Faculty of Management Science and
Informatics
University of Zilina

GIS Accessibility

Project Development Plan

Michal Bors
Pedro Maurício Costa
Tiago Reis
Vladimír Dobos

October 30, 2006

Contents

1 Overview of the Project

1.1 Purpose and Scope

The *GIS Accessibility* project intends to build a free stable platform to do the accessibility analysis of a geographical area, using the public transportation network, based on a defined measurement unit.

The development team was proposed with two different approaches to the problem: a deep analysis to a small geographical area or a more general analysis to a larger area. The last approach was chosen due to its potential to do a better analysis, taking in consideration the possibility to introduce some complexity by adding variables such as public transportation, as buses, and different kinds of roads as well as diverse demographic information.

The analysis will focus on the transportation needs of people who wants to frequent *Entertainment/Recreation* infrastructures - *Swimming pools, Sky centers* and *Movie theatres* - within three counties around the city of *Zilina* - *Zilina, Martin* and *Dolny Kubin*. The *Entertainment/Recreation* points allow us also to do a more detailed and complex analysis in different parts of the year, since the *swimming pools* are more visited during the summer, the *sky centers* during the winter and the *movie theatres* along the year. To do so, the measurement unit chosen was the distance and time from the origin to destination point.

1.2 Objectives

The main goals that the team intends to reach for the system are:

- release an application that supports the accessibility analisys of a geographical area, under *GNU General Public License*[GNU, 2006];
- to develop a multi-platform well documented *GIS Accessibility* application - operating system independent;
- to build a stable *Application Core:*
  - that implements accessibility algorithms based on *distance/time* using transportation network*;*
  - geographical area independent;
  - destination points independent;
  - well defined *Communication Module* with the *Visualizer*;
  - *Visualizer* independent;
- to build a *Visualizer*, that will act together with the *Core* as a standalone *Application;*
- to write detailed documentation that will help its future development.

1.3 Project Deliverables

The *GIS Accessibility* project is composed by the following deliverables:

- *Project Development Plan & Briefing Presentation* will serve to present how the team plans to develop the *Software Application*, by delivering a formal *Project Development Plan* and present it as a short *Briefing Presentation*.
- *Software Application & User Manual* consists on the final usable product.
- *Project Development Report & Final Presentation* have the purpose to present the final project results, including a formal *Project Development Report* and an *oral presentation* where will be shown not only the final product capabilities, but also an overview of the development process.
- *Website & Poster* are the *marketing* components of the project, since they will publicity and present the final product capabilities.

1.4 Schedule Summary

The project consists mainly in three stages: *planning*, *development* and *presentation* phases. See the *GIS Accessibility GANTT chart* in the *Appendices* section for a full detailed schedule.

Planning

This stage takes approximately two weeks long to be complete and it's dedicated to planning the evolution of the project as well as the necessary data collecting, to be used in the next phase.

Development

The final product will be developed at this stage, for a five weeks period, and includes all technical planning aspects, implementation and *debugging*.

Presentation

This stage is dedicated mainly to documentation production during the last two weeks: *marketing* components, as well as *user* and *software application* documentation.

2 Project Organization

2.1 Internal Structure

The organization of the project has a very simple structure, as is possible to see in the *Organization chart*.

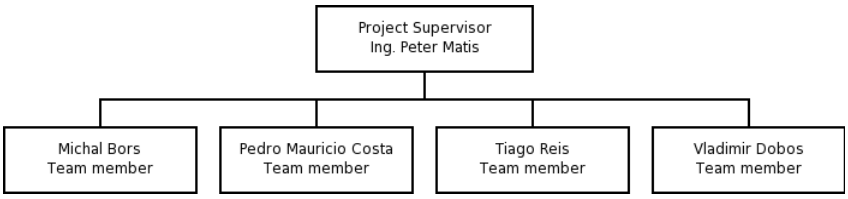

Figure 1: Organization chart

Each member has the same level of responsability within the project. The team is not divided in smaller sections, since it has few collaborators, and it intends to follow an informal, yet organized and well planned, development process.

2.2 Roles and Responsibilities

The roles and responsibilities existent in the project are listed below, as well as the person who plays it, in the *Project Roles and Responsibilities*.

