SCORE Contest project

Team members:

Adis Mustedanagić	[adis.mustedanagic@fer.hr]
Dominik Pavlović	[dominik.pavlovic@fer.hr]
Martin Vrkljan	[martin.vrkljan@fer.hr]
Pavel Chen	[pcn10003@student.mdh.se]
Ramesh Nagilla	[rna10002@student.mdh.se]
Zhixiang Gao	[zgo10001@student.mdh.se]

January, 2011
Västerås, Zagreb

Executive summary

Schematic maps are widely used today for displaying topographic information where exact relations between important locations are not important, the most common example of these maps are metro maps. Schematic maps however, are still mostly drawn by hand with the largest amount of work being done by designers. Unlike most other maps, schematic maps do not preserve topographic relations, but rather relative relations between important landmarks, such as stations and lines between them. Today, there are few commercially available tools that would provide an easier overall process of creating a schematic map.

The Schematizing Maps project aims to provide a user friendly tool with emphasis on user interaction which will help the designer of a map overcome the hardest part of map creation – positioning lines and stations on the map in a schematized way.

The project was developed within the Distributed Software Development course [2] held in parallel at Mälardalen University [3] in Västerås, Sweden, and Faculty of Electrical Engineering and Computing [4], University of Zagreb, Croatia. The goal of the course is to provide the experience and insights into difficulties and problem solving techniques which arise when developing in a distributed environment. The Schematizing maps team consists of six team members, with three team members stationed in Västerås, and the other three in Zagreb. The project was developed over the time span of 18 weeks, from September 2010 to January 2011. During this time, the team had to quickly adapt to a new, distributed environment and come up with an efficient way to communicate and develop software. We have soon settled for personal communication using mailing lists and Skype, and our software development method was Feature Driven Development approach, which is based on product iterations with adding new features in every following iteration and revising already existing features.

The technologies used for developing the software were Java programming language and a small number of supporting libraries (Swing [12], JUNG2 [8]). The goal was to create a standalone desktop application which can be used on any computer or any operating system capable of running Java. Graphical user interface is designed with aims to be user-friendly, by implementing controls that are intuitive and simple to use.

1 Introduction

The use of schematic maps is common nowadays. Figure 1 shows a schematic map of Stockholm metro lines (partly), which was generated by our application. The schematic maps have benefits such as readability, flexibility in extending them, and applicability in various areas making it essential for people to use schematic maps instead of other maps which represent drawn and connected objects. Transportation is one of the areas where schematic maps are commonly used to find out an optimal route to take or how various places are located geographically. Instead of using regular geographic maps there exist simplified visual representations of transportation networks which are suitable for most of public transportation systems. On these maps, important objects such as stations are placed with correct relations; however they are not geographically precise which makes them simpler and easy to read. This project has a goal to develop an application that can help generating an accurately arranged map of a transportation network where relative geometrical locations from an original map are preserved.

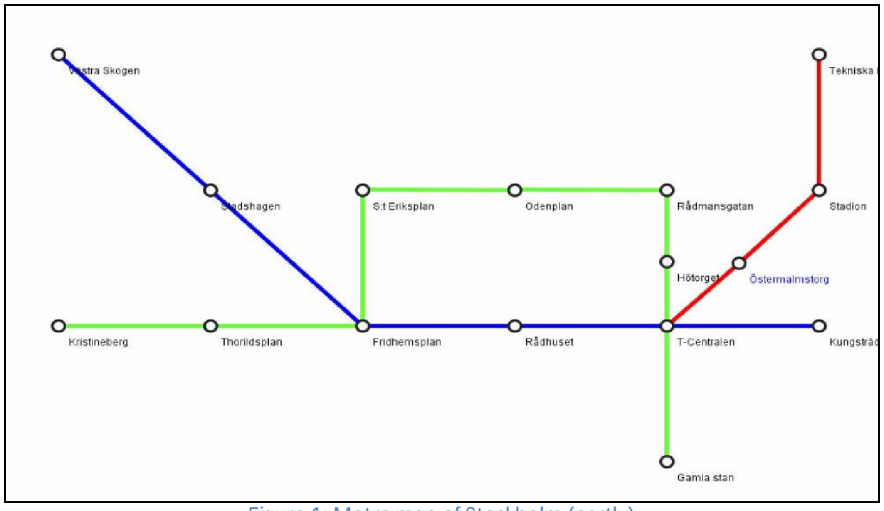

Figure 1: Metro map of Stockholm (partly)

The complexity of drawing a schematic map grows with the size of the area that needs to be covered with the schematic maps. A typical metro map consists of a fairly large number of vertices (stations) and edges (metro lines). Schematizing a map as such is a complex and a time consuming job.

The main problem the project aims to solve is the problem of aligning and placing stations and edges in a way that it is easier to read and thus reduce time and effort a designer has to spend during this crucial part of a map creation. With software giving at least a partially desirable solution, the rest of the process should require far less time than doing the entire job manually. Although there have already been several attempts to solve this problem [9] [11], they are mostly based on solutions built around mixed integer programming, and attempting to solve the well know 3-SAT problem [5] (proof of problem equivalency and details will be omitted in this document). Solutions such as these are more advanced than the solution delivered in this project; however, they require advanced knowledge of mathematics, logic and computer algorithms in order to understand the solution. Also, such solutions do not offer any interaction from the user other than defining hard and soft algorithm constraints, which, for a less technologically-experienced user, might be a big obstacle.

The solution given in this project is built with user interaction in mind first. The user is active during all stages of map schematization, and is also allowed to influence the outcome of the algorithm as well as the final map solution.

This project involved six students from Croatia and Sweden, and this arrangement was setup within Distributed Software Development (DSD) [2] course according to preferences and qualifications of students. In this document we will describe summary report of this project.

This document is organized as follows. Section 2 (Scope of the project and main challenges) describes the scope of this project, students involved along with their experiences and challenges that the team has faced or concerns that have arisen prior to beginning of the project. Section 3 (Development process) explains the development process methodology we have chosen and why. It is followed by Section 4(Project plan and performance) which talks about resources allocation, important milestones with comparison to real outcomes, and communication approaches. Sections 5, 6 and 7 are related to Requirements Specifications, Architectural Design and Implementation of

the final product. Section 8 (Verification and integration) closes the development topics and describes the test strategies. Section 9 concludes this report with outcomes, lessons learned and section 10 is the summary of this report.

2 Scope of the project and main challenges

With the project being developed as a part of DSD course as well as SCORE competition, the goal was to follow the project proposal [14] presented by Michal Young, who is also our project customer.

The final objectives were to overcome challenges of working in a distributed environment, adjust to cultural differences caused by different backgrounds of team members and to develop an application that will satisfy the problem presented to the team.

The biggest challenge was to create a good working atmosphere for team members, to keep the communication at high level at all times and to make sure that there are no misunderstandings between the team members in Sweden and Croatia. This challenge was tackled by defining a way of communicating with the team members, who included mailing lists for important information, weekly Skype meetings, and weekly to-do lists for team members as well as a large number of informal meetings held using Skype at any given time.

