagnitude # total magnitude of flows in shp
 7      global CalStatMaxFlow # max. flow magnitude in shp
 8      global CalStatMinFlow # min. flow magnitude in shp
 9      global CalStatMean # average flow magnitude in shp
10      global CalStatMeanNotZero # avg.mag.excl.feat.having mag.=0
11      global CalStatStdDev # standard deviation of flows in shp
12      global CalStatStdDevNotZero # std.dev.excl.feat.having mag.=0
13
14      # open and read shapefile by using "ogr" driver
15      shp = ogr.Open(str(InputShpFilterDirectory),1)
16      layer = shp.GetLayer(0)
```

Table 5.9. Python Code for Calculating Descriptive Statistics (cont.)

Code snippet for calculating descriptive statistics

```
# get number of features in shapefile
CalStatNoOfFeatures = str(layer.GetFeatureCount())

feature = layer.GetNextFeature() # read first feature
counterNotZero = 0
CalStatSumMagnitude = 0
CalStatList = [] # create list object
while feature:
    magnitude = feature.GetField('magnitude') # get flow mag.
    if magnitude > 0:
        CalStatList.append(int(magnitude))
        CalStatMinFlow = min(CalStatList)
        CalStatMaxFlow = max(CalStatList)
        counterNotZero = counterNotZero + 1
    CalStatSumMagnitude = CalStatSumMagnitude + magnitude
    feature.Destroy() # destroy feature
    feature = layer.GetNextFeature() # read next feature
# assign values to variables
CalStatMean = str((CalStatSumMagnitude / \
              int(CalStatNoOfFeatures)))
CalStatMeanNotZero = str(CalStatSumMagnitude / \
                     int(counterNotZero))
CalStatSumMagnitude = str(CalStatSumMagnitude)
CalStatNoOfFeaturesZero = str(int(CalStatNoOfFeatures)- \
                          int(counterNotZero))
CalStatMaxFlow = str(int(CalStatMaxFlow))
CalStatMinFlow = str(int(CalStatMinFlow))
layer.ResetReading() # reset reading shapefile

feature = layer.GetNextFeature() # read first feature
var = 0
varNotZero = 0
while feature:
    magnitude = feature.GetField('magnitude') # get flow mag.
    var = var + ((magnitude - float(CalStatMean))**2)
    if magnitude > 0:
        varNotZero = varNotZero + ((magnitude - \
                     float(CalStatMeanNotZero))**2)
    feature.Destroy() # destroy feature
    feature = layer.GetNextFeature() # read next feature
# assign values to variables
CalStatStdDev = str((var / int(CalStatNoOfFeatures))**0.5)
CalStatStdDevNotZero = str((varNotZero / \
                       int(counterNotZero))**0.5)
layer.ResetReading() # reset reading shapefile

# close data source
shp.Destroy()
```

As given in Table 5.9, in order to calculate basic statistical indicators regarding flow features, first global variables are defined at the beginning of “CalStatFilter” function. As these indicators are assigned to global variables, once calculated they can be accessed and used in other functions or methods. Shapefile layer holding flow features is accessed via “ogr” driver (Lines 14 – 16). Number of features in the shapefile is returned by the “GetFeatureCount” method (Line 18). Then, first feature in the shapefile is accessed (Line 20) and value of desired attribute field (e.g. magnitude, Line 25) for that feature is appended as an element to the list object “CalStatList[ ]” (Line 27). This operation is performed iteratively until the last feature in the shapefile is accessed (Lines 24 – 33). By using list functions for “CalStatList[ ]” object, indicators such as minimum, maximum, mean are calculated (Lines 35 – 43). After calculating the mean value for dataset (Lines 35 – 38), a similar iterative operation is performed by using another “while” loop to calculate the standard deviation (Lines 49 – 60). After calculating all required statistics, data source is closed (Line 64).

In Chapter 2, during review of visual clutter reduction techniques, filtering was cited by Tobler (1987) and by Ellix and Dix (2007) as a method for avoiding visual clutter on flow maps. Tobler (1987) defines filtering operation as applying filters on flow tables based on descriptive statistics. Ellix and Dix (2007) define filtering as the selection of a subset from the whole dataset that satisfies the criteria set by the user. In FlowMapper, users can perform several filtering operations by accessing the following forms: (i) “Filter flow lines by magnitude”, (ii) “Filter flow lines by length” and (iii) “Filter flow lines by node and direction”.

Filtering operation is implemented in Python code by executing the “ogr2ogr” command line utility in the background with user defined parameters. As a part of GDAL/OGR library, this utility can be used for conversion of vector data to supported formats as well as attribute based subsetting of input data during conversion. For example, to extract features from the input shapefile which have magnitudes greater than the mean, following command should be executed.

```
ogr2ogr -f "ESRI Shapefile" -where "magnitude > MeanValue" OutputShapefileName.shp InputShapefileName.shp
```

Similarly, to extract incoming flows to a node selected by the user, following command should be executed. In the following command, “name_x2y2” corresponds to the attribute field that holds destination node names for flow lines.

```
ogr2ogr -f "ESRI Shapefile" -where "name_x2y2 = "UserSelectedNodeName"" OutputShapefileName.shp InputShapefileName.shp
```

FlowMapper offers several symbology options for better cartographic representation and perception of flow phenomena. For example, direction of each flow line is calculated automatically and can be depicted with an arrow head that is overlaid on the center of flow line. Besides, if the user prefers graduated symbology representation rather than single symbology mode, flow lines are drawn with widths proportional to their magnitudes and rendered in graduated colors which shade from light green to dark blue by default. Similarly, for flow nodes, diameter of each circle representing a point feature is drawn proportional to the cumulative magnitude of incoming, outgoing or gross flows on that node. In graduated symbology mode, the user can determine up to eight subclasses and manually adjust width and color for each subclass or can select one of the following algorithms in order to generate representation subclasses automatically; (i) equal size classes, (ii) equal interval classes, (iii) defined interval classes and (iv) standard deviation classes. For example, in Table 5.10, algorithm implemented for automatically creating equal interval subclasses is given.

Table 5.10. Python Code for Generating Graduated Symbology with Equal Interval Classes

Python code to automatically generate equal interval classes

```python
...
...
# determine feature geometry
def validatedDefaultSymbol(geometryType):
    symbol = QgsSymbolV2.defaultSymbol(geometryType)
    if symbol is None:
        if geometryType == QGis.Point:
            symbol = QgsMarkerSymbolV2()
        elif geometryType == QGis.Line:
            symbol =  QgsLineSymbolV2()
        elif geometryType == QGis.Polygon:
            symbol = QgsFillSymbolV2()
    return symbol
# create symbology
def makeSymbologyForRange(layer,min,max,label,colour,alpha,width):
    symbol = validatedDefaultSymbol(layer.geometryType())
    if dlg.ui.comboBoxSelectSymbology.currentText() <> "Single \
    Symbol" and dlg.ui.ShowDirectioncheckBox.isChecked() == True:
...
...
    elif dlg.ui.comboBoxSelectSymbology.currentText() <> "Single \
    Symbol" and dlg.ui.ShowDirectioncheckBox.isChecked() == False:
        symbol.setColor(colour)
        symbol.setAlpha(alpha)
        symbol.setWidth(width)
        range = QgsRendererRangeV2(min, max, symbol, label)
    return range
# set shapefile layer name, path and driver
vlayer = QgsVectorLayer(str(SaveShpName),str(SaveShpName),'ogr')
myTargetField = 'magnitude' # set target attribute field
# base line width and graduated symbology width multiplier factor
global basesymbolwidth, multipliersymbolwidth

if dlg.ui.comboBoxSelectSymbology.currentText() <> "Single Symbol"
and dlg.ui.ShowDirectioncheckBox.isChecked() == True:
...
...
elif dlg.ui.comboBoxSelectSymbology.currentText() <> "Single
Symbol" and dlg.ui.ShowDirectioncheckBox.isChecked() == False:
    global basesymbolwidth, multipliersymbolwidth
    basesymbolwidth = 0.25 # base line width
    multipliersymbolwidth = 0.75 # grad. sym. width multiplier

# graduated symbology mode: CREATE EQUAL INTERVAL CLASSES
if dlg.ui.comboBoxSelectSymbology.currentText()=="Equal Interval":
    # get number of classes set by user
    GradSymNoOfClasses = dlg.ui.spinBoxClasses.value()
    # calculate class interval by dividing range to no. of classes
    GradSymInterval = round(((int(GradSymMax) - int(GradSymMin)) \
    / float(GradSymNoOfClasses)),0)
    myRangeList = [] # create list to store renderer range

    # iterate and set symbology parameters for each grad. class
    for i in range(GradSymNoOfClasses):
        if i == 0:
            # set class label for the FIRST class
            classlabel=str(GradSymMin)+" - "+str(int(GradSymMin) \
            + GradSymInterval)
```

Table 5.10. Python Code for Generating Graduated Symbology with Equal Interval Classes (cont.)

```
#   Python code to automatically generate equal interval classes
59              # define symbology of first class
60              myRangeList.append(makeSymbologyForRange(vlayer, \
61              int(GradSymMin), int(GradSymMin) + GradSymInterval, \
62              classlabel, QColor(0,255-(255*i/GradSymNoOfClasses), \
63              255*i/GradSymNoOfClasses), 1, basesymbolwidth ))
64          elif i == (GradSymNoOfClasses - 1):
65              # set class label for OTHER classes except the last
66              classlabel=str(int(GradSymMin)+(GradSymInterval*i) \
67              +0.001) + " - "+str(GradSymMax)
68              # def.symbology of other classes except the last class
69              myRangeList.append(makeSymbologyForRange(vlayer, \
70              (GradSymInterval*i)+0.001, int(GradSymMax),
71              classlabel, QColor(0,255-(255*i/GradSymNoOfClasses), \
72              255*i/GradSymNoOfClasses), 1, (i+1)* \
73              multipliersymbolwidth))
74          else:
75              # set class label for LAST class
76              classlabel=str(int(GradSymMin)+(GradSymInterval*i) \
77              +0.001)+" - "+str(int(GradSymMin)+GradSymInterval \
78              *(i+1))
79              # define symbology of last class
80              myRangeList.append(makeSymbologyForRange(vlayer, \
81              int(GradSymMin)+(GradSymInterval*i)+0.001, \
82              int(GradSymMin)+GradSymInterval*(i+1),classlabel, \
83              QColor(0,255-(255*i/GradSymNoOfClasses), 255*i/ \
84              GradSymNoOfClasses), 1, (i+1)*multipliersymbolwidth ))
85
86      # replace renderer of current layer
87      myRenderer = QgsGraduatedSymbolRendererV2(myTargetField, \
88      myRangeList )
89      vlayer.setRendererV2( myRenderer )
90      # add layer to map and show success message
91      SuccessMessage = str(SaveShpName) + " created successfully !"
92      QMessageBox.information(self.iface.mainWindow(), "info", \
93      SuccessMessage, "Close")
94      QgsMapLayerRegistry.instance().addMapLayer( vlayer )
95  ...
96  ...
```

In Table 5.10, in order to create graduated symbology representation with equal interval classes, first a blank symbology is created according to the geometry of feature layer (Lines 15 - 27). Feature geometry is then validated (Line 16) by using the “validateDefaultSymbol” function (Lines 4 – 13). Afterwards the data source and the target attribute field to which graduated symbology will be applied are defined (Lines 29 - 30). Line width to be used for the first class is assigned to the global variable “basesymbolwidth” (Line 41) and similarly line width multiplier factor to be used for the rest of the classes is assigned to “multipliersymbolwidth” variable (Line 42). By accessing the number of graduated classes set by the user (Line 47), equal interval class value is calculated by dividing the range to the number of classes (Lines 49 - 50). Class label that will appear on legend and symbology parameters for each class (e.g. min., max., line color and width) are calculated iteratively (Lines 54 – 84). After setting all required parameters for each graduated class, symbology is rendered (Lines 87 – 88) and added into map (Line 94).

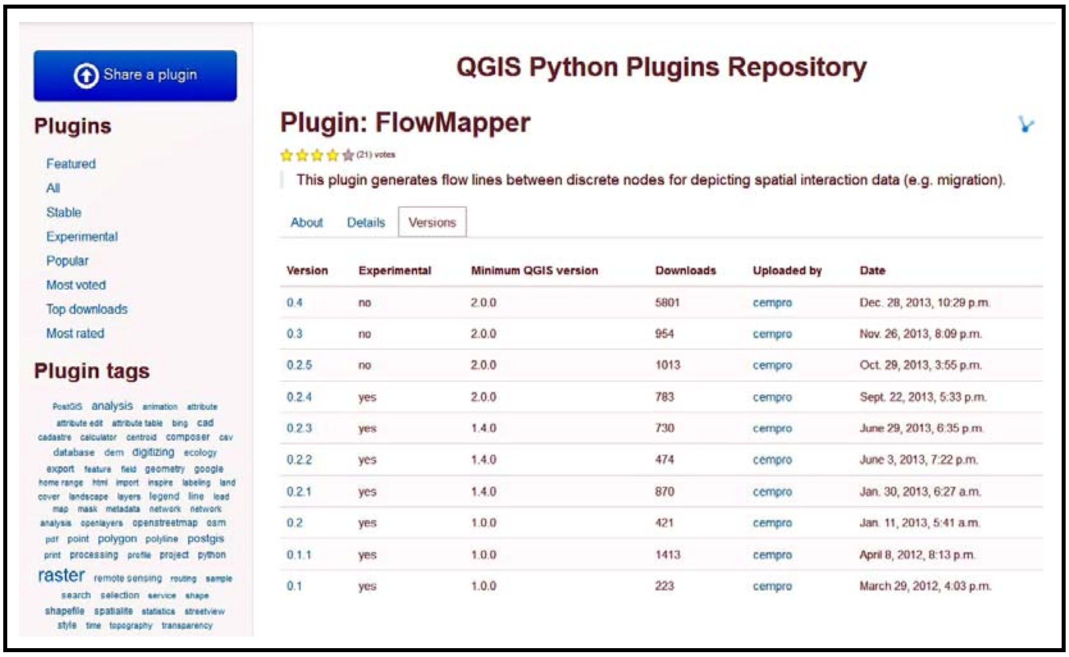
Up to this point, the architecture of FlowMapper plugin is reviewed in detail. First, folder structure of FlowMapper and the purpose of each file located under this plugin directory is explained with respect to the general structure of a Python plugin for QGIS. Then, details regarding GUI development are reviewed. Finally, capabilities and code structure of two major modules; (i) main module and (ii) flow generator module, are examined in detail. In addition, a conceptual diagram (Figure A.1) is given in Appendix A in order to portray the inner structure of the plugin and interaction of it with QGIS.

5.3. Releasing FlowMapper Plugin

The last stage of agile development methodology defined in the PyQGIS developer cookbook (QGIS, 2014a) involves releasing the plugin and publishing the source code. Coding of FlowMapper plugin in Python started at the beginning of December 2011. After about four months, at the end of March 2012, both the source code and the initial release v0.1 were uploaded to the QGIS official plugin repository.

When a plugin is uploaded to a repository, it is ready for the use of community. A plugin published in a repository takes the advantage of easy installation via “QGIS Plugin Manager” which otherwise requires manually downloading and deploying the contents of plugin under QGIS plugins directory. Unlike most of the core plugins written in C/C++, external plugins of QGIS are written in

Python and they are stored either in third party repositories maintained by external developers or in the QGIS official repository. Managed by the QGIS community, offical plugin repository of QGIS is being hosted on “http://plugins.qgis.org/plugins” and as of August 2014 more than 350 Python plugins are being hosted including FlowMapper (Figure 5.6).


Share a plugin
Plugins
Featured
All
Stable
Experimental
Popular
Most voted
Top downloads
Most rated
Plugin tags
PostGIS analysis animation attribute attribute edit attribute table bing cad cadastre calculator centroid composer csv database dem digitizing ecology export feature field geometry google home range html import inspire labeling land cover landscape layers legend line lead map mask metadata network network analysis openlayers openstreetmap osm pdf point polygon polyline postgis print processing profile project python raster remote sensing routing sample search selection service shape shapefile spatialite statistics streetview style time topography transparency
(21) votes
About
Details
Versions

QGIS Python Plugins Repository

Plugin: FlowMapper

This plugin generates flow lines between discrete nodes for depicting spatial interaction data (e.g. migration).

Version	Experimental	Minimum QGIS version	Downloads	Uploaded by	Date
0.4	no	2.0.0	5801	cempro	Dec. 28, 2013, 10:29 p.m.
0.3	no	2.0.0	954	cempro	Nov. 26, 2013, 8:09 p.m.
0.2.5	no	2.0.0	1013	cempro	Oct. 29, 2013, 3:55 p.m.
0.2.4	yes	2.0.0	783	cempro	Sept. 22, 2013, 5:33 p.m.
0.2.3	yes	1.4.0	730	cempro	June 29, 2013, 6:35 p.m.
0.2.2	yes	1.4.0	474	cempro	June 3, 2013, 7:22 p.m.
0.2.1	yes	1.4.0	870	cempro	Jan. 30, 2013, 6:27 a.m.
0.2	yes	1.0.0	421	cempro	Jan. 11, 2013, 5:41 a.m.
0.1.1	yes	1.0.0	1413	cempro	April 8, 2012, 8:13 p.m.
0.1	yes	1.0.0	223	cempro	March 29, 2012, 4:03 p.m.

Figure 5.6. Webpage of FlowMapper Python Plugin Hosted at the QGIS Official Repository

Building a third party plugin repository for QGIS is quite easy. It just needs some space on a web server to host the plugin files in “zip” packaged format and a definition file written in XML language. Repository built for FlowMapper is hosted on the following address: “http://95.9.195.180/plugins.xml”. Although it is not mandatory, if an “xsl” stylesheet is supplied for transformation of this “xml” file, repository can be displayed as a webpage in web browsers. Source code and screenshot of the plugin repository built for FlowMapper is given respectively in Table 5.11 and Figure 5.7.

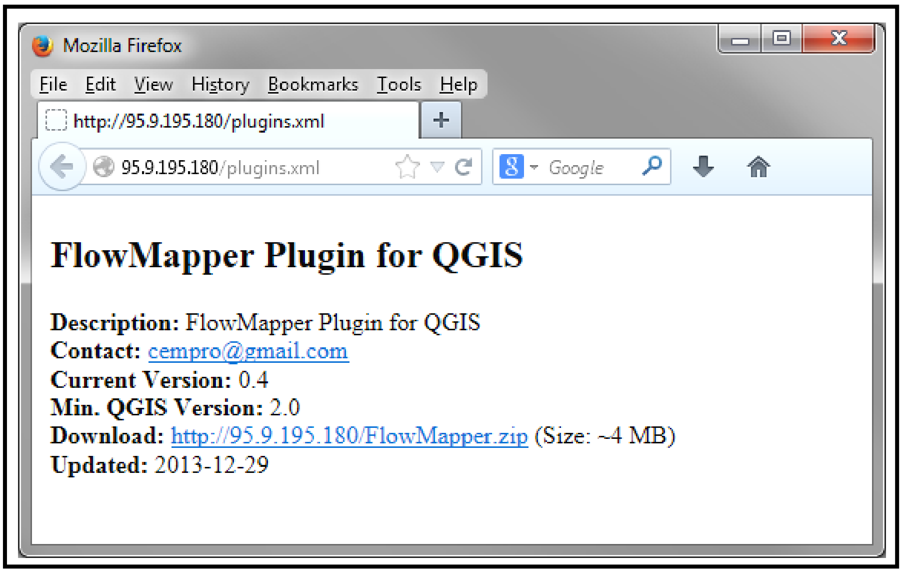
Table 5.11. Source Code of the Repository for FlowMapper: (a) xml file, (b) xsl stylesheet

(a) Source code of repository XML file

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<?xml-stylesheet type='text/xsl' href='/plugins.xsl'?>
<plugins>
  <pyqgis_plugin name='FlowMapper' version='0.4'>
    <description>FlowMapper Plugin for QGIS</description>
    <version>0.4</version>
    <qgis_minimum_version>2.0</qgis_minimum_version>
    <homepage>http://95.9.195.180</homepage>
    <file_name>FlowMapper.zip</file_name>
    <author_name>Cem GULLUOGLU</author_name>
    <download_url>http://95.9.195.180/FlowMapper.zip</download_url>
    <uploaded_by>Cem GULLUOGLU</uploaded_by>
    <create_date>2013-12-29</create_date>
    <update_date>2013-12-29</update_date>
  </pyqgis_plugin>
</plugins>
```

