SCORE GPXCleaner project

DSD GPXCleaner

Tin Tvrtković, tin.tvrtkovic@fer.hr
Tihomir Bregović, tihomir.bregovic@fer.hr
Federico Ciccozzi, fci08001@student.mdh.se
Jenny Jutterström, jjm05001@student.mdh.se
Josip Labor, josip.labor@fer.hr
Pablo Santibañez Jara, psz05002@student.mdh.se
Coen Tempelaars, cts08001@student.mdh.se

January, 2009

Faculty of Electrical Engineering and Computing,
University of Zagreb

Mälardalen University

1. Executive Summary

Global positioning system [1] (GPS) devices are used in a wide range of outdoor activities, such as cycling, hiking and running, to allow their users to record and analyze their performed excursions. Services for displaying the gathered GPS records on a map and for sharing the recorded adventures between others are therefore becoming increasingly popular, but the raw GPS data obtained directly from the GPS device is often poorly suited for instant sharing. The GPS data files often contain unnecessary information, such as too many points or unwanted sections which may not be interesting for sharing. The GPS data may also be partly distorted with travel gaps or misplaced points due to a lost satellite connection. Therefore, the DSD GPXCleaner application was developed to enable users to manipulate the information received from their GPS devices before sharing it with others. For example, DSD GPXCleaner allows users to view and edit the records of their travels on a map, to join several excursions into one and to reduce the total number of recorded points with a minimal deviation from the original track.

The application was developed in eleven weeks during the Distributed System Development [2] (DSD) course at Mälardalen University [3] (MdH) in Sweden and at the Faculty of Electrical Engineering and Computing [4] (FER), University of Zagreb in Croatia. The course goal is to give students an insight in the complexity of distributed software development, provide training for working in distributed teams and to use the available techniques to facilitate distributed development. The project was therefore performed in a distributed environment with seven project members (of five different nationalities) located in two countries, this implicates cultural differences, teleconferences and the need for virtual cooperation without the possibility of meeting in person. The information exchange and frequent communication were therefore of the great importance for the project. Except from local meetings, the whole project team performed regular meetings every week both using Skype [5] and a video conference system. The contact with our customer Michal Young has also been important during the project process to avoid misunderstandings. Since we could not meet our customer, email communication involving written reports and illustrated PDF documents was used to validate the requirements and provide progress reports. The project had predefined milestones, such as document delivery dates and application prototype demonstrations, which made the waterfall model well suited as a start of the developing process. When most of the document milestones were met, we were able to use a more iterative model during the last process phases, implementation and testing, to avoid integration problems and to ensure that each component worked correctly.

The DSD GPXCleaner system design was modeled using Unified Modeling Language [6] (UML) to define the system structure and behavior and to allow easy discussion between all project members, before the distributed implementation. The system design follows a Model-View-Control pattern and the application is implemented in Java. We used commercial off-the-shelf software developed by NASA for the embedded map component, which is open source and therefore could be modified during the implementation to satisfy our needs. The application components were tested during the whole project and the application was verified at the end by using a requirement based test method. The project result is an application that fulfills all customer reqirements and provides adiditional features such as a simple view mode for most frequently used features.

Before starting the DSD GPXCleaner project, team members were all inexperienced in distributed development at the project start and have now learned distributed developing techniques and also received valuable experience in a global environment.

2. Introduction

GPX [7] data files received from a GPS unit often contain information which the user wants to manage and manipulate, such as discard travel parts or automatically reduce the number of track points without losing valuable information. Our solution for handling the recorded files is called DSD GPXCleaner. The application was developed during eleven weeks by seven project members located in two countries. The project was a part of the university course DSD at MdH and at the FER. The goal of the project was therefore to get experience in working in a distributed environment, focus on communication and to develop a product which satisfies the requirements of our customer.

The report begins with an introduction to the product requirements by defining the problem statement in Section 3, followed by a section regarding the project scope and the main project challenges. The following sections (Section 5, Section 6 and Section 7) describe the development process applied and define the project and management plan. Section 8, contains the product requirements and includes an example *use case*, and a sample of the complete list of requirements agreed upon by the customer. Section 9 describes the system

design, and includes conceptual, architectural, structural and behavioral design examples. The system implementation and the mathematical algorithms are discussed in Section 10. Section 11, contains information about the test methods used to verify the DSD GPXCleaner application and its components. Finally, Section 12 concludes the report by mentioning the project outcomes and team member experiences of the project.

3. Requirements: Problem statement

GPS units are growing in popularity, from specialized ones to brand new mobile phones with integrated GPS device. As such, more and more people want to share or store their tracks. However stored tracks often contain multiple thousands of points, most of them useless to the user but taking up unnecessary storage and bandwidth, providing very slow rendering on maps. Since a much smaller number is needed to display the track correctly, the key requirement of the project is an ability to reduce a number of points of user-recorded GPS tracks conserving the track shape as closely as possible, while providing the user with a measure of precision of the reduced track compared to the original one.

Beside the main problem – a large number of track points, there are several smaller ones that a user of a GPS device could face: the user might have turned on the device too early or too late and now would like to remove those points, or if a user had two tracks recorded on different dates but would like to join them together, or he would like to delete a part of a track. During track recording, due to weather disturbances, device errors, signal loss, etc., “wild points” can happen. A wild point is a point that is recorded by GPS device by error, often placed away from the rest of the track, and should be removed. Finally, the user would probably wish to rename his track, to help with organization.

Let’s say that we are at home and are going to university. We turn our device on and make ourselves on the way. During our walk we meet a friend and agree to go for a coffee with him. After the coffee we are of course late and decide to take subway. A quick walk from subway and there we are, ready on time for second part of the course. When we get home we look at the map and we have something to see. There’s our fieldtrip for coffee but because we are going to show this to our parent we want to cut it out, luckily we have an older track from which we can take the part we need and paste it to fit the erased one. Because we took the subway a lot of disturbances occurred and there are bunch of wild points we want to remove. Finally we were at university, but since we were late we forgot to run the device right away and now we have bunch of points in the same place. So we use GPXCleaner application to drop from two thousand points to mere 50. Happily we rename the track and upload it online for our parents to see it. The following figures illustrate our path before and after managing the track. (Note that there is no subway or coffee house − the map is just to illustrate real world scenario.)

Figures 3.1 and 3.2: Path simplification

4. Scope of the Project and Main Challenges

Since the DSD GPXCleaner application was developed during the university DSD course, the project scope includes both, properties defined by our supervisors to enhance the importance of the distributed progress and also qualities to achieve a good project process considering the SCORE [8] competition.

The course progressed 11 weeks in total with a workload of 20 hours per person per week. One of the objectives of this course was to accomplish a good result in the SCORE competition by performing a good

collaboration in the project team, having regular contact with our customer and considering the difficulties of distributed development and dealing with cultural differences.

The project team consists of seven members located in Croatia and Sweden, with university supervisors positioned in both countries. The project team is organized in two local teams, with the project leader located at FER and a Swedish team leader for the local project management at MdH. All project members have a computer science background, however with different specialties and from several universities. Two of the project members located in Sweden are international master students, originally studying at the University of L'Aquila in Italy and at the Eindhoven University of Technology in the Netherlands. The scope of this project was to develop a product which satisfies our customer, by performing a distributed developing process with project members located at separate places and from different background and cultures. Part of the project scope was also to develop comprehensive documentation detailing both the finished product and the process used, including: the project plan, weekly reports, the requirement specification, the design specification and the acceptance test plan [9].