Another big challenge and the hardest decision making went into handling the algorithm implementation. As already mentioned in the previous chapter, every other solution available to this problem is based on extensive knowledge of mathematical logic, graph theory and solving of NP-complete problems. This presented a rather large obstacle, considering that the amount of knowledge necessary in these areas is not trivial and most of the team members lacked the background knowledge for these specific areas. A common solution (such as given in Dr. Nöllenburg's thesis [9] which was used as a starting point) would reduce the problem of schematizing a map to a 3-SAT problem, and then use a method called MIP (mixed integer programming) with an implementation of a LP-solver (linear programming solver) and a set of soft and hard constraints to generate a map that satisfies the constraints in the best way possible. These solutions however, don't allow for any user interaction, other than defining constraints, which we have considered to be too advanced for users not versed in the mathematical areas mentioned.

Because of this, the team has decided to take a different approach to the problem and develop a different algorithm which will be usable even for users who do not understand the problems of schematic map generation. The solution developed is not as powerful as solutions based on LP-solvers but it has the major advantage of being interactive, and allowing the user to intervene any time during the process of map generation as well as giving the user complete control over the final map layout.

Although all team members have a good base in computer science, none of us have been presented with a problem similar to this one before, so one of the challenges was also to study the problem deeper and understand how the possible solutions might arise. Before implementing the solution, it was imperative to achieve a good and a modular base for programming which would allow us to concentrate more on problem solving, rather than programming every bit needed for the software. Such a base was found in the JUNG2 library [8] which provided us with a great base for

project development. JUNG2 already provides libraries for handling the graph logic, including vertices, edges and their positions, as well as means of passing that information to Swing library for visual representation. Having a library that already handles the low level modeling and information allowed us to concentrate more on algorithm implementation, GUI and visual design as well as interactivity to improve user experience.

Finally, we kept the communication lines to our supervisors and customer open at all times so that we could quickly adapt and implement any features they requested.

3 Development process

The project proposal gave much freedom for choosing a potential customer, product functionalities and various non-functional requirements. For instance, a user could be an amateur or a professional graphics designer, the way of tracing map elements was not set, and technical tools or solutions were not limited within a specific framework. However this flexibility might not be advantageous because at the end a product can be very different from what has been expected. Hence we have decided to follow an approach described further.

We selected Feature Driven Development (FDD) [7] as our main development methodology. In this method developers emphasize on features rather than tasks. In addition it is an agile methodology which proved to be successful during the past decade. What makes it agile is the process of continuous revision of features and review of system overall model. Those features were maintained in the feature list document. The feature list has become an inalienable part of project documentation and an artifact of periodical planning activities.

FDD has proved to be successful in projects of various sizes, and even though we have less people than recommended (not less than 10) in this project we utilized it by assigning several roles to the same person. There are several other reasons behind selecting this development method:

- Uncertain requirements – abstract requirements for product features and technical solutions where FDD can help emphasize on features and add, change or remove them rapidly at any time depending on customer feedback or requirements changes.
- Timeframe was short – the time given to complete the project was short so we needed a process that could deal with this constraint (agile, rapid) and we also could rearrange our features priorities fast.
- The need of continuous integration – our goal was to reduce the integration time at the end and make sure that we do not run into problems, so that early builds and integration tasks could take place; this adds up agility flavor.
- The need of robust software – we wanted to achieve the goal of delivering stable software build on the base of features we are going to select (with features with the highest priorities as the most expected).

Due to strict milestones (alpha, beta, release candidate and final release) set by DSD course lecturers it was important to fit FDD according to the given schedule. Hence we have had 4 iterations of features implementation and evaluation, with several feature list revisions within such iterations. On approaching each of those milestones we had a working system with planned features. It is not uncommon to overestimate or underestimate effort required for a feature and at times we either

delivered more features than we had planned or less features, however all that we had planned in the initial stage of this project was completed.

Figure 2 shows a graphical representation of our development process, where a revision of feature list document occurs periodically which is followed by a plan, design and implementation (build):

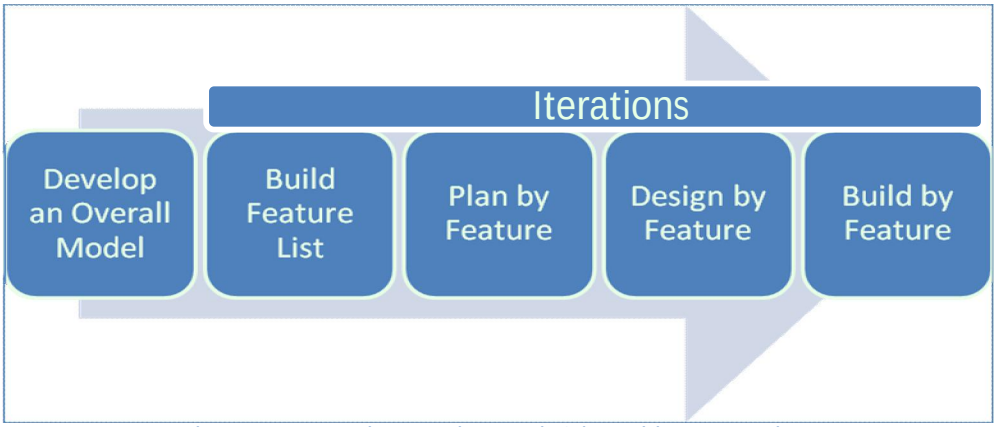

Figure 1: Feature Driven Development in Schematizing Maps project

It is worth pointing out that FDD helped us to accomplish the goals set. Instead of separating the implementation by modules or tasks and maintaining a list with descriptions of what each task involves we relied on picking features after a group discussion and working on them within a specified timeframe. This simplicity removed discussion and management overheads.

For implementation part of the project we have selected object-oriented programming with Java. Since we look at the overall model of our software as a collection of various objects it saves time for programmers to start coding by looking at this model from object-orientation perspective and not spending much time on figuring out how things should work.

First part of testing took place together with programming where a programmer ensured that committed code is working correctly and system build does not fail. Integration is also a part of this process. This requirement is set in the project policy and had to be followed strictly in order not to cause problems for other programmers and minimize the time spent on integration. Acceptance testing was conducted after our Release Candidate milestone and included test of all the features completed by that date. In parallel the programmers were expected to complete the remaining features and during the rest of the time fix bugs found by a tester.

4 Project plan and performance

4.1. Project Plan

The initial project plan is based on waterfall model in order to meet DSD course requirements. However, our actual activity is more flexible, which is decided by our development process (FDD based) as describe in previous section.

Week is the basic time unit in each task. Figure 3 shows our project activity plan. Blue color stands for initial plan and green stands for extra activities in practice. We stuck to the initial plan and

had more iterations in both documentation and development, which is more consistent with agile development.