(b) Source code of XSL stylesheet

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match=" / ">
  <html> <body>
  <h2>FlowMapper Plugin for QGIS</h2> <xsl:apply-templates/>
  </body> </html>
</xsl:template>
<xsl:template match="description">
    <b> Description: </b> <xsl:value-of select="."/> <br></br>
    <b> Contact: </b>
    <a href="mailto:cempro@gmail.com" > cempro@gmail.com </a>
    <br></br>
</xsl:template>
<xsl:template match="qgis_minimum_version">
    <b> Min. QGIS Version: </b><xsl:value-of select="."/> <br></br>
</xsl:template>
<xsl:template match="download_url">
    <b> Download: </b>
    <a href="FlowMapper.zip"> <xsl:value-of select="."/> </a>
    (Size: ~4 MB) <br></br>
</xsl:template>
<xsl:template match="update_date">
    <b> Updated: </b> <xsl:value-of select="."/> <br></br>
</xsl:template>
<xsl:template match="version">
    <b> Current Version: </b> <xsl:value-of select="."/> <br></br>
</xsl:template>
<xsl:template match="create_date"> </xsl:template>
<xsl:template match="file_name"> </xsl:template>
<xsl:template match="author_name"> </xsl:template>
<xsl:template match="uploaded_by"> </xsl:template>
<xsl:template match="homepage"> </xsl:template>
</xsl:stylesheet>
```


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://95.9.195.180/plugins.xml
95.9.195.180/plugins.xml
Google

FlowMapper Plugin for QGIS

Description: FlowMapper Plugin for QGIS
Contact: cempro@gmail.com
Current Version: 0.4
Min. QGIS Version: 2.0
Download: http://95.9.195.180/FlowMapper.zip (Size: ~4 MB)
Updated: 2013-12-29

Figure 5.7. Plugin Repository Webpage

Source code of the “xml” file describing the repository is given in Table 5.11(a). Properties of FlowMapper plugin are defined within the “pyqgis_plugin” tag (Lines 4 – 15) which is contained by the “plugins” tag (Lines 3 – 16). It is also possible to store more than one plugin within a single repository by creating a new “pyqgis_plugin” tag for each plugin that is to be hosted. In Table 5.11(a), after declaring “xml” version (Line 1) and “xsl” stylesheet (Line 2), a short description regarding the plugin is given (Line 5). Version of the plugin (Line 6) and the minimum version of QGIS that the plugin could run are defined within “qgis_minimum_version” tag (Line 7). Homepage, plugin package name, author name and download URL for the plugin are defined between Lines 8 – 11. Besides, details regarding by whom and when the plugin is uploaded are defined between lines 12 – 14. In Table 5.11(b), source code of the “xsl” stylesheet is given. By using a stylesheet file, web browsers can interpret and show the repository like a webpage. For example; between Lines 4 – 8, heading of the page is set. Between Lines 5 – 28, page items are set by matching styles to desired descriptions in the “xml” file. For example; in Table 5.11(b) between Lines 26 – 28, plugin version data defined under the “version” tag that resides under the “xml” file is set to be shown on the webpage with a prefix text: “Current Version:”.

The first release of FlowMapper (v0.1) was uploaded to the official repository at the end of March 2012 after about four months of development. Excluding the automatically compiled “resources.py” file, FlowMapper v0.1 is comprised of about 400 lines of Python code which includes the main module containing almost one hundred lines of code and the flow generator module containing more than 150 lines of code. Following this initial release, several major and minor releases were uploaded to the official repository within two years period during development of FlowMapper. In Table 5.12, all major releases of FlowMapper are given side by side along with the menu structure and GUI of main module in order to portray how the plugin evolved gradually in time.

Table 5.12. Releases of FlowMapper Plugin

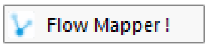
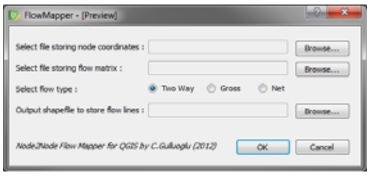
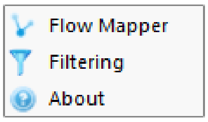
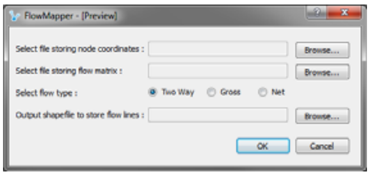
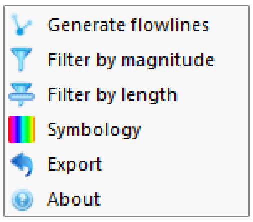
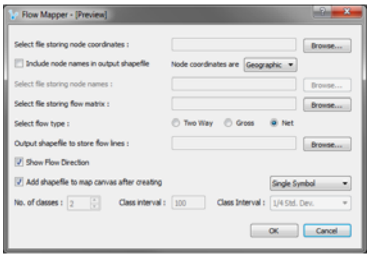
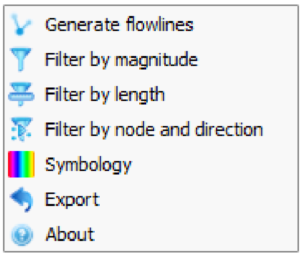
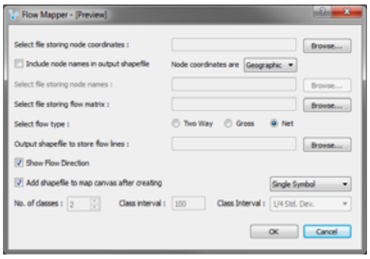
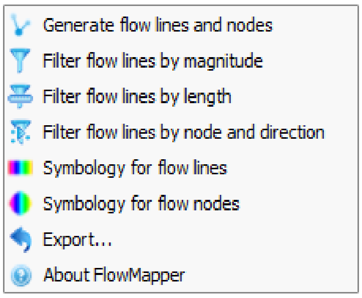
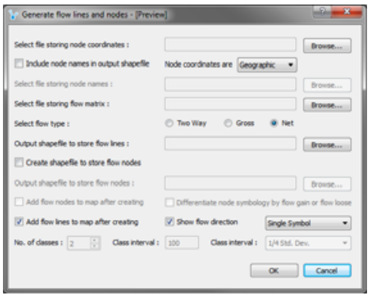
Major Release	**Release Date**	**Minimum QGIS Version**	**Plugin Menu**	**GUI of Main Module**	**Minor Revisions**
0.1	29 Mar. 2012	1.0	 Flow Mapper !	 FlowMapper - [Preview] Select file storing node coordinates : Browse... Select file storing flow matrix : Browse... Select flow type : ● Two Way ○ Gross ○ Net Output shapefile to store flow lines : Browse... *Node2Node Flow Mapper for QGIS by C.Gulluoglu (2012)* OK Cancel	N/A
0.2	11 Jan. 2013	1.0	 Flow Mapper Filtering About	 FlowMapper - [Preview] Select file storing node coordinates : Browse... Select file storing flow matrix : Browse... Select flow type : ● Two Way ○ Gross ○ Net Output shapefile to store flow lines : Browse... OK Cancel	0.1.1
0.2.5	29 Oct. 2013	2.0	 Generate flowlines Filter by magnitude Filter by length Symbology Export About	 Flow Mapper - [Preview] Select file storing node coordinates : Browse... □ Include node names in output shapefile Node coordinates are Geographic Select file storing node names : Browse... Select file storing flow matrix : Browse... Select flow type : ○ Two Way ○ Gross ● Net Output shapefile to store flow lines : Browse... ☑ Show Flow Direction ☑ Add shapefile to map canvas after creating Single Symbol No. of classes : 2 Class interval : 100 Class Interval : 1/4 Std. Dev. OK Cancel	*for 1.4 < QGIS <2.0:* 0.2.1 0.2.2 0.2.3 *for QGIS >2.0:* 0.2.4

Table 5.12. Releases of FlowMapper Plugin (cont.)

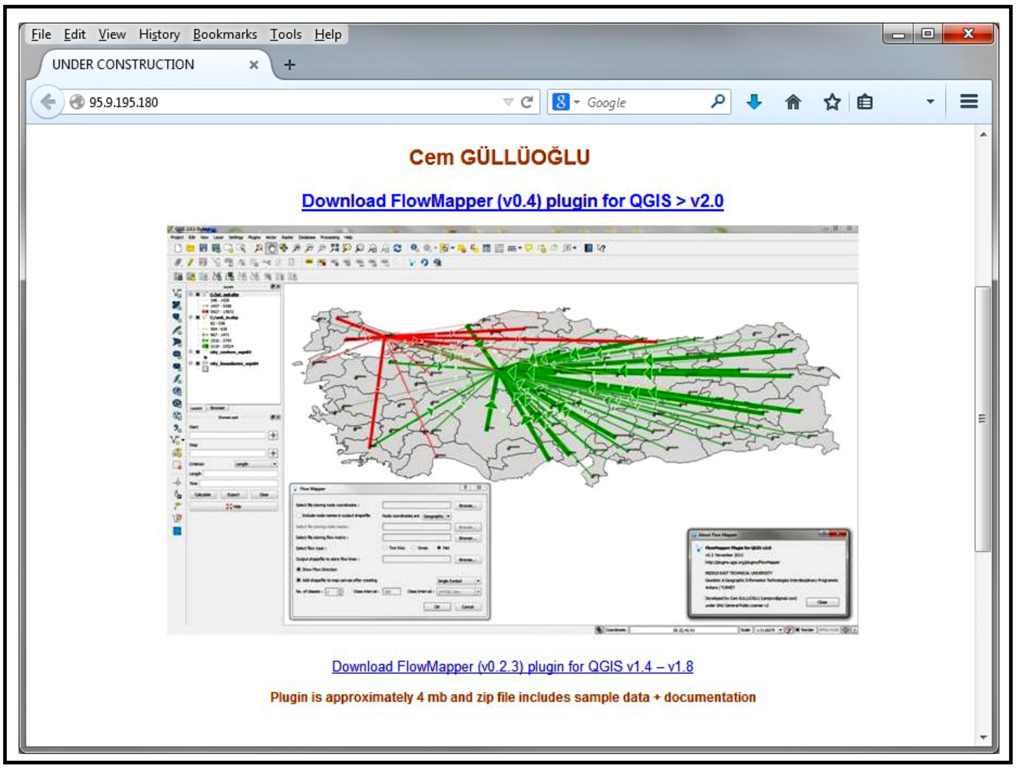
Major Release	Release Date	Minimum QGIS Version	Plugin Menu	GUI of Main Module	Minor Revisions
0.3	26 Nov. 2013	2.0	 Generate flowlines Filter by magnitude Filter by length Filter by node and direction Symbology Export About	 Flow Mapper - [Preview] Select file storing node coordinates : Browse... □ Include node names in output shapefile Node coordinates are Geographic Select file storing node names : Browse... Select file storing flow matrix : Browse... Select flow type : ○ Two Way ○ Gross ● Net Output shapefile to store flow lines : Browse... ☑ Show Flow Direction ☑ Add shapefile to map canvas after creating Single Symbol No. of classes : 2 Class interval : 100 Class Interval : 1/4 Std. Dev. OK Cancel	N/A
0.4	28 Dec. 2013	2.0	 Generate flow lines and nodes Filter flow lines by magnitude Filter flow lines by length Filter flow lines by node and direction Symbology for flow lines Symbology for flow nodes Export... About FlowMapper	 Generate flow lines and nodes - [Preview] Select file storing node coordinates : Browse... □ Include node names in output shapefile Node coordinates are Geographic Select file storing node names : Browse... Select file storing flow matrix : Browse... Select flow type : ○ Two Way ○ Gross ● Net Output shapefile to store flow lines : Browse... □ Create shapefile to store flow nodes Output shapefile to store flow nodes : Browse... □ Add flow nodes to map after creating □ Differentiate node symbology by flow gain or flow loose ☑ Add flow lines to map after creating ☑ Show flow direction Single Symbol No. of classes : 2 Class interval : 100 Class interval : 1/4 Std. Dev. OK Cancel	N/A

During development of FlowMapper, QGIS has also received updates. In December 2011, when development of FlowMapper started, the latest stable release of QGIS was v1.7.3. In September 2013, QGIS was updated to v2.0.1 from v1.8.0 which was released in June 2012. This was a major update and result some changes in the PyQt and PyQGIS API. While SIP v1 was used to create Python bindings for Qt and QGIS API in QGIS v1.x, SIP v2 is preferred in QGIS v2.x. Since API was subject to chance, QGIS v2.x is not fully backward compatible with the plugins which were initially written for QGIS v1.x. Thus, FlowMapper needed to be modified to be compatible with QGIS v2.x. In September 2013, just two weeks after the release of QGIS v2.0.1, FlowMapper v0.2.4 was uploaded to the repository as one of the first QGIS v2.x compatible plugins. At the end of the year, in 28 December 2013, FlowMapper v0.4 was released on the repository and received more than 5.000 downloads in 8 months. After about two years development period, latest release of FlowMapper (v0.4) is comprised of more than 6.500 lines of code (excluding resources.py) which is almost 15 times more than the amount contained by the first release (v0.1). Besides, the length of main module code reached up to 3.000 lines which was about one hundred in v0.1 and flow generator module contains more than 400 lines of code which was slightly more than 150 lines in v0.1. As of August 2014, considering all releases hosted on the QGIS official repository, FlowMapper has been downloaded more than 12.000 times. This corresponds more than 10 downloads per day and justifies the main intention of this study which is integration of flow mapping tools to a popular open source desktop GIS application. In Table 5.13, download counts retrieved from the official repository and daily download rates for each release is presented.

Table 5.13. Number of FlowMapper Downloads from the QGIS Official Repository

Version	v0.1	v0.1.1	v0.2	v0.2.1	v0.2.2	v0.2.3	v0.2.4	v0.2.5	v0.3	v0.4
Release Date	29.03 2012	08.04 2012	11.01 2013	30.01 2013	03.06 2013	29.06 2013	22.09 2013	29.10 2013	26.11 2013	28.12 2013
Down. by Rel.	220	1.406	421	865	471	724	769	1.011	952	5.452
Daily Down.	22	5	22	7	18	9	21	36	30	23
Total Down.	220	1.626	2.047	2.912	3.383	4.107	4.876	5.887	6.839	**12.291**
* Based on download counts listed in the official repository in 24 August 2014.										

A dedicated website is built for FlowMapper which is available from the URL “http://95.9.195.180”. During development of FlowMapper, latest release is published on the site for download. As of August 2014, site hosts two versions of plugin: (i) v0.2.3 which is the last release compatible with QGIS v1.x and (ii) v0.4 which is the latest release compatible with QGIS 2.x. Besides, brief documentation about the usage of plugin and several test datasets are included in the plugin package which can be directly downloaded from the site. In Figure 5.8, screenshot of the plugin website is given.



Cem GÜLLÜOĞLU

Download FlowMapper (v0.4) plugin for QGIS > v2.0

Download FlowMapper (v0.2.3) plugin for QGIS v1.4 – v1.8

Plugin is approximately 4 mb and zip file includes sample data + documentation

Figure 5.8. FlowMapper Plugin Website

It is possible to get download counts from the official plugin repository, however it is not possible to retrieve any further information regarding these downloads such as distribution of downloads by country or downloads by operating system. On the other hand, details about the users visiting the plugin website can be tracked by means of Google Analytics service. Basically, Google Analytics service monitors the site and generates statistics regarding traffic sources (Wikipedia, 2014b). Between February 2013 and August 2014, the site was visited approximately 1.200 times. Since nearly % 60 of the total traffic to the website is redirected traffic from the official plugin repository, website visitor statistics can also be taken as a sample to reveal some details regarding downloads from official repository. In Table 5.14, distribution of visits between February 2013 and August 2014 is given by countries.

Table 5.14. Number of Visits to the Plugin Website by Countries

Rank	Country	Visits	%
1	United States	155	12,8
2	France	134	11,0
3	Italy	84	6,9
4	Turkey	81	6,7
5	Germany	67	5,5
6	Brazil	60	4,9
7	United Kingdom	56	4,6
8	Spain	46	3,8
9	Mexico	45	3,7
10	Canada	39	3,2
11	Japan	34	2,8
12	Poland	29	2,4
13	Netherlands	21	1,7
14	Sweden	19	1,6
15	Portugal	17	1,4
16	Australia	16	1,3
17	Switzerland	16	1,3
18	Belgium	15	1,2
19	Indonesia	15	1,2
20	Austria	14	1,2
21	Reunion	14	1,2
22	Malaysia	13	1,1
23	South Korea	13	1,1
24	Armenia	12	1,0
25	Argentina	11	0,9
26	China	10	0,8
27	India	10	0,8
28	Norway	9	0,7
29	South Africa	9	0,7
30	Greece	8	0,7
31	Taiwan	8	0,7
32	New Zealand	7	0,6
33	Singapore	7	0,6
34	Thailand	7	0,6
35	Czech Rep.	6	0,5
36	Finland	6	0,5
37	Morocco	6	0,5
38	Chile	5	0,4
39	Denmark	5	0,4
40	Hungary	5	0,4
41	Puerto Rico	5	0,4
42	Israel	4	0,3
43	Peru	4	0,3
44	Romania	4	0,3
45	Russia	4	0,3
46	Bulgaria	3	0,2
47	Colombia	3	0,2
48	Croatia	3	0,2
49	Estonia	3	0,2
50	Philippines	3	0,2
51	Ukraine	3	0,2
52	Bolivia	2	0,2
53	Burkina Faso	2	0,2
54	Costa Rica	2	0,2
55	Hong Kong	2	0,2
56	Ireland	2	0,2
57	Kenya	2	0,2
58	Malawi	2	0,2
59	Singapore	2	0,2
60	Slovakia	2	0,2
61	U.A. Emirates	2	0,2
62	Albania	1	0,1
63	Argentina	1	0,1
64	Bangladesh	1	0,1
65	Cuba	1	0,1
66	Cyprus	1	0,1
67	Dominican R.	1	0,1
68	Ecuador	1	0,1
69	Egypt	1	0,1
70	Ethiopia	1	0,1
71	Fiji	1	0,1
72	Ghana	1	0,1
73	Guatemala	1	0,1
74	Guyana	1	0,1
75	Honduras	1	0,1
76	Iran	1	0,1
77	Latvia	1	0,1
78	Luxembourg	1	0,1
79	Nigeria	1	0,1
80	Pakistan	1	0,1
81	Palestine	1	0,1
82	Venezuela	1	0,1
83	Vietnam	1	0,1

Visitor statistics: 24 August 2014

Starting from the release v0.2.2, FlowMapper offers multi platform support. It can work with QGIS installations running on Windows, Linux and MacOSX operating systems. According to visitors data acquired by Google Analytics, excluding the % 2 of visits performed by using mobile devices; % 78 of visitors was using Windows while % 15 of them was using MacOSX and % 7 was using Linux. These findings provide evidence regarding the cross platform utilization of FlowMapper plugin.

CHAPTER 6

CAPABILITIES OF FLOWMAPPER PLUGIN AND RESULTS

In previous chapter, methodology and steps involved in development of FlowMapper plugin are explained. This is mainly performed by reviewing the structure of a QGIS Python plugin and by giving technical details about GUI development and by examining the source code of plugin modules. In this technical context, Chapter 5 is intended to inform Python plugin developers for QGIS.

In this chapter, the capabilities of FlowMapper plugin are explored. At the beginning, installation methods are explained. Then, plugin capabilities are reviewed by exploring the GUI. Afterwards, input data structure is explained by reviewing several test datasets. Finally, sample flow maps are created from the test datasets by using different capabilities of plugin. Besides, cartographic quality of these flow maps is also discussed. Unlike Chapter 5 which aims to inform plugin developers, Chapter 6 keeps focus particularly on the usage of plugin. In other words, Chapter 6 is organized like a “user manual” and “tutorial document”.

6.1. Installation of FlowMapper Plugin

FlowMapper is designed as a Python plugin to QGIS. Thus, QGIS must be already installed on the target system that the FlowMapper plugin will run. Similar to QGIS, FlowMapper offers multi platform support and can run on Windows, Linux, BSD, MacOSX systems. To benefit from additional capabilities of FlowMapper such as filtering, for only MacOSX users, installation of GDAL/OGR framework is needed since it is not automatically installed during QGIS installation.

Assuming that QGIS is already exists on the system; it is possible to install FlowMapper plugin in two ways. In the first method, plugin package should be downloaded from the web and files in the package should be manually placed by user under the OS specific path that contains QGIS Python plugins directory. In this directory, each plugin is stored under a separate folder with the name of the plugin. So, “FlowMapper” folder in the plugin package should be copied under plugins directory to complete the installation. Afterwards, plugin can be enabled or disabled from the plugin manager when needed. This manual installation process might be a bit confusing for novice users since it requires user’s knowledge about OS specific paths. In Table 6.1, addresses of download resources and OS specific locations for Python plugins are listed.

Table 6.1. URLs for Downloading FlowMapper and OS Specific Paths for QGIS Python Plugins

Web resources for downloading the plugin package		
FlowMapper v0.4 for QGIS > 2.0	http://plugins.qgis.org/plugins/FlowMapper/version/0.4/download/	
	http://95.9.195.180/FlowMapper-0.4.zip	
FlowMapper v0.2.3 for 1.4 < QGIS < 2.0	http://plugins.qgis.org/plugins/FlowMapper/version/0.2.3/download/	
	http://95.9.195.180/FlowMapper-0.2.3.zip	
OS	**Location of QGIS Python plugins directory to extract plugin package**	
Windows	C:\Program Files\QGIS Dufour\apps\qgis\python\plugins\	for all users in system
	C:\Users\*$USERNAME*\\.qgis\python\plugins\	specific for user (v1.x)
	C:\Users\*$USERNAME*\\.qgis2\python\plugins\	specific for user (v2.x)
Linux	./usr/share/qgis/python/plugins/	for all users in system
	home/*$USERNAME*/.qgis/python/plugins/	specific for user (v1.x)
	home/*$USERNAME*/.qgis2/python/plugins/	specific for user (v2.x)
MacOSX	./Applications/QGIS.app/Contents/Resources/python/plugins/	for all users in system
	./Users/*$USERNAME*/.qgis/python/plugins/	specific for user (v1.x)
	./Users/*$USERNAME*/.qgis2/python/plugins/	specific for user (v2.x)
(i) If the plugin is needed only for a specific user, variable *$USERNAME* should be replaced with the user name of that user. (ii) v1.x stands for QGIS 1.x releases up to 2.0. v2.x stands for QGIS releases greater than v2.0.		

In the second installation method, QGIS Plugin Manager is used. Brief information about this utility has already been given in Chapter 5 (Figure 5.2). In this method, download and extraction operations are performed silently by means of the plugin manager. Thus, this is the recommended method for novice users. By default, address of the official plugin repository is already defined in the plugin manager and the user can immediately start searching for any Python plugin that is hosted on the official repository. Besides, by using the plugin manager interface that was previously shown on Figure 5.2(c), users can also add addresses of third party repositories to install Python plugins released by external developers. In QGIS interface, users can access to the plugin manager by clicking on the "Manage and Install Plugins…" item that is listed under the "Plugins" menu. Then user finds FlowMapper from the list and when user clicks install button plugin gets installed. In this method, by default plugin gets installed under the user specific Python plugins directory. Thus, when QGIS starts, plugin is loaded only for the user who has installed it. Once installed, FlowMapper icon appears on the plugins toolbar. FlowMapper can be enabled, disabled or completely uninstalled by using the plugin manager interface which was previously given in Figure 5.2(b).

6.2. Exploring GUI and Capabilities of FlowMapper

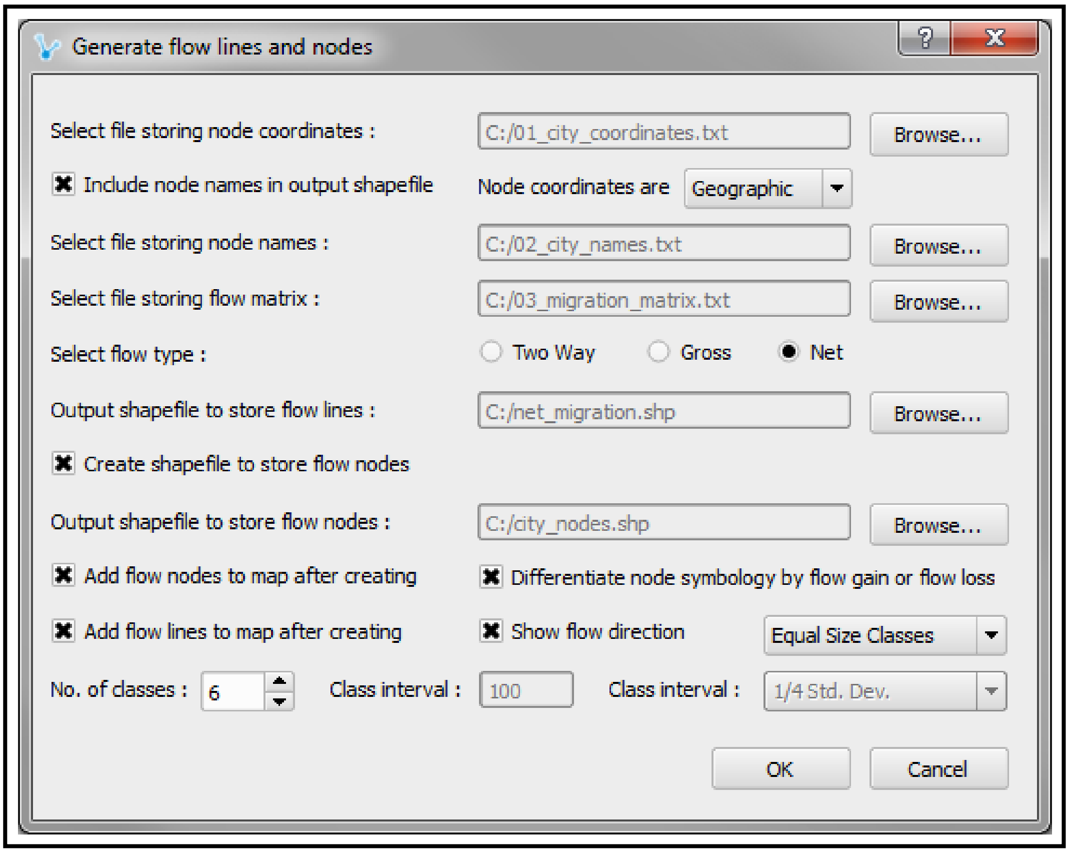
Under this heading, GUI and menu structure of FlowMapper will be reviewed. Besides, purpose of each form will be explained by examining the functions offered in each form. In Table 6.2, each item on FlowMapper menu is listed together with a short description explaining its purpose.

Table 6.2. FlowMapper Menu Items

Plugin Menu	Main Purpose
Generate flow lines and nodes	This item resides at the core of plugin. It is used to create flow lines based on a set of coordinates and a corresponding interaction matrix that holds flows between nodes. Besides, flow nodes can also be created as point features by using the same data.
Filter flow lines by magnitude	In this tool, by entering a threshold value and then by choosing the desired arithmetic operator user can perform filtering on flow lines with respect to their magnitude.
Filter flow lines by length	In this tool, by entering a threshold distance and then by choosing the desired arithmetic operator user can filter flow lines by their length.
Filter flow lines by node and direction	In this tool, user chooses a flow node to filter flow lines incoming to that node or outgoing from that node or both cases.
Symbology for flow lines	By using this form, user can define desired symbology and representation options for flow line features.
Symbology for flow nodes	By using this form, user can define desired symbology and representation options for point features, which represent discrete flow nodes.
Export...	User can export shapefiles storing flow features to Google Earth "kml" or MapInfo "tab" formats.
About FlowMapper	This form gives basic information about the plugin. It does not offer any functions.

6.2.1. Creating Flow Lines and Flow Nodes

The first item on the plugin menu is "Generate flow lines and nodes" tool (Figure 6.1). This tool resides at the core of FlowMapper and is used to create flow lines based on a given set of node coordinates and an interaction matrix that holds flows between these nodes. Besides, it is also possible to create flow nodes as point features by using this form. These operations are performed by means of the flow generator module "flowpyv07.py" which was reviewed in Chapter 5.



Generate flow lines and nodes

Select file storing node coordinates : C:/01_city_coordinates.txt

☒ Include node names in output shapefile

Node coordinates are Geographic

Select file storing node names : C:/02_city_names.txt

Select file storing flow matrix : C:/03_migration_matrix.txt

Select flow type : ○ Two Way ○ Gross ● Net

Output shapefile to store flow lines : C:/net_migration.shp

☒ Create shapefile to store flow nodes

Output shapefile to store flow nodes : C:/city_nodes.shp

☒ Add flow nodes to map after creating

☒ Differentiate node symbology by flow gain or flow loss