Role	Responsabilities	Person
Project Supervisor	Supervise overall project and assure development in accordance to requirements	Ing. Peter Matis
Data Collector	Collect and compile necessary data to the project analisys	Michal Bors, Vladimír Dobos
Project Planning	Define main guidelines to project development	Pedro Maurício Costa, Tiago Reis
Developer	Define technical details, implementation and testing	Michal Bors, Pedro Maurício Costa, Tiago Reis, Vladimír Dobos
Documentation Writer	Produce technical and user oriented documentation	Michal Bors, Pedro Mauricio Costa, Tiago Reis

Table 1: Project Roles and Responsibilities

3 Technical Process Plan

3.1 Process Model

To implement the project it will be used a mixture of two software engineering methodologies, *extreme programming (XP)* and *spiral model*, which are both agile software development concepts. This chapter will provide a small introduction to both of the methods and then the definition on how they are going to be applied in this project

3.1.1 Extreme Programming

Extreme Programming (XP) is a software engineering methodology, like other agile methodologies, but it differs from traditional methodologies primarily in placing a higher value on adaptability than on predictability. Proponents of *XP* regard ongoing changes to requirements as a natural, inescapable and desirable aspect of software development projects; they believe that being able to adapt to changing requirements at any point during the project life is a more realistic and better approach than attempting to define all requirements at the beginning of a project and then expending effort to control changes to the requirements.

3.1.2 Spiral Model

The *spiral model* is a software development process combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts.

The *spiral model* was defined by Barry Boehm in his article A Spiral Model of Software Development and Enhancement from 1985. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration matters. As originally envisioned, the iterations were typically 6 months to 2 years long.

Each phase starts with a design goal and ends with the client (who may be internal) reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

3.1.3 Spiral Model & Extreme Programming

In this project we will apply a something of these two techniques, from extreme programming we will obtain the adaptability and the standard to produce code, "Two is better than One" this means that at no one is supposed to be implementing code alone, it's supposed two always have two persons looking to the same monitor, to the same code, with this we plan to seek a code with less errors and less bugs.

The spiral model will provide us the biggest part of or coding phase of the project, with our project divided in two major parts, the features first, and then the visualizer. This model enable us two have two distinct parts of the project

created exactly the same way. The spiral that we are going to implement is, due to the schedule of the project, simplified, this means that we will not use all the parts of that model.

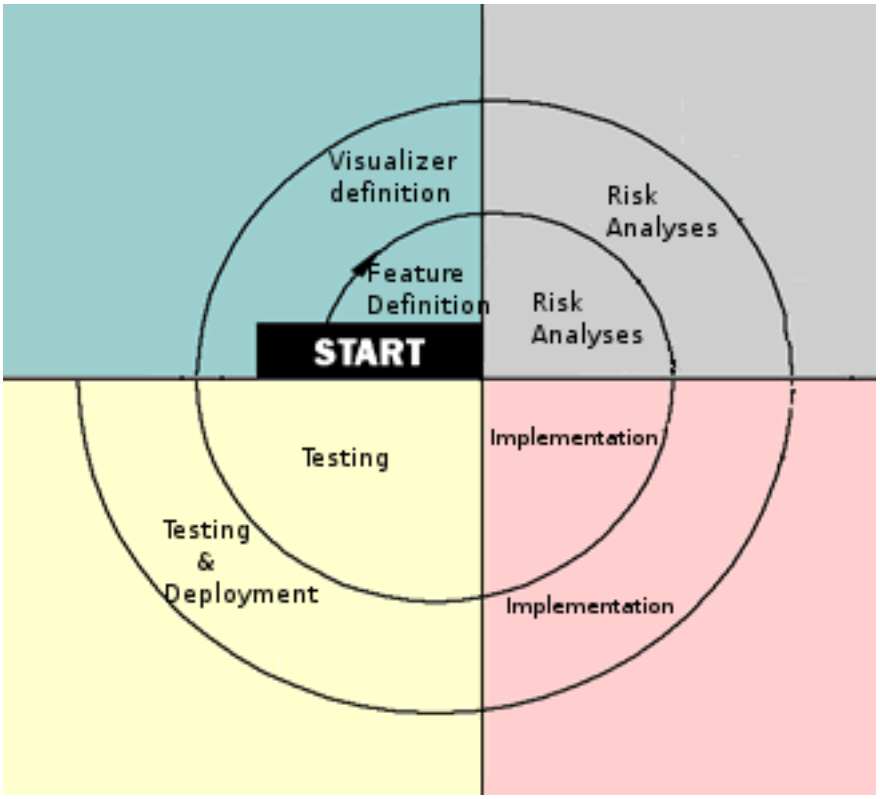

Figure 2: Spiral Model

As it is possible to see in the above model, the project is divided in two distinct implementation phases, the first one is composed by the creation of the core functionalities of the program and the second with the visualizer of the program. In both parts before the implementation is necessary to define very well all the features in each phase so that the implementation team won't loose itself with work not necessary. Next it's necessary to study what can go wrong and how to avoid it. Following that study the phase of project is implemented, when the implementation is over all the features should be tested with unit tests [Wp Test, 2006].

3.2 Methods, Tools and Techniques

3.2.1 Code Standards

We will define some code standards to ensure that all the code is created in the same style.