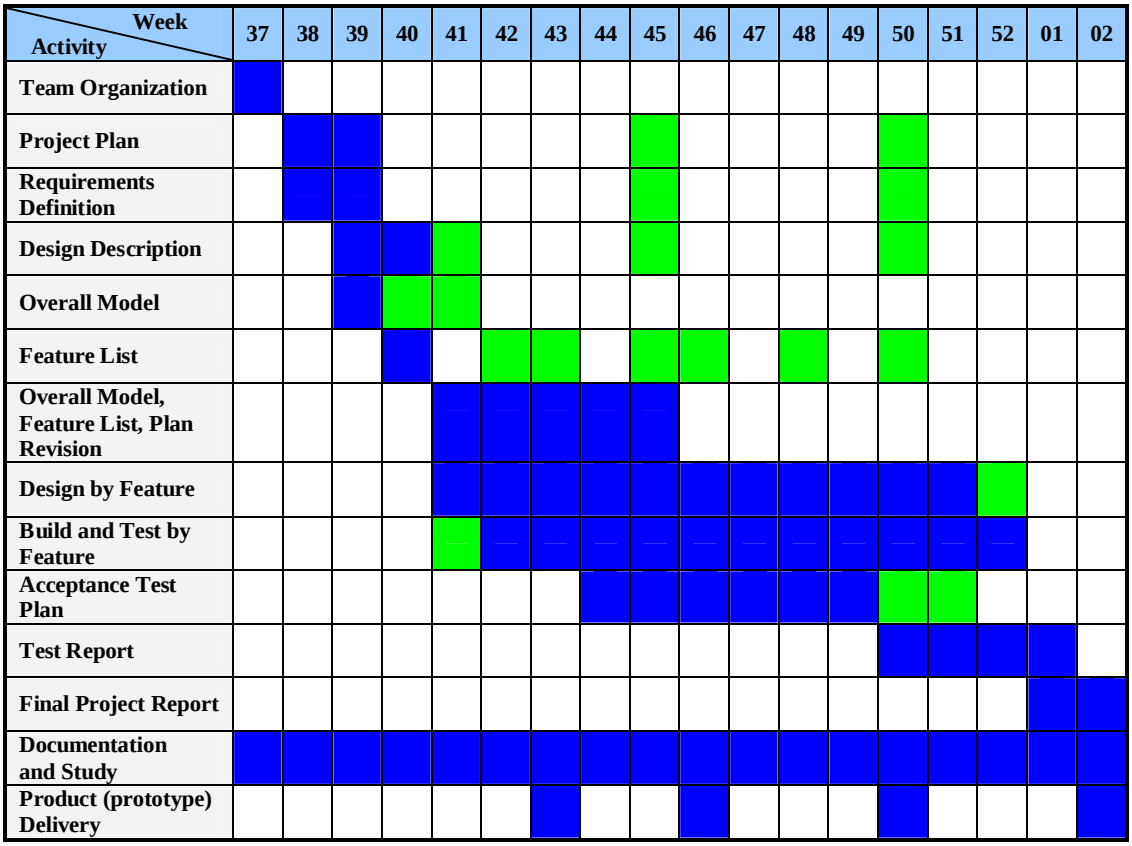

Figure 3: Project plan – Gantt chart

A list of milestones was decided along with initial project plan as shown in table 1. There are 4 main milestones (highlighted in table 1) which are Alpha prototype (2010-10-26), Beta prototype (2010-11-16), Release Candidate version (2010-12-14) and final project (2011-01-11). On each of these 4 milestones, we reviewed project status, summarized both good and bad experience, delivered latest product to our customer and created plans for the next stage. We also made a presentation of our progress for each of the main milestones in DSD course.

Table 1: Milestones

ID	Milestone Description	Finished Week	
		Plan	Actual
M001	Project plan	39	39
M002	Requirements analysis	39	39
M003	Initial Overall Model	39	40
M004	Design description	40	40
M005	Initial Feature List	40	40
M006	All policy documents	40	40
M007	Alpha prototype	43	43
M008	Model, feature list, plan revision	45	45
M009	Beta prototype	46	46
M010	Acceptance test plan	49	49

M011	Release candidate	50	50
M012	Test report	01	01
M013	Installation guide	02	02
M014	User manual	02	01
M015	SCORE report	02	02
M016	Final Project	02	02

Several kinds of potential risk were defined at the beginning of project. The experience from previous teams in DSD course was considered when we predicted risks. We also made a review of these risks at the end of the project in order to get experiences. Most of these risks were avoided or did not happen at all during the project, which was contributed a lot by this precaution.

Table 2: Project Risks

Possibility	Risk	Preventive action	Review
High	Beyond the deadline	Members follow up schedule strictly.	We delivered all documents and prototypes on time.
High	Miscommunication	Communicate when a problem happens. Document important information.	Never happened. We followed preventive action strictly.
Medium	Lack of technical competence	Members should help each other. Play one's strength and improve weaknesses.	Study and discussion has been proved quite effective when this risk happened.
Medium	Version control problem	Members follow the subversion [15] policy. Backup data periodically.	Never happened. Everyone followed policy and server was stable.
Low	Lack of human resources	Plan well and everyone completes/submits work on time. Assign the tasks according to team member's skills.	Never happened. We always had the right person to handle the right thing, which contributed by good communication.
Low	Loss of team member	Every team member should be aware of other members' responsibility.	Never happened.

4.2. Team organization

Our team is a distributed development team that consists of 6 members, 3 from University of Zagreb in Croatia and 3 from Mälardalen University in Sweden. All team members are graduate students who major in software engineering. Table 3 lists role distribution between team members.

Table 3: Team members and responsibilities

Name	University/Nationality	Responsibility (roles)
Adis Mustedanagić	FER/Croatia	FER team leader algorithm programming, visual design
DominikPavlović	FER/Croatia	GUI design, programming, documentation
Martin Vrkljan	FER/Croatia	Software design, resource gathering, documentation
Pavel Chen	MDH/Uzbekistan	Software design, programming, documentation
Ramesh Nagilla	MDH/India	SVN administration, testing, documentation

Zhixiang Gao	MDH/China	Project leader, MDH team leader document and team coordination, algorithm programming

4.3. Communication and Collaboration

We had various ways to ensure enough communication among the team. Specifically there were two kinds of communication in our team.

- Synchronous communication
  We tended to have face to face discussion among local team since this is the most efficient way to communicate. On the other hand, online communication (mostly voice and text) over Skype is our primary choice for the whole team. We adopted synchronous communication in meetings, technical discussions, collaborations and casual chat. We used phone call and SMS if necessary.

- Asynchronous communication
  We chose asynchronous communication when we had important information to share and it is not so urgent. In detail, we used Google group to share drafts of documents and tell all members important project activities. We used email to communicate with customer and supervisor and share important information from DSD course, SCORE, and other minor external information source.

Some collaboration tools were adopted in our project. They are:

- Subversion for version control;
- Project home page under DSD course web page as project documents repository;
- Online whiteboard Scriblink [6] for visual collaboration.

4.4. Week routines

We had a series of week routines since our project was planned in weeks. Usually, we started a week with a weekly TODO list which was posted by the project leader. Tasks were prioritized and assigned to team members in this list; then we had at least one all hands meeting during the week (mostly on Friday). The meeting usually took about one hour in order to make team decision rather than technical discussion. At the end of the week the team members were supposed to send a week report to project leader who would make a summary report of the week. Figure 4 shows week routines circle.