The main challenges in this project involve both distributed development risks as well as project management difficulties when producing a product in a short time. Since the project members where located in two countries with different cultures, one important challenge was to create a good team spirit to facilitate the collaboration. Because the whole group could not meet in person, frequent and agile communication was important through the whole project process and to never let the distance be an excuse for not having a meeting. We therefore performed regular team meetings via a teleconference system using Skype, with detailed meeting notes (Minutes of Meeting documents published to project web pages) to avoid any misunderstandings and to ensure that persons who could not attend still got the information. We also used Google Groups [10] to discuss subjects outside the meetings, which could be read and answered by everyone. An available video conference system was also a good technique when we needed to discuss, explain or show something for the whole group, by using a white board for example (for an exhaustive list of communication channels used, see section 7). Another challenge in the distributed environment is to make sure all knowledge is spread; everyone should know what information exists and where to find it, therefore we put documents and files regarding the project on both our Subversion server and on the Google Group to make sure we all have the same information. One other challenge is the possibility of integration problems when the application is being developed concurrently at two sites. This was prevented by all members using the same repository and thereby continuously integrated small parts during the development.

The challenges which did not regard the distributed environment are connected to the project management to ensure the final application achieves a successful verification and validation. Since the project was performed in a short time and we needed to use unfamiliar tools, one challenge was to divide the work well from the start in order to accomplish a good final result. We managed to use off-the-shelf software to save time, e.g. NASA World Wind [11] (NWW) for the embedded map and an Eclipse [12] plugin called Jigloo [13] to ease the graphical user interface (GUI) development. We also stressed the importance of a working prototype to make sure we could handle these tools and determine that it was possible, considering the time constrains, to still implement all the requirements we had agreed on with our customer. Another project challenge was to ensure that the customer requirements where clearly defined to avoid any misunderstandings with our customer. Therefore, we had continual contact with our customer to get feedback and we got the requirement document approved early. We also appreciated opinions from the customer regarding the GUI and the user interaction in the beginning of the process, to have time to perform any changes and improve the usability.

5. Development Process

As a result of DSD GPXCleaner being a university project, worked on by students without experience in collaborative software development, no formal development process was used. Rather, an agile, improvised design process has emerged from the interaction between the students themselves, and between the students and local supervisors, incorporating elements of the waterfall model, the iterative model, and evolutionary prototyping. This section deals with that design process in detail.

Since the project was developed as part of a structured course (see section 4) with clearly defined and mandated milestones, parts of the waterfall model were used, especially at the early stages of development. Several project stages along with pertaining documentation were required to be completed at set dates. These were: the project description, the project requirement definition and the design description, along with three written feature descriptions, in that order. This part of the development process has elements closely

corresponding to the first three phases of the waterfall model (the requirements phase leading to the design phase, leading to the implementation phase).

After the initial requirements document was compiled and approved by both local and SCORE supervisors, the development process shifted toward the iterative model, with the rotating phases of analysis,

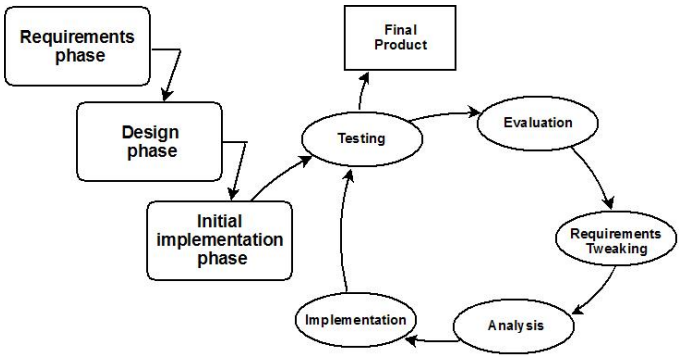

Figure 5.1: the DSD GPXCleaner development process

implementation, testing, evaluation and slight tweaking of requirements. The addition of the "simple mode" is probably the most obvious example of performing additional development iterations after evaluation had shown a need for additional functionality.

Evolutionary prototyping, defined as building robust prototypes in a structured manner, which can be constantly and continuously refined, was used to great success in designing the data manipulator class, which started out having only a few basic functions, and was expanded gradually as needed (the expansion was generally triggered by the gradual completion of the GUI module, or by adopting new items into the project requirements specification at the behest of the supervisors).

Commercial off-the-shelf software was used for the embedded map component and GPX reading and writing functionality. Both of these components were implemented by pre-existing code from the NASA World Wind Java code base. Although this code is licensed under the NASA Open Source Agreement [14], version 1.3, it still falls into many definitions of the term "commercial item".

Our development process can be said to have had three major iterations over its course. These iterations were mandated by the three strict feature milestones imposed on the project by the local (university) supervisors. For more information on the milestones, consult section 6, "Project Plan".

6. Project Plan

The development of GPXCleaner was directed by a number of milestones. This section deals with the development plan of the project in detail, and includes a list of milestones, both supervisor assigned and internal, brief overviews of these milestones and a Gantt chart of the entire project.

The development was week-based, with the tasks for the upcoming week being discussed and assigned at the weekly Skype meetings and the Google Group. For more information on this, see section 7. Important to note is that the development of DSD GPXCleaner differed in certain aspects from the recommended SCORE guidelines: the number of team members was 7 (instead of the recommended 5), and the development schedule was shorter than suggested (3 months, versus the recommended 5).

The following is a listing of all the milestones, along with time stamps (in week numbers, where 46-52 represent weeks 46 through 52 of 2008, and 01-03 represent corresponding weeks of 2009).

Id	Milestone Description	Supervisor assigned?	Finished week	
			Plan	Actual
M001	Project description	yes	46	46
M002	Initial requirements definition	yes	47	47-03
M003	Project design	yes	47	47
M004	GPX reading and writing	no	49	49
M005	Expert mode GUI, basic	no	49	49

M006	Embedded map segment visualization	no	49	49
M007	First “State of the project” presentation	yes	49	49
M008	Expert mode GUI, complete	no	51	51
M009	KML [15] reading and writing	no	51	dropped
M010	Path reduction algorithms	no	51	51
M011	Embedded map, complete	no	51	51
M012	Second “State of the project” presentation	yes	51	51
M013	Simple mode GUI, complete	no	03	03
M014	Time parsing and editing	no	03	02
M015	Wild point detection	no	03	02
M016	Final “State of the project” presentation	yes	03	03
M017	SCORE documentation	yes	03	03
M018	Final product delivery	yes	05	

Table 6.1: List of project milestones

The following is a Gantt chart detailing the activity plan of the project. Boxes colored indigo signify work being done on the items, and boxes colored green signify maintenance (refining, polishing, testing, bug-fixing).

Activity	w45	w46	w47	w48	w49	w50	w51	w52	w1	w2	w3	w4	w5
Requirements definition													
Team definition and managing													
Design description													
GPX input and output													
Expert mode GUI													
Simple mode GUI													
Embedded map													
Reduction algorithms													
Time parsing													
Wild point detection													
Integration													
Testing													
Documentation													
Final delivery													

Table 6.2: Project Gantt chart

At the start of development, we tried listing all the risks the project was likely to face and coming up with potential solutions. The following table enumerates the risk scenarios we thought likely and remarks on how they were avoided or dealt with.

Possibility	Risk	Preventive action	Remarks
High	Project is late.	Work overtime, define internal milestones.	Avoided by fine-tuning requirements and working with enthusiasm
Medium	Team member has problem with their assignment.	Reorganize team members.	Dealt with by reassigning team members.
Medium	Customer not satisfied with product.	Have constant communication with customer.	Avoided by communication.
Medium	Problems with integrating various project parts together.	Use UML, make out product design and assign team member/s to project integration.	Avoided by assigning a team member to integration.
Low	Problems with SVN.	Assign a person to take care of SVN and make backups.	Never happened.
Low	Team member leaves.	Reorganize team members.	Never happened.
Low	Communication problems.	Use several communication tools and have weekly meetings.	Dealt with by having several communication channels.

Table 6.3: Project risks

7. Management Plan

DSD GPXCleaner placed great importance on long-distance collaboration and communication. Four members of the team and one supervisor were based in Västerås, Sweden, and three members of the team and one supervisor in Zagreb, Croatia. In order to facilitate efficient project organization and management, a number of communication methods had to be used in parallel. This section lists all of these methods in detail.