Names

- All the classes names must respect *Pascal Casing* [1]

```
public class HelloWorld { ... }
```

Figure 3: Class name standard

- File name should match with class name, in the previous example the file should be named *helloworld. (extension)*
- All method names must respect *Pascal Casing*

```
public class HelloWorld
{
        void SayHello(string name) { ... }
}
```

Figure 4: Method name standard

- All variables and method parameters must respect *Camel Casing*[2]

```
public class HelloWorld
{
    int totalCount = 0;
    void SayHello(string name)
    {
        string fullMessage = "Hello " + name;
        ...
    }
}
```

Figure 5: Variables and method parameters standard

- Do not use "_" in variable names

[1] Pascal Casing - First character of all words are Upper Case and other characters are lower case.

[2] Camel Casing - First character of all words, except the first word are Upper Case and other characters are lower case.

- Use Meaningful, descriptive words to name variables

Indentation and Spacing

- Use *TAB* for indentation, do not use *SPACES;*
- Comments should be in the same level as the code;
- Curly braces (*{}*) should be in the same level as the code outside the braces;
- Use one blank line to separate logical groups of code;

```
bool SayHello ( string name )
    {
        string fullMessage = "Hello " + name;
        DateTime currentTime = DateTime.Now;

        string message = fullMessage + ", the time is : " + currentTime.ToShortTimeString();

        MessageBox.Show ( message );

        if ( ... )
        {
            // Do something
            // ...

            return false;
        }

        return true;
    }
```

Figure 6: Indentation and Spacing standard

- There should be one and only one single blank line between each method inside the class;
- The curly braces should be on a separate line and not in the same line as *if, for*, etc.

```
if ( ... )
        {
            // Do something
        }
```

Figure 7: Curly braces standard

- There should be one and only one single blank line between each method inside the class;
- Use a single space before and after each operator and brackets;

```
if ( showResult == true )
        {
            for ( int i = 0; i < 10; i++ )
            {
                //
            }
        }
```

Figure 8: Operators and brackets space standards

3.3 Project Logic Architecture

The Logic Architecture adopted by the Development Team is composed by three main components: *Core, Visualizer* and *Communication Interface*. The main goal of this type of architecture is to allow the development of a modular application that can easily be perfected in the future. This way the *Core* module will be first developed using *Java* supported by an existent *GIS Library*, keeping in mind that it must have a well defined *Communication Interface* to connect to the *Visualizer* module. This language was chosen due to it's strong platform independent capabilities.

The *Visualizer* module will be developed above the previous *Core* module, using the *Communication Interface* to interact with it. The *Visualizer* to be developed will be also based in *Java* as a standalone application, however, it is possible to develop other types of *Visualizers*. For instance, a web-based application basing the *Visualizer* on *ASP.NET*, communicating with the *Core* in the same way as the standalone version.

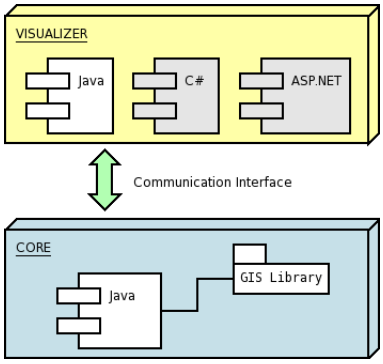

Figure 9: Software Application Architecture

References

[DIR, 1993] Department of Information Resources (1993), *DIR - Planning Template: Project Development Plan*, [available at: http://www.dir.state.tx.us/eod/qa/planning/projplan.htm]

[DOTNET, 2006] DOTNET Spider (2006), *Coding Standards and Best Programming Practices*, [available at: http://www.dotnetspider.com/tutorials/BestPractices.aspx]

[GanttProject, 2006] GanttProject.org (2006), *GanttProject.org - Home*, [available at: http://ganttproject.sourceforge.net/]

[GNU, 2006] Free Software Foundation - GNU Project (2006), *GNU General Public Licence*, [available at: http://www.gnu.org/copyleft/gpl.html#SEC1]

[Wp Agile SD, 2006] Wikipedia (2006), *Agile software development*, [available at: http://en.wikipedia.org/wiki/Agile_software_development]

[Wp XP, 2006] Wikipedia (2006), *Extreme Programming*, [available at: http://en.wikipedia.org/wiki/Extreme_Programming]

[Wp Architecture, 2006] Wikipedia (2006), *Software* Architecture, [available at: http://en.wikipedia.org/wiki/Software_architecture]

[Wp Spiral, 2006] Wikipedia (2006), *Spiral model*, [available at: http://en.wikipedia.org/wiki/Spiral_model]

[Wp Test, 2006] Wikipedia (2006), *Unit test,* [available at: http://en.wikipedia.org/wiki/Unit_testing]

Appendices

GIS Accessibility GANTT chart