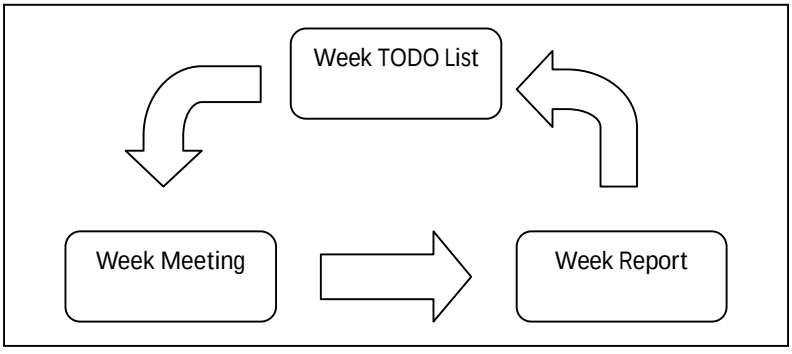

Figure 4: Week Routines

4.5. Work organization

Usually a task was assigned to individual instead of a group so that working time could be easier to arrange and responsibility could be more unambiguous. For a feature implementation task, responsible person is supposed to build the whole system before committing any new code. For a documentation task, there is always at least one reviewer who helps to revise the draft before document delivery. A team member always informs other members as soon as he has made any progress so that everyone could be aware of project status in time. Furthermore, brain storm happened in meetings and discussions sometimes, which helped a lot especially in inspiring new ideas. Flexibility and information sharing were highly emphasized in our project.

5 Requirements specification

The customer has asked for an application which aids a map designer in creating schematic maps. As a team, we have been given a lot of freedom in defining requirements. The customer gave us certain guidelines within which those requirements should be made. We have decided to concentrate on the basic functionalities and create a simple, but robust application, which could be used by amateurs and professionals alike. To describe what the application is required to do in general, we can describe what the user should provide as an input, and what the desired output would be.

On an abstract level, all that the application needs from the user are two sets of different entities: stations and lines. The stations are fixed locations, while the lines connect those stations and create different routes. On a more realistic level, those stations and lines can represent bus, tram or metro stations and lines, or even possible guidelines for notable landmarks, bicycle routes, or any other information that could be schematized in order to provide a more organized view of geographical locations.

Once the user has defined the input data, the output should be a schematic map. A schematic map is a special type of map which is drawn with a particular set of constraints in mind. The purpose of such a map is to offer a more organized view of geographical paths (routes, lines, etc.) which contains the information of geographical relation of stations and lines, but not necessarily their distance, as it will be explained in the following sections.

For the purpose of easier understanding of the Schematizing maps domain and better development process, we divided requirements into five main groups. We have business requirements, user requirements, functional requirements, non-functional requirements and algorithm requirements. This way, we came to better understanding of the problem we were solving and to approach the problem from different sides.

5.1. Business requirements

Business requirements describe the need for building this application. Requirements that should be met to satisfy customers' (or potential users') businesses are following:

- The application should offer simple and quick response of schematic maps.
- The application should produce a schematic map in a form that is printable, publishable on digital media, and easily distributable to users of customer's business.

5.2. User requirements

User requirements describe the tasks that the user should be able to perform while using this application. These tasks closely relate to the use case models shown in Section 5.6.

- The application should enable the user to easily create a schematic map based on his knowledge of network of stations and lines in question, regardless of his skill as a map designer.
- The application should provide the user with a quality end result in form of a schematic map, regardless of his knowledge or skill in mathematical algorithms, graph theory or graphical design.

5.3. Functional requirements

Functional requirements define the expected behavior of the application and operations it can perform in order to enable user to perform his tasks.

- The application should enable the user to define input data by drawing stations and lines that connect them over an image of a geographical map.
- The application should enable the user a certain amount of control over how the schematic map will look, such as the route's color, station's name. This should be handled by giving the user an option to work in a basic, fast mode with predefined settings, and the option to work in an advanced mode with the ability to adjust certain settings.
- The application should provide map schematizing functionality based on user input.
- The application should enable the user to save the input data and the output result for optional editing in the future.
- The application should enable the user to save the output result in a form that is printable and publishable on a digital media.

5.4. Non-functional requirements

Non-functional requirements describe the constraints and contracts the application should comply with. Any requirements on the graphical interfaces, performances, implementation, standards and similar should be listed here.

- The application should be simple and easy to use, for amateurs and professionals alike.
- The user interface should be simple and unobtrusive. The main use of each interface should be clearly recognized, with only necessary controls displayed and neatly organized.
- On the output schematic map, different lines should be displayed in different, contrasting colors. It must be clear which line continues in which direction and through which stations.

5.5. Algorithm requirements

Schematizing algorithm is the main part of the application. It takes stations and lines drawn by the user as an input and transforms them into a schematic map. Algorithm requirements describe the expected behavior of schematizing algorithm and constrains which the algorithm should comply.

- The schematizing algorithm should roughly support the geographical relationships of stations. A station which is located north from another station should not be positioned south of it on the schematic map.
- The schematizing algorithm should respect the input topology of the user's drawing; if that drawing represents the input graph, the embedding of that graph must be preserved in the schematic map.

5.6. Use case

Figure 5 is the main use case diagram which visualizes the process of creating a schematic map. A typical usage is like this: a user opens the application, imports an image of geographical map, then draws important locations and routes on it. After this the user applies schematizing algorithm on his drawing and soon gets a schematic map based on his input.

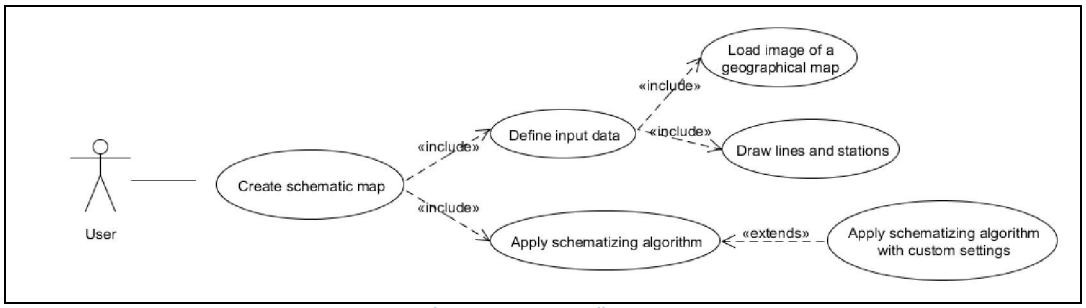

Figure 5: use case diagram

6 Architectural design

6.1. Conceptual design

We had a lot of freedom at choosing what additional features can be useful in our project and will be implemented according to the project proposal. So we opted for an architecture that is easily extendable and in which new features can be added with minimal modifications of the already implemented functionalities.

The application is realized as a Java desktop application. Reasons for this were mainly usability and ease of defining input data. We have decided that a desktop application would be a much better solution, from the user's point of view, and the developers'. Using a desktop solution, we were able to include certain libraries which proved to be invaluable to the development team.