7.1 An overview of the management process

The team was split into two sub-teams, based on the location of the team members: the “Swedish team” (consisting of 4 students) and the “Croatian team” (consisting of 3 students, one of which was chosen to be the project manager). Initial assignments were distributed at the first weekly Skype meeting (see section 7.3 below, “Organizational channels of communication”), along with the position of *Swedish team manager* (since the project manager was part of the “Croatian team”, a sub-manager was needed for the Swedish part of the project team).

The responsibilities of the project manager were: coordinating all efforts of the whole team, customer communication, communication with the university supervisors (among other things, by compiling meeting reports and summary weekly reports), assignment of tasks (although this was in practice done on a voluntary basis) and organizing the weekly Skype meetings.

The responsibilities of the Swedish project manager were the supervision of the members of the Swedish side, and the gathering of weekly reports from the members of the Swedish team.

The responsibilities all individual team members, aside from their assigned implementation or documentation tasks, were attending the weekly Skype meeting and compiling a weekly report summarizing their activities in the past week. These reports were then compiled into one summary report and submitted to the university supervisors.

The assignment of tasks, both initial and during the project course, was done at the weekly Skype meetings, on a voluntary basis. During task assignment, distance wasn’t taken into consideration, so often sub-teams in charge of a task had members of both the Croatian and Swedish sides; we found this to be in the spirit of the university course.

7.2 Technical channels of communication

These channels were used mainly for exchange of technical information, such as various technical documents and source code.

DSD web pages. FER hosted the communal web pages available to each DSD project. These web pages were used for communication between local supervisors and the teams; in particular, for sharing files required by the university-assigned milestones (mostly documents and presentations), for sharing team summary weekly reports, and for sharing minutes of meeting of internal team meetings.

Subversion. FER made available to each DSD project a server running the Subversion software[16]. This server was used for all collaborative code development. Subversion is a software package that enables multiple people to collaborate on source code, by providing special source code repositories and an easy way of merging source code files (so different team members were able to work on different parts of the same source code file at the same time, and seamlessly merge their contributions).

7.3 Organizational channels of communication

These channels were used mainly for organizing the team, by commenting on the current project status, assigning and reassigning members to sub-teams, assigning workload, and for communication with the local and SCORE supervisors.

Teleconferencing. MdH in Västerås, Sweden and FER made their teleconferencing facilities available to the staff and students of the DSD course once a week. Usually, this time was used by the university supervisors to communicate with the students (and vice versa, in the case of a number of presentations), but on occasion it was made available to the students to communicate amongst themselves. This form of communication was almost exclusively verbal.

Weekly Skype meetings. Early in the development process, the team members agreed to weekly Skype meetings in order to discuss the current project status and to more effectively and with more flexibility distribute workload among the team members. Skype is well-known freeware software used to make telephone calls (and

conference calls) on the Internet. This form of communication was practiced totally independently of the respective universities, and was exclusively verbal.

The GPXCleaner Google Group. A special Google Group was created in order to provide a forum-like venue of communication. The Group was used to discuss development topics in depth, as an addition to the weekly Skype meetings, and for collaborative creation of documentation. Although this channel of communication was created and used independently of the respective universities, local supervisors had access to it. This form of communication was used to report when assignments were completed by sub-teams, or to request additional time or resources as needed.

Email and instant messaging. Email and instant messaging were used for communication between individual team members, and especially for communication between the project manager and both local and SCORE supervisors. This form of communication was mostly used for person-to-person communication, while the other methods were used for group communication.

8. Requirements Specification

As DSD GPXCleaner is a SCORE project, basic requirements and outlines were given by SCORE supervisor Michal Young. But not only being a SCORE project but also a part of university course, requirements were discussed first and mostly among team members. From a document provided by the SCORE supervisor Michal Young, we took the required part and discussed which other options should we provide that would be useful. After we had the first requirements specification, it was given for review to course supervisors and the SCORE supervisor and changed where needed. As this is our first distributed course and one with “real” customer requirements were constantly changing and were analyzed as we found new ways to implement a feature or add a new functionality. All this was constantly discussed within the team and supervisors.

The customer required an application for handling (reading, writing and manipulating) GPX files and automatic reduction of GPS track points given in GPX file format. This was the main thought that led our project and all requirements were built on it. This led to designing of two types of GUI; a simple GUI with just basic features a user most often uses and more complex GUI with all features available in our solution.

GPX file is made of tracks with attributes like name, time of making etc. Every track has one or more segments with segment attributes like name etc. Segments have one or more GPS points with attributes like latitude, longitude, elevation, time etc. One such file contains route that we recorded during one or more of our trips.

The application required a reduction algorithm that would reduce the number of GPS points in a chosen segment by a specified amount and still retain the original segment shape as closely as possible, while providing the user with a measure of precision of the reduced segment compared to the original segment. Both user interfaces would need to feature wild point detection and removal functionality.

From the decision that we were going to create two types of interfaces, we had to agree on what features to enable in each. Both would need to have the embedded map for viewing tracks. The embedded map component would enable displaying of track segments overlaid onto satellite imagery. In the simple interface a user can use the reduction algorithm (wild points are detected and deleted automatically) and he can split segments and remove them. In the complex GUI, alongside all these features, user can also add single points, remove them, add segments, remove segments, change point information (name, time, GPS position), and change segment and track information, move points/segments from one segment/track to another, reorder tracks, segments, points etc.

For agreed and given requirements we worked out *use case* models for all functionalities. Here we present an example of a *use case*:

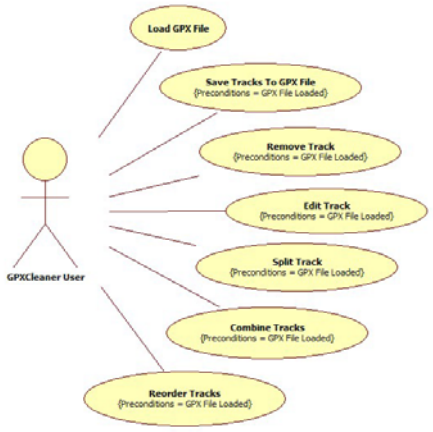

Figure 8.1: Use case model – Track manipulation

Use case "Load GPX File"

Initiator: *GPXCleaner User*
Goal: *Load a GPX file into memory*
Main Scenario: *1. A standard file selection dialog opens*
2. The user navigates to the desired file and selects it
3. The user clicks the "Open File" button on the dialog
4. The newly opened tracks are added to the tracks in memory
5. The list of loaded tracks and track segments updates

Extensions: *If the user clicks the close button, the action is aborted*
If the selected file doesn't contain well-formed GPX data, the operation is aborted and the user is notified

The following is a listing of defined requirements, some of them were defined at the beginning, some were added later and some were dropped. Keep in mind these are only partial requirements definition tables to give example how they were organized, for full requirements definition see official project documentation.