☒ Add flow lines to map after creating

☒ Show flow direction

Equal Size Classes

No. of classes : 6

Class interval : 100

Class interval : 1/4 Std. Dev.

Figure 6.1. GUI for Generate Flow Lines and Nodes Tool

Two input files must be supplied in order to create flow features. Both files are in plain text format and either tab or space delimited. The first input file contains coordinate pairs for every flow node (Table 6.3.a). Each flow node corresponds to a location that gives and receives flows. A flow node is analogous to a point feature and defined by easting and northing coordinates. For each flow node, one pair of either geographic or Cartesian coordinate must appear on each line of the input file. For example, if there are four different locations, coordinates of first location must appear in the first line and coordinates of the last location must appear on the fourth line. The second mandatory input file is the interaction matrix or namely flow matrix (Table 6.3.c). It contains interaction data between flow nodes in square matrix form which has equal dimensions in terms of the number of rows and columns in the matrix. The number of rows and columns in the interaction matrix must be equal to the number of flow nodes appearing in the coordinates file. Besides, the order of flow nodes appearing on the rows and columns of the flow matrix must be exactly same with the order that flow nodes appear in the coordinates file. While each matrix element appearing on the same row corresponds to the magnitude of an outgoing flow from the node represented by that row number, elements appearing on the same column correspond to the magnitude of incoming flows to the node represented by that column number. For example, if there are four locations, flow matrix must have 4 rows and 4 columns. Elements appearing on the first row are magnitudes of outgoing flows from the first node to others; similarly elements on the first column are incoming flows to the first node from others while diagonal elements should be assigned to zero since no internal flows occur. In addition to these two mandatory files, user can also provide an input file in plain text format which lists the name of each flow node in a row (Table 6.3.b). If this optional file is supplied, flow nodes will appear with their names in the attribute tables rather than their row number which is automatically calculated based on the order that they appear in the coordinates file. Typical structure of a sample flow dataset is given in Table 6.3. It comprises of four flow nodes as cities of Turkey, corresponding coordinates pairs, names and a 4 by 4 square interaction matrix. By using this dataset it is possible to organize mandatory and optional input files in plain text format which is accepted by FlowMapper.

Table 6.3. Sample Flow Data Format: (a) Coordinates of nodes, (b) Names of nodes, (c) Interaction matrix storing magnitude of flows between nodes

Raw Node Data:
(a) Node coordinates *, (b) Node names **

No.***	Easting	Northing	Names
1	28.90	41.10	Istanbul
2	30.70	36.90	Antalya
3	41.80	41.20	Artvin
4	40.70	37.30	Mardin

* Coordinate pairs must be supplied in a text file one pair in a row separated either by tab or white space to be used with FlowMapper.
** Names must be listed in a text file one under the other, one name appearing in each row to be used with FlowMapper.
*** No. column should not be included in input file.

Raw Flow Data:
(c) Interaction matrix ****

	To / From	Magnitude of flows received			
		Node 1	Node 2	Node 3	Node 4
Flows sent	1	0	270	10	5
	2	200	0	40	15
	3	120	130	0	20
	4	70	60	25	0

**** Magnitude of all flows must be supplied in a text file to be used with FlowMapper. Order of flows must be listed in the same order that they appear in the coordinates data. Header rows or columns should not appear in the input file. Each flow magnitude should be separated either by tab or white space.

After providing all required input files, user can choose desired flow type that is to be calculated. There are three options: (i) two way, (ii) gross and (iii) net. In “two way” mode, two flow lines are created in opposite directions between two discrete nodes. One depicts incoming flow and the other depicts outgoing flow (e.g. incoming and outgoing migration between two cities). Magnitude of each flow line is assigned from the corresponding flow matrix element. In “gross” mode, only one flow line is created to depict interaction between two discrete locations. Magnitude of a gross flow line is the sum of incoming and outgoing flows between these two locations. Thus, “gross” flows do not imply direction (e.g. total trade volume between two countries). In “net” mode, similar to “gross” mode, only one flow line is created to depict interaction between two locations. However, “net” flows imply direction and calculated by subtracting magnitudes of incoming and outgoing flows. While absolute difference is assigned as the “net” flow magnitude, positive or negative result determines the direction of flow line. In Figure 6.2, a basic illustration is provided for each flow type.

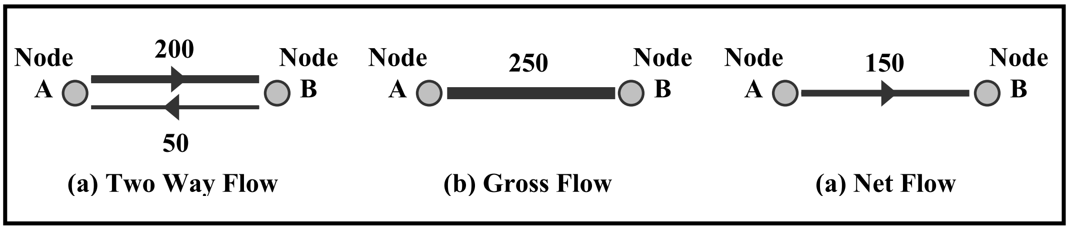
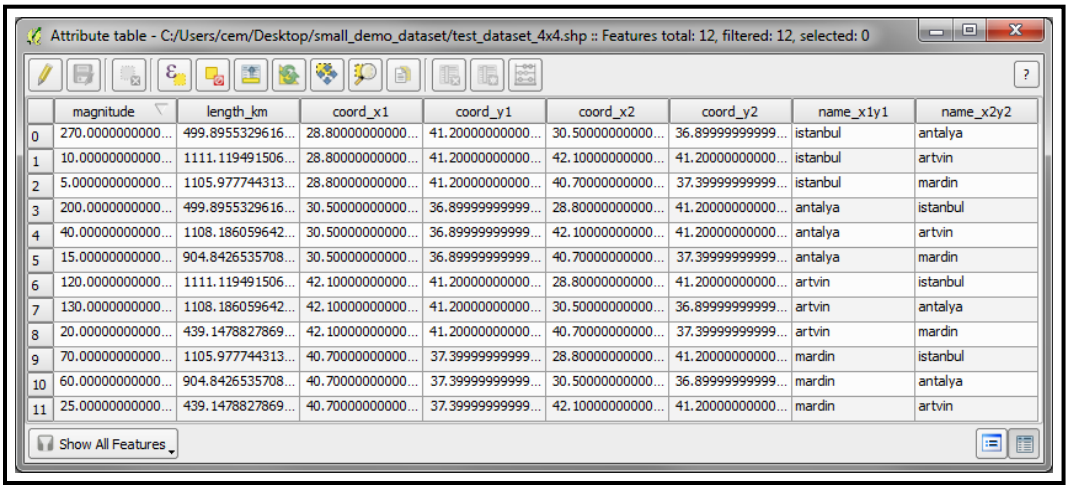


Figure 6.2. Types of Flow Calculations: (a) Two way, (b) Gross, (c) Net

After defining input files and determining type of flow calculation, user needs to type output file name to generate flow lines. Besides, if user wants to create flow nodes as point features, desired name for output file can be entered by clicking on the “Create shapefile to store flow nodes” checkbox. Upon clicking the “OK” button on the form, flow lines and nodes are created in shapefile format.

FlowMapper not only builds geometry for flow phenomena, but also automatically calculates and populates attribute field values for created flow features (Figure 6.3). For example, when flow lines are created, following feature attributes are automatically calculated: (i) “magnitude”: magnitude of flow line, (ii) “length_km”: length of flow line in kilometers to represent interaction distance, (iii) “coord_x1”: easting of origin node, (iv) “coord_y1”: northing of origin node, (v) “coord_x2”: easting of destination node, (vi) “coord_y2”: northing of destination node, (vii) “name_x1y1”: name of origin node, (viii) “name_x2y2”: name of destination node.

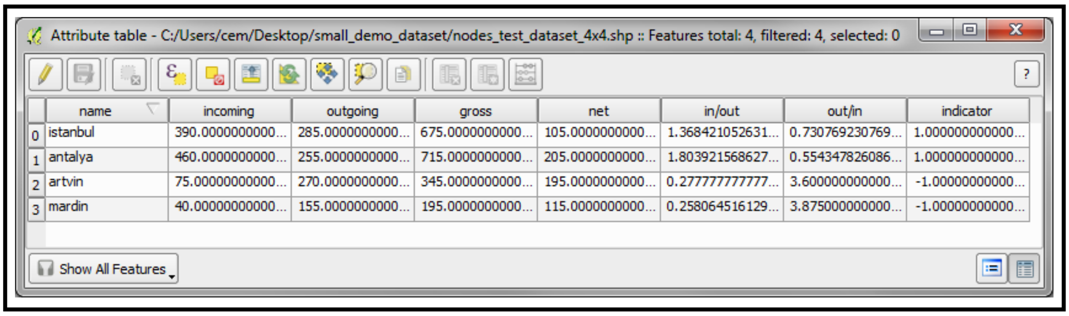


Attribute table - C:/Users/cem/Desktop/small_demo_dataset/test_dataset_4x4.shp :: Features total: 12, filtered: 12, selected: 0

	magnitude	length_km	coord_x1	coord_y1	coord_x2	coord_y2	name_x1y1	name_x2y2
0	270.0000000000...	499.8955329616...	28.80000000000...	41.20000000000...	30.50000000000...	36.89999999999...	istanbul	antalya
1	10.00000000000...	1111.119491506...	28.80000000000...	41.20000000000...	42.10000000000...	41.20000000000...	istanbul	artvin
2	5.000000000000...	1105.977744313...	28.80000000000...	41.20000000000...	40.70000000000...	37.39999999999...	istanbul	mardin
3	200.0000000000...	499.8955329616...	30.50000000000...	36.89999999999...	28.80000000000...	41.20000000000...	antalya	istanbul
4	40.00000000000...	1108.186059642...	30.50000000000...	36.89999999999...	42.10000000000...	41.20000000000...	antalya	artvin
5	15.00000000000...	904.8426535708...	30.50000000000...	36.89999999999...	40.70000000000...	37.39999999999...	antalya	mardin
6	120.0000000000...	1111.119491506...	42.10000000000...	41.20000000000...	28.80000000000...	41.20000000000...	artvin	istanbul
7	130.0000000000...	1108.186059642...	42.10000000000...	41.20000000000...	30.50000000000...	36.89999999999...	artvin	antalya
8	20.00000000000...	439.1478827869...	42.10000000000...	41.20000000000...	40.70000000000...	37.39999999999...	artvin	mardin
9	70.00000000000...	1105.977744313...	40.70000000000...	37.39999999999...	28.80000000000...	41.20000000000...	mardin	istanbul
10	60.00000000000...	904.8426535708...	40.70000000000...	37.39999999999...	30.50000000000...	36.89999999999...	mardin	antalya
11	25.00000000000...	439.1478827869...	40.70000000000...	37.39999999999...	42.10000000000...	41.20000000000...	mardin	artvin

Figure 6.3. Attribute Table for Flow Lines

Similarly, based on the interaction matrix given in Table 6.3(c), following attributes are automatically calculated for flow nodes (Figure 6.4). Respectively these attributes are (i) “name”: name of the flow node, (ii) “incoming”: magnitude of all incoming flows to a node which is calculated by summing all values listed under the corresponding column in flow matrix, (iii) “outgoing”: magnitude of cumulative outgoing flows from a node which is calculated by summing all values listed on the corresponding row in flow matrix, (iv) “gross”: magnitude of all gross flows calculated by summing all flows incoming to a node and outgoing from a node, (v) “net”: magnitude of all net flows calculated by taking the absolute difference between all flows incoming to a node and outgoing from a node, (vi) “in/out”: ratio of cumulative incoming flows to outgoing flows calculated by dividing “incoming” field value to “outgoing” field value, (vii) “out/in”: ratio of cumulative outgoing flows to incoming flows calculated by dividing “outgoing” field value to “incoming” field value, (viii) “indicator”: an indicator field calculated by subtracting “incoming” field value from “outgoing” field value. If indicator is “-1”, it means that node is losing flows at total (e.g. assuming that interaction matrix given in Table 6.3.c corresponds to an internal migration dataset, “-1” denotes cities losing population due to outgoing migration). If indicator is “1”, this means that magnitude of incoming flows to that node is greater than the outgoing flows from that node (e.g. based on the same assumption, it denotes cities gaining population due to incoming migration).



Attribute table - C:/Users/cem/Desktop/small_demo_dataset/nodes_test_dataset_4x4.shp :: Features total: 4, filtered: 4, selected: 0

	name	incoming	outgoing	gross	net	in/out	out/in	indicator
0	istanbul	390.0000000000...	285.0000000000...	675.0000000000...	105.0000000000...	1.368421052631...	0.730769230769...	1.000000000000...
1	antalya	460.0000000000...	255.0000000000...	715.0000000000...	205.0000000000...	1.803921568627...	0.554347826086...	1.000000000000...
2	artvin	75.00000000000...	270.0000000000...	345.0000000000...	195.0000000000...	0.277777777777...	3.600000000000...	-1.00000000000...
3	mardin	40.00000000000...	155.0000000000...	195.0000000000...	115.0000000000...	0.258064516129...	3.875000000000...	-1.00000000000...

Figure 6.4. Attribute Table for Flow Nodes

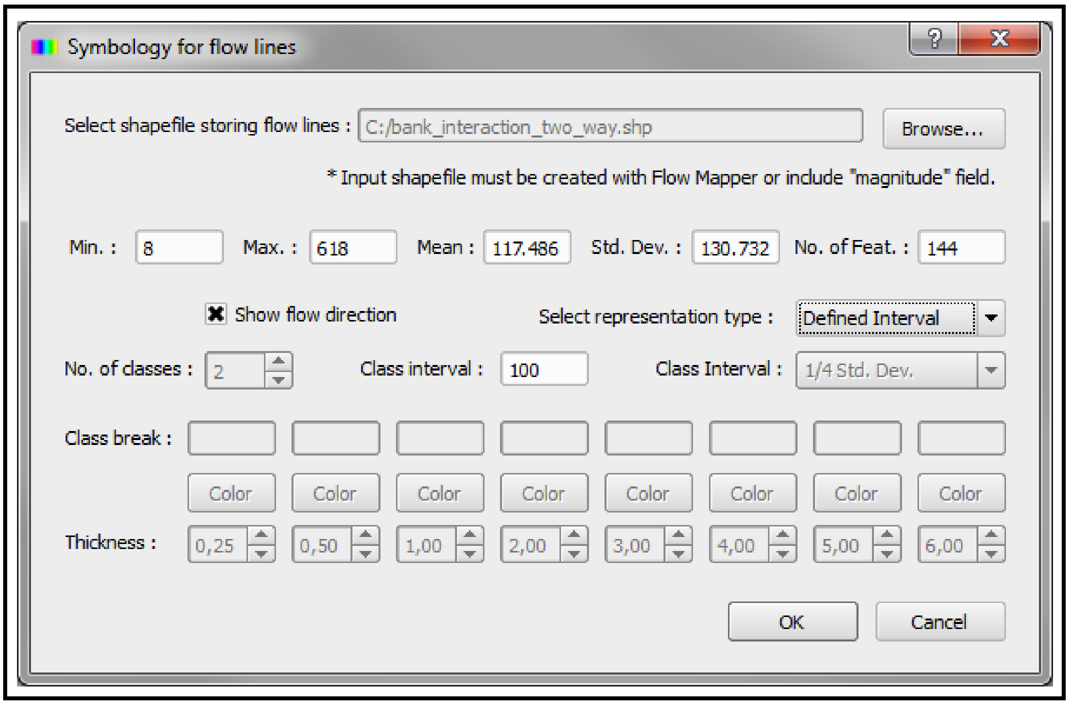
6.2.2. Symbology Capabilities

In their studies, both Tobler (1987) and Ellis and Dix (2007) mentioned visual clutter problem arising in flow maps due to data intensive interaction matrices. They discussed several methods, including utilization of different representation techniques, to overcome this problem. On this basis, FlowMapper offers tools to represent flow features with desired symbology. In Table 6.4, a list of all symbology options included in FlowMapper is given.

Table 6.4. Symbology Options Offered in FlowMapper

Symbology Option		User Defined Independent Variable	System Calculated Dependent Variable	Default Rendering Size	Default Rendering Color
Single Symbology		N/A	N/A	Fixed, all same size	Fixed, same color
Graduated Symbology	Equal Size	Number of classes	Class Intervals	Line thickness or point size gradually increases proportional to the target attribute field value.	Flow lines gradually shade from light green to dark blue proportional to flow magnitude. Flow nodes are rendered in grey.
	Equal Interval	Number of classes	Class Intervals		
	Defined Interval	Class interval	Number of classes		
	Standard Deviation	Class interval as a factor of Std. Dev.	Number of classes		
	Manual Interval	Number of classes Class interval Color of each class Line thickness or Point size	N/A	As defined by user	As defined by user
Show Flow Directions		N/A	N/A	Arrow size is proportional to the line thickness	Arrow color is identical to the line color
Differentiate symbology by flow gain and loss		N/A	N/A	Fixed, all same size	Green for flow gaining nodes Red for flow losing nodes Grey for neutral nodes

Excluding following two forms, “Export…” and “About FlowMapper”, on all forms a symbology tab exists along the bottom of the form window (Figure 6.1). Besides, there are two separate tools one of which offers options for adjusting symbology of flow lines (Figure 6.5) while the other provides similar options for representation of flow nodes (Figure 6.6). Three types of representations are supported with these two tools: (i) single symbology mode, (ii) graduated symbology mode with automatically adjusted classes or intervals, (iii) graduated symbology mode with fully manual, user adjusted parameters. While the first two modes are available in all forms, manual mode only exists in “Symbology for flow lines” and “Symbology for flow nodes” tools.



Symbology for flow lines

Select shapefile storing flow lines : C:/bank_interaction_two_way.shp

* Input shapefile must be created with Flow Mapper or include "magnitude" field.

Min. : 8 Max. : 618 Mean : 117.486 Std. Dev. : 130.732 No. of Feat. : 144

☒ Show flow direction

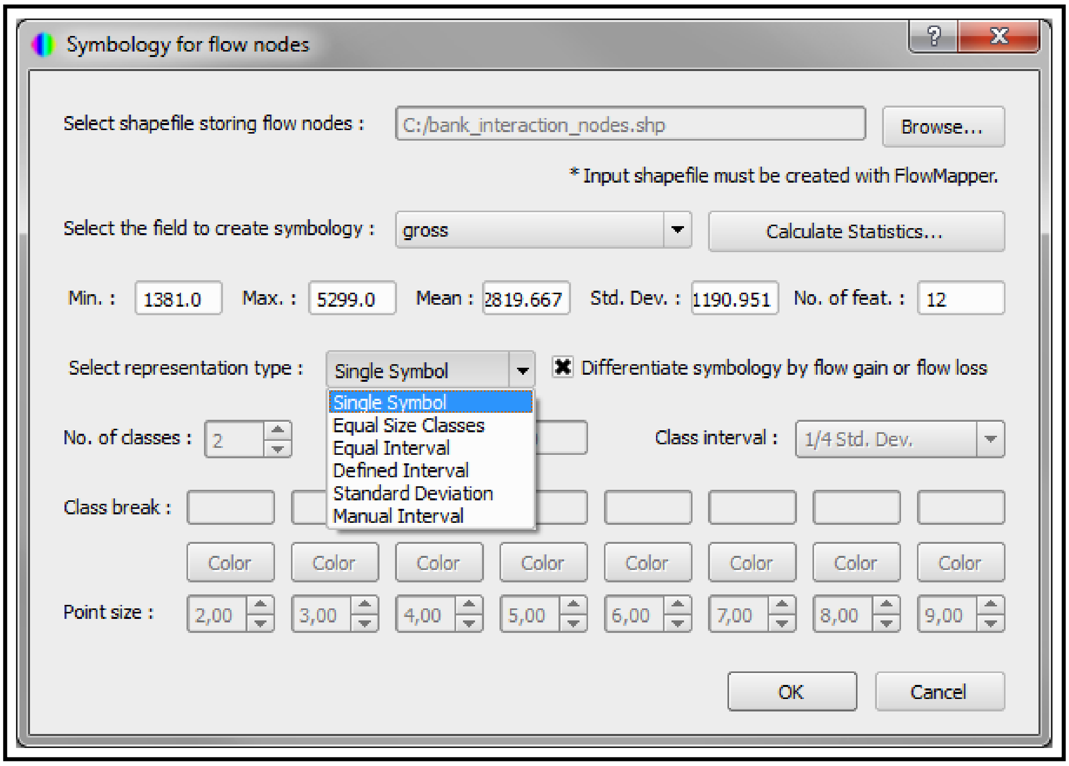
Select representation type : Defined Interval

No. of classes : 2 Class interval : 100 Class Interval : 1/4 Std. Dev.

Class break :

Thickness : 0,25 0,50 1,00 2,00 3,00 4,00 5,00 6,00

Figure 6.5. Symbology Adjustment Tool for Flow Lines



Symbology for flow nodes

Select shapefile storing flow nodes : C:/bank_interaction_nodes.shp

* Input shapefile must be created with FlowMapper.

Select the field to create symbology : gross

Min. : 1381.0 Max. : 5299.0 Mean : 2819.667 Std. Dev. : 1190.951 No. of feat. : 12

Select representation type : Single Symbol

☒ Differentiate symbology by flow gain or flow loss

No. of classes : 2 Class interval : 1/4 Std. Dev.

Class break :

Point size : 2,00 3,00 4,00 5,00 6,00 7,00 8,00 9,00

Figure 6.6. Symbology Adjustment Tool for Flow Nodes

In order to use “Symbology for flow lines” tool, user must first browse and select the shapefile storing flow lines. Upon selecting input shapefile, descriptive statistics for the magnitude field are automatically calculated and displayed on the form. These basic indicators provide prior knowledge about the dataset before determining the parameters for selected symbology type (e.g. class interval for graduated symbology representation). In single symbology mode, all flow features are depicted with plain lines in one color. In graduated symbology mode, thickness of a flow line is drawn proportional to its magnitude while by default the color schema gradually shades from light green for minor flows to dark blue for major flows. For this type of representation, there are four available algorithms which automatically generate graduated classes and the rendering parameters for each class such as line color and thickness. The first graduated symbology algorithm is “Equal Size Classes”. In this mode, user sets the number of graduated classes and algorithm automatically adjusts the interval of each class to keep equal number of elements in each class. In “Equal Interval” mode, algorithm gets the whole data range and divides it to the number of graduated classes defined by user in order to generate classes having equal intervals. To use “Defined Interval” mode, user must first enter desired value for class interval. Then, algorithm divides the whole data range to the user defined class interval value to calculate required number of graduated classes. Afterwards, graduated classes are generated with user defined intervals. Algorithm of “Standard Deviation” mode is similar to the algorithm used in “Defined Interval” mode. However, in this mode, class interval is not entered by user; instead multiples or divisions of standard deviation value are used as class interval values. This value can be chosen by user as one or two standard deviation or as half or quarter of the standard deviation. The last mode, “Manual Interval”, is only available on symbology forms and allows user control on all parameters. In this mode, user can generate up to eight graduated classes and can define class intervals for each class. Besides, user can also control rendering parameters of each class such as color and thickness.

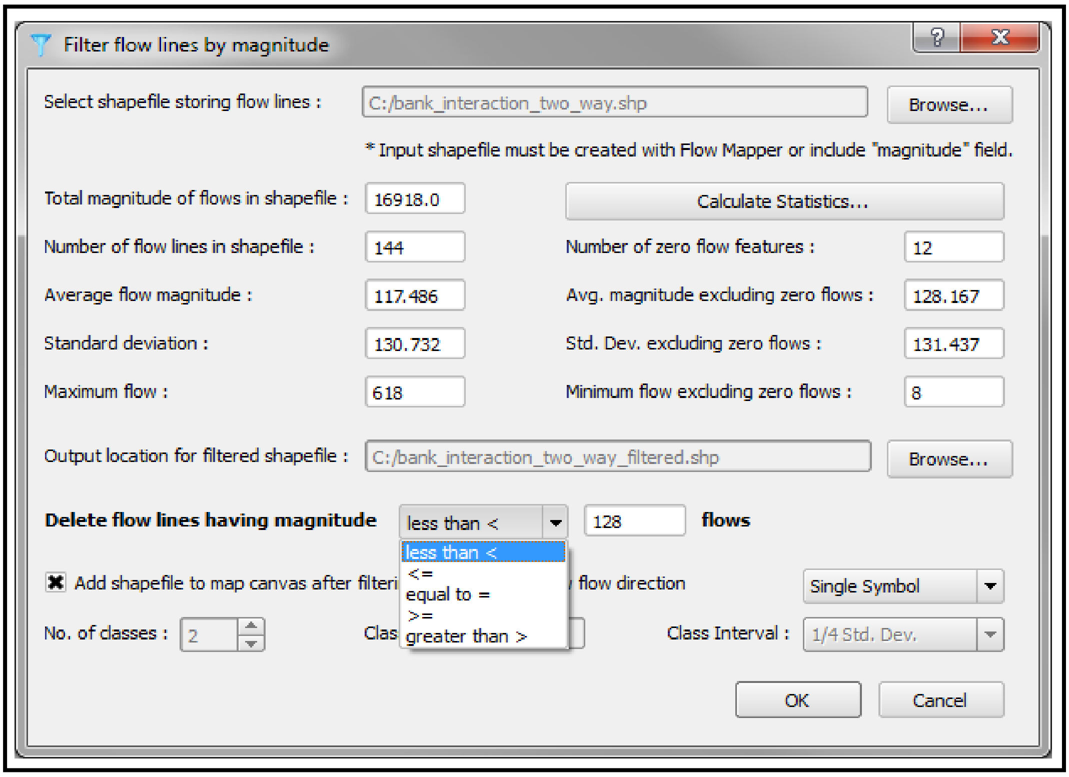
Capabilities of “Symbology for flow nodes” tool are quite similar to “Symbology for flow nodes” tool. Yet, in the “Symbology for flow nodes” tool user can choose target attribute field (e.g. incoming, outgoing, net, gross etc.) upon loading the shapefile that stores flow nodes. Then, by choosing one of the graduated symbology algorithms, size of point markers can be rendered proportional to the target field selected by user. While point markers with bigger size represent major interactions taking place on a node, smaller point markers denote minor event interactions. By default, all flow nodes are rendered in grey.

In addition to these representation options, there are two specific representations one of which is used for displaying flow directions and the other one is used for creating specific symbology in order to differentiate flow gaining nodes from flow losing nodes. In order to display flow directions on map, user should click on the “Show flow direction” checkbox in “Symbology for flow lines” window. Then, direction of each flow line is shown by overlaying an arrow head on the center of each line. In single symbology mode, all arrow heads are drawn the same size. In graduated symbology mode, the size of the arrow head is rendered proportional to the magnitude of flow. If user wants to distinguish flow gaining locations from flow losing locations, “Differentiate symbology by flow gain and loss” checkbox in “Symbology for flow lines” window should be clicked. Then, “flow gaining nodes” will be rendered with green makers while “flow losing nodes” will be rendered in red. The term “flow gaining node” is used to describe a node which receives more incoming flows than outgoing. On the contrary, if a node gives more outgoing flows than incoming, it is a “flow losing node”. In this mode, all point markers are drawn the same size and overlaid on top of flow nodes so that user can easily distinguish locations gaining or losing flows.

6.2.3. Filtering Capabilities

Performing filtering on dense flow data was cited by Tobler (1987) and by Ellix and Dix (2007) as a simple and straightforward method for avoiding or reducing visual clutter problem in flow maps. Filtering can be defined as extraction of a subset from the whole dataset based on user defined criteria. FlowMapper offers three tools to perform different filtering operations on flow lines. These tools are (i) “Filter flow lines by magnitude”, (ii) “Filter flow lines by length” and (iii) “Filter flow lines by node and direction”.

Interfaces of “Filter flow lines by magnitude” (Figure 6.7) and “Filter flow lines by length” (Figure 6.8) tools are quite similar. In both forms, user first clicks “Browse…” button and selects the shapefile storing flow lines. Then, by clicking on “Calculate Statistics…” button, it is possible to calculate some basic statistics about the dataset. While these statistics are calculated for the “magnitude” field in the “Filter flow lines by magnitude” form, in the “Filter flow lines by length” form they are calculated for the “length_km” column. By reviewing these statistics, user gets some prior knowledge about the dataset before performing filtering operation. For example, if user wants to keep flow lines having magnitudes greater than the mean value, by referring to the textbox where average magnitude is displayed, user can learn the value. To complete filtering operation, user selects the desired operator and enters the threshold value to delete matching records (e.g. delete flow lines having magnitudes less than < average value or delete flow lines longer than > average flow length). Since user may need to keep original input data, filtered flow features are written into a new shapefile with a file name to be provided by user.



Filter flow lines by magnitude

Select shapefile storing flow lines : C:/bank_interaction_two_way.shp

* Input shapefile must be created with Flow Mapper or include "magnitude" field.

Total magnitude of flows in shapefile : 16918.0

Number of flow lines in shapefile : 144

Number of zero flow features : 12

Average flow magnitude : 117.486

Avg. magnitude excluding zero flows : 128.167

Standard deviation : 130.732

Std. Dev. excluding zero flows : 131.437

Maximum flow : 618

Minimum flow excluding zero flows : 8

Output location for filtered shapefile : C:/bank_interaction_two_way_filtered.shp

Delete flow lines having magnitude less than < 128 **flows**

☒ Add shapefile to map canvas after filteri

flow direction

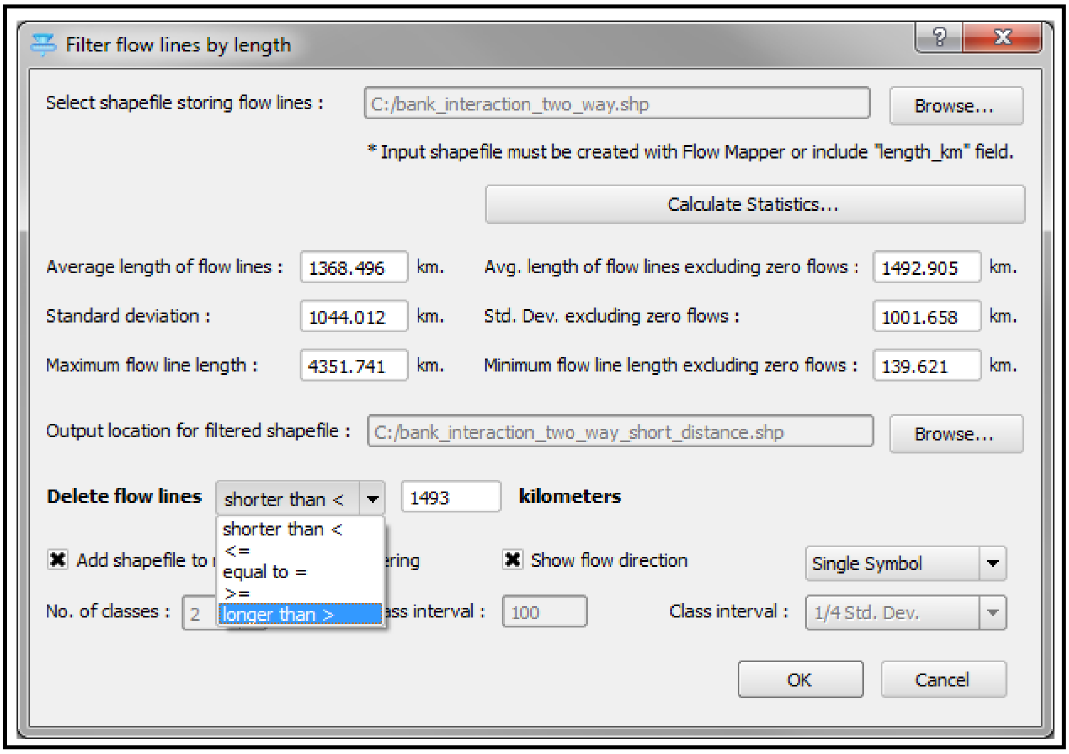
Single Symbol

No. of classes : 2

Clas

Class Interval : 1/4 Std. Dev.

Figure 6.7. User Interface of Filter Flow Lines by Magnitude Tool



Filter flow lines by length

Select shapefile storing flow lines : C:/bank_interaction_two_way.shp

* Input shapefile must be created with Flow Mapper or include "length_km" field.

Average length of flow lines : 1368.496 km.

Avg. length of flow lines excluding zero flows : 1492.905 km.

Standard deviation : 1044.012 km.

Std. Dev. excluding zero flows : 1001.658 km.

Maximum flow line length : 4351.741 km.