The software architecture of Schematizing maps application is a modified MVC (Model-View-Controller) [16] architecture, as shown in Figure 6. The model is application logic layer and it will present schematic maps algorithm along with two sets of entities: stations and routes. The controller is a link between model and view.

When the user inputs the stations or routes, then the underlying map topology is modified. In case the user selects schematic map generation the controller reacts, and invokes the algorithm which provides a modified map. After that the controller delegates that data from model to view and the user can see output.

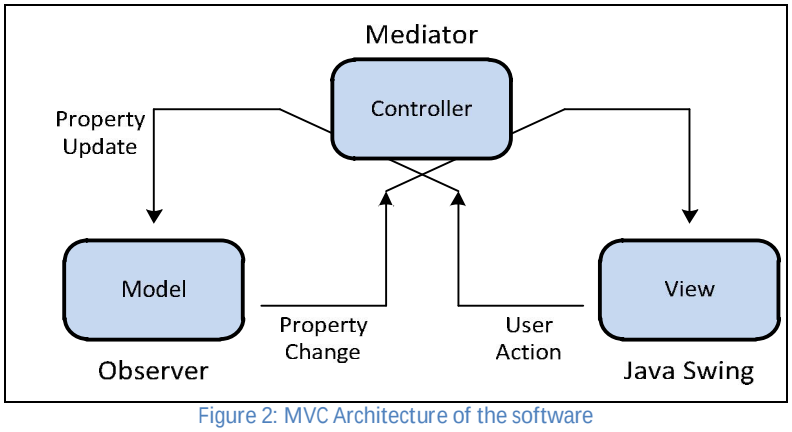

Figure 2: MVC Architecture of the software

The view includes the Graphical User Interface (GUI) of the application. This is where user will input data: mark the stations and draw routes of a schematic map, click the "Generate" (or "Advanced Generate") button and after that the algorithm reformats the map, this is where the result will be displayed for a user, and so he can validate the result, and make some additional changes if needed. The controller consists of a GraphController. The controller will take data that user inputs and delegate it to a model. When the user clicks on generate button or advanced generate button the algorithm will invoke, form a schematic map from the data that user inputs, and return the result. Then the controller takes result and delegates it to view for displaying. There is no need for a database; the final output (fully finished schematic map) of the application can be saved to disk directly.

6.2. System specification

This project is developed completely in Java, which enables it to run on any platform, independent of the operating system. Java Runtime Environment (minimal version 1.5 or newer) is required to be installed in order to run Schematizing Maps application. In order to be able to import and use web images as template maps upon which the schematic map will be drawn, a computer with access to the internet is required. For everything else, any computer with standard U/I peripherals is enough.

6.3. Main software components

In implementation of the architecture described above we rely on the singleton pattern [13]. For this purpose we have identified several components which will serve for handling input, map generation and output operations. The data flow among those components can be seen in Figure 7:

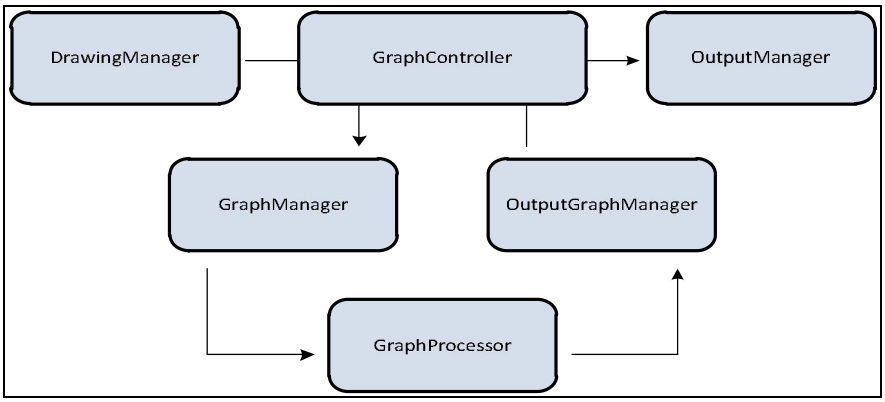

Figure 3: Data flow of schematic map generation

All of the components conform to the singleton pattern design. The application has exactly one copy of each of them. By following this pattern, it allows us to access commonly used components in the application in an easier manner and manage the logic with less impacts on the whole application. Table 4 is a brief description of each component:

Table 4: Components Description

Components	Descriptions
DrawingManager	defines visual settings of a map: routes and stations color, shape, and width
GraphController	handles all the modifications of the maps
GraphManager	contains the input map and related data
GraphProcessor	processes the input map and provides the schematic map as output
OutputGraphManager	contains the output map
OutputManager	defines the visual settings of the output map

Some important classes are defined in the system. They are:

- Edge and Vertex – represent edges (route sections) and vertices (stations) of a graph
- Route – represents a route in a graph and associated with edges and vertices
- ToolMode and GenerateMode – enumerations for tools and types of map generations
- VertexFactory and VertexType – define how vertices are created
- MdmFile – represents a custom file to be used for reading and writing ".mdm" files to a file system

Below are sequence diagrams for drawing and generating a map:

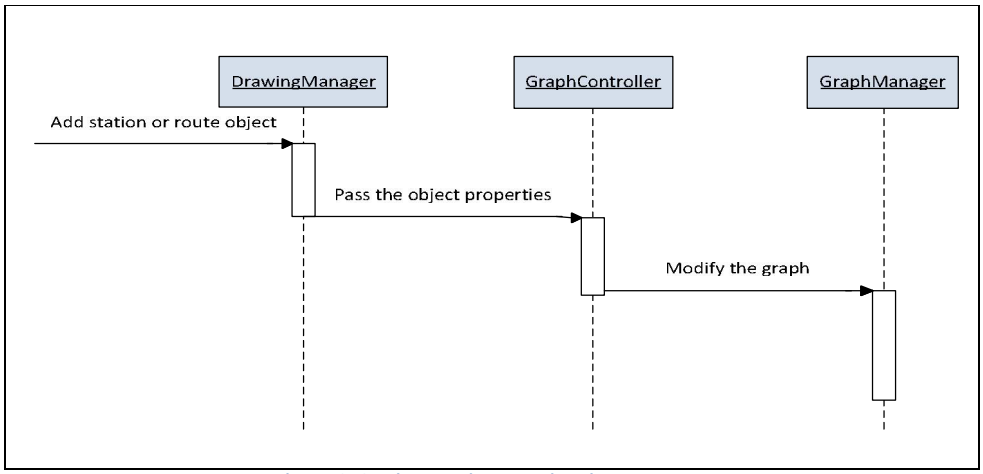

Figure 8: Stations and routes drawing sequence

Figure 8shows that once a station or a route is placed by a user, that object is passed to the DrawingManager which invokes the GraphController by passing that object using a relevant method. The GraphController calls a method that would add a station or a route to the global graph object.

In Figure 9 the process of schematic map generation is described. Similarly to the previous case, once a user decides to generate a schematic map, the GraphController will get the global graph

object and pass it to the GraphProcessor which applies the algorithm to a graph copy. After the graph copy is modified it is shown to a user.