Identification	Requirement Group
SC	System Controller
ME	Math Engine
PA	Parser
NWW	NASA World Wind
FIO	File Input/Output
SH	Segment Handler

Table 10.1: Requirement Group Definitions

Source	Description
Ctm	Customer (Michal Young)
Sys	Required as a consequence of system design (contractor's requirement)
Ds	Developers suggestion

Table 10.2: Requirement Sources

Identity	Status	Priority	Description	Source
			System Controller	
SC-1	I	1	Core logic, connecting all parts together	Sys
			Math Engine	
ME-1	I	1	Automatic point reduction	Ctm
ME-1-1	I	1	Point reduction based on custom algorithm	Sys
ME-1-2	I	2	Point reduction based on time point was added	Ds
ME-2	I	1	Removal of wild points	Ctm
			Parser	
PA-1	I	1	Parse data read with FIO	Ctm
PA-1-1	I	1	Read tracks	Sys
PA-1-1-1	I	2	Read track names	Sys
PA-1-3	I	1	Read points	Sys

PA-1-3-1	I	2	Read point name	Sys
PA-1-3-2	I	1	Read longitude	Sys
PA-1-3-3	I	1	Read latitude	Sys
			NASA World Wind	
NWW-1	I	1	Show tracks in embedded map, view mode	Sys
NWW-1-1	I	2	Points can't be selected	Sys
NWW-1-2	I	2	Show number of points based on zoom	Sys
NWW-2-3	D	2	Allow dragging of points(changing its GPS position)	Ds
NWW-2-4	I	2	Allow removal of selected point	Ds
NWW-2-5	A	3	Allow changing selected points GPS position by selecting the point and clicking on new position	Ds
			Graphical User Interface	
GUI-1	I	1	Open GPX file	Ctm
GUI-2	I	1	Save GPX file	Ctm
GUI-3	I	1	Call automatic point reduction	Ctm
GUI-3-1	I	1	Select how many point it should remove	Sys
GUI-7-1	I	2	Call SH to change various properties of tracks	Sys
GUI-8	I	1	Edit segments	Sys
GUI-9	A	3	Open simple GUI	Ds
GUI-9-1	A	3	Call automatic point reduction and wild point removal	Ds
			File Input/Output	
FIO-1	I	1	Read GPX file content	Ctm
FIO-2	I	1	Save current tracks, points, segments into GPX file	Ctm
FIO-2-1	I	1	Make backup of old GPX file	Ctm
FIO-3	D	3	Save current tracks, points, segments as KML file	Ds
			Segment Handler	
SH-1	I	1	Edit track	Sys
SH-1-1	I	1	Edit track name	Sys
SH-1-2	I	1	Delete track	Sys
SH-1-3	I	1	Add new track	Sys
SH-3	I	1	Edit point	Sys
SH-3-1	I	1	Edit point coordinates	Sys
SH-3-2	I	2	Edit point name	Sys
SH-3-3	I	1	Delete point	Sys
SH-3-4	I	1	Add new point	Sys
SH-3-5	I	1	Edit point time	Sys
SH-3-6	I	3	Edit additional point properties	Ds
SH-3-7	I	3	Move point to another segment	Ds

Table 10.3: Partial requirements definitions

Requirement status:

I = initial (this requirement has been identified at the beginning of the project),
D = dropped (this requirement has been deleted from the requirement definitions),
H = on hold (decision to be implemented or dropped will be made later),
A = additional (this requirement was introduced during the project course).

9. Architectural Design

9.1 Conceptual design

The following conceptual design outlined in Figure 9.1.1., is useful for getting an idea about the structure of the final product.

In the conceptual design, the main part is a SystemController. The SystemController communicates with the GUI, which uses an I/O component and the NASA World Wind component. The I/O component is for reading from and writing to GPX files, the NWW component is there for visualizing the GPX track on a globe. The SystemController has three separated sets of operations which can be initiated by the user. The parser is there for parsing GPX files. The math operations include track reducing algorithms and wild-point removing algorithms. The SegmentHandler contains operations for combining and splitting tracks or segments.

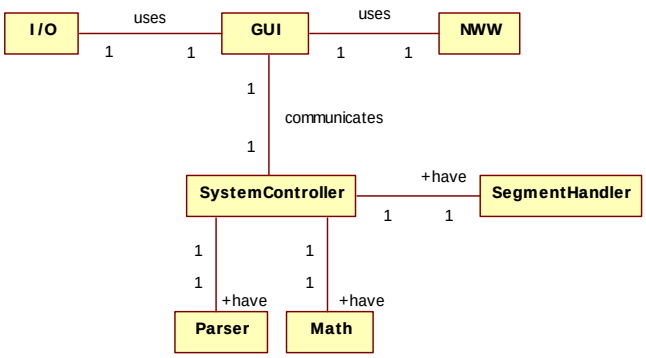

Figure 9.1.1: Conceptual design

9.2 Structural design

The structural design is based upon the model-view-controller pattern. Several versions of the structural design exist. The version that conforms to the implementation of the project is outlined below in Figure 9.2.1.

Figure 9.2.1 shows a collection of nine classes, logically grouped into the three parts of the model-view-controller pattern. The class and method names in the structural design below conform to the actual classes in the source code. Note however that the set of methods in the design below is incomplete. The methods are there to give the reader a quick notion of the purpose of every class, instead of being there for some kind of reference. Furthermore note that the UIFrame class is designed as an empty class. This class has not been implemented as an empty class. The implemented class contains a large constructor which puts all widgets on the main frame of the user interface, some methods that update the contents of the different lists on the main frame and a lot of ActionListeners that react upon user input.

The model part is represented by the GPX_DataObject folder. It shows the data structure that represents a GPX file. Such a GPX_DataObject contains zero or more tracks, which in turn contain zero or more *reducible* segments. Such a reducible track segment is a wrapper for a GPX track segment. A GPX track segment contains zero or more track points.

The view part of the structural design is the graphical user-interface, called GPX_GUI. The user can choose between a simple GUI and an expert GUI. In the both cases, the main frame of the GUI is constructed in the UIFrame class. This main frame contains an I/O FileHandler and the embedded map, which is called EmbeddedMapWWJ.

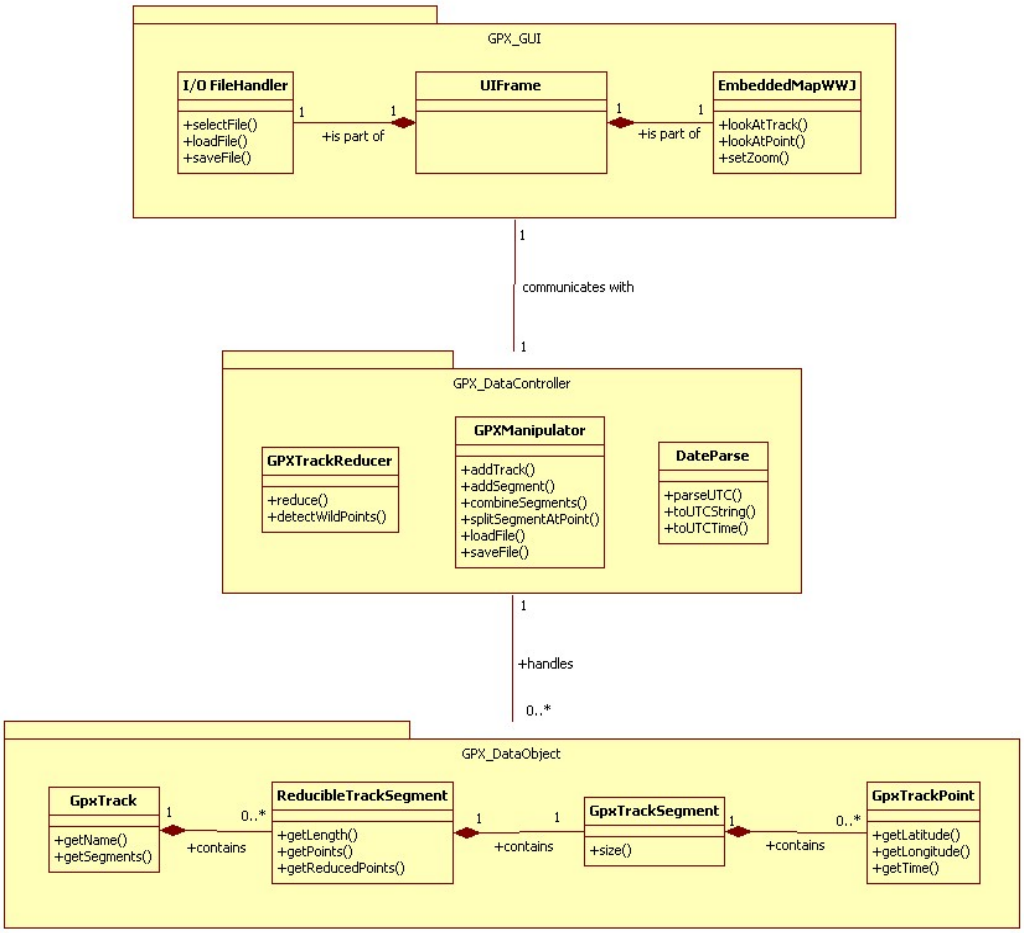

Figure 9.2.1: Structural design

The controller part of the design is called GPX_DataController. It contains three classes that are responsible for manipulating the data structure. The GPXManipulator class implements many smaller methods for manipulating tracks, segments and track points. The GPXTrackReducer implements a few methods for reducing an entire track and finding wild points in a track. Finally, the DateParse class is there for converting dates in the GPX format towards human readable dates and vice versa.