Minimum flow line length excluding zero flows : 139.621 km.

Output location for filtered shapefile : C:/bank_interaction_two_way_short_distance.shp

Delete flow lines shorter than < 1493 **kilometers**

☒ Add shapefile to r ring

☒ Show flow direction

Single Symbol

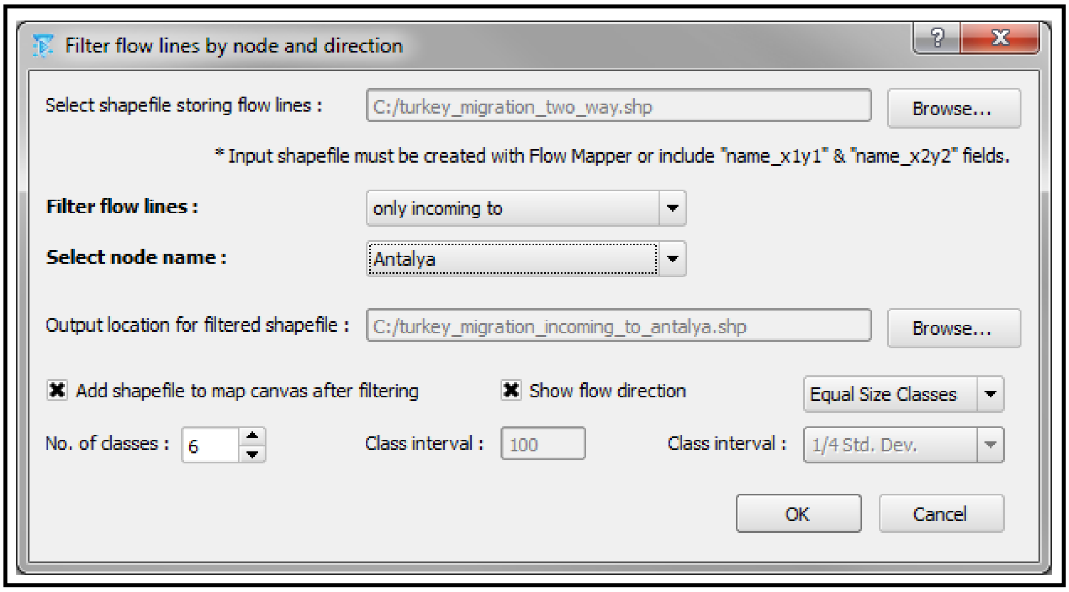
No. of classes : 2

ass interval : 100

Class interval : 1/4 Std. Dev.

Figure 6.8. User Interface of Filter Flow Lines by Length Tool

Interface of “Filter flow lines by node and direction” tool differs from the first two filtering tools (Figure 6.9). This tool is used to extract incoming, outgoing or both incoming and outgoing flows for a selected node. To perform this operation, user browses and selects the input shapefile storing flow lines. When the input file is selected, “Select node name” combobox is automatically populated by parsing “name_x1y1” and “name_x2y2” columns in attribute table. If the selection on “Filter flow lines” combobox is set to “only incoming to”, then “Select node name” combobox is populated by parsing “name_x2y2” column since this column stores destinations of flow lines. If combobox is set to “only outgoing from”, then node names are populated by “name_x1y1” column since it stores origins of flow lines. If value is set to “both incoming to & outgoing from”, then both columns are used. After making two selections from the comboboxes, user types an output file name for the shapefile to store filtered flow lines and clicks “OK” button on the form to complete filtering operation. Once filtered flow files are added to map, they look like “star” diagrams which imply many radial lines originating from a point. If symbology tool in this form is used in graduated symbology mode, by default all incoming flows to a node are rendered in shades of green while all outgoing flows are rendered in shades of red.



Filter flow lines by node and direction

Select shapefile storing flow lines : C:/turkey_migration_two_way.shp

* Input shapefile must be created with Flow Mapper or include "name_x1y1" & "name_x2y2" fields.

Filter flow lines : only incoming to

Select node name : Antalya

Output location for filtered shapefile : C:/turkey_migration_incoming_to_antalya.shp

☒ Add shapefile to map canvas after filtering

☒ Show flow direction

Equal Size Classes

No. of classes : 6

Class interval : 100

Class interval : 1/4 Std. Dev.

Figure 6.9. User Interface of Filter Flow Lines by Node and Direction Tool

6.3. Input Data Structure and Describing Test Datasets

Every FlowMapper release includes a set of sample data. Bundling test data with the plugin package provides users immediate chance of trying FlowMapper. A typical test dataset is provided with the following files: (i) node coordinates, (ii) node names, (iii) interaction matrix. While the files storing node coordinates and interaction matrix are mandatory, file storing node names is optional.

When preparing dataset to be used with FlowMapper, all input files must be created in plain text format and values must be separated either by using whitespace or tab. All input files must be prepared by using ANSI encoding and special characters should not be used when typing node names. Besides, if a node name consists of more than one word, underscore "_" character should be used instead of whitespace (e.g. This_is_a_node_name). For numeric entries, such as flow magnitudes and node coordinates, point "." should be used as the decimal separator (e.g. Easting: 33.123456).

Node coordinates file consists of coordinate pairs listed as rows. Separated either by whitespace or tab, easting and northing values of each flow node must appear on a separate row. Hence, the number of rows in this file is equal to the number of flow nodes in the case dataset. In Table 6.5, content of a "node coordinates file" is given. This first file belongs to a test dataset which is distributed with "Flowpy" script developed by Glennon (2009). It includes 12 flow nodes corresponding to locations of several banks in the United States and a flow matrix storing amount of banking interactions such as money transfers.

Table 6.5. Structure of Input File Storing Node Coordinates

Coordinates of 12 Banks in the U.S.	
Node No.	**File Content (Easting & Northing)**
1	`-71 42`
2	`-74 41`
3	`-75 40`
4	`-81 41`
5	`-77 37`
6	`-84 33`
7	`-88 42`
8	`-90 39`
9	`-93 45`
10	`-94.5 39`
11	`-96.75 32.75`
12	`-122.5 37.75`

Node names file holds the names of locations or any other explanatory text to label flow nodes. Similar to the structure used in node coordinates file, in node names file name of each node must appear on a new row. So, row lengths of coordinates file and names file are equal. This file must be prepared in ANSI format and language specific characters (e.g. İ, ı, Ğ, ş) should not be used when typing node names. Besides, underscore character should be preferred to whitespace when writing node names consisting of two or more words. In Table 6.6, content of a "node names file" is given.

Table 6.6. Structure of Input File Storing Node Names

Names of 12 Bank Locations in the U.S.	
Node No.	**File Content (Names of flow nodes)**
1	`Boston`
2	`NewYork`
3	`Philadelphia`
4	`Cleveland`
5	`Richmond`
6	`Atlanta`
7	`Chicago`
8	`StLouis`
9	`Minneapolis`
10	`Kansas`
11	`Dallas`
12	`SanFrancisco`

Together with the file storing node coordinates, interaction matrix file is the other mandatory input that is needed by FlowMapper to generate flow lines. Created in plain text format and arranged as a square matrix, it holds the magnitudes of flows taking place between nodes. Dimension of this square flow matrix is determined according to the number of flow nodes. For example; if there 10 pairs of coordinates listed in the nodes coordinates file, there must be 10 rows and 10 columns in the interaction matrix which implies 100 flows. It is also critical that row and column order of the flow matrix should be arranged in the same order that is used for the coordinates and names files. Meaning that if coordinates of Node A is listed on the first row and Node Z is listed on the last row of coordinates file, flows outgoing from Node A should be listed on the first row and Node Z should be listed on the last row in the flow matrix. In the flow matrix, magnitudes listed on the same row correspond to outgoing flows and magnitudes listed on the same column correspond to incoming flows. In theory, it is also possible to assign values to diagonal elements; however in practice they are set to "0" zero since no self flows occur. Like other input files, interactions file is plain text based and each element appearing on the same row is separated either by whitespace or tab. In Table 6.7, content of an "interaction file" or namely "flow matrix file" is given.

Table 6.7. Structure of Input File Storing Interaction Matrix

Interactions (Money Transfer) between 12 Bank Locations in The U.S.													
To		Amount of Money Received by Node											
From		1	2	3	4	5	6	7	8	9	10	11	12
Amount of Money Sent from Node	1	0	289	47	52	137	118	90	10	16	15	13	80
	2	602	0	231	209	388	307	286	15	48	26	18	261
	3	143	414	0	84	342	130	134	8	25	10	10	80
	4	68	192	47	0	171	177	618	16	44	43	19	131
	5	150	266	158	226	0	578	295	20	62	54	22	152
	6	122	159	57	186	319	0	439	30	51	78	102	189
	7	97	155	39	496	143	266	0	74	278	100	40	290
	8	31	56	14	142	80	201	573	0	46	128	47	109
	9	14	26	11	32	29	41	295	10	0	51	14	138
	10	20	41	8	55	40	71	215	33	129	0	86	247
	11	31	41	8	38	46	165	125	20	37	253	0	203
	12	82	81	23	84	114	106	251	22	127	128	43	0

Five different datasets are provided for testing purposes. Each dataset includes three input files: (i) node coordinates, (ii) node names and (iii) interaction matrix or namely flow matrix. Each input file is created in plain text format and edited to be compatible with the input data structure of FlowMapper described above. The first dataset includes 4 nodes and a 4 by 4 interaction matrix. Each node corresponds to a location of city center in Turkey and flows taking place between these city centers are stored in the interaction matrix. This small dataset is prepared for quick testing purposes during development of FlowMapper and does not imply any real case or interaction. As a second set of sample data, the test dataset supplied with the Flowpy script is used. Developed by Gelennon (2009) under GPL, this Python script creates geographic flow lines between discrete locations and some parts of this script are reused in the flow generator module of FlowMapper. This dataset includes 12 nodes each of which corresponds to a location of a bank in the U.S. Magnitude of interactions between these banks are stored within a 12 by 12 square matrix in plain text format. Third flow dataset pertains to internal migration of Turkey regarding five years period between 2007 and 2012. This data is available from the web site of TurkStat on yearly basis (TurkStat, 2013). This is a medium size dataset and includes 81 nodes each of which corresponding to a city of Turkey. Interaction matrix is prepared in plain text format holds magnitudes of nearly 6.500 flow lines corresponding to human migration between cities. Fourth test dataset is about long term or permanent global immigration. This is a fairly large dataset comprising of 226 countries. Hence, it is a suitable test dataset for testing the filtering capabilities of FlowMapper. Data is available from the web page of DRC (2007). It includes a 226 by 226 matrix which is generated by disaggregating the information on migrant stock in each destination country as given in its census (DRC, 2007). Reference period of the dataset is the 2000 round of population censuses. Fifth and the last test dataset refer to a different scenario rather than usual geographic flows. This dataset is about passing distributions between the team players in a soccer match. Interaction matrix displaying passing distributions regarding the FIFA 2010 World Cup Final, which was played between Spain and the Netherlands, can be accessed from

FIFA Official Documents Repository (FIFA, 2010). Passing distribution dataset comprises of 14 players while 3 of them were substitute players. In order to create a compatible dataset to be used with FlowMapper, players are regarded as flow nodes and since the location of the Soccer City Stadium where the match is played in Johannesburg is known, coordinates for each node are determined by looking at the arrangement of players on the soccer field. Besides, node names file is created by using the names of players.

Totally five different test datasets are created in different sizes. While the smallest dataset has just four nodes, the largest has more than two hundred nodes. Practically FlowMapper does not imply any specific limitations regarding the size of data; however in theory any limitation that applies to shapefile format also applies to FlowMapper since flow features are created in this format. Yet, remembering that each component of the shapefile format (e.g. shp, shx, dbf) is limited to a size of 2 GB, FlowMapper has still enough space to handle more than six millions of flow lines including their automatically calculated attributes. In other words, FlowMapper is theoretically capable of handling a square matrix that stores two way interactions taking place between slightly more than 2.500 flow nodes. In Table 6.8, properties of selected test datasets are summarized. Under the next heading, several flow maps will be generated to test the capabilities of FlowMapper by using each of these datasets.

Table 6.8. Properties of Selected Test Datasets

	Small Synthetic Dataset	US Bank Interactions	Passes in a Soccer Match	Internal Migration of Turkey	Global Migrants
Intended Purpose	For quickly testing plugin's capabilities during development.	For testing symbology and filtering capabilities on a small dataset	To show how the plugin can be useful for different scenarios	For testing symbology and filtering capabilities on a medium sized dataset	For testing symbology and filtering capabilities on a large dataset
Description	The simplest dataset that can be used to test FlowMapper.	Bank interactions between 12 different locations in the U.S.	Distribution of passes between team players during a soccer match	Migration between 81 cities of Turkey between 2007 and 2012	Global immigration data between 226 different countries
Type	Synthetic	Synthetic	Real Data	Real Data	Real Data
Dataset Size	Very Small	Small	Small	Medium	Large
No. of Nodes	4	12	14	81	226
Matrix Size	4x4	12x12	14x14	81x81	226x226
Projection & Datum	Geographic (in Long. – Lat. order) & WGS84				
File Structue	Full text in ANSI encoding with space delimited variables				
Date of Data	N/A	N/A	2010	2007 – 2012	2000
Data Source	N/A	Glennon (2009)	FIFA (2010)	TurkStat (2013)	DRC (2007)

6.4. Generating Flow Maps with FlowMapper and Discussing Results

In this part, by using the test datasets introduced in the previous heading, several flow maps will be generated based on different scenarios in order to demonstrate functional and cartographic capabilities of FlowMapper.

At first, it will be convenient to start with the smallest dataset. Since this dataset contains only four nodes, output flow map will not be subject to any visual clutter problem. In this way, basic characteristics of flow maps generated by FlowMapper could be better understood since no details are hindered. In Figure 6.10, three flow maps displaying respectively two way, net and gross type of flows are given. All three map layouts are prepared by using FlowMapper plugin and QGIS map composer manager utility. To be more precise, a view from Google Maps is added as a background layer by using another QGIS plugin named as "OpenLayers plugin".

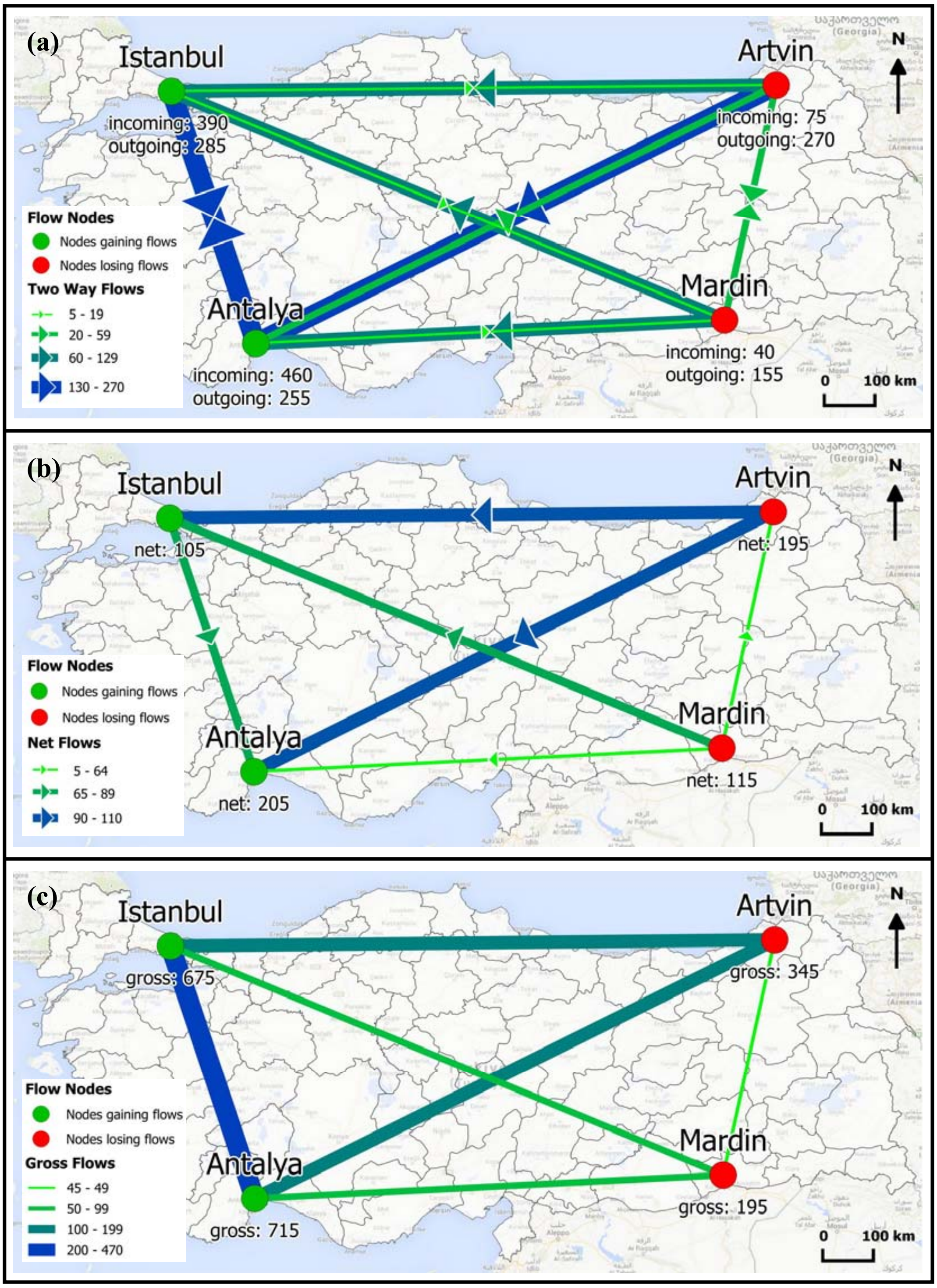


Figure 6.10. Flow Maps Generated by Using 4x4 Interaction Matrix: (a) Two Way Flows, (b) Net Flows, (c) Gross Flows

FlowMapper uses straight lines in order to generate flow lines between discrete nodes. During literature survey, this was identified as the most fundamental cartographic technique for displaying spatial interactions. In Figure 6.10(a), “two way” flows are mapped as straight line segments between discrete locations. In this mode, all incoming and outgoing flows are shown on the map. To prepare this map, first flow features are created in shapefile format by using the “Generate flow lines and nodes” tool. Then flow lines are added into map with graduated symbology representation. Since “Equal size classes” is selected, there are equal numbers of features in each of these 4 graduated classes. By default, line colors of graduated classes shades from light green to dark blue when graduated symbology renderer is selected. Besides, line thickness of each graduated class increases proportional to the magnitude of flow. Since “Show flow direction” option is also selected, an arrow head is overlaid on the center of each flow line in order to show the flow direction. Similar to line thickness, size of this direction arrow increases proportional with the flow magnitude. As it can be seen in Figure 6.10(b), flow lines having less magnitude are rendered on top of major flow lines having greater magnitudes. This is a technique which is also referred by Tobler (1987). By using this technique, flow lines displaying minor flows are not suppressed by thicker flow lines showing major flows. In addition to flow lines, flow nodes are also added to map and shown with two different representations. Since “Differentiate symbology by flow gain and loss” option is selected, by accounting all cumulative incoming and outgoing flows, if the magnitude of incoming flows to a node is greater than the magnitude of outgoing flows from that node, then this node is rendered in green; otherwise it appears red. As can be seen from the maps given in Figure 6.10, while Antalya and İstanbul are shown in green as gaining nodes, Artvin and Mardin are shown in red since they lose flows.

In Figure 6.10(b) and Figure 6.10(c), flow maps respectively displaying net flows and gross flows are given. To prepare these maps, net and gross type of flow lines are created in shapefile format by using the “Generate flow lines and nodes” tool. Then these flow lines are added into map with graduated symbology representation. For net flow lines, “Equal size classes” representation is selected. So, there are equal numbers of features in each of these 3 classes. For gross flow lines, 4 different graduated representation classes are defined in “Manual Interval” mode by using “Symbology for flow lines” tool.

Magnitude of net flow between two nodes is calculated by subtracting magnitudes of two flows in opposite directions. Depending on the subtraction result, which gives either a positive or a negative value, direction of net flow is determined. For example, by looking at the attribute table of two way flow lines, it can be inferred that magnitude of flows outgoing from Antalya to Mardin is 15; in the opposite direction there are 60 flows originating from Mardin to Antalya. Hence, as shown in Figure 6.10(b), net flow magnitude between Antalya and Mardin is 45 and the direction is towards Antalya. Calculation of gross flow magnitude between two flow nodes is straightforward; it is just aggregation of incoming and outgoing flows between two nodes. Since gross flow is an indicator of all interactions between two locations and it does not imply any direction. For example, in Figure 6.10(c), gross interaction between Antalya and Mardin is shown as 75 by adding the magnitudes of two flows in opposite directions.

U.S. bank interactions dataset is a small test dataset comprising of 12 flow nodes. However, since it is larger than the first synthetic dataset which only contains 4 nodes; bank interactions dataset can be used for demonstrating basic filtering capabilities of the plugin such as magnitude filtering. In Figure 6.11(a), a flow map rendered with graduated symbology is given to show gross interactions which have magnitudes greater than average gross flow magnitude.

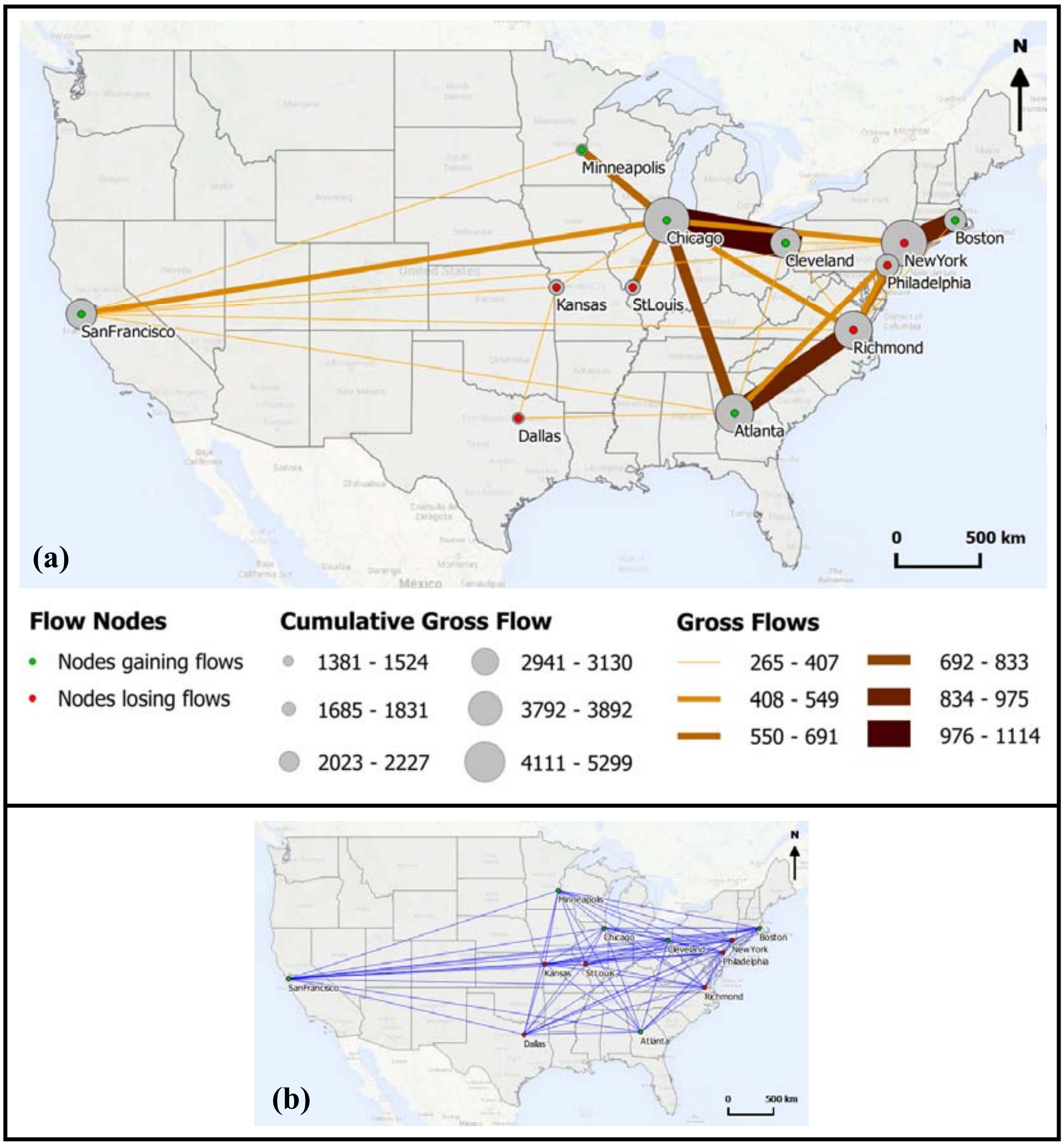


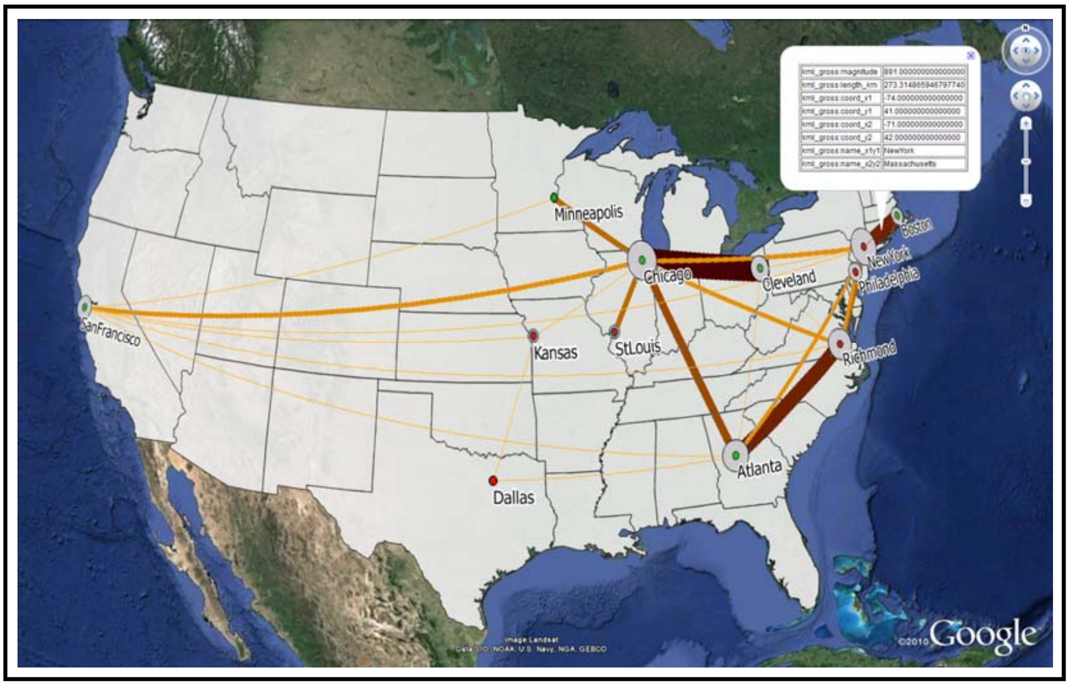
Figure 6.11. Flow Maps Generated by Using Bank Interactions Dataset: (a) Gross interactions greater than the average magnitude with graduated symbology, (b) Unfiltered gross interactions with single symbology

As the number of flow nodes increases, resulting number of two way flow lines increases in a quadratic way. Thus, Tobler (1987) suggests utilization of computerized techniques when dealing with flow datasets containing more than ten nodes. When mapped, these datasets produce more than one hundred flow lines; hence cartographic concerns arise such as feature overlaps or crossings. For example, when bank interactions data is mapped to show two way flow lines, more than a hundred line features will be generated. Assuming that only gross flow lines are created, this will still generate about 70 flow lines. In Figure 6.11(b), all gross flow lines created by using bank interactions data is presented with single symbology representation. It is obvious that unless some clutter reduction techniques are applied, resulting map is far from being informative.

Filtering, together with utilization of cartographic representation techniques such as size, color and transparency, are mentioned by Tobler (1987) and by Ellis and Dix (2007) as two common techniques to reduce visual clutter problem in flow maps. With this in mind, in Figure 6.11(a), gross flow features only having magnitudes greater than the average are mapped. In other words, minor flows are filtered from whole dataset by using "Filter flow lines by magnitude" tool and only major gross flows are mapped in six equal interval graduated representation classes.

As it can be seen in Figure 6.11(a), in addition to flow lines, flow nodes are also shown on map with grey circles. Similar to line thickness which is rendered proportional to flow magnitude, diameter of each circle overlaid on a flow node is adjusted proportional to the magnitude field selected by user, which is gross magnitude for this instance. Besides, red or green markers are also overlaid on top of these grey circles in order to indicate whether a node gains or loses flows. For example, by looking at the diameter of the grey circle on New York, user can interpret that this node is subject to more than 4000 gross interactions; yet there is a red marker on New York since the amount of cumulative outgoing flows from New York is higher than the amount of incoming cumulative flows.

It is also possible to display results on Google Earth. This can be performed in several ways. User can export flow features to "kml" format by using FlowMapper's "Export…" tool or can use built-in export capabilities of QGIS. Besides, by using another QGIS Python plugin named as "GEarthView", map layout can be draped on Google Earth as geo-referenced raster. Both the vector flow features and the map layout are overlaid on Google Earth and presented in Figure 6.12.



kml_gross:magnitude	891.0000000000000
kml_gross:length_km	273.314865946797740
kml_gross:coord_x1	-74.000000000000000
kml_gross:coord_y1	41.000000000000000
kml_gross:coord_x2	-71.000000000000000
kml_gross:coord_y2	42.000000000000000
kml_gross:name_x1y1	NewYork
kml_gross:name_x2y2	Massachusetts

Figure 6.12. Gross Interactions Displayed in Google Earth