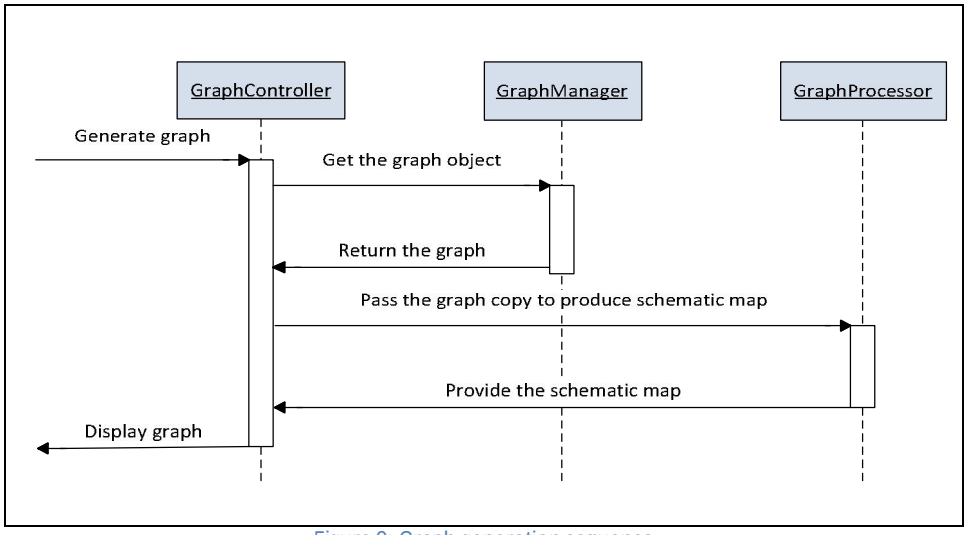

Figure 9: Graph generation sequence

7 Implementation

We set up several roles during the implementation, which was in light of FDD.

- Project Manager– an administrative lead of the project
- Chief Architect– a designer of the overall system
- Development Manager– an administrator of daily activities
- Chief Programmers–developers and testers of software units
- Tester– a tester of the whole system

Domain Experts– customers and supervisors

We chose NetBeans [10] as our Integrated Development Environment (IDE) since most team members had prior experience with it. Java is the only programming language in our project.

7.1. GUI Implementation

We have decided to create a simple and easy to use application with the GUI being as simple as possible. As the input data is defined by drawing and visually identifying the routes and stations, GUI elements needed to be unobtrusive and easy to use.

The GUI consists of Toolbox, which is supposed to provide user with the necessary tools for drawing, modeling and editing the schematic map, and of the tab pane with two tabs. First tab represents the panel in which user can draw a sketch of a schematic map, and the second tab represents the panel in which schematic map will be displayed after the algorithm is executed and a schematic map generated. Figure 10 shows the GUI of application.

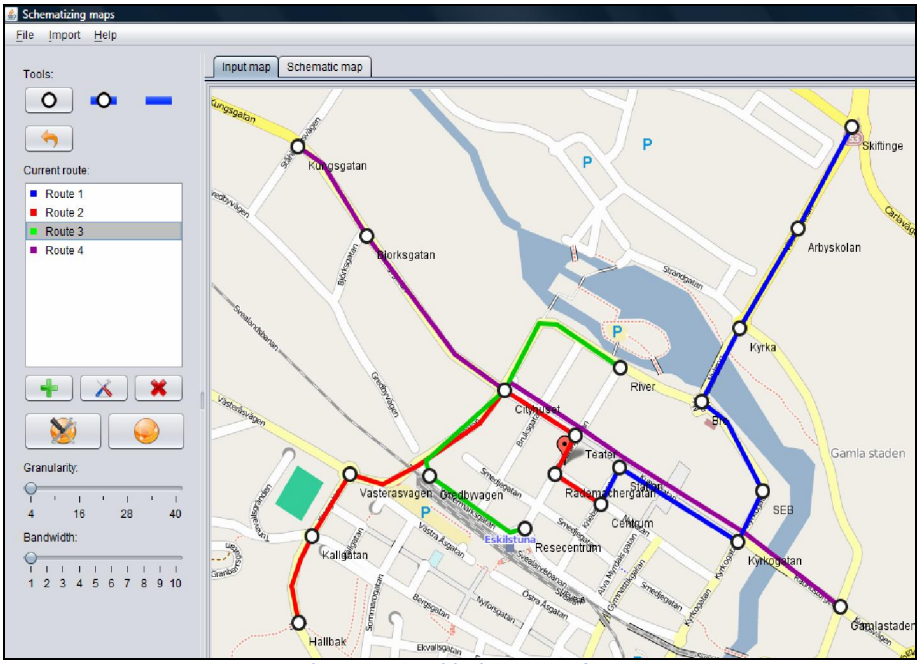

Figure 10: Graphical User Interface

Basic tools for drawing routes and stations are grouped in a toolbox panel which enables the user to perform all necessary actions. Action buttons define the currently drawing tool, the drawing mode and the useful undo function. Routes are neatly organized and listed, along with an icon representing the route color. Routes can easily be added, edited and removed if necessary. In case the user wishes to use more advanced features and have more control over the layout algorithm, the advanced controls are easily accessible.

7.2. Drawing implementation

The idea of defining input data is to keep the drawing process as fluid as possible, and allow a user to fix eventual mistakes and save the current work for later. Drawing implementation relies heavily on JUNG2. We have extensively studied JUNG2 and managed to customize some of the included classes and default behavior to suit our application. This helps us to use and modify the graph layout to visualize the input map, while keeping the graph structure in memory and ready to be processed by the schematizing algorithm.

Drawing of an input map can be separated into two distinct modes: placing stations and defining routes which connect them. There is also a hybrid mode called “Auto Connect” which combines the two main modes by connecting stations automatically as the user places them on a map.

The drawing functionality relies exclusively on mouse events. Parts of the JUNG2 library that we have mostly customized relate to classes which represent mouse behavior and the dynamic switching of modes. When station placing mode is active, the user is able to place stations on the map, move them around, and choose to rename them or delete them, all with simple mouse clicks and gestures. On the other hand, when the route mode is active, those clicks and gestures place routes and route points and connect them to stations. Route points are graph vertices which help the user to keep the input map neat and readable, as they allow the user to follow the underlying map

with as much precision as needed. Figure 10 shows an example of drawing with 4 bus lines in Eskilstuna, Sweden.

7.3. Algorithm implementation

Once the user has finished drawing the input data by placing stations and routes on the map, the graph structure is sent to the GraphProcessor for processing. The graph structure contains the coordinates of all stations and all connections between stations. The basic idea of the layout algorithm in GraphProcessor is to adjust all stations to proper places on a grid which stands for the drawing panel. Two important parameters are needed to be explained before introducing the algorithm.

- Granularity –indicates the shortest distance between two adjacent junction points on the grid.
- Bandwidth – indicates the width of band which the grid is spited into.