9.3 Behavioral design

Two sequence diagrams have been created in the design phase, conforming to the core functionality of the system. The first sequence diagram shows the behavior of the system when a user wants to open a GPX file. This is shown in Figure 9.3.1. The second sequence diagram, as depicted in Figure 9.3.2, shows the behavior of the system when the user decides to reduce the currently loaded track.

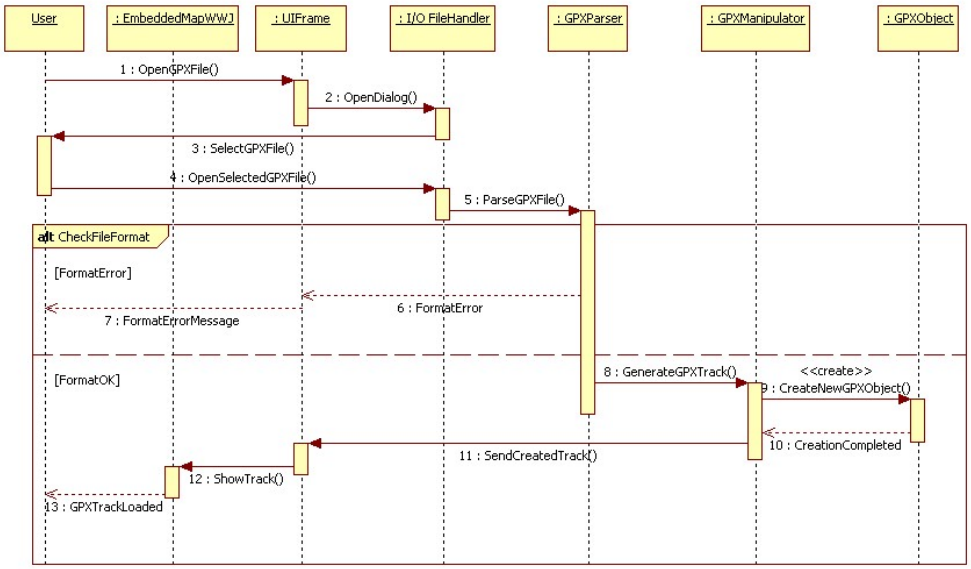

Figure 9.3.1: LoadGPX sequence diagram

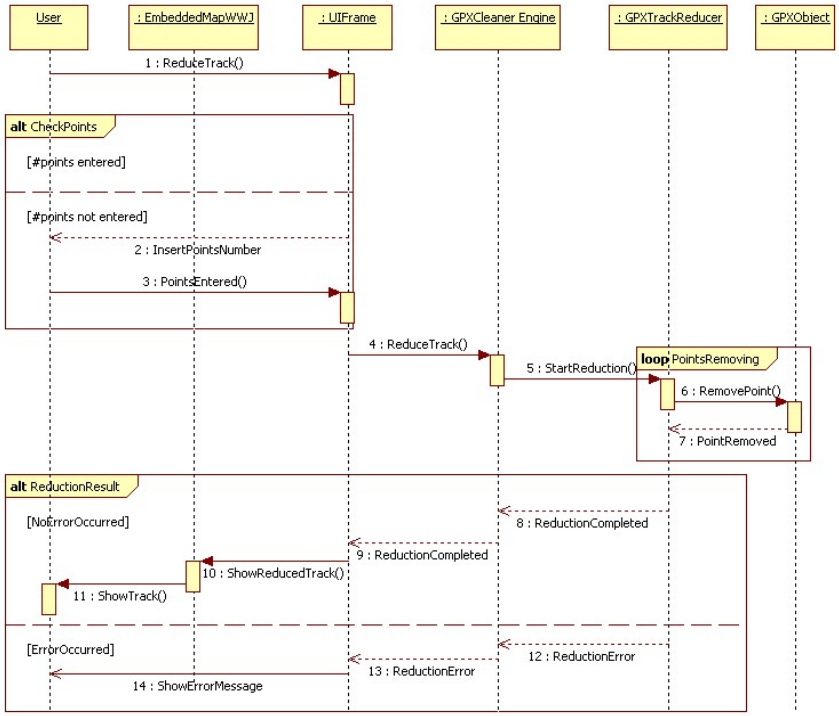

Figure 9.3.2: TrackReduction sequence diagram

10. Implementation

The final application was realized as a desktop application written in Java [17], because all of the project members were familiar with the Java language and because the tools for Java development (such as Eclipse, which was used by the project) are free and widely available. The NASA World Wind Java software development kit was used for the realization of the embedded map component and the file reader and writer components (because using commercial off-the-shelf software (COTS) can dramatically decrease the development time needed for these components). The graphical user interface was mostly implemented by specialized tools (such as Jigloo, see the sections below), but parts of it were coded by hand.

Project members were assigned into a number of sub-teams, each with a special development focus; this led to highly parallelized implementation. The following table illustrates how the sub-teams were assigned at the start of the project, and in the last week of the project.

Task	Start of the project	Last week of development
GUI implementation	2 members	2 members
Embedded map	1 member	no members
File input/output	1 member	no members
Math-related tasks	2 members	1 member
Documentation	1 member	4 members

Table 10.1: Sub-team assignments

This section deals with each of those tasks in detail, except the documentation task and sub-group.

10.1 GUI implementation

We knew from the start that we needed a GUI for users to interact with. We then stood with the choice of making the GUI by hand coding or trying to find a code generating tool that would help us make one. Coding by hand was quickly discarded as an option because the GUI is big, in the sense that it contains many objects which needs to be put together in a complex way. We therefore needed an application to give us a quick overview of what we were doing i.e. see component positions and they should be easy to edit by just dragging them around. We chose to use a plug-in for Eclipse called Jigloo to implement the GUI. Jigloo is a graphical editor for GUI implementation; we chose it as it is a good tool to get overview how the GUI was going to look for the user while we were developing it.

The main focus in the development of the GUI was to have a reasonably big window for the map to be shown in so the user wouldn't find it too hard to interact with the map. Another concern was that we needed to implement the buttons to cover the required functionality. To do so, the different buttons were grouped according to what they are intended to edit, thus the edit track buttons were grouped together and similar solution had to be done for the rest of the buttons so the user could access the functions quickly when editing one part. We wanted to have, from a user point of view, the map in the middle of our GUI so we put the point edit buttons to the left side of the GUI; this is so that the map would be the main focus and the buttons always would be visible so the user doesn't need to shift the eyes so much in the while working with the program. Another thing to have in mind is that we wanted to let the user to have as much information that would be relevant for each section so a user could get a quick overview of the editing.