Internal migration data of Turkey is a medium size dataset comprising of 81 city nodes and a corresponding migration matrix. Given an 81 by 81 flow matrix, this will result more than 6.000 flow lines to be rendered which is inconvenient to map unless visual clutter reduction techniques are implemented. Thereby, all filtering capabilities of FlowMapper will be used together with graduated symbology representation capabilities in order to generate informative and cartographically appealing flow maps. By using this migration dataset, four flow maps are generated. The first map given in Figure 6.13(a) shows the amount and direction of major net migrations between cities. The second map, given in Figure 6.13(b), is a derivate map which also shows direction of major net migration between cities but this time for those closer than 300 kilometers. Generation of first map involves creation of net flow lines and then filtering of them by magnitude in order to select major flows. Besides, to generate the second map, this subset should be filtered again based on flow length in order to determine short distance flows. The third map, given in Figure 6.14, shows amount of major gross migrations between cities which are not farther than 300 kilometers. To prepare this map, created gross flow lines should be filtered based on their magnitudes and lengths. The last map, in Figure 6.15, prepared by using this migration dataset shows net flows either received or given by Ankara. To create this kind of thematic map which shows interactions related to a specific node, flow lines should be filtered by using the “Filter flows by node and direction” tool.

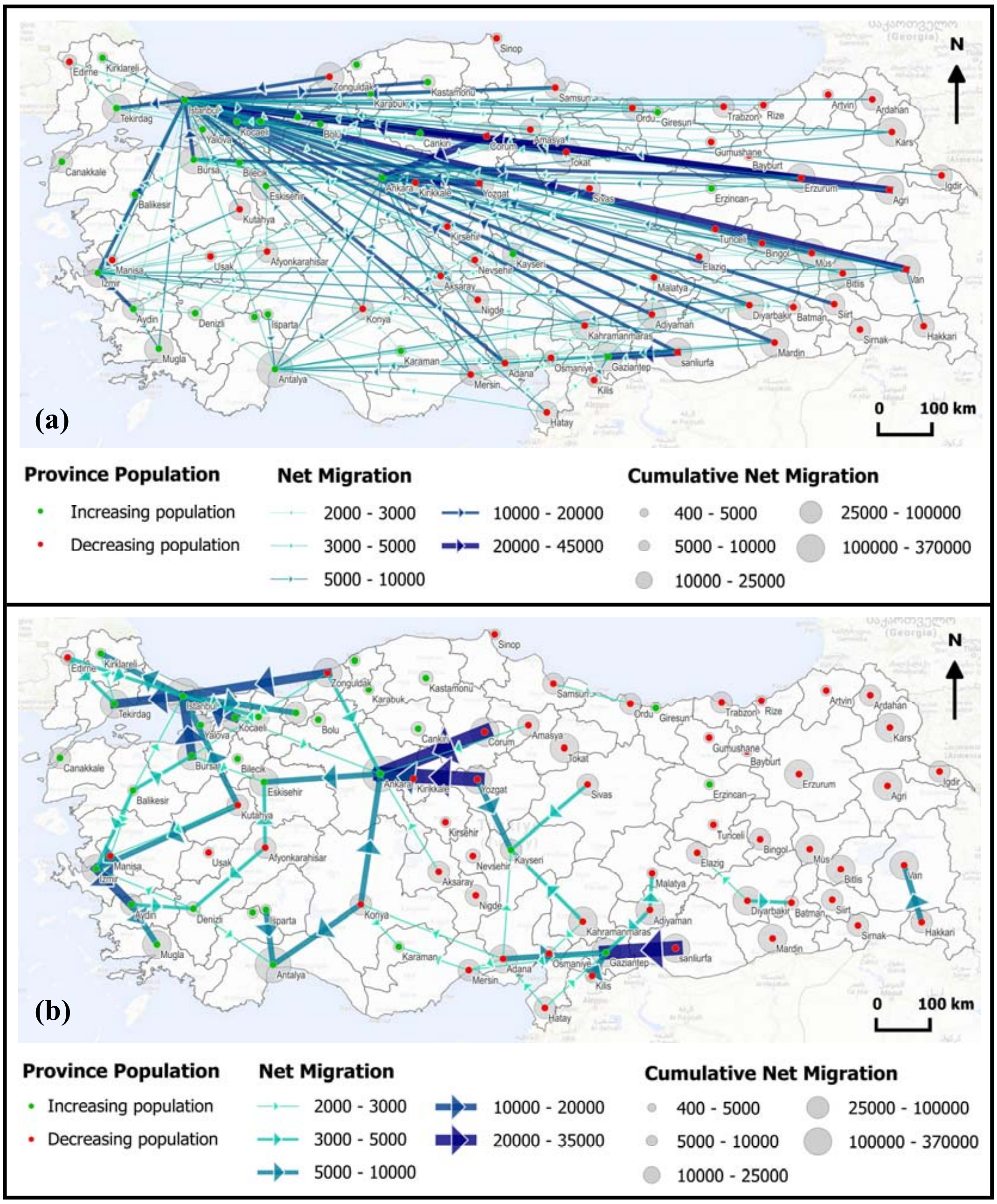


Figure 6.13. Internal Migration Patterns of Turkey between 2007 and 2012: (a) Major net migrations greater than 2.000 people, (b) Major net migrations shorter than 300 kilometers and greater than 2.000 people

The flow map given in Figure 6.13(a) illustrates major net internal migration patterns in Turkey during five years period between 2007 and 2012. In order to identify major migration patterns, net flow lines having magnitudes less than 2.000 people are ignored. This is performed by using “Filter flow lines by magnitude” tool. Although major net migrations are mapped with graduated symbology representation, this map is subject to some level of visual clutter due to crossings of flow lines. However, considering that the number of net flow lines generated from an 81 by 81 matrix is more than 3.000, this migration map can still be taken as informative to reveal magnitude and direction of major net migration patterns. For example; Istanbul can easily be identified as one of the most outstanding city that receives major amounts of migration especially from the eastern parts of Turkey. If further examined, Antalya can be identified as one of the most appealing cities which receives major amount of net migrations almost from all cities including metropolises such as İstanbul and Ankara. Besides, by looking at the color and size of point markers on city centers, it is possible to identify the amount of population increase or decrease for that city. For example; İstanbul, Ankara and Antalya can be identified as cities receiving net migration of more than 100.000 people while Van, Diyarbakır and Erzurum can be identified as cities with outmigration of more than 25.000 people.

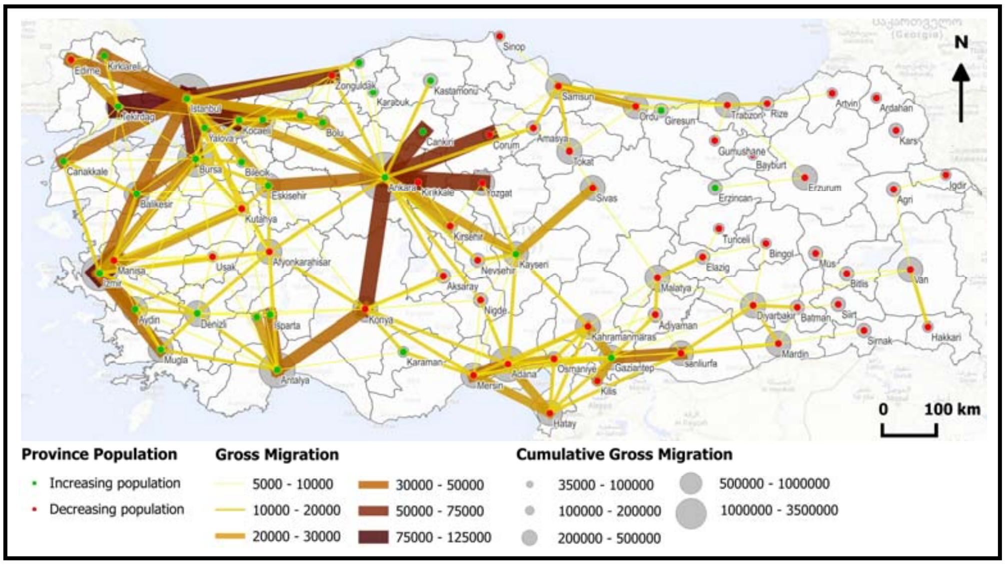


Figure 6.14. Major Gross Migration Patterns of Turkey Shorter than 300 Kilometers and Greater than 5.000 People

In Figure 6.13(b) and Figure 6.14, flow lines are filtered based on distance in order to reveal migration patterns between neighboring cities which are closer to each other than 300 kilometers. This operation is performed by using “Filter flow lines by length” tool. The first map, given in Figure 6.13(b), depicts amount and direction of net migrations between neighboring cities which involve more than 2.000 people. The second map, given in Figure 6.14, shows gross migration patterns between neighboring cities which involve more than 5.000 people. Besides, by looking at the diameter of grey circles overlaid on city nodes, Istanbul and Ankara can be identified as two prominent cities which are subject to more than one million cumulative gross migrations. Compared to the flow map given in Figure 6.13(a) which is prepared by filtering the data based on magnitude, these two maps are less exposed to visual clutter problem since they depict a subset which is also filtered based on distance in addition to magnitude. As a result of ignoring long distance flow lines, the amount of overlaps and line crossings decrease significantly and this provides cartographically more appealing flow maps.

In Figure 6.15, magnitude and direction of all incoming and outgoing net migration patterns are depicted for Ankara by using graduated symbology representation. This is performed by filtering two way flow lines dataset by using the “Filter flow lines by node and direction” tool. In this tool, when graduated symbology representation is selected, by default all incoming flow lines are rendered in shades of green while all outgoing flow lines are rendered in shades of red. Besides, thicknesses of flow lines and size of direction arrows are rendered proportional to the magnitude of flow which is the net migration for this case. With reference to this map, it can be inferred that the most significant incoming net migration sources of Ankara appear as Çorum and Yozgat with more than 5.000 people while Antalya and Eskişehir appear as the two most significant destinations for outgoing net migrations more than 10.000 people.

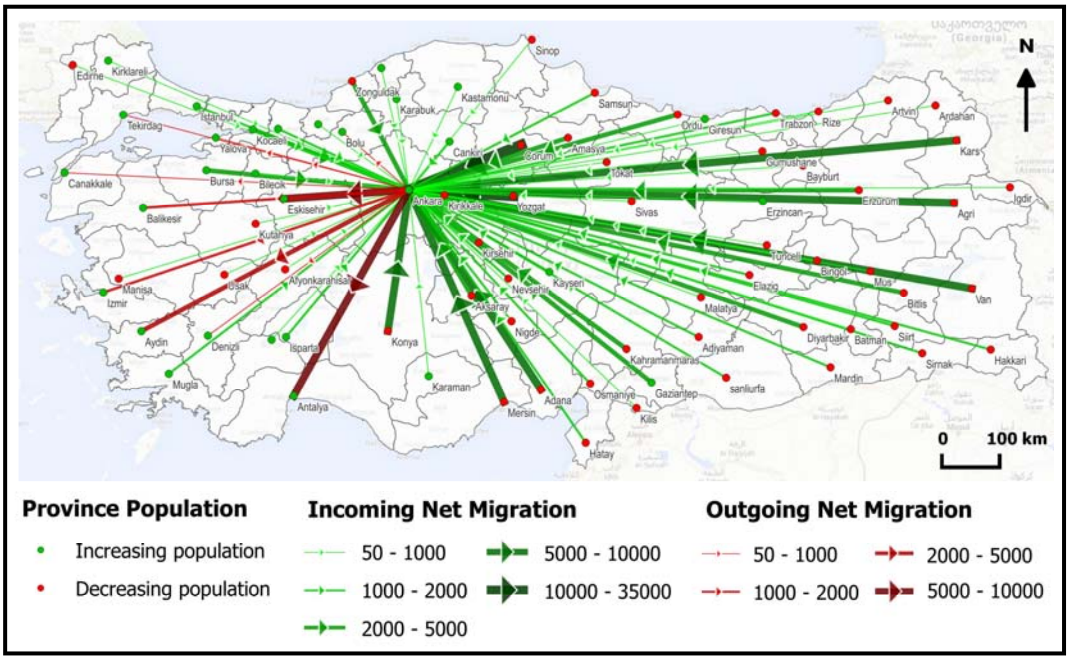


Figure 6.15. Magnitude and Direction of Incoming and Outgoing Net Migration Patterns for Ankara

In Figure 6.16, major destinations of Turkish immigrants are mapped at global scale. With reference to a study carried by DRC (2007) about long term or permanent immigration until 2000, this fairly large dataset involves 226 countries including Turkey.

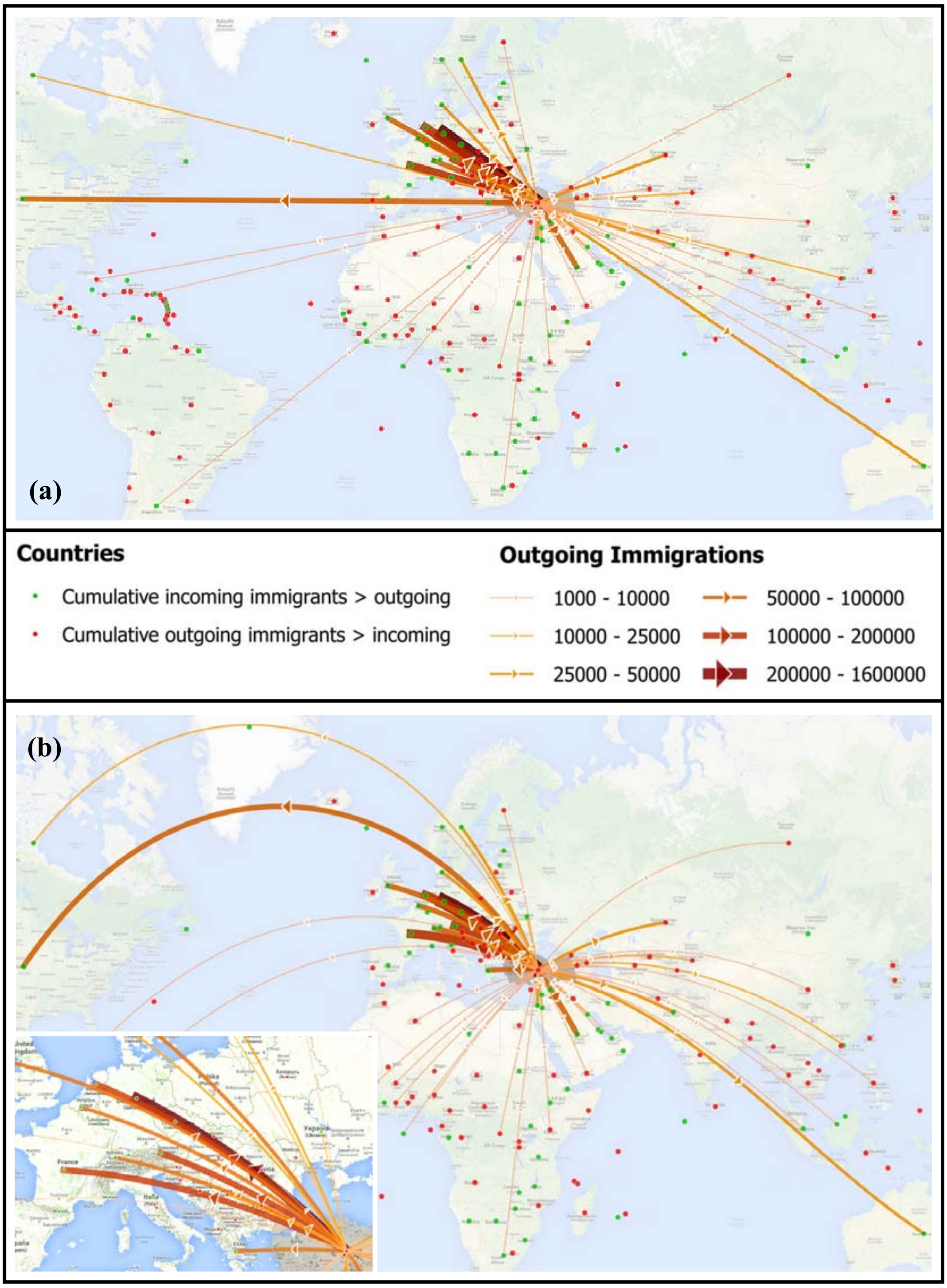


Figure 6.16. Major Destinations of Turkish Immigrants: (a) With straight flow lines, (b) With curved flow lines

When an interaction matrix having dimensions of 226 by 226 is given to FlowMapper and executed in “two way” flows mode, this will result more than 40.000 flow lines which is inconvenient to map simultaneously. Thus, in order to produce cartographically appealing flow maps about Turkish immigrants, this dataset should be filtered to reveal necessary information. For this case, this is performed in three steps: (i) generating all two way flows by using the 226 by 226 immigration matrix, (ii) extracting a subset for flow lines outgoing from the capital city Ankara by using the “Filter flow lines by node and direction” tool, (iii) filtering this subset by using the “Filter flow line by magnitude” tool in order to determine immigration patterns involving more than 1.000 people. Afterwards, all flow lines in this subset are rendered with graduated symbology representation to compose the map layout. When examined in detail, more than 50 countries can be identified as destinations for Turkish immigrants. The most popular destination can be distinguished as Germany with slightly more than one and a half million immigrants. Besides, France, Netherlands and Austria can be identified as other popular destinations in the Continental Europe with more than 100.000 immigrants. On the other hand, in terms of transoceanic long distance immigrations, United States of America is the most popular destination for long term or permanent Turkish immigrants with a magnitude of slightly less than 100.000 people.

As given in Figure 6.16(a), FlowMapper links locations (flow nodes) with straight segments (flow lines) to depict spatial interactions. In Chapter 2, during literature survey, this was mentioned as the most fundamental way of displaying spatial interactions. However, instead of using straight segments, utilization of curves in displaying flows between locations is also possible. This is not a new issue (Thornthwaite, 1934 in Tobler, 1987) and sometimes gives cartographically more appealing results especially for long distance flows since curves represent the globe surface better. The map given in Figure 6.16(a) which shows immigration patterns with straight segments is transformed into a new map that displays interactions with curved features, Figure 6.16(b). However, there are few issues to overcome when preparing flow maps with curved links. First of all, features such as curves, arcs, splines and Bezier curves are not supported in shapefile format by default. If mandatory, such features are represented by inserting a large number of vertices into polyline features to reshape them so that they resemble curves. However, this is a workaround and not supported by FlowMapper. Thus, a third party tool is needed. For example, in the popular proprietary GIS software ArcGIS Desktop, "XY to Line" tool offers such functionality. Given a shapefile created with FlowMapper that stores flow lines, this tool transforms these features into geodesic curves. However, since this involves inserting a large number of vertices into features, total size of the shapefile increases significantly. For example, the total size of the shapefile storing flow lines shown in Figure 6.16(a) is about 25 kilobytes; conversely the shapefile storing flow lines shown in Figure 6.16(b) is about 3.500 kilobytes, which is about 140 times bigger to store the same number features.

The last use case deals with a different scenario rather than usual geographic flows. It involves passing distributions on the field among team players in a soccer match (Figure 6.17). This use case is selected to demonstrate how FlowMapper can be utilized in diverse application areas as a generic tool in displaying different types of spatial interactions.

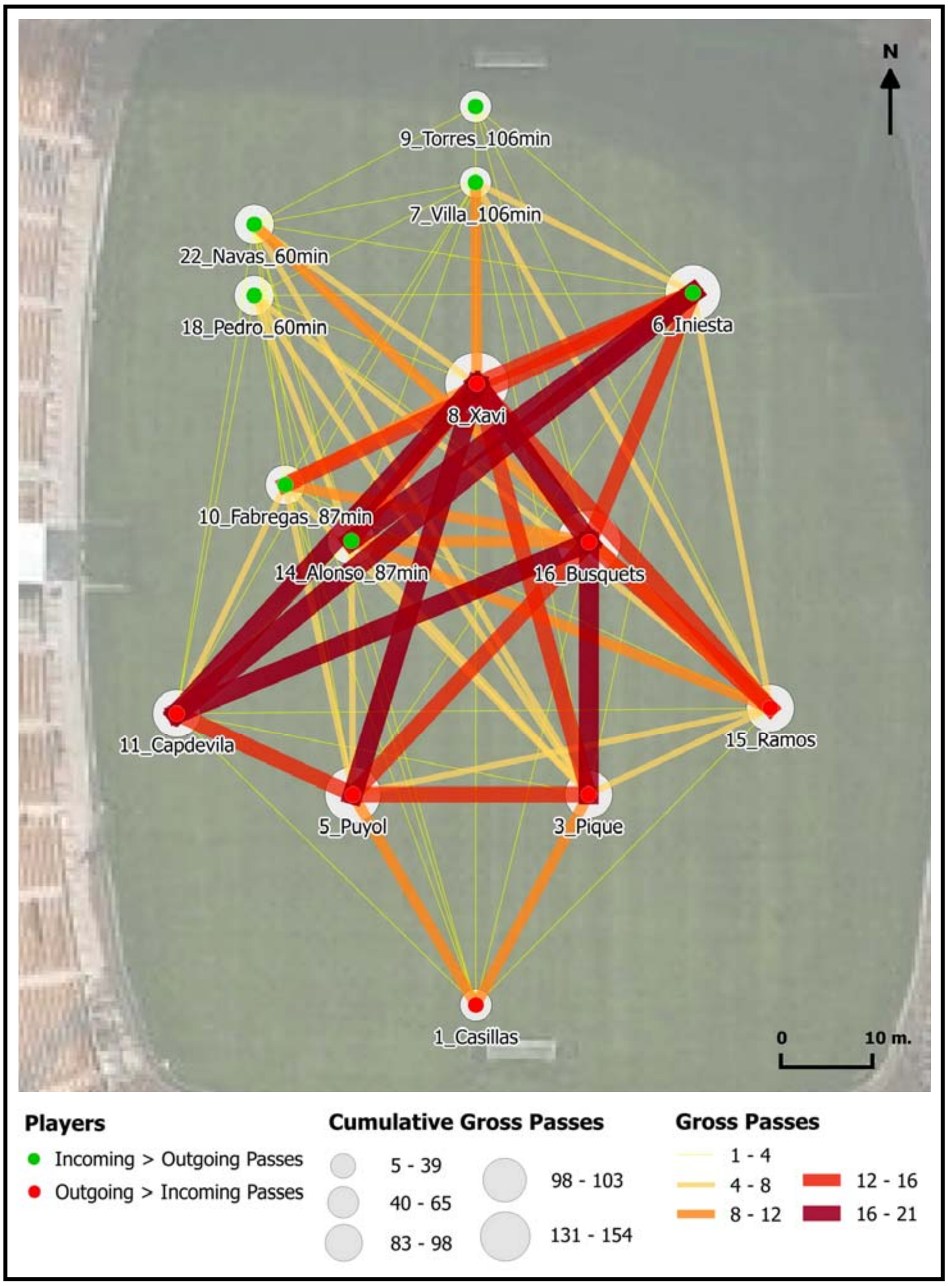


Figure 6.17. Magnitude of Gross Pass Distributions between Players of Spain during Final Match of FIFA 2010

Flow map given in Figure 6.17 illustrates gross pass distributions between players of Spain during final match of FIFA 2010 World Cup. Including extra time, this match lasted about 130 minutes and involved 14 players including the 3 substitutes as Navas, Fabregas and Torres. According to the match statistics provided by FIFA (2010), a total of 542 passes were successfully completed. In order to visualize these passes, each player is assigned to a flow node and coordinates for each node is determined according to the formation of players on the soccer field. Then, by using the pass distributions matrix (FIFA, 2010), gross flow lines are created by using FlowMapper plugin. As given in Figure 6.17, flow lines are rendered with graduated color and thickness which deviates proportional to the magnitude of gross passes. Besides, in order to show amount of cumulative gross passes, a grey circle having diameter proportional to the magnitude is placed on the location of each player. For example, Xavi holding the jersey number 8, can be identified as the most significant player who gave or received more than 130 gross passes. Besides, by looking at the red point marker located on Xavi, it can be concluded that the number of total passes made by Xavi is greater than the number of total passes received by him. In other words, as a midfield player, Xavi gave more passes than he received.

In this part, the capabilities of FlowMapper plugin are demonstrated based on different scenarios by using five different case datasets. Consequently, FlowMapper emerges as a fully GIS integrated general purpose flow mapping tool for displaying spatial interactions between discrete locations in today’s highly mobilized world. Besides, FlowMapper offers additional filtering and cartographic representation capabilities to aid discovering interesting patterns in dataset while striving to avoid visual clutter problem without distorting the essence of flow data.

CHAPTER 7

CONCLUSIONS AND RECOMMENDATIONS

In this chapter, brief summary of the study is presented together with the conclusions that can be derived. Then, recommendations are presented for further research and development of FlowMapper.

7.1. Conclusions and Discussions

In today's highly mobilized world, a significant part of change arises from geographical flows such as the movement of people, ideas, information, money, energy or materiel. In his several studies Tobler (1976, 1981, 1987) drew attention to the importance of geographical flows by focusing on the visualization of spatial interaction data. Soon after Tobler drew attention to geographic movement and flow phenomena; in his pioneering publication "The Informational City", Castells (1989) introduced the concept of "Space of Flows" by reconceptualizing new forms of spatial mobilities with respect to technological paradigms which portray a new type of space allowing real time distant interactions (Castells, 2004). In other words, in contrast to geographically bounded spaces, the "Space of Flows" concept pertains to human actions and interactions circulating dynamically among organizational nodes (Stalder, 2003). From this point of view, representing geographic movements, displaying flows and spatially dynamic events are key activities in perceiving today's highly mobilized world which is subject to flows of people, information, money, goods, etc. on an ever increasing scale…

This study was initiated with the motivation of developing a general purpose flow mapping software which is intended to help people better understand today's highly mobilized world by displaying spatial interactions and flows between locations. Considering extensive capabilities of today's computers to process large amounts of data and the proven potential of GIS in handling spatial datasets offers great opportunities in terms of discovering the potential of spatial interaction data. However, considering the efforts that have been made in the last few decades to reveal the full potential of GIS almost in every aspect, it is surprising to find how full potential of flow mapping remained relatively less developed under GIS. For example, either commercial or open source, there are no special tools included in any of the popular GIS software such as ArcGIS, MapInfo, GRASS or QGIS. In other words, none of this software offers out of the box functionality to depict spatial interactions. With the development of FlowMapper plugin, which is fully coupled with QGIS, this gap is partially filled in terms of integration of dedicated flow mapping tools to desktop GIS.

Although GIS provides substantial tools and functions for dealing with spatial data, handling of spatially dynamic events such as flows posses challenges for cartographers. These challenges can be evaluated from two perspectives: (i) problems arising from the essence of flow phenomena and spatial interaction data; (ii) problems arising from the absence of dedicated flow mapping tools integrated to off-the-shelf GIS software. For the first case, the most common challenge can be expressed as visual clutter problem which arises from overlapping flow features while displaying large flow matrices. Due to the structure of interaction matrices, as the number of flow nodes increases linearly, the number of corresponding flow lines increase in a quadratic way. As also suggested by Tobler (1987), to map flow lines and calculate corresponding attributes (e.g. net or gross flow magnitudes) for datasets including especially more than ten flow nodes, computerized techniques should be used. However at this point, the absence of GIS integrated flow mapping tools emerges as another challenge. By considering these challenges, FlowMapper plugin was designed in plugin form as a fully GIS integrated software while implementing several spatially non-distorting visual clutter reduction techniques which were discussed by Ellis and Dix (2007) as filtering (e.g. subsetting by a given criteria) and advanced symbology (e.g. graduated color rendering, proportional symbol size adjustment etc.)

One of the reasons of the absence of integrated flow mapping tools to popular commercial desktop GIS applications may be due to weak demand of commercial market for such domain specific tools. In other words, the number of potential users demanding to pay for such tools may not be commercially enough to attract companies to initiate development. Considering that flow mapping is one of these areas where the market is not strong enough to attract commercial developers, free and open source development should be embraced as the preferred methodology for development of applications in specific research domains such as flow mapping. On this basis, free and open source development emerged as an agile methodology in development of FlowMapper plugin for QGIS desktop application which is also an open source GIS software supplied freely without any licensing fee.

After researching fundamental concepts in flow mapping, examining historical development together with ongoing challenges in displaying spatial interaction data and after all determining a gap in seamless integration of flow mapping tools to GIS, the aim of this study was shaped under these circumstances. In short, the main objective of this study is to develop free and open source software for flow mapping integrated to GIS. With this motivation, several structural and functional requirements were figured out for this domain specific software based on the findings of literature survey carried in Chapter 2 and review of open source software concept and development methodology reviewed in Chapter 3. For example, some of the most important structural and functional requirements of FlowMapper plugin were determined as follows: (i) FlowMapper should be developed as a plugin to QGIS for seamless GIS integration and offered to all potential users freely as an open source software product, (ii) FlowMapper should be GUI based for ease of user interaction and must be fully functional out-of-the box without needing installation of any other library that is not already included with QGIS installation, (iii) Input and output data structure should be lucid and must use widely accepted formats such as plain text and shapefile, (iv) FlowMapper should be capable of generating flow lines between discrete nodes and be capable of making calculations for net, gross or two way flows, (v) To avoid visual clutter and to reveal desired patterns in data, plugin should offer some basic flow filtering capabilities together with several cartographic representation options for flow lines and nodes.

FlowMapper plugin is built on top of QGIS. By this way, users of FlowMapper can also benefit from all existing capabilities of QGIS. In Chapter 3, several free and open source GIS applications were reviewed and among all QGIS and GRASS were identified as the most functional and mature desktop GIS applications. However, owing to its user friendly GUI, cross platform support including native Windows binaries and stable development environment supported with up-to-date API documentation; QGIS was selected as the most promising desktop GIS application upon which to develop FlowMapper plugin. Besides, as an open source software project continuing with collaborative efforts of community, popularity of QGIS was proven with over half million downloads performed during 2012 (QGIS, 2013c). It is obvious that the popularity and widespread usage of QGIS is promising for a growing active community which guarantees the continuity of QGIS and strengthens the stability of the platform on which the FlowMapper is built.

As a QGIS plugin, development environment of FlowMapper plugin is similar to the environment that is portrayed in PyQGIS developer cookbook (QGIS, 2014a). As reviewed in Chapter 4, following components are used for the development of FlowMapper plugin: (i) Python as the programming language and IDLE Python IDE; (ii) Qt4 cross platform application framework, Qt Designer for GUI development, Python bindings for Qt (PyQt) and its command line tools “pyrcc4”, “pyuic4”; (iii) Python bindings for QGIS API (PyQGIS); (iv) OGR simple features library and its command line utility “ogr2ogr”. In order to fulfill functional requirements of FlowMapper, plugin is written by using these development components with an open source and agile methodology. During development of FlowMapper, this methodology provided some advantages which can be summarized as follows: (i) Providing the plugin free of charge under QGIS, which also does not require any licensing fee, led the plugin to reach more than 12.000 people almost in two and a half year; (ii) Considering that all users of FlowMapper are also native testers, plugin was tested thousands of times and valuable feedbacks were received by email; (iii) Since Python is an interpreted language that runs directly from human readable source code, when a bug is found in the plugin, users identified it and reported the erroneous line of code based on the warning message that is raised by the integrated Python interpreter in QGIS; (iv) GNU GPL v2, under which the QGIS is licensed, guarantees the source code of both QGIS and its plugins to remain public in the future for collaborative development; (v) Providing the source code of plugin to researchers paves the road for further development of FlowMapper and also enables creation of derivative works with code reuse; (vi) Since open source development environment components of QGIS support multi platform, FlowMapper plugin gained native support for multiple platforms without needing any additional coding.