The application provides two options to generate a schematic map, which are auto generation and advanced generation. The difference between these two options is that user can adjust granularity and bandwidth in advanced generation. And in advanced generation, these two parameters are automatically set. All these two options fall into the procedure as follow:

1. Preprocessing. If user chooses auto generation then algorithm sets the granularity with the shortest distance of every two stations and bandwidth with default value. Otherwise it sets these two parameters with user chosen values.
2. Formatting. Algorithm initializes the grid with the parameter granularity and then adjusts all stations to their nearest junction points on the grid. There is a precaution program for stations overlap.
3. Vertical sweep. The grid is divided into vertical bands with the parameter bandwidth. Then algorithm sweeps each band and adjusts those hit stations' X coordinates to the middle of band.
4. Horizontal sweep. The grid is divided into horizontal bands with the parameter bandwidth. Then algorithm sweeps each band and adjusts those hit stations' Y coordinates to the middle of band.

The algorithm ends after finishing the sweeping on all vertical and horizontal bands. The processed graph data will be sent to OutputGraphManager to generate schematic map. Figure 11 shows the schematic map that generated based on the user input in figure 10.

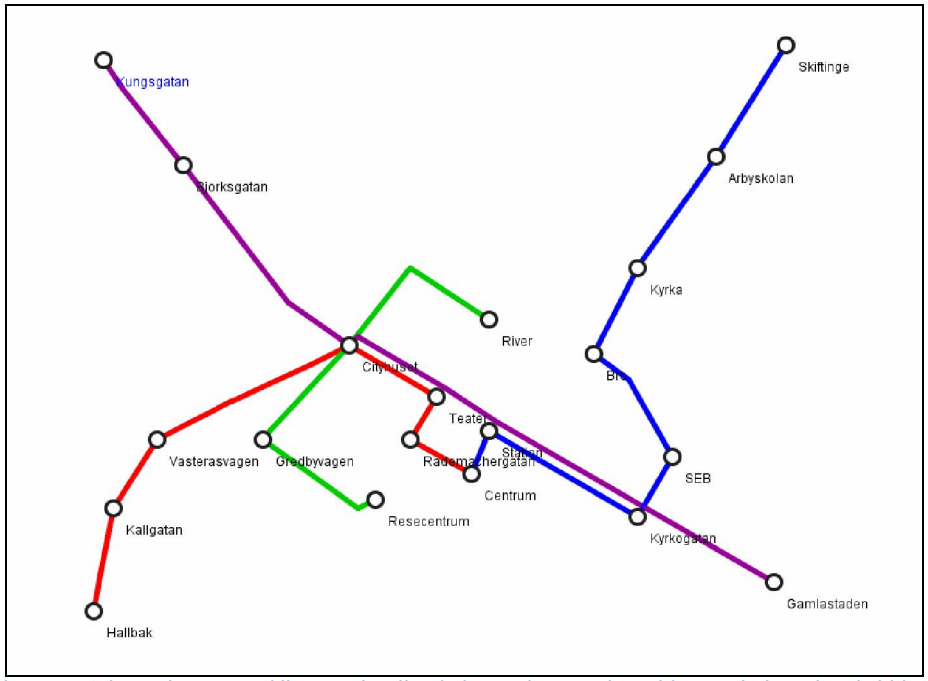

Figure 11: Schematic map – Eskilstuna's bus line (advanced generation with granularity 4, bandwidth 2)

7.4. File input and output

In order to make drawing as easy as possible, user can save his work in a project file and continue to work on the saved project later. For this purpose we have created our own file format with ".mdm" extension. An ".mdm" file contains data such as the underlying map image, graph structure and a list of created routes. To achieve this functionality we have applied the method of serialization and deserialization of Java objects.

Importing of the underlying map is based on loading an image from a local file system. The program can recognize several image formats (.jpeg, .png, .tiff) which after loading fill the drawing panel and the size of scrollbars is adjusted accordingly so that the entire image is visible to a user.

The application supports import of static maps based on their URLs also. The program sends an HTTP request to the image resource which is specified by a user and fills the drawing panel in the same way as it is in importing an image from a local file system.

One more feature that involves file manipulation is saving a schematic map as an image file. This feature is based on a basic file saving operation and the file format used for an output is ".jpeg".

8 Integration and verification

The process of internal integration and verification has involved two activities: continuous integration and acceptance test. Continuous integration was an ongoing activity throughout the whole project and involved work from all programmers. Acceptance test took place after Release Candidate milestone and was done as a separate activity with a tester role dedicated to it. Also the product was given to a customer for a feedback.

Continuous integration was one of the main policy items since beginning of the project. It was obligatory for a programmer responsible for a particular part to test the implemented feature

thoroughly and verify that it works without affecting the work of other programmers. This helped us to avoid problems with non-working code and there wasn't even a single case when a programmer who had downloaded the latest code revision found a compilation error or an irresolvable merge conflict. Also by having the process of continuous integration we eliminated the need in an integration test, because at any point in time we had a working system.

Acceptance test was an essential activity required not only by the DSD course, but also by the nature of the project. For conducting an acceptance test we have prepared *Acceptance Test Plan* document, where we have described in details what must be done in order to verify that features we have developed work correctly and to validate that features we developed are what the customers have asked for. The main tasks in acceptance test activity were to write the test cases and execute those test cases. This was assigned to a tester. Test cases were separated into six categories. Each category contained several test cases with such specifications as an identification code, inputs and expected outputs (if any), precondition, description and type of a test. Some of test cases were relying on other test cases and referenced them accordingly. The categories are:

- User interface – resolution issues, user friendliness, intuitive layout and controls, words spelling, controls presence
- Map generation – accurate application output (a schematic map) with respect to user specified parameters
- Toolbox – all the available tools and their behavior
- Import and file operations – opening and saving files, loading images
- Error handling – behavior of the application in situations specified in Design Description document (Section 3.3)
- Miscellaneous – any features that do not fall into categories above (i.e. performance)

Due to the nature of the project topic, it was not easy to define expected outputs of the main feature – schematic map generation. In order to test this feature the tester had to rely on his perception of what is a good or a bad result. Together with testing this feature the tester had made screenshots or wrote down descriptions of positive and negative outputs with relevant inputs, so that a programmer would be able to recreate a problem.

There were 30 test cases in total for all categories created to test all features. Out of those test cases 3 were marked as canceled due to dropped features. Out of remaining 27, all test cases passed, and as a result we can claim that all implemented features work correctly.

9 Outcomes and Lessons learned

9.1. Project outcomes

The requirement of this project was to develop an application that will provide schematic map designers with simple but quality solution for making the schematic maps. The main requirement was accomplished successfully.