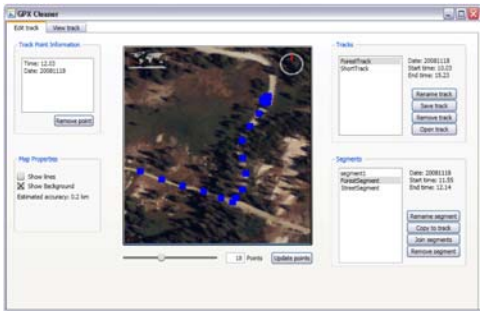

Figure 10.1.1: Milestone M005 mockup

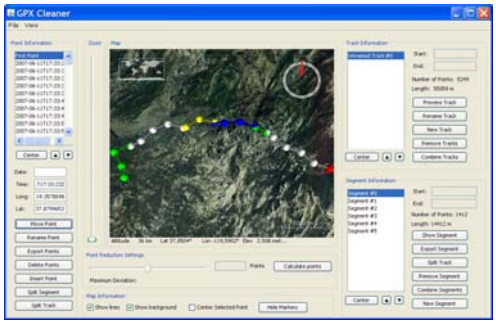

Figure 10.1.2: Milestone M008 GUI

Initially, we made a mockup to present and complete milestone M005 (see table 6.1: "List of project milestones"). After some customer feedback on what to improve and after adding the last missing buttons in each section, we presented a more complete GUI at the completion of milestone M008. As this version proved to be too complex for a new user we decided to make a simplified version so that a user could do some basic editing without getting intimidated by the complexity of the original GUI. We sat down and designed a simplified version which, after being reviewed by our customer, we implemented into the application.

During GUI development we ran into problems when we needed to edit components directly in the code. As the graphical view of the GUI shown by the plug-in was rendered directly as code were added, it could make the program unresponsive as it started rendering the view before the written line was finished, thus trying to render a view with errors in it. Another thing we later noticed is how the tool was unable to handle component

adding or removal by hand e.g. removing a button could break the code structure when the code was reworked by the plug-in. We dealt with this by creating a skeleton GUI in Jigloo and finishing the implementation by hand.

10.2 Embedded map

The embedded map component was implemented by modifying pre-existing, COTS software. This section contains descriptions of the software used, changes made to it, and challenges faced during development.

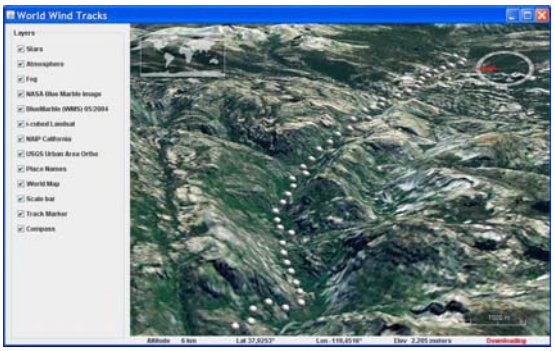

Figure 10.2.1: World Wind Tracks

NASA World Wind is a free, open source virtual globe application developed by NASA and the open source community for use on personal computers running Microsoft Windows, first released in 2004. The program overlays NASA and USGS satellite imagery, aerial photography, topographic maps and publicly available GIS data on 3D models of the Earth and other planets. The program is written in C# and licensed under the NASA Open Source Agreement, ver. 1.3. In 2007, a new version of World Wind had been developed in Java, referred to as World Wind Java. This new version has an API-centric architecture with functionalities off-loaded to modular components, making World Wind itself a plug-in. These features made World Wind Java ideal for adaptation and use in the DSD GPXCleaner project.

A full description of World Wind's implementation is well beyond the scope of this document. Therefore, we will limit our focus to parts of World Wind that were specially modified for the needs of GPXCleaner.

World Wind uses the concept of layers to present the user with information and controls in its window. These layers can include, but are not limited to, components such an on-screen compass, a miniature world map, satellite and aerial imagery overlaid onto the virtual globe, and icons and geometric shapes rendered on the surface of the globe. One of these layers, called TrackMarkerLayer, is shipped with the World Wind Java package, as part of an example application called Tracks (these example applications are supposed to demonstrate the capabilities of World Wind to novice software developers). This layer has the ability to display GPX tracks as icons rendered on the surface for the virtual globe, and the ability to programmatically vary the number, density and size of the icons based on the level of camera zoom (in a nutshell: the more the user zooms in, the track is shown in more detail). This layer was the base for the two layers GPXCleaner uses to render and allow interaction with GPX tracks; these layers are called SimpleMarkerLayer and ComplexMarkerLayer.

SimpleMarkerLayer is a layer heavily based on World Wind Java's TrackMarkerLayer. Features shared with TrackMarkerLayer are: programmatic changes to the size, number and density of icons rendered based on the level of camera zoom, and the ability to display both tracks and segments (this is relevant since the other custom layer, ComplexMarkerLayer, can only render segments). Improvements made over the original TrackMarkerLayer are: introduction of the selection concept (if a user selects a point in the graphical user interface, SimpleMarkerLayer will never skip rendering the selected point and will render it in a distinct color), awareness of reduced segments (SimpleMarkerLayer will not consider points that have been reduced away, if there are any), optional rendering of lines connecting neighboring points and the ability to render different segments of a track in rotating colors (the first segment is rendered in white, the second in green, etc.). This layer is the basis of GPXCleaner's preview functionality; it can be used to display whole tracks (which may contain a very large mount of points) without making the whole application unresponsive due to a heavy burden being placed upon the CPU.

ComplexMarkerLayer is a custom built layer based on World Wind Java's TrackMarkerLayer. The main differences between the two are: ComplexMarkerLayer displays each and every point of a given segment, can only render individual segments, is aware of the current point selection (will render selected points differently),

can optionally render lines connecting neighboring points, and is aware of segments that have been reduced. This layer is also “pickable” (in the context of World Wind Java, this means the embedded map component will generate events in response to an element of this layer being clicked on with a mouse). Since this layer doesn’t skip any points while rendering and since its icons can be selected with the mouse, this layer is the base of precision segment editing functionality in GPXCleaner. Due to a potentially large number of icons rendered, and due to additional CPU-time being necessary to detect its icons being clicked on, this layer places a greater strain on the CPU.

Figure 10.2.2: a side-by-side comparison of SimpleMarkerLayer and ComplexMarkerLayer

SimpleMarkerLayer and ComplexMarkerLayer were built as complementing layers, meant to be used together. For example, selecting a segment in GPXCleaner’s “simple mode” will render the track containing the chosen segment using SimpleMarkerLayer (except the selected segment) and the selected segment using ComplexMarkerLayer. This arrangement is thought to be a good compromise between giving the user as much useful information as possible while keeping the application responsive in normal usage conditions. The same effect can be achieved in “expert mode” as well, but there it requires explicit activation.

Since World Wind Java provided basic, but easily upgradeable components for rendering GPX data on the virtual globe, the biggest challenges faced during the implementation of the embedded map component were: understanding the existing architecture, modifying it for GPXCleaner’s specific needs, and further optimizing it. As the embedded map component was judged to be mature, the member assigned to it was reassigned to work on general integration.

10.3 File input/output

One of the main requirements for GPXCleaner was the reading and writing of GPX files. GPX is a GPS navigation device data format based on XML [18], which is device independent. It is used for points of interest, tracks, and route points. Except for GPX, there is one more format commonly in use for presenting geographical data – Keyhole Markup Language (KML) which is a language schema for expressing geographic annotation and visualization on two-dimensional maps and three-dimensional Earth browsers.

After consulting the project supervisors during the initial requirements gathering phase, it was decided to support only a subset of the GPX standard and if time permits, a subset of the KML standard. The elements and attributes chosen for support were the ones most likely to be used in commercial GPS devices to record tracks.

Due to the short development schedule and time constraints, KML support was dropped near the end of the project.

The GPX input and output functionality was achieved by using an existing part of the NASA World Wind Java package (in particular: the GPXReader and GPXWriter classes) and subsequently modifying them to support the attributes and elements important to GPXCleaner. The usage of COTS software for this task has significantly cut development time and made it possible to reassign the member in charge of file input/output to help with GUI development.

The list that follows illustrates which parts of the GPX standard are supported by GPXCleaner and which aren’t. The supported elements and attributes are listed in **bold**, and the unsupported ones are ~~stricken-through~~. Unsupported elements and attributes are simply ignored by GPXCleaner.