During literature survey, fundamental concepts of flow mapping were reviewed. With the guidance of this survey, interactions taking place between discrete locations over uncertain paths were determined as one of the most common form of flow scenarios. Thus, aiming to offer a general purpose interaction mapping tool, FlowMapper is primarily developed to visualize this type of flows by representing each inter-nodal interaction with a straight line object. Besides, based on the review that visual clutter reduction techniques are evaluated (Ellis and Dix, 2007); filtering and graduated symbology rendering are implemented in FlowMapper as spatially non-distorting techniques for reducing visual clutter in flow maps.

Methodology and implementation stages involved in development of FlowMapper plugin follows the five stage agile methodology defined in the PyQGIS developer cookbook (QGIS, 2014a). These stages involve; (i) formation of the idea, (ii) creation of files, (iii) coding, (iv) testing and (v) publishing the plugin. First stage involves setting the goal and determination of requirements prior to initiating coding. Next stage involves building the skeleton of the QGIS Python plugin. For FlowMapper, this is performed by using another Python plugin named as "QGIS Plugin Builder". Third and fourth stages involve Python code writing, debugging, GUI development and testing. This routine were performed by using the development environment components required for writing an external Python plugin to QGIS which were previously mentioned as IDLE IDE, Qt, PyQt and PyQGIS. For the last stage, FlowMapper and its source code was published in the QGIS official plugin repository to make it available for public. Based on the indicator that FlowMapper was downloaded more than 12.000 times within almost two and a half year period and considering that every user of the plugin is also a potential tester that may contribute to development by sending feedbacks about bugs or asking for new features, FlowMapper reached a mature state with four major and several minor versions.

Coding of FlowMapper started at the beginning of December 2011 and after about 4 months of development, at the end of March 2012, initial release v0.1 was published on the QGIS official plugin repository. This initial release is comprised of nearly 400 lines of Python code which includes the main module containing almost one hundred lines of code and the flow generator module containing more than 150 lines of code. Following this initial release, three major and several minor releases were published. FlowMapper v0.4 was released at the end of December 2013 and comprises of more than 6.500 lines of Python code which is almost 15 times more than the total amount code written for the first release (v0.1). Besides, the length of the main module reached up to 3.000 lines and length of the flow generator module reached up to more than 400 lines.

Development of FlowMapper was performed under Windows. However, since development environment components of QGIS support multi platform, FlowMapper plugin gained cross platform support without needing any extra additional coding. The only modification in the code was needed for making the file paths compatible with the POSIX file system which is used by Linux based and MacOSX systems. As a result of this modification, starting from v0.2.2, all further releases of FlowMapper plugin offer cross platform support and known to be fully functional under Linux. In addition to multi platform support, one of the requirements in development of FlowMapper was to provide a user friendly GUI by which required functions could be accessed via simple menu driven structure. For this purpose, GUI was designed by using the Qt Designer tool. By means of all these efforts, FlowMapper came up as fully GIS integrated free and open source flow mapping tool that can operate on multiple platforms with fully GUI based menu driven structure.

Architecture of the FlowMapper plugin is similar to the architecture of a generic Python plugin portrayed in QGIS developer cookbook (QGIS, 2014a). This implies a modular structure that requires creation of several files such as; metadata, resource and initialization files, a Python file to store the main class and some additional files to store form interface classes. Besides, another module was written for FlowMapper to generate flow features by interacting with the main module. By means of this modular structure that promotes agile development and successive releases, new functions and interfaces were added on top of existing ones in each release without needing to modify the whole source code. In other words, since the main skeleton of the FlowMapper plugin is already established, it just needs some extra coding effort and design of corresponding form interface if a new function needs to be added.

As an integral part of open source development under GNU GPLv2 license, all releases of FlowMapper and their source codes are freely available for download from the QGIS official plugin repository hosted on "http://plugins.qgis.org/plugins/FlowMapper". Additionally, a private repository was also built for FlowMapper and is being hosted on "http://95.9.195.180/plugins.xml". As of August 2014, FlowMapper has been downloaded more than 12.000 times from the QGIS official repository. This means more than 10 downloads a day and justifies the need for development of

FlowMapper. Besides, it also reveals the demand for integration of flow mapping tools to a popular open source desktop GIS application.

In addition to download repositories, a dedicated website was built for the plugin and published at “http://95.9.195.180”. During the period between February 2013 and August 2014, site received approximately 1.200 visits from more than 80 countries. According to the visitor statistics acquired by Google Analytics, almost fifty percent of total visits were performed by the first five countries which can be respectively listed as United States (%13), France (%11), Italy (%7), Turkey (%7) and Germany (%6). Besides, %77 of visitors were using Windows while %15 of them were using MacOSX, %6 were running Linux and %2 were on mobile platforms. Based on this finding, it may be concluded that FlowMapper is being utilized on multiple platforms.

FlowMapper plugin has drawn interest from tens of QGIS users from different countries and took various contributions by emails. Most of these emails include questions about the usage of plugin and preparation of nodal interaction data to use it with FlowMapper. Considering these feedbacks, starting from v0.1.1, FlowMapper plugin package was prepared to include sample datasets and brief documentation that defines the structure of input data format. There were also requests from few users demanding several new functions to be added into FlowMapper such as inclusion of origin and destination node names in the attribute table of flow lines or implementation of node clustering algorithms for reducing visual complexity. However, considering that FlowMapper is intended to be developed as a general purpose interaction mapping tool that only includes spatially non-distorting clutter reduction techniques; implementation of algorithms such as node clustering or flow line bundling were not included to the functional requirements list of FlowMapper. Moreover, there are numerous of clutter reduction techniques reviewed by Ellis and Dix (2007) that may be considered to be implemented in FlowMapper. However, accounting the time budget of this study, only spatially non-distorting techniques such as filtering and advanced symbology techniques were implemented.

Inherently free and open source development promotes collaboration among developers and users. Although FlowMapper is coded just by one developer and there were no active participation to coding; owing to previous development efforts of open source developers, FlowMapper benefited from code reuse which was very valuable especially at the early stages of development. For example, a code snippet derived from Flowpy, which is a Python script originally coded by Glennon (2009), was adapted to be reused in FlowMapper while coding the flow generator module. Besides, GIS forum of StackExchange network was used for asking questions and searching answers about QGIS API related issues.

In Chapter 6, capabilities of FlowMapper were demonstrated by using several test datasets having different sizes. Generated flow maps refer to several common scenarios such as internal country migration and global immigration as well as an uncommon one that involves passing distributions between players on a soccer field. By generating flow maps with different scenarios, FlowMapper proved itself as general purpose flow mapping tool in displaying spatial interactions almost at all scales from local to global. Besides, even working with large interaction matrices, for example 81 by 81 or 223 by 223, desired patterns can be easily extracted by performing simple filtering operations in FlowMapper. Moreover, advanced symbology techniques included in FlowMapper, such as graduated color rendering of flow lines or proportional symbol size for flow nodes, aids creation of cartographically more appealing and spatially informative flow maps.

Considering the previous works of researchers in understanding geographically dynamic events and ongoing efforts of developers to develop case specific mapping tools, it is obvious that FlowMapper is neither the first nor the last software developed for interaction mapping. As mentioned at the end of Chapter 2, there are already several mapping tools available for displaying spatial interactions such as Tobler’s FlowMapper as one of the earliest contributions (Tobler, 2003; CSISS, 2004), Flow Data Model Tools for ArcGIS to visualize different types of flow scenarios (Glennon and Goodchild, 2004), VIS-STAMP software for utilization of multivariate flow data analysis (Guo, 2009), JFlowMap to create maps either with flow binding or node clustering (Boyandin et al., 2010) and Flow Map Layout for utilizing hierarchical clustering on flow lines (Phan et al., 2005). Besides, Flowmap software developed in participation of Utrecht University in the Netherlands and Gadjah Mada University in Indonesia (Geertman et al., 2003), Caliper's transportation planning software TransCAD and general purpose, open source graphing tool Gephi (Bastian et al., 2009) can be counted on the list of applications which are capable of displaying interactions between discrete nodes. However, among all, neither of these applications completely meets the requirements defined for FlowMapper. Although FlowMapper may be criticized as lacking some special algorithms such as flow bundling, node clustering and hierarchical routing; it can be regarded as a fully GIS integrated, general purpose free and open source flow mapping tool that meets many needs of average users with its easy installation process, fully menu driven structure, advanced symbology options and practical filtering functions. In other words, FlowMapper is not the most advanced flow mapping software that includes complex algorithms for specific needs; but it is functionally one of the best balanced software in terms of answering expectations of an average user which was proven with more than 12.000 downloads in two and a half year period.

The most prominent contribution of this study is the FlowMapper software that is fully integrated to QGIS in plugin form. As a fully GUI based general purpose interaction mapping tool, FlowMapper does not require any licensing fee and its source code is publicly open for further development or for code reuse. FlowMapper offers filtering and cartographic representation capabilities to aid discovering interesting patterns in dataset while striving to avoid visual clutter problem without distorting the spatial arrangement and essence of flow data. In other words, it enables the flow data to speak for itself. As a plugin to one of the most popular desktop GIS platform, FlowMapper paves the way from spatial interaction data to spatial knowledge and helps people in understanding today's highly mobilized world which was conceptualized by Castells (1989) as "space of flows".

7.2. Recommendations

FlowMapper was written just by one developer as a general purpose flow mapping tool to aid users of desktop GIS in understanding spatial interactions between discrete nodes. As of August 2014, almost two and a half year after the first release, FlowMapper has reached more than 12.000 users. This number can be increased by adding new functions to plugin with the collaborative efforts of QGIS community developers. In addition to development of incremental releases which offer new features that will attract more users; preparation of up-to-date user manuals, tutorials, making announcements on the OSGeo email lists and preparation of a dedicated wiki page will also assure retaining the community active.

Continuity of an open source project is possible with an active community. Owing to its powerful “Pythonic” API (PyQGIS) and more than half million downloads in 2012, QGIS ensures itself as a stable and promising platform for developers to build plugins on it. Although FlowMapper received more than 12.000 downloads in two and a half year, it is still being coded by one developer. In order to implement further algorithms that will improve analysis and visualization capabilities of FlowMapper, active participation of other developers is needed. At this point, since all source code of FlowMapper is publically open on the QGIS plugins repository, QGIS community developers emerge as an alternative for giving contributions to the plugin. Although this will not guarantee the sustainability of the plugin, an open invitation for collaboration or hand over can be performed anytime by announcing this issue on the QGIS community emailing list.

FlowMapper is developed as an open source, free and fully GIS integrated flow mapping tool. It focuses on displaying inter-nodal spatial interactions which is identified as the most common and fundamental form of flow mapping in the literature. As a user friendly, fully GUI based tool, FlowMapper runs fast and does not imply any specific limitations regarding the size of input data. On a quad core

PC with 8GB ram, more than ten thousand flow lines are created with their attributes in less than five seconds. However since FlowMapper creates outputs in shapefile format, 2 GB maximum size limitation of each shapefile component also applies to FlowMapper. Yet, there is still enough space to handle more than six millions of flow lines including their attributes. This means FlowMapper can work with flow matrices storing two way interactions taking place between slightly more than 2.500 locations which is quite enough for many needs. If a larger matrix is involved, either creation of several flow attributes should be eliminated by modifying the source code to reduce output file size or utilization of more capable vector data formats should be preferred.

As a general purpose flow mapping tool, FlowMapper offers only spatially non-distorting visual clutter reduction techniques such as filtering of flow features based on attributes, proportional symbology rendering, graduated color representations etc. Considering that much of the research efforts in flow mapping domain are focusing on techniques for visual clutter reduction and effective representation of large interaction matrices; FlowMapper may be criticized as lacking some advanced algorithms such as flow bundling, node clustering or hierarchical routing. However, considering the extendable, modular structure of FlowMapper described in Appendix A and agile development environment of Python, implementation of the visual clutter reduction methods reviewed by Ellis and Dix (2007) will be a further development issue for a developer fluent in Python.

Although it is not considered as a clutter reduction method, utilization of curved flow links instead of straight line segments, namely using geodesic flow lines between flow nodes, can be offered as an option in FlowMapper that will improve aesthetic quality especially for representing long distance global flows.

FlowMapper was developed for visualizing interactions between discrete nodes where the actual paths or routes of flows are either unknown or unimportant. Two other types of flows mentioned by Glennon and Goodchild (2004) were listed as:

(i) flows taking place on network links or through known routes (e.g. traffic density on road segments) and (ii) situations where node-to-node and network flows occur in proximity (e.g. hydrological flows with sources and sinks). With the implementation of additional flow data models, capabilities of FlowMapper can be extended to cover these types of flow scenarios.

There are many algorithms that can be adapted to be implemented in FlowMapper. By utilization of external libraries, it is theoretically possible to write more complex algorithms in FlowMapper such as hierarchical clustering (Phan et al., 2005) or SOM (Guo, 2009). However, as these algorithms are being added to extent the capabilities of plugin, this will also expose a risk of deviating from the core requirements of FlowMapper one of which is defined as avoiding implementation of distorting visual clutter reduction techniques that involve spatial rearrangement of flow data. While others are defined as offering a general purpose, user friendly tool that is fully functional out of the box without needing installation of any third party library. Thus, any further function to be included in FlowMapper should be evaluated in detail and designed properly to keep the balance between functionality and user friendliness.

REFERENCES

Andrienko, G., Andrienko, N., Kopanakis, I., Ligtenberg, A. and Wrobel, S., (2008), Visual Analytics Methods for Movement Data Mobility, book chapter in Mobility, Data Mining and Privacy: Geographic Knowledge Discovery, Springer Publishing Co., pp. 375-410

Bastian, M., Heymann, S., Jacomy, M., (2009), Gephi: An Open Source Software for Exploring and Manipulating Network, in Proceedings of the 3rd International Conference on Weblogs and Social Media, San Jose, California, USA

Beck K., Beedle M., Bennekum A., Cockburn A., Cunningham W., Fowler M., Grenning J., Highsmith j., Hunt A., Jeffries R., Kern J., Marick B., Martin R. C., Mellor S., Schwaber K., Sutherland J., Thomas D., (2001), Manifesto for Agile Software Development, Agile Alliance, retrieved from http://agilemanifesto.org (visited 26 May 2013)

Börner, K., Hardy, E.F., (Eds.), (2008), 4th Iteration: Science Maps for Economic Decision Makers, Places and Spaces: Mapping Science, retrieved from http://dev.scimaps.org/maps/map/europe_raw_cotton_im_3 (visited 3 Jun 2012)

Boyandin, I., Bertini, E., Lalanne, D., (2010), Using Flow Maps to Explore Migrations Over Time, in Proceedings of Geospatial Visual Analytics Workshop in conjunction with the 13th Agile International Conference on Geographic Information Science, Guimaraes, Portugal

Brodbeck, B., Chalmers, M., Lunzer, A., Cotture, P., (1997), Domesticating Bead: Adapting an Information Visualization System to a Financial Institution, in Proc. InfoVis'97, Phoenix, IEEE, pp. 73-80

Brown, M. C., (2001), Python: The Complete Reference, McGraw-Hill/Osborne Media, U.S.

Brunza, P.D., Weide, Th.P., (1989), The Semantics of Data Flow Diagrams, in N. Prakash (Eds.) Proceedings of the International Conference on Management of Data, Hyderabad, India

Caliper Corporation, (2014), TransCAD Pricing and Ordering, in the official website of TransCAD Transportation Planning Software, retrieved from http://www.caliper.com/TransCAD/TransCADVersions.htm (visited 6 Jul 2014)

Card, S.K., Mackinlay, J.D., Shneiderman, B., (1999), Readings in Information Visualization: Using Vision to Think, Chp. 1-2, Morgan Kaufmann

Cascadoss, (2007a), Development of a trans-national cascade training program on Open Source GIS&RS Software for environmental applications, Inventory and Analysis of OSS Business Models, Deliverable 1.5, rev. final, pp. 9-11, 21, 41 retrieved from http://www.cascadoss.eu/en/PDFs/D1.5_Business_Models.pdf (visited 26 May 2013)

Cascadoss, (2007b), Development of a trans-national cascade training programme on Open Source GIS&RS Software for environmental applications, Evaluation of Desktop Applications, retrieved from http://www.cascadoss.eu/en/index.php?option=com_content&task=view&id=14&Itemid=14 (visited 26 May 2013)

Castells, M., (1989), The Informational City: Information Technology, Economic Restructuring and the Urban Regional Process, Oxford: Blackwell, UK

Castells, M., (2004), An Introduction to the Information Age, in The Information Society Reader, editors: Webster, F., Blom, R., Karvonen, E., Melin, H., Nordenstreng, K., Puoskari, E., London and New York: Routledge, pp. 138-149

Chen, D., Shams, S., Carmona-Moreno, C., Leone, A., (2010), Assessment of Open Source GIS Software for Water Resources Management in Developing Countries, in Journal of Hydro-environment Research 4, pp. 253-264

Chen, K., Liu, L., (2003), A Visual Framework Invites Human into the Clustering Process, in Proc. Int. Conf. Scientific and Statistical Database Management, IEEE, pp. 97-106