Besides the main requirement, the application offers a lot of other features, developed to improve the possibilities of the application and quality of the programs output, schematic map. The program offers importing of web image as a template upon which a schematic map can be drawn. This way, it is very much easier for the designer of the schematic map to mark the stations and draw

routes that connect those stations cause his able to actually see the real geographic positions of them. Schematizing maps application offers easy way of drawing the stations and routes, editing them and removing them. Stations can be given names, renamed, removed, moved across the drawing panel or inserted on already existing route. Routes can also be removed, partially or completely, moved across the drawing panel; it is possible to change the routes color. It is also possible to draw stations with auto-connect option. That means that the stations will be auto-connected with the selected route at the same time as they are drawn. Designer has the freedom to choose whether he will draw a schematic map completely manually or just mark the stations and routes and let the algorithm, which is the core of this application, do the automated schematizing of the input map. The automated schematization comes in two variations: auto and advanced. The algorithm is based on the concept of adjusting the stations on a two-dimensional grid by aligning them to the nearest grid junctions and calculating distance based on neighboring stations. The process of adjusting stations coordinates is based on two parameters: granularity and bandwidth. The auto schematization does not take granularity and bandwidth into the schematic map generation, while advanced schematization does. In addition, edit on the schematic map that generated by algorithm is also supported. The application enables user to adjust the positions by dragging stations. Figure 12 and 13 shows another example of Bangkok river area's bus lines map.

Figure 12
Input map. Bangkok river area's bus line (partly)

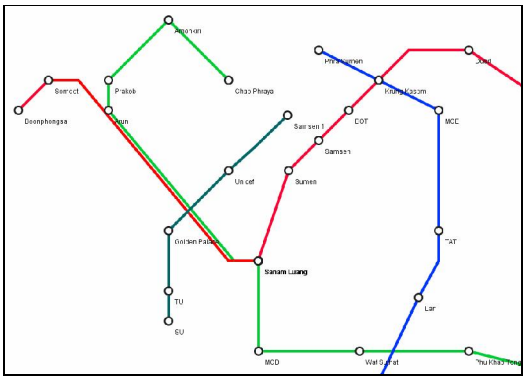

Figure 13
Output map (Advanced generation with granularity 6, bandwidth 10)

The schematic map does not have to be finished in one attempt; designer can save his current work and continue later. For the saving purpose a custom file format (".mdm") is developed which allows designers to save the template map along with the graph of stations and routes they have drawn.

All of these features makes the Schematizing maps application easy for use for both professionals and amateurs. Along with the application we have written a complex and extensive documentation that describes in detail all of the features implemented in this application and how the development process unfolded.

9.2. Lessons learned

For all team members of Schematizing maps project, this was the first distributed project in which we participated, so in the beginning, no one knew what to expect, how the cultural differences

and the impossibility to meet in person will reflect on the success and quality of this project. But we must say that we can consider us very lucky, because we all ended up in a team in which every team member showed great responsibility, great will and ability to complete the tasks that were presented to him.

We all took the early warnings about importance of the communication and cultural differences, especially in such a distributed team, very seriously. The result of this was that from the very beginning of our cooperation we were solving our tasks in a calm and sensible way and we are proud to say that we never had any conflict between any members of our team. We always had our final goal, a solid and quality application, on our minds and always tried to distribute tasks in the beast way on our path to achieve that goal.

In order to have and to keep the relationships between team members and attitude towards the project on a high level we managed to set up a Google Group for bigger discussions and draft documents and meetings over Skype at least once a week, or whenever a need arises and email. For distribution and management of our code we used a subversion server.

Based on the capabilities of each team member and on the requirements of our project, we divided the roles in the team and the development process of the Schematizing maps application started. We opted for a Java standalone desktop application because we found it more suitable for such type of application. During the development of this application, we had written a lot of documentation, we held the several presentations of our project and the progress of our project, we communicated often with ours project supervisor and we informed our customer periodically of the progress of the development and presented him results and constant improvements of this product.

Beside the successfully completed project and the satisfaction that we managed to complete the tasks given to us, we gained great and priceless experience that can only benefit us greatly in our future. Before all, we learned a lot about team work and how important a good team spirit is, about distributed systems development, cultural differences and new technologies.

10 Summary

Schematizing Maps application provides an easy way to create schematic maps. We have developed this software with designer and interaction as our first and foremost goals. Projects created by the application can be used with any other image editing software, outcome can be easily tweaked, and there are many other functions that can be found in most of other desktop applications aimed for frequent use. We did not manage to finish all features that set up at the beginning of project due to tough schedule. However, it'd not be difficult to implement new functions on our current application since we already have a robust, extendable and fully documented platform.

The application was developed according to the proposal given by our customer – Michal Young. We have also tried to follow the guidelines given to us during our course by our teachers and supervisors. Considering this application is a part of Distributed Software Development course, we had to complete our project in a rather short time, which required extensive planning and a lot of work from each of our six team members. We had to take into consideration the fact that we come

from four different countries and break the cultural barriers in order to keep our communication error free.

The experience gained during the development of this project is invaluable to us and will certainly be a great asset in our future projects. Considering the circumstances and the fact that none of us had any previous experience, either in distributed development, or in problems of this kind, we can say that we are very content with our final result, and judging by the initial feedback, so are the potential users.

11 Acknowledgements

At the end we would like to show our gratitude to the people who helped us during the project. We are thankful to our project proposal and also our customer Michal Young from University of Oregon, who gave us a lot of valuable comments and suggestions for our prototypes and documents. We thank our project adviser Martin Nöllenburg from University of Karlsruhe, who shared experience on layout algorithm in paper [9]. We thank our DSD course supervisor Ana Petričić from University of Zagreb, who gave us a lot of guide and encouragement during the project. We thank Prof. Ivica Crnković from Mälardalen University, who gave us instructions on both DSD course and SCORE contest. And we thank all teachers from DSD course. They provided us such an amazing opportunity to work and study with friends from distributed locations. Lastly, we offer our regards and blessings to all of those who helped and supported us during the project.

12 Reference

[1] Tube map of London, http://en.wikipedia.org/wiki/File:Tube_map_thumbnail.png
[2] Distributed Software Development, DSD, http://www.fer.hr/rasip/dsd
[3] Mälardalen University, http://www.mdh.se/
[4] Faculty of Electrical Engineering and Computing, http://fer.hr/en
[5] 3-SAT problem, http://en.wikipedia.org/wiki/3-SAT#3-satisfiability
[6] Scriblink, online collaboration tool, http://www.scriblink.com/
[7] Feature Driven Development, FDD, http://www.agilemodeling.com/essays/fdd.htm
[8] JUNG2 library, http://jung.sourceforge.net/
[9] Martin Nöllenburg, “Automated drawing of metro maps”, Internal report. Faculty of computer science, University of Karlsruhe; 2005.25, ISSN: 1432-7864
[10] NetBeans, Java IDE, http://netbeans.org/
[11] Stott, J.; Rodgers, P.; Martínez-Ovando, J.C.; Walker, S.G.; “Automatic Metro Map Layout Using Multicriteria Optimization”, IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. 17, NO. 1, JANUARY 2011.
[12] Swing, a primary Java GUI widget, toolkithttp://en.wikipedia.org/wiki/Swing_(Java)
[13] Singletonpattern, one of software design patterns, http://en.wikipedia.org/wiki/Singleton_pattern
[14] Michal Young, “Schematizing Maps”, http://score-contest.org/2011/projects/Young.SchemaMap.pdf
[15] Subversion, An open-source revision control system, http://subversion.tigris.org/
[16] MVC, Model-View-Controller, a software's architecture, http://en.wikipedia.org/wiki/Model-View-Controller