Element name	Description of element	Supported attributes	Unsupported attributes	Supported elements	Unsupported elements
gpx	root element of a GPX file	-	~~version~~, ~~creator~~	**trk**	~~metadata, wpt, rte, extensions~~
trk	a GPX track, part of gpx	-	-	**name**, **trkseg**	~~cmt, desc, src, link, number, type, extensions~~
trkseg	a segment of a GPX track	-	-	**trkpt**	~~extensions~~
trkpt	a point on the surface of the Earth, part of trkseg	**lat**, **lon**	-	**ele**, **time**, **name**	~~magvar, geoidheight, cmt, desc, src, link, sym, type~~

Table 10.3.1: GPX elements and attributes

GPXCleaner also supports creating back-up copies of files about to be overwritten by its output. If creating a back-up copy is allowed in the directory containing the file to be overwritten (by file system permissions, available disk space, etc.), the original file is copied with a changed name (the date and time of the back-up is appended to the file name).

10.4 Math-related tasks

The path reduction algorithm represents the core functionality of the GPXCleaner system. This algorithm does not work in a straightforward way, therefore some explanation is needed. The path reduction algorithm is designed as a combination of two different approaches.

10.4.1 First approach

The first approach towards a path reduction algorithm resulted in a quite straightforward way of reducing paths. It works as follows. For every point in the track except the start point and end point, a distance *d* is calculated. This *d* is the shortest distance between the point and the line through both direct neighbors of the point. This is outlined in Figure 10.4.1. Note that this *d* can be calculated using basic mathematics.

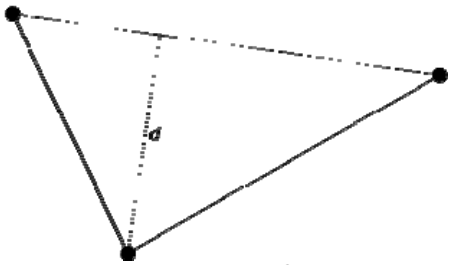

Figure 10.4.1: Calculating d

After *d* has been calculated for every point (except the start point and end point), the points with the smallest *d*'s are removed from the track. This could either be a number of points defined by the user or it could be all points with a *d* below some threshold. This approach results in a reduced track that is quite similar to the original track. The approach is good for removing points that are (almost) on one line with their direct neighbors.

10.4.2 Second approach

The first approach algorithm bases the decision to keep or remove a point on its position relative to its direct neighbors. The second approach is an improvement in the sense that it decides to remove a point after comparing its position to the most important points in the entire track.

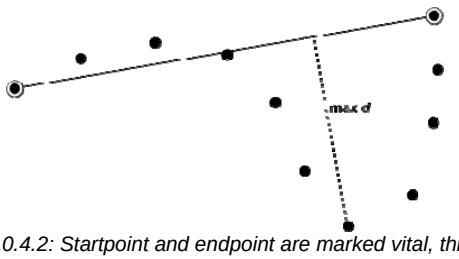

Figure 10.4.2: Startpoint and endpoint are marked vital, third vital point found

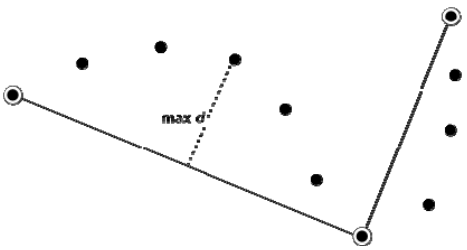

Figure 10.4.3: Fourth vital point found

In the second approach, a number of points can be marked 'vital'. The last action of the algorithm is to remove all non-vital points from the track. In the beginning, no points are marked vital, except for the start point and the end point of the track. While the maximum number of vital points has not been reached, the algorithm marks a non-vital point as being vital, exactly one for a repetition of the loop. The point being marked vital is the point for which *d* is maximal, where *d* is the shortest distance between the point and the line through both vital neighbors of the point. The behavior of the second approach is outlined in Figures 10.4.2 and 10.4.3.

10.4.3 Final approach

The final algorithm is a combination of the two approaches. The algorithm starts by finding all points that are (almost) on one line with both direct neighbors using the first approach. These points are marked 'least important'. The remaining points are ranked in order of importance; the start point and end point are of highest importance, the third vital point is next, etc. After this calculation is done, the path reduction remains a simple operation on the list of track points.

10.4.4 Wild point detection

A GPX track can contain “wild points”; points for which the position has been measured incorrectly. A fairly simple algorithm has been implemented for the detection of wild points. This algorithm requires all points to contain a position and a time attribute. Using these attributes, one can compute the average traveling speed between any two points. The algorithm computes the average traveling speed between any two consecutive points. If the average speed exceeds some user-defined top speed, the destination point is considered to be a wild point. The algorithm returns a vector of wild points. The top speed is not a fixed value, because for example a hiking user would set a different top speed from a biking user.

10.4.5 Converting from coordinates to distances

The algorithm as presented above depends on the distance between any two points. The input given to the application consists of coordinates. The conversion from coordinates to distances is not a trivial one, because the earth is a sphere. The conversion function in the implementation is a standard function taken from the Internet.

11. Verification and Validation

During the project development, three types of testing were performed. This section describes all three types and when they were performed.

The first type of testing was performed by the individual code developers immediately after implementing a feature. This type of testing was component and integration based. The developers were expected to test their features against all boundary cases they could think of and the exact testing methods were left to their discretion. These tests exposed the most bugs.

The second type of testing was performed before milestones M012 and M016 (for more information see section 6), because those milestones involved live demonstrations of the application. For every demonstration, a common usage scenario involving newly implemented features of DSD GPXCleaner was created and thoroughly manually tested before the demonstration itself. This testing exposed a number of bugs at the system and integration levels.

The third and most significant type of testing was the final acceptance testing performed on the application, at the system and integration level. Acceptance testing is a black-box testing technique which aims to verify that the final product meets the specified requirements. Our approach included manual requirements-based functional testing, which aimed to verify that each functional area of the system behaves correctly from the user and business perspective according to the specified requirements, and stress testing, which was performed to measure and ensure a certain level of effectiveness under unfavorable conditions (this test was carried out by overloading the map handler with huge input data to measure its availability under this condition). This section contains a brief overview of our project acceptance test plan, for a complete overview testing see the “DSD GPXCleaner Acceptance Testing Plan” document included in the project documentation [9].

First, we created the functional requirements definitions, a few of which are shown below.

ID	Description
TR1	Load tracks from a GPX file
TR2	Save tracks into a GPX file
TR12	Reduce points of a segment
TR23	Switch from Simple mode to Expert mode and vice versa

Table 11.1: Partial functional requirements definition

Then, we defined a number of test cases to test the functional requirements described above.

ID	Description
TC1	Perform the loading GPX file operation by using an empty file. The tool shouldn't raise any error; just not open the empty file. It tests the requirement TR1.
TC2	Perform the loading GPX file operation by using a badly formatted file. The tool shouldn't open the wrong file and advice the user about the failure. It tests the requirement TR1.
TC3	Perform the loading GPX file operation by using a correctly formatted file. The tool should open the file and visualize the parsed information. It tests the requirement TR1.
TC37	Reducing points of several selected segments. The tools should not reduce anything since it is not possible to reduce several segments at the same time. It tests the requirement TR12.
TC61	Switching from simple mode to expert mode. The user in simple mode clicks on the "Window->Switch to Expert mode" and the tool should delete every item from the main frame of the GUI, and repopulate it with the simple GUI ones. The lists will maintain the info from the previous mode and the embedded map should keep its state. It tests the requirement TR23.

Table 11.2: Partial test cases definition

Then, a responsibility matrix was created, assigning individual tests and test cases to particular testers. The tests were carried out and the detected failures documented and corrected.

At the time of writing of this document, the testing hasn't been fully completed. The following table describes the results of four latest failed test cases, the cause of the failure, and the steps taken to correct the problems.