CISS, Center for Spatial Integrated Social Science, (2004), Tobler's Flow Mapper, website including description, functionalities and documentation of software, retrieved from http://csiss.ncgia.ucsb.edu/clearinghouse/FlowMapper (visited 5 June 2011)

Cogliati, J., (2005), Non-Programmers Tutorial for Python, retrieved from http://jjc.freeshell.org/easytut/easytut.pdf (visited 04 Feb 2012)

Corradini, G., Racicot, A., (2011), QGIS Workshop v1.0.0 documentation, Python in QGIS, retrieved from http://www.qgisworkshop.org/html/workshop/python_in_qgis_intro.html (visited 21 Dec 2013)

Cui, W., Zhou, H., (2008), Geometry-based edge clustering for graph visualization, Transactions on Visualization and Computer Graphics, 14(6), Nov-Dec 2008, pp.1277-1284

Darrell, R.R., (1991), Reading Source Code, in CASCON '91 Proceedings of the 1991 conference of the Centre for Advanced Studies on Collaborative research, IBM Press, pp. 3-16 de Bruijn, O., Spence, R., (2000), Rapid serial visual presentation: a space-time trade-off in information presentation, in Proc. AVI 2000, Trento, Italy, ACM Press, pp. 51-60

de Jong, T., van Eck Ritsema, J. R., (1996), Location profile based measures as an improvement on accessibility modeling in GIS, in Computers, Environment and Urban Systems, Vol. 20, pp. 181-190

Derthick, M., Christel, M.G., Hauptmann A.G., Wactlar, H.D., (2003), Constant Density Displays Using Diversity Sampling, in Proc. InfoVis'03, Seattle, IEEE, pp. 137-144

Dix, A., Ellis, G.P., (2002), By chance: enhancing interaction with large data sets through statistical sampling, in Proc. AVI'02, L'Aquila, Italy, ACM Press, pp. 167-176

DRC, Development Research Centre on Migration, Globalisation and Poverty, (2007), Global Migrant Origin Database, retrieved from http://www.migrationdrc. org/research/typesofmigration/global_migrant_origin_database.html (visited 23 Apr 2013)

Dykes, J. A., Mountain, D. M., (2003), Seeking structure in records of spatio-temporal behaviour: visualization issues, efforts and applications, in Computational Statistics and Data Analysis, 43 (Data Visualization II Special Edition), pp. 581-603

Ellis, G., Dix, A., (2007), A Taxonomy of Clutter Reduction for Information Visualisation, in IEEE Transactions on Visualization and Computer Graphics, Nov/Dec 2007, Vol. 13, No. 6, pp. 1216-1223

ESRI, Environmental Systems Research Institute, (2012), ESRI Developer Network, About the EDN Subscription Program, How to subscribe?, retrieved from http://edn.esri.com/index.cfm?fa=misc.program#item6 (visited 09 Jun 2013)

ESRI, Environmental Sciences Research Institute, (2013), Definition of Desktop GIS in ESRI GIS Dictionary, retrieved from http://support.esri.com/en/knowledge base/GISDictionary/term/desktop%20GIS (visited 16 Jun 2013)

ESRI, Environmental Sciences Research Institute, (2014), Products: ArcGIS for Desktop, Extensions, in the official website of ESRI, retrieved from http://www.esri.com/software/arcgis/arcgis-for-desktop/extensions (visited 14 Jun 2014)

Fekete, J.D., Plaisant, C., (2002), Interactive Information Visualization of a Million Items, in Proc. InfoVis'02, pp., IEEE, pp. 117-124

FIFA, 2010, Official Documents Repository, (2010) FIFA World Cup South Africa, Final: Neatherlands - Spain, Passing Distribution, retrieved from http://www.fifa.com/mm/document/tournament/competition/01/27/28/23/64_0711_ned-esp_passingdistribution.pdf (16 Nov 2013)

Flowmap, (2013), Flowmap Home, official website of Flowmap including information about software and its features, retrieved from http://flowmap.geog.uu.nl/index.php (visited 2 Feb 2014)

Fogel, K., (2006), Producing Open Source Software, O'Reilly, pp. 57-60, 152-154, retrieved from http://producingoss.com/en/producingoss.pdf (visited 02 Jun 2013)

Friendly, M., (2002), Visions and re-visions of Charles Joseph Minard, in Journal of Educational and Behavioral Statistics, 27(1), pp. 31-52

FSF, Free Software Foundation, (2013a), What is free software, The Free Software definition, retrieved from http://www.fsf.org/licensing/essays/free-sw.html (visited 19 May 2013)

Fua, Y. H., Ward, H.O., Rundensteiner, E.A., (1999), Hierarchical Parallel Coordinates for Exploration of Large Datasets, in Proc. Visualization'99, Los Alamitos, CA, IEEE, pp. 43-50

GDAL, (2014a), GDAL/OGR 1.10.1 release, GDAL Raster Formats, retrieved from http://www.gdal.org/formats_list.html (visited 05 Jan 2014)

GDAL, (2014b), GDAL/OGR 1.10.1 release, OGR Vector Formats, retrieved from http://www.gdal.org/ogr/ogr_formats.html (visited 05 Jan 2014)

Geertman, S., de Jong, T., Wessels, C., (2003), Flowmap: A support tool for strategic Network analysis, in Geertman S. & Stillwell J. (Eds.), Planning support systems in practice, Springer Verlag, Berlin

Glennon, A., (2009), Flowpy: Geographic Flow Line Creator Python Script, retrieved from http://enj.com/software/flowpy/ (visited 01 Oct 2011)

Glennon, A., Goodchild, M., (2004), A GIS Flow Data Model, Flow White Paper v03 Revised at 18.04.2005, unpublished paper retrieved from http://parker.ou.edu/~gdi/flow/flowwhitepaperv03.doc (visited 14 May 2011)

GRASS, Geographic Resources Analysis Support System, (2013), Release Notes for GRASS GIS 6.4 for MS-Windows-Native, retrieved from http://grass.osgeo.org/grass64/binary/mswindows/native/ (visited 16 Jun 2013)

Gülgeç, İ., (1998), Ulaşım Planlaması, Özsan Matb., Bursa, Türkiye, p.2, 199, 209

Guo, D., Jin, C., MacEachren, A. M., Liao, K., (2006), A visualization system for space-time and multivariate patterns (VIS-STAMP), in Transactions on Visualization and Computer Graphics, 12(6), IEEE, pp. 1461-1474

Guo., D., (2009), Flow mapping and multivariate visualization of large spatial interaction data, in Transactions on Visualization and Computer Graphics, 15(6), IEEE, pp. 1041-1048

Hägerstrand, T., (1970), What about people in regional science?, in Papers of the Regional Science Association, Vol. 24, pp. 7-21

Healey, C.G., Booth, K.S., Enns, J., (1995), Visualizing Real-Time Multivariate Data Using Preattentive Processing, in Trans. Modeling and Computer Simulation, 5(3), pp. 190-221

Hedley, N. R., Drew, C. H., Lee, A., (1999), Hagerstrand Revisited: Interactive Space-Time Visualization of Complex Spatial Data, Informatica: International Journal of Computing and Informatics, Vol. 23(4), pp. 155-168

Holten, D., Wijk, J.J., (2009), Force-Directed edge bundling for graph visualization, in Computer Graphics Forum, 28(3), pp. 983-990

Hugentobler, M., (2008), Quantum GIS, in Shekhar, S., Xiong, H. (Eds), Encyclopedia of GIS, New York, Springer, pp. 171-188

Jain, A.K., Murty, M.N., Flynn, P.J., (1999), Data Clustering: A Review, in ACM Computing Surveys, 31(3), Sept 1999, pp. 264-323

Janssen, C., (2013), Codebase, Definition - What does codebase mean?, Technopedia Webpage, retrieved from http://www.techopedia.com/definition/23962/codebase (visited 26 May 2013)

Keim, D.A., (1997), Visual Techniques for Exploring Databases, invited tutorial Knowledge Discovery in Databases KDD'97, Newport Beach, CA, available from http://www.dbs.informatik.uni-muenchen.de/~daniel/KDD97.pdf (visited 2 Jun 2012)

Kepoğlu, V., (2010), Development of Free/Libre and Open Source Spatial Data Analysis System Fully Coupled with Geographic Information Systems, Unpublished Ph.D. thesis submitted to the Graduate School of Natural and Applied Sciences of Middle East Technical University, Ankara, Turkey, p. 74

Kern, R., Ruston, G., (1969), MAPIT, a computer program for the production of flow maps, dot maps, and graduated symbol maps, in The Cartographic Journal, 6, (2), pp. 131-137.

Kraak, M. J., (2003), Geovisualization illustrated, in ISPRS Journal of Photogrammetry and Remote Sensing, Vol. 57, Issues 5-6, April 2003, Challenges in Geospatial Analysis and Visualization, pp. 390-399

LangPop, (2013), Programming Language Popularity: Normalized Comparison, retrieved from http://langpop.com/#normalized (visited 28 Dec 2013)

Leung, Y.K., Apperley, M.D., (1994), A Review and Taxonomy of Distortion-Oriented Presentation Techniques, in ACM Trans. Computer-Human Interaction, 1(2), June 1994, pp. 126-160

Longley, P. A., Goodchild, M. F., Maguire, D. J., Rhind, D. W., (2005), Geographical information systems and science, 2nd ed., Chichester, Wiley, p.13

MapTools.org, (2013), List of projects retrieved from the free and open source geospatial portal MapTools.org, retrieved from www.maptools.org (visited 09 Jun 2013)

Minard, C. J., (1869), Tableaux Craphiques et Cartes Figuratives de M. Minard, 1845-1869, a portfolio of his work held by the Bibliotheque de'Ecole Nationale des Ponts et Chaussees (ENPC), Paris

Mockus, A., Fielding, R.T., Herbsleb, J., (2000), A case study of open source software development: the Apache server, in Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000), Limerick, Ireland, pp. 263-272

Mountain, D., (2005), Chapter 5: Visualizing, querying and summarizing individual spatio-temporal Behaviour, in Exploring Geovisualization, Elsevier, pp. 181-200

Mountain, D., Dykes, J., (2002), What i did on my vacation: Spatio-temporal log analysis with interactive graphics and morphometric surface derivatives, in Proceedings of GIS Research UK, April 2002, UK

Mountain, D., Raper, J., (2001), Modelling human spatio-temporal behaviour: A challenge for location-based services, in proceedings of Geocomputation 2001, 6th Int. Conference on Geocomputation, Sep 2001, Australia

Murtagh, F., (2000), Clustering in Massive Data Sets, Chemical Data Analysis in the Large, in Proc. Beilstein-Institut Workshop, May 2000, Bozen, Italy

O'Malley, K., (1998), Desire line maker v2 for ArcView 3.0, retrieved from http://arcscripts.esri.com/details.asp?dbid=11200 (visited 2 June 2012)

Ohloh, Ohloh the open source network, (2013), Compare Repositories, retrieved from http://www.ohloh.net/repositories/compare (visited 26 May 2013)

OSGeo, Open Source Geospatial Foundation, (2013), List of supported and incubating projects, in OSGeo web portal, retrieved from http://www.osgeo.org (visited 09 Jun 2013)

OSGeo, Open Source Geospatial Foundation, (2014a), FAQ - General, What does OGR stand for?, retrieved from http://trac.osgeo.org/gdal/wiki/FAQGeneral#WhatdoesOGRstandsfor (visited 04 Jan 2014)

OSGeo, Open Source Geospatial Foundation, (2014b), GDAL/OGR Info Sheet, retrieved from http://www.osgeo.org/gdal_ogr (visited 04 Jan 2014)

OSI, Open Source Initiative, (2013a), What is free software and is it the same as open source?, retrieved from http://opensource.org/faq#free-software (visited 19 May 2013)

OSI, Open Source Initiative, (2013b), The Open Source Definition (Annotated) v 1.9, retrieved from http://opensource.org/osd-annotated (visited 19 May 2013)

OSI, Open Source Initiative, (2013c), Open Source Licenses, Licenses by Name, retrieved from http://opensource.org/licenses/alphabetical (visited 19 May 2013)

OSRC, Open Source Resource Center, (2013), Open Source License Data, Top 20 Most Commonly Used Licenses in Open Source Projects, retrieved from http://osrc.blackducksoftware.com/data/licenses (visited 19 May 2013)

Oxford University Press, (2012), Keyword search “flow” in Oxford English Dictionary, retrieved from http://oxforddictionaries.com/definition/flow?q=flow (visited on 27 May 2012)

Paul, R., (2010), Document Foundation forks OpenOffice.org, liberates it from Oracle, Ars essay in Technica Technology Portal, retrieved from http://arstechnica.com/information-technology/2010/09/document-foundation-forks-openofficeorg-to-liberate-it-from-oracle (visited 02 Jun 2013)

Phan, D., Xiao, L., Yeh, R., Hanrahan, P. and Winograd, T., (2005), Flow Map Layout, in IEEE Symposium on Information Visualization, 2nd Edt., Pearson Education Prentice Hall, New York

Pieke, B., Krüger, A., (2007), Flow Maps - Automatic Generation and Visualization in GIS, in Proceedings of the 5th Geographic Information Days, GI-Days 2007, Probst, F., Kessler, C. (Eds.), 10-12 Sep 2007, Münster, Germany, ISBN: 978-3-936616-48-4

Pilgrim, M., (2004), Dive Into Python - Python from novice to pro, p. 343, retrieved from http://www.diveintopython.net/download/diveintopython-pdf-5.4.zip (visited 04 Jan 2014)

Prechelt, L., (2000), An Empirical Comparison of Seven Programming Languages, in Computer, Vol. 33, Issue 10, pp. 23-29

PSF, Python Software Foundation, (2013a), About Python, retrieved from http://www.python.org/about (visited 29 Dec 2013)

PSF, Python Software Foundation, (2013b), About Python, Application Domains, retrieved from http://www.python.org/about/apps (visited 29 Dec 2013)

PSF, Python Software Foundation, (2013c), History and License, History of the software, retrieved from http://docs.python.org/2/license.html (visited 29 Dec 2013)

PSF, Python Software Foundation, (2013d), Releases, retrieved from http://www.python.org/download/releases (visited 29 Dec 2013)

PSF, Python Software Foundation, (2013e), Python 2.7.6 Documentation, Installing Python Modules, Introduction, retrieved from http://docs.python.org/2/install/#introduction (visited 29 Dec 2013)

QGIS, Quantum GIS Development Team, (2010), Quantum GIS Coding and Compilation Guide Version 1.6 “Copiapo”, retrieved from http://download.osgeo.org/qgis/doc/manual/qgis-1.6.0_coding-compilation_guide_en.pdf (visited 08 Mar 2014)

QGIS, Quantum GIS, (2013a), About QGIS, in official website of QGIS project, retrieved from http://qgis.org/en/about-qgis.html (visited 16 Jun 2013)

QGIS, Quantum GIS, (2013b), Download QGIS, retrieved from http://hub.qgis.org/projects/quantum-gis/wiki/Download (visited 26 May 2013)

QGIS, Quantum GIS, (2013c), 2012 statistics for qgis.org, retrieved from http://www.qgis.org/cgi-bin/awstats.pl?urlfilter=%2Fdownloads%2F&urlfilterex=l&config=qgis&year=2012&month=all (visited 09 Jun 2013)

QGIS, Quantum GIS, (2014a), PyQGIS developer cookbook Release 2.0, retrieved from http://docs.qgis.org/2.0/pdf/QGIS-2.0-PyQGISDeveloperCookbook-en.pdf (visited 23 Feb 2014)

QGIS, Qunatum GIS, (2014b), QGIS API Documentation, 2.1.0-Master, Modules, retrieved from http://qgis.org/api/modules.html (visited 05 Jan 2014)

QGIS, Quantum GIS, (2014c), QGIS User Guide Release 2.0, retrieved from http://docs.qgis.org/2.0/pdf/QGIS-2.0-UserGuide-en.pdf (visited 10 Mar 2014)

Qt Project, (2012), Qt Versions, retrieved from http://qt-project.org/wiki/QtVersions (visited 28 Dec 2013)

Qt Project, (2013), Qt Public Roadmap: Qt 5, retrieved from http://qt-project.org/wiki/Qt_5.0 (visited 28 Dec 2013)

Rae, A., (2009), From Spatial Interaction Data to Spatial Interaction Information? Geovisualisation and Spatial Structures of Migration from the 2001 UK Census, in Computers, Environment and Urban Systems, Vol. 33, pp. 161-178

Ramsey, P., (2007), The state of open source GIS, in FOSS4G 2007 Conference, Vancouver, BC, Canada, p.6, retrieved from http://www.refractions.net/expertise/whitepapers/opensourcesurvey/survey-open-source-2007-12.pdf (visited 02 Jun 2013)

Raymond, E.S., (1999), The Cathedral and the Bazaar, O'Reilly Media

Riverbank Computing Limited, (2014a), What is PyQT, retrieved from http://www.riverbankcomputing.com/software/pyqt/intro (visited 04 Jan 2014)

Riverbank Computing Limited, (2014b), License, retrieved from http://www.riverbankcomputing.com/software/pyqt/license (visited 04 Jan 2014)

Robinson, A., (1982), Early Thematic Mapping in The History of Cartography, University of Chicago Press, Chicago

Robles, G., (2004), A Software Engineering approach to Libre Software, in Gehring, R.A., Lutterbeck, B. (eds.), Open Source Jahrbuch, Lehmanns Media, Berlin

Sarkar, M., Snibbe, S.S., Tversky, O.J., Reiss, S.P., (1993), Stretching the rubber sheet: a metaphor for viewing large layouts on small screens, in Proc. UIST'93, Atlanta, Georgia, ACM Press, pp. 81-91

Shapiro, J. S., David, A. P., (2008), Community-Based Production of Open Source Software: What Do We Know about the Developers Who Participate?, p. 50, retrieved from http://ssrn.com/abstract=1286273 (visited 02 Jun 2013)

Sherman, G., Pasotti, A., Sucameli, G., (2014), QGIS Plugin Builder v2.0.3, retrieved from http://plugins.qgis.org/plugins/pluginbuilder/ (visited 14 Mar 2014)

Stalder, F., (2003), "The Status of Objects in the Space of Flows", Dissertation, University of Toronto, p.3, retrieved from http://felix.openflows.org/html/objects_flows.pdf (visited 25 May 2014)

Steiniger S., Bocher, E., (2009), An Overview on Current Free and Open Source Desktop GIS Developments, in International Journal of Geographical Information Science, Vol. 23, Iss. 10, pp. 1345-1370

The Gephi Consortium, (2014), Features of Gephi, in offical website of Gephi, retrieved from http://gephi.github.io/features/ (visited 5 Jul 2014)

The Linux Information Project, (2006), Freeware Definiton, retrieved from http://www.linfo.org/freeware.html (visited 19 May 2013)

Thompson, W., Lavin, S., (1996), Automatic generation of animated migration maps, in Cartographica Journal, 33(2), pp. 17-29

Thornthwaite, W., (1934), Internal Migration in the United States, University of Pennsylvania Press, Philadelphia, USA in Computational Statistics & Data Analysis, Vol. 43/4, pp. 581-603

TIOBE, (2014), TIOBE Index for January 2014: Very Long Term History, Programming Language Hall of Fame, retrieved from http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (visited 05 Jan 2014)

Tobler, W., (1976), Spatial Interaction patterns, in Journal of Environmental Systems, Vol. VI, pp. 271-301

Tobler, W., (1981), A Model of Geographical Movement, in Geographical Analysis, Vol. 13, pp. 1-20

Tobler, W., (1985), Derivation of a Spatially Continuous Transportation Model, in Transportation Research, Vol. 19-A, pp. 169-172

Tobler, W., (1987), Experiments in Migration Mapping by Computer, in The American Cartographer, Vol. 14, pp. 155-163

Tobler, W., (2003), "Movement Mapping", Santa Barbara: Center for Spatially Integrated Social Science, University of California, unpublished paper retrieved from http://csiss.ncgia.ucsb.edu/clearinghouse/FlowMapper/MovementMapping.pdf (visited 14 May 2011)

Tufte, E., (2007), The Visual Display of Geographic Information, 2nd. Ed., 5th Printing, Aug 2007, Cheshire, Connecticut, Graphic Press, p. 25, 41

TurkStat, Turkish Statistical Institute, (2013), Address Based Population Registration System Results: Migration Statistics, Internal Migration, Migration across Provinces, retrieved from http://tuikapp.tuik.gov.tr/adnksdagitapp/adnks.zul?kod=4&dil=2 (visited 18 May 2013)

van Eck Ritsema, J. R., de Jong, T., (1999), Accessibility analysis and spatial competition effects in the context of GIS-supported service location planning, in Computers, Environment and Urban Systems, Vol. 23, pp. 75-89 van Rossum, G., (2006), Python Enhancement Proposals, PEP 3000 - Python 3000, retrieved from http://www.python.org/dev/peps/pep-3000 (visited 14 Dec 2013)

Vries, H., Oshri, I., (2008), Standards Battles in Open Source Software: The Case of Firefox, Palgrave Macmillan, UK.

Ward, M.O, (2002), A taxonomy of glyph placement strategies for multidimensional data visualization, in Information Visualization, Vol. 1, Issue 3/4, Dec 2002, pp. 194-210

Warmerdam, F., (2008), The Geospatial Data Abstraction Library, in Open Source Approaches in Spatial Data Handling, ed. by G. Brent Hall and Michael G. Leahy, Springer Berlin Heidelberg, pp. 87-104

Warsta, J., Abrahamsson, P., (2003), Is open source software development essentially an agile method? In 3rd Workshop on Open Source Software Engineering, Portland, Oregon, USA

Wegman, E.J., Luo, Q., (1996), High Dimensional Clustering Using Parallel Coordinates and the Grand Tour, in Computing Science and Statistics, 28, July 1996, pp. 352-360

Wikipedia, (2014a), Qt (Software), retrieved from http://en.wikipedia.org/wiki/Qt_%28software%29 (visited 05 Jan 2014)

Wikipedia, (2014b), Google Analytics, retrieved from http://en.wikipedia.org/wiki/Google_analytics (visited 23 Apr 2014)

Wong, P.C., Bergeron, R.D., (1997), 30 Years of Multidimensional Multivariate Visualization, Scientific Visualization: Overviews, Methodologies & Techniques, Washington, IEEE, pp. 3-33

Wood, J., Dykes, J., Slingsby, A., Radburn, R., (2009), Flow trees for exploring spatial trajectories, in Proceedings of the GIS Research UK 17th Annual Conference, Durham, UK, pp. 229-234

Xiao, N., Chun, Y., (2009), Visualizing migration flows using kriskograms, in Cartography and Geographic Information Science, 36(2), pp. 183-191

Zelle, J. M., (1999), Python as a First Language, in Proceedings of 13th Annual Midwest Computer Conference, March 1999, Lisle, Illinois, U.S., retrieved from http://mcsp.wartburg.edu/zelle/python/python-first.html (visited 14 Dec 2013)

APPENDIX A

STRUCTURE OF FLOWMAPPER AND INTERACTION WITH QGIS

In Figure A.1, a conceptual diagram is given in order to portray inner structure of the plugin and interaction of plugin with QGIS.

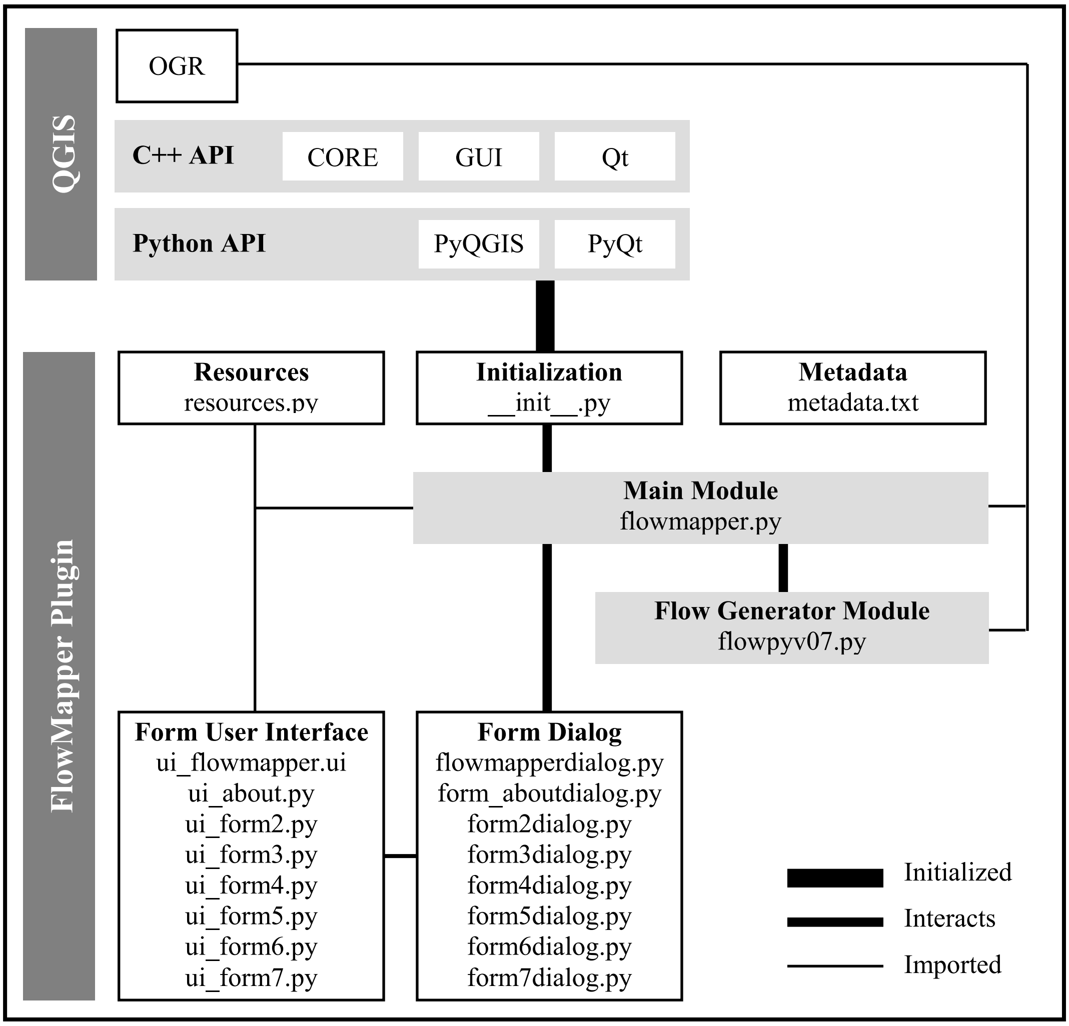


Figure A.1 Conceptual Diagram of FlowMapper Plugin

APPENDIX B

SOURCE CODE OF FLOW GENERATOR MODULE

In order to create flow lines and flow nodes in shapefile format and to automatically perform attribute calculations (e.g. flow magnitude, origin – destination coordinate pairs, node names etc.) “shapefilemaker” function must be called from “flowpyv07.py” file with the parameters defined in the main module based on user preferences (Table 5.8, Lines 160 – 164). This requires importing “flowpyv07.py” at the beginning of “flowmapper.py” (Table 5.8, Line 24). In Table B.1, full source code of “flowpyv07.py” is given.

Table B.1. Source Code of Flow Generator Module: “flowpyv07.py”

Source Code of flowpyv07.py in Python