ID	Test case description	Description of the encountered problem	Description of the solution
TC2	Perform the loading GPX file operation by using a wrong formatted file. The tool shouldn't open the wrong file and advice the user about the failure. It tests the requirement TR1.	The test case was failed: a file containing a track point with the latitude attribute set outside the accepted range was not rejected.	The subroutines reading attributes and elements with limited ranges were modified to throw exceptions in case of an illegal value and the GUI module was modified to catch the exception and notify the user.
TC5	Perform the saving GPX file operation when the user does not have the right to write. The tool should not perform the operation since the user does not have the rights for the requested operation. It tests the TR2.	The test case was passed, but the error message was unclear (and contained debugging information).	The error dialog was modified to present the user with a clearer error message.
TC9	Split a selected track at a selected segment. The tool should create a new track and put into it the selected segment; lists are updated. It tests the requirement TR4	The test case was passed, but the embedded map did not immediately refresh after the action, to reflect the new state.	The track splitting subroutines were modified to refresh the map immediately.

TC17	Move up a selected track. The tool should move up the selected track in the list of one position. It tests the requirement TR6	The test case was passed, but the selected track was dropped out of preview mode, and the selected segment out of editing mode (if selected).	The subroutine in question was changed to reapply the preview and editing mode settings if they were previously set.

Table 11.3: Example test case verification results

During tests that require properly formatted GPX data, three files were used as input data.

Filename	Description
tuolumne.gpx	A file containing 1 track, 1 segment and 5249 points. This file was obtained from the World Wind package. Contains data from a GPS device.
Track-2007-10-13.gpx	A file containing 1 track, 4 segments and 2282 points. Obtained from the customer. Contains data from a GPS device.
test1.gpx	A file containing 1 track, 1 segment and 6 points. Created with the DSD GPXCleaner application.

Table 11.4: Partial list of test files

12. Outcomes and Lessons learned

The requirement of the project was to build a program that would allow reading data recorded by GPS devices in GPX files and making it suitable for use with online services by reducing the number of recorded points. This key requirement was fulfilled.

The program not only features the main requirement but also a lot of other features. We have embedded the NASA World Wind map for visualization of user tracks and their easy manipulation; the number of points displayed on the map is based on the current zoom on the map. With both simple and complex interface modes, a user can either access only the most commonly used features or use all the features the program has. DSD GPXCleaner can detect wild points, reduce them with or without user supervision, reduce the number or GPS points by user given amount and calculate the deviation from original track. With DSD GPXCleaner it is possible to add tracks, remove them, join them together, split them, arrange their order in GPX files, and do the same with segments inside of every track. It's possible to add and remove points, move them on the map, change the time associated with them, and rename them. DSD GPXCleaner also has an automatic backup feature. We use only a part of the GPX standard that is most important and most used with internet services; a future development implementing the support for all features could be made. Also the KML standard for GPS tracks is becoming more popular so support for it could be a good improvement to the application.

With the program, we have also written complex documentation describing all the features, how they work, how they were built and what was the development process. All of this can be found on the web page hosted by the DSD course: http://www.fer.hr/rasip/dsd/projects/gpxcleaner.

The road to here wasn't an easy one. For all of us this was the first distributed project with members located in different counties with different cultures without the possibility to meet in person. The distribution part was the largest problem we faced. We had to arrange teleconference meetings; we used Skype for weekly meetings and used a Google Group and email for a lot of discussion about the project and problems. A Subversion server was used for code management between all the members.

As this wasn't only a SCORE project but also a part of a university course, "Distributed Software Development", all members of the team had DSD supervisors to whom we had to provide weekly reports, periodically present the state of the project and meet milestones; all of this was new to us.

After all the new things came the project. We had basic requirements given to us by our customer Michal Young. With them we proceeded to work out what functionalities would our program offer, and make documentation on how they would be implemented. But those weren't final either, the whole time the project ran, we had communication going on relation team–supervisors and team–customer to be sure that our project is doing well and that the customer is satisfied.

For the program itself, we agreed to build it in Java as all of us are familiar with the Java programming language and we found excellent open source software that we could use with our application. During development we faced several challenges we expect to find in the real world, such as shifting requirements (most notably, the addition of the "simple mode"), difficult problems with deadlines (such as the path reduction algorithms) and the need to perform thorough testing both during and after the application implementation.

Besides the concrete outcome – a GPXCleaner application, we believe that the "non-visible" outcomes of this project are even more important; this project resulted in a lot of valuable experience on how to work in a team and how to cope with distance with members that are not always around to help you. We managed to arrange the distributed project and communicate over long distances. Furthermore, this gave us "real world" experience as we learned how to communicate with supervisors and a customer, work out problems and come to solutions together.

For final notice it was a great new experience that will benefit us greatly in future and that no one should miss.

13. Summary

The DSD GPXCleaner application provides several features to allow users to easily manage their GPX data files received from GPS units. The application can be used to handle personal excursion recordings and to set up GPX data files before sharing them with others. DSD GPXCleaner was developed for our customer Michal Young, with whom the application requirements was decided and defined during the project process. The final product fulfills all customer requirements and also provides adiditional features such as a simple view mode for the most frequently used functionality.

The DSD project was run in eleven weeks by a distributed project team consisting of seven team members from five different countries positioned at two separate sites; Croatia and Sweden. The project was part of the university course, DSD, and the work load for each person was therefore defined as 20 hours per week maximum. The project resulted not only in a successful application that satisfied our customer, but also in the achieved knowledge in distributed development and the valuable experiences from the distributed environment considering culture differences, importance of communication and the new techniques we needed to use.

We believe DSD GPXCleaner has potential to become widely used software due to the increasing use of GPS units in outdoor activities and the need for handling the recorded files. Our application is an excellent solution since it is both easy to use and a strong tool for manipulating these records.

14. References

[1] Global Positioning System [online] http://www.gps.gov/ [Accessed 14 Jan 2009]
[2] Distributed Software Development 2008, CDT402 [online] http://www.idt.mdh.se/kurser/cd5610/2008/ and http://www.fer.hr/rasip/dsd [Accessed 13 Jan 2009].
[3] Mälardalen University [online] http://www.mdh.se/ [Accessed 13 Jan 2009]
[4] Faculty of Electrical Engineering and Computing, University of Zagreb [online] http://www.fer.hr/en [Accessed 13 Jan 2009]
[5] Skype [online] http://www.skype.com/ [Accessed 13 Jan 2009]
[6] Unified Modeling Language™ [online] http://www.uml.org/ [Accessed 14 Jan 2009]
[7] GPX (the GPS Exchange Format) [online] http://www.topografix.com/gpx.asp [Accessed 14 Jan 2009]
[8] SCORE Software Engineering Contest [online] http://score.elet.polimi.it/ [Accessed 14 Jan 2009]
[9] DSD GPXCleaner documentation [online] http://www.fer.hr/rasip/dsd/projects/gpxcleaner [Accessed 14 Jan 2009]
[10] Google Groups [online] http://groups.google.com/ [Accessed 14 Jan 2009]
[11] NASA World Wind [online] http://worldwind.arc.nasa.gov/index.html [Accessed 13 Jan 2009]
[12] Eclipse [online] http://www.eclipse.org/ [Accessed 13 Jan 2009]
[13] Jigloo [online] http://www.cloudgarden.com/jigloo/ [Accessed 13 Jan 2009]
[14] NASA Open Source Agreement [online] http://opensource.arc.nasa.gov/page/nosa-software-agreement/ [Accessed 13 Jan 2009]
[15] KML [online] http://code.google.com/apis/kml/ [Accessed 14 Jan 2009]
[16] Subversion [online] http://subversion.tigris.org/ [Accessed 14 Jan 2009]
[17] Java [online] http://www.java.com/ [Accessed 14 Jan 2009]
[18] Extensible Markup Language (XML) [online] http://www.w3.org/XML/ [Accessed 14 Jan 2009]