```python
# flowpyv07.py was initially written by A. Glenneon (2009) in Python.
# flowpyv07.py is further developed by C. Gulluoglu between 2011 – 2013
# as the flow generator module for QGIS FlowMapper plugin. Besides, its
# capabilities are extended to automatically generate several attribute
# fields (e.g. origin-destination names, coordinates, length etc.) and to
# generate flow nodes in addition to flow lines.

# This file takes a square interaction matrix and corresponding set of
# coordinate pairs to generate flow lines and flow nodes. While interaction
# matrix and coordinate pairs files are mandatory inputs, node names file
# is an optional input.

def shapefilemaker(typeofcalculation,CreateShpNodes,IncludeNodeNames, \
fulldirectorystring,outputfilename,outputfilenamenodes,fulldirODinput, \
fulldirPTinput,fulldirPTNamesinput,combotext):

    import os, sys, ogr # ogr is required to handle shapefile format
    import math # math is required to make great circle distance calc.

    odmatrixfilename = fulldirODinput # location of interaction matrix
    nodefilename = fulldirPTinput # loc. of file storing node coordinates
    nodenamesfilename = fulldirPTNamesinput # loc. of node names file

    # flow calculation type: two way = 1, gross = 2, net = 3
    try:
        # open input text files read only
        odmatrix = open(odmatrixfilename, 'r')
        nodes = open(nodefilename, 'r')
```

Table B.1. Source Code of Flow Generator Module: “flowpyv07.py” (cont.)

Source Code of flowpyv07.py in Python

```python
        names = open(nodenamesfilename, 'r')
    except IOError, e:
        print 'file open error:', e

    numberofnodes = 0
    for line in nodes: # get node count, each line represents 1 node
        numberofnodes += 1
    nodes.close

    # interaction matrix is read below, result is myodmatrix[rows][columns]
    rows = 0
    myodmatrix = [] # create list to store flow magnitudes between nodes
    for icounter in xrange(numberofnodes):
        myodmatrix.append([])
        for jcounter in xrange(numberofnodes):
            myodmatrix[icounter].append(icounter+jcounter)
    for eachLine in odmatrix:
        separatestrings = eachLine.split()
        columns = 0
        while columns < numberofnodes:
            onevalue = float(separatestrings[columns])
            myodmatrix[rows][columns] = onevalue
            columns += 1
        rows += 1
    odmatrix.close

    # coordinates are read below, result is mypoints[ptrows][ptcolumns]
    ptrows = 0
    mypoints = [] # create list to store input node coordinates
    for kcounter in xrange(numberofnodes):
        mypoints.append([])
        for lcounter in xrange(2):
            mypoints[kcounter].append(kcounter+lcounter)
    nodesagain = open(nodefilename,'r')
    for eachLine2 in nodesagain:
        separatestrings2 = eachLine2.split()
        ptcolumns = 0
        while ptcolumns < 2:
            onevalue2 = float(separatestrings2[ptcolumns])
            mypoints[ptrows][ptcolumns] = onevalue2
            ptcolumns += 1
        ptrows += 1
    nodesagain.close

    # node names are read below if supplied by the user
    mypointnames = [] # create list to store input node names
    for eachLine3 in names:
        mypointnames.append(eachLine3.split())
    names.close

    # START: create shapefile to store flow NODES and attributes
    if CreateShpNodes == 1:
        driver2 = ogr.GetDriverByName('ESRI Shapefile') # get ogr driver
        if os.path.exists(outputfilenamenodes):
            driver2.DeleteDataSource(outputfilenamenodes)
        # create data source and layer
        ds2 = driver2.CreateDataSource(outputfilenamenodes)
        layer2 = ds2.CreateLayer('node', geom_type=ogr.wkbPoint)
        # define attribute field name and set data type
        fieldDefn20 = ogr.FieldDefn('name', ogr.OFTString)
        fieldDefn21 = ogr.FieldDefn('incoming', ogr.OFTReal)
        fieldDefn22 = ogr.FieldDefn('outgoing', ogr.OFTReal)
        fieldDefn23 = ogr.FieldDefn('gross', ogr.OFTReal)
        fieldDefn24 = ogr.FieldDefn('net', ogr.OFTReal)
        fieldDefn25 = ogr.FieldDefn('in/out', ogr.OFTReal)
        fieldDefn26 = ogr.FieldDefn('out/in', ogr.OFTReal)
        fieldDefn27 = ogr.FieldDefn('indicator', ogr.OFTReal)
```

Table B.1. Source Code of Flow Generator Module: “flowpyv07.py” (cont.)

Source Code of flowpyv07.py in Python

```python
        # create attribute fields
        layer2.CreateField(fieldDefn20)
        layer2.CreateField(fieldDefn21)
        layer2.CreateField(fieldDefn22)
        layer2.CreateField(fieldDefn23)
        layer2.CreateField(fieldDefn24)
        layer2.CreateField(fieldDefn25)
        layer2.CreateField(fieldDefn26)
        layer2.CreateField(fieldDefn27)
        # FINISH: create shapefile to store flow NODES and attributes

        # START: insert nodes as point features into shapefile
        counterA = 0
        sumincoming = 0
        while counterA < numberofnodes:
            linester2 = ogr.Geometry(ogr.wkbPoint)
            linester2.SetPoint \
            (0, mypoints[counterA][0],mypoints[counterA][1])
            featureDefn2 = layer2.GetLayerDefn()
            feature2 = ogr.Feature(featureDefn2)
            feature2.SetGeometry(linester2)
            # set node names
            if IncludeNodeNames == 1:
                name = str(mypointnames[counterA])
                name = name[2:-2]
                feature2.SetField('name',name)
            elif IncludeNodeNames == 0:
                name = "Node_"+str(counterA+1)
                feature2.SetField('name',name)
            # sum magnitude of all outgoing flows from that node
            myodmatrixA=myodmatrix[counterA]
            sumoutgoing=sum(myodmatrixA)
            feature2.SetField('outgoing',sumoutgoing)
            # sum magnitude of all incoming flows to that node
            counterB = 0
            while counterB < numberofnodes:
                myodmatrixB = myodmatrix[counterB][counterA]
                sumincoming=sumincoming+int(myodmatrixB)
                counterB += 1
            feature2.SetField('incoming',sumincoming)
            # sum magnitude of gross for that node
            gross = sumincoming + sumoutgoing
            feature2.SetField('gross',gross)
            # calculate magnitude of net flows for that node
            net = sumincoming - sumoutgoing
            net_abs = abs(net)
            feature2.SetField('net',net_abs)
            # calculate ratio of all incoming flows to all outgoing flows
            in_DIV_out = sumincoming / sumoutgoing
            feature2.SetField('in/out',in_DIV_out)
            # calculate ratio of all outgoing flows to all incoming flows
            out_DIV_in = sumoutgoing / sumincoming
            feature2.SetField('out/in',out_DIV_in)
            # set indicator field: +1 gains flow, -1 looses flow, 0 in=out
            if net < 0:
                indicator = -1
            elif net > 0:
                indicator = 1
            elif net == 0:
                indicator = 0
            feature2.SetField('indicator',indicator)
            sumincoming = 0
            layer2.CreateFeature(feature2)
            counterA += 1
        linester2.Destroy() # destroy geometry
        feature2.Destroy() # destroy feature
ds2.Destroy() # close data source
```

Table B.1. Source Code of Flow Generator Module: “flowpyv07.py” (cont.)

Source Code of flowpyv07.py in Python

```python
        # FINISH: insert nodes as point features into shapefile

    # START: create shapefile to store flow LINES and attributes
    driver = ogr.GetDriverByName('ESRI Shapefile') # get ogr driver
    if os.path.exists(outputfilename):
        driver.DeleteDataSource(outputfilename)
    # create data source and layer
    ds = driver.CreateDataSource(outputfilename)
    if ds is None:
        print 'Could not create file'
        sys.exit(1)
    layer = ds.CreateLayer('flow', geom_type=ogr.wkbLineString)
    # define attribute field name and set data type
    fieldDefn = ogr.FieldDefn('magnitude', ogr.OFTReal)
    # define attribute field name and set data type
    fieldDefn2 = ogr.FieldDefn('length_km', ogr.OFTReal)
    fieldDefn3 = ogr.FieldDefn('coord_x1', ogr.OFTReal)
    fieldDefn4 = ogr.FieldDefn('coord_y1', ogr.OFTReal)
    fieldDefn5 = ogr.FieldDefn('coord_x2', ogr.OFTReal)
    fieldDefn6 = ogr.FieldDefn('coord_y2', ogr.OFTReal)
    fieldDefn7 = ogr.FieldDefn('name_x1y1', ogr.OFTString)
    fieldDefn8 = ogr.FieldDefn('name_x2y2', ogr.OFTString)
    # define attribute fields
    layer.CreateField(fieldDefn)
    layer.CreateField(fieldDefn2)
    layer.CreateField(fieldDefn3)
    layer.CreateField(fieldDefn4)
    layer.CreateField(fieldDefn5)
    layer.CreateField(fieldDefn6)
    layer.CreateField(fieldDefn7)
    layer.CreateField(fieldDefn8)
    # FINISH: create shapefile to store flow LINES and attributes

    # START: insert TWO WAY flows as line features into shapefile
    if typeofcalculation == 1:
        counter1 = 0
        counter2 = 0
        while counter2 < numberofnodes:
            while counter1 < numberofnodes:
                linester = ogr.Geometry(ogr.wkbLineString)
                linester.AddPoint \
                (mypoints[counter2][0],mypoints[counter2][1])
                linester.AddPoint \
                (mypoints[counter1][0],mypoints[counter1][1])
                featureDefn = layer.GetLayerDefn()
                feature = ogr.Feature(featureDefn)
                feature.SetGeometry(linester)
                # set magnitude of flow line
                feature.SetField \
                ('magnitude',myodmatrix[counter2][counter1])
                # set origin and destination node coordinates
                x1 = mypoints[counter2][0]
                y1 = mypoints[counter2][1]
                x2 = mypoints[counter1][0]
                y2 = mypoints[counter1][1]
                feature.SetField('coord_x1',x1)
                feature.SetField('coord_y1',y1)
                feature.SetField('coord_x2',x2)
                feature.SetField('coord_y2',y2)
                # set origin and destination node names
                if IncludeNodeNames == 1:
                    Namex1y1 = str(mypointnames[counter2])
                    Namex1y1 = Namex1y1[2:-2]
                    feature.SetField('name_x1y1',Namex1y1)
                    Namex2y2 = str(mypointnames[counter1])
                    Namex2y2 = Namex2y2[2:-2]
                    feature.SetField('name_x2y2',Namex2y2)
```

Table B.1. Source Code of Flow Generator Module: “flowpyv07.py” (cont.)

Source Code of flowpyv07.py in Python

```
                elif IncludeNodeNames == 0:
                    Namex1y1 = "Node_"+str(counter2+1)
                    feature.SetField('name_x1y1',Namex1y1)
                    Namex2y2 = "Node_"+str(counter1+1)
                    feature.SetField('name_x2y2',Namex2y2)
                # calculate length of flow line
                if combotext == "Cartesian":
                    length_m = (((x1-x2)**2)+((y1-y2)**2))**0.5
                    feature.SetField('length_km',length_m / (1000))
                else: # spherical distance (based on wgs84 ellipsoid)
                    lon1 = float(x1)
                    lat1 = float(y1)
                    lon2 = float(x2)
                    lat2 = float(y2)
                    length_km = math.atan2(math.sqrt((math.cos((math.atan \
                    (1.0)/45.0)*lat2)*math.sin((math.atan(1.0)/45.0)*lon2 \
                    -(math.atan(1.0)/45.0)*lon1))**2+(math.cos((math.atan \
                    (1.0)/45.0)*lat1)*math.sin((math.atan(1.0)/45.0)*lat2 \
                    math.sin((math.atan(1.0)/45.0)*lat1)*math.cos \
                    ((math.atan(1.0)/45.0)*lat2)*math.cos((math.atan(1.0) \
                    /45.0)*lon2-(math.atan(1.0)/45.0)*lon1))**2),
                    math.sin((math.atan(1.0)/45.0)*lat1)*math.sin ( \
                    (math.atan(1.0)/45.0)*lat2)+math.cos((math.atan (1.0) \
                    /45.0)*lat1)*math.cos((math.atan(1.0)/45.0)*lat2) \
                    *math.cos((math.atan(1.0)/45.0)*lon2-(math.atan(1.0) \
                    /45.0)*lon1)) * 6367.9375
                    feature.SetField('length_km',length_km)
                layer.CreateFeature(feature)
                counter1 = counter1 + 1
            counter2 = counter2 + 1
            counter1 = 0
            # FINISH: insert TWO WAY flows as line features into shapefile


    # START: insert GROSS flows as line features into shapefile
    if typeofcalculation == 2:
        g = 0
        h = 0
        while g < numberofnodes:
            while h < numberofnodes:
                if (g <= h):
                    linester = ogr.Geometry(ogr.wkbLineString)
                    linester.AddPoint(mypoints[g][0], mypoints[g][1])
                    linester.AddPoint(mypoints[h][0], mypoints[h][1])
                    if h==g:
                        grossmagnitude = \
                        (myodmatrix[g][h] + myodmatrix[h][g])/2
                    else:
                        grossmagnitude = \
                        (myodmatrix[g][h] + myodmatrix[h][g])
                    featureDefn = layer.GetLayerDefn()
                    feature = ogr.Feature(featureDefn)
                    feature.SetGeometry(linester)
                    # set gross magnitude of flow line
                    feature.SetField('magnitude',grossmagnitude)
                    # set origin and destination node coordinates
                    x1 = mypoints[h][0]
                    y1 = mypoints[h][1]
                    x2 = mypoints[g][0]
                    y2 = mypoints[g][1]
                    feature.SetField('coord_x1',x1)
                    feature.SetField('coord_y1',y1)
                    feature.SetField('coord_x2',x2)
                    feature.SetField('coord_y2',y2)
                    # set origin and destination node names
                    if IncludeNodeNames == 1:
                        Namex1y1 = str(mypointnames[h])
```

Table B.1. Source Code of Flow Generator Module: “flowpyv07.py” (cont.)

Source Code of flowpyv07.py in Python

```
                        Namex1y1 = Namex1y1[2:-2]
                        feature.SetField('name_x1y1',Namex1y1)
                        Namex2y2 = str(mypointnames[g])
                        Namex2y2 = Namex2y2[2:-2]
                        feature.SetField('name_x2y2',Namex2y2)
                    elif IncludeNodeNames == 0:
                        Namex1y1 = "Node_"+str(h+1)
                        feature.SetField('name_x1y1',Namex1y1)
                        Namex2y2 = "Node_"+str(g+1)
                        feature.SetField('name_x2y2',Namex2y2)
                    # calculate length of flow line
                    if combotext == "Cartesian":
                        length_m = (((x1-x2)**2)+((y1-y2)**2))**0.5
                        feature.SetField('length_km',length_m / (1000))
                    else: # spherical distance (based on wgs84 ellipsoid)
                        lon1 = float(x1)
                        lat1 = float(y1)
                        lon2 = float(x2)
                        lat2 = float(y2)
                        length_km = math.atan2(math.sqrt((math.cos(( \
                        math.atan(1.0)/45.0)*lat2)*math.sin((math.atan \
                        (1.0)/45.0)*lon2-(math.atan(1.0)/45.0)*lon1))**2+ \
                        (math.cos((math.atan(1.0)/45.0)*lat1)*math.sin(( \
                        math.atan(1.0)/45.0)*lat2)-math.sin((math.atan \
                        (1.0)/45.0)*lat1)*math.cos((math.atan(1.0) /45.0) \
                        *lat2)*math.cos((math.atan(1.0)/45.0)*lon2-( \
                        math.atan(1.0)/45.0)*lon1))**2),math.sin(( \
                        math.atan (1.0)/45.0)*lat1)*math.sin((math.atan \
                        (1.0)/45.0)*lat2)+math.cos((math.atan(1.0)/45.0) \
                        *lat1)*math.cos((math.atan(1.0)/45.0)*lat2)* \
                        math.cos((math.atan(1.0)/45.0)*lon2-(math.atan \
                        (1.0)/45.0)*lon1)) * 6367.9375
                        feature.SetField('length_km',length_km)
                    layer.CreateFeature(feature)
                h += 1
            h = 0
            g += 1
            # FINISH: insert GROSS flows as line features into shapefile

    # START: insert NET flows as line features into shapefile
    if typeofcalculation == 3:
        g = 0
        h = 0
        while g < numberofnodes:
            while h < numberofnodes:
                if (g <= h):
                    if h==g:
                        netmagnitude=(myodmatrix[g][h]+myodmatrix[h][g])/2
                    else:
                        netmagnitude=(myodmatrix[g][h]-myodmatrix[h][g])
                    if netmagnitude < 0: # if net magnitude has minus value
                        linester = ogr.Geometry(ogr.wkbLineString)
                        linester.AddPoint(mypoints[h][0], mypoints[h][1])
                        linester.AddPoint(mypoints[g][0], mypoints[g][1])
                        featureDefn = layer.GetLayerDefn()
                        feature = ogr.Feature(featureDefn)
                        feature.SetGeometry(linester)
                        # set net magnitude of flow line (absolute value)
                        feature.SetField('magnitude',netmagnitude * (-1))
                        # set origin and destination node coordinates
                        x1 = mypoints[h][0]
                        y1 = mypoints[h][1]
                        x2 = mypoints[g][0]
                        y2 = mypoints[g][1]
                        feature.SetField('coord_x1',x1)
                        feature.SetField('coord_y1',y1)
                        feature.SetField('coord_x2',x2)
```

Table B.1. Source Code of Flow Generator Module: “flowpyv07.py” (cont.)

Source Code of flowpyv07.py in Python

```
                        feature.SetField('coord_y2',y2)
                        # set origin and destination node names
                        if IncludeNodeNames == 1:
                            Namex1y1 = str(mypointnames[h])
                            Namex1y1 = Namex1y1[2:-2]
                            feature.SetField('name_x1y1',Namex1y1)
                            Namex2y2 = str(mypointnames[g])
                            Namex2y2 = Namex2y2[2:-2]
                            feature.SetField('name_x2y2',Namex2y2)
                        elif IncludeNodeNames == 0:
                            Namex1y1 = "Node_"+str(h+1)
                            feature.SetField('name_x1y1',Namex1y1)
                            Namex2y2 = "Node_"+str(g+1)
                            feature.SetField('name_x2y2',Namex2y2)
                        # calculate length of flow line
                        if combotext == "Cartesian":
                            length_m = (((x1-x2)**2)+((y1-y2)**2))**0.5
                            feature.SetField('length_km',length_m / (1000))
                        else : # spherical dist. (based on wgs84 ellipsoid)
                            lon1 = float(x1)
                            lat1 = float(y1)
                            lon2 = float(x2)
                            lat2 = float(y2)
                            length_km = math.atan2(math.sqrt((math.cos \
                            ((math.atan(1.0)/45.0)*lat2)*math.sin(( \
                            math.atan(1.0)/45.0)*lon2-(math.atan(1.0) \
                            /45.0)*lon1))**2+(math.cos((math.atan (1.0) \
                            /45.0)*lat1)*math.sin((math.atan(1.0)/45.0)* \
                            lat2)-math.sin((math.atan(1.0)/45.0)*lat1)* \
                            math.cos((math.atan(1.0)/45.0)*lat2)*math.cos \
                            ((math.atan(1.0)/45.0)*lon2-(math.atan(1.0) \
                            /45.0)*lon1))**2),math.sin((math.atan(1.0) \
                            /45.0)*lat1)*math.sin((math.atan(1.0)/45.0) * \
                            lat2)+math.cos((math.atan(1.0)/45.0)* lat1)* \
                            math.cos((math.atan(1.0)/45.0)*lat2)* \
                            math.cos((math.atan(1.0)/45.0)*lon2-(math.atan
                            (1.0)/45.0)*lon1)) * 6367.9375
                            feature.SetField('length_km',length_km)
                        layer.CreateFeature(feature)
                    else: # if net magnitude has positive value
                        linester = ogr.Geometry(ogr.wkbLineString)
                        linester.AddPoint(mypoints[g][0], mypoints[g][1])
                        linester.AddPoint(mypoints[h][0], mypoints[h][1])
                        featureDefn = layer.GetLayerDefn()
                        feature = ogr.Feature(featureDefn)
                        feature.SetGeometry(linester)
                        # set net magnitude of flow line
                        feature.SetField('magnitude',netmagnitude)
                        # set origin and destination node coordinates
                        x1 = mypoints[g][0]
                        y1 = mypoints[g][1]
                        x2 = mypoints[h][0]
                        y2 = mypoints[h][1]
                        feature.SetField('coord_x1',x1)
                        feature.SetField('coord_y1',y1)
                        feature.SetField('coord_x2',x2)
                        feature.SetField('coord_y2',y2)
                        # set origin and destination node names
                        if IncludeNodeNames == 1:
                            Namex1y1 = str(mypointnames[g])
                            Namex1y1 = Namex1y1[2:-2]
                            feature.SetField('name_x1y1',Namex1y1)
                            Namex2y2 = str(mypointnames[h])
                            Namex2y2 = Namex2y2[2:-2]
                            feature.SetField('name_x2y2',Namex2y2)
                        elif IncludeNodeNames == 0:
                            Namex1y1 = "Node_"+str(g+1)
```

Table B.1. Source Code of Flow Generator Module: “flowpyv07.py” (cont.)

Source Code of flowpyv07.py in Python

```
                            feature.SetField('name_x1y1',Namex1y1)
                            Namex2y2 = "Node_"+str(h+1)
                            feature.SetField('name_x2y2',Namex2y2)
                        # calculate length of flow line
                        if combotext == "Cartesian":
                            length_m = (((x1-x2)**2)+((y1-y2)**2))**0.5
                            feature.SetField('length_km',length_m / (1000))
                        else: # spherical dist. (based on wgs84 ellipsoid)
                            lon1 = float(x1)
                            lat1 = float(y1)
                            lon2 = float(x2)
                            lat2 = float(y2)
                            length_km = math.atan2(math.sqrt((math.cos \
                            ((math.atan(1.0)/45.0)*lat2)*math.sin(( \
                            math.atan(1.0)/45.0)*lon2-(math.atan(1.0) \
                            /45.0)*lon1))**2+(math.cos((math.atan (1.0) \
                            /45.0)*lat1)*math.sin((math.atan(1.0)/45.0)* \
                            lat2)-math.sin((math.atan(1.0)/45.0)*lat1)* \
                            math.cos((math.atan(1.0)/45.0)*lat2)*math.cos \
                            ((math.atan(1.0)/45.0)*lon2-(math.atan(1.0) \
                            /45.0)*lon1))**2),math.sin((math.atan(1.0) \
                            /45.0)*lat1)*math.sin((math.atan(1.0)/45.0) * \
                            lat2)+math.cos((math.atan(1.0)/45.0)* lat1)* \
                            math.cos((math.atan(1.0)/45.0)*lat2)* \
                            math.cos((math.atan(1.0)/45.0)*lon2-(math.atan
                            (1.0)/45.0)*lon1)) * 6367.9375
                            feature.SetField('length_km',length_km)
                        layer.CreateFeature(feature)
                h += 1
            h = 0
            g += 1
            # FINISH: insert NET flows as line features into shapefile

    linester.Destroy() # destroy geometry
    feature.Destroy() # destroy feature
    ds.Destroy() # close data source
```

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name:	Güllüoğlu, Naim Cem
Nationality:	Turkish (T.C.)
Date and Place of Birth:	18 July 1980, Antalya
Marital Status:	Married
Phone:	+90 532 550 1150
Email:	cempro@gmail.com , cemgulluoglu@gmail.com

EDUCATION

Degree	Institution	Graduation
M.Sci.	METU, Geodetic and Geographic Information Technologies Department	2005
B.Sci.	Gazi University, City and Regional Planning Department	2002
High School	Antalya High School	1998

WORK EXPERIENCE

Year	Place	Enrollment
2013 – Current	Ministry of Urbanism & Environment	Urban Planner
2011 – 2013	Golder Associates Turkey	GIS Expert
2008 – 2011	Encon Environmental Consulting	GIS Expert
2007 – 2008	Sampaş Inc.	GIS Expert
2006 – 2006	Kutluay Planning Office	RS Expert

LANGUAGES

Turkish (Native), English (Advanced)

PUBLICATIONS

Aksoy, A., Akyıldız, K., Berke, M., Büke, A., Çalışkan, M., Çeşmeci, H., Dıvrak, B., Duran, M., Göcek, Ç., Güllüoğlu, C., Gültekin, M., Sürücü, B., 2012, Büyük Menderes Havza Atlası, S Basım San., İstanbul, Türkiye, ISBN 978-605-62927-1-2 (in Turkish)

Ok, A.Ö., Güllüoğlu, C., 2008, Orthorectification of Basic-Level Quickbird Panchromatic Imagery with Different Sensor Models, 2nd Remote Sensing and Geographic Information System Symposium, 13 – 15 Oct. 2008, Kayseri, Turkey (in Turkish)