Report

Restaurant Finder

Enrique Cordero

Sergiu Ghita

Hampus Gunnrup

Linhang Nie

Table of contents

Introduction

Background

With many types of resources in the area, with plenty of seafood from the coast, fresh vegetables form the south and mushrooms from the northeast it is not weird that Gothenburg is rich in its food culture. There are more than 650 restaurants in the Gothenburg city area and over 300 more restaurants in the surrounding areas, so Gothenburg is definitely a restaurant rich city. There are four exclusive Michelin stars awarded restaurants. There are plenty of streets that are food and entertainment oriented and the culture incentivizes you to enjoy them. With the classis Swedish fika, the brunch, lunch and dinner, restaurants and cafes have a big influence in the social life of the Gothenburg inhabitants.

Purpose

The purpose of the project is to have a more centralized and simple way to find the type of restaurant you want to visit at a specific point in time. The application will help the customer find a restaurant based on different parameters. It will also provide a way to rate them and to store those reviews in order to remember which restaurants they have visited and what they thought about it. From the restaurant owner point of view, the application will help the owner to see first hand what customers thought about their experience and to modify the information provided along with the restaurant. The application is targeted for restaurant owner who want to reach their customers in a better way and for people who want a tool to find the right restaurant experience they were looking for.

Material

Several tools were used to help with the development of the project, with the main tool for developing the application being Netbeans. This IDE was chosen mostly because of the GUI building feature, but proved useful in other areas as well, like the support for maven, which was used to organize imported packages. Maven did, however, make it difficult for the group to make an executable file of the program, so time was spent on fixing this, which the group did not expect. The GUI- building feature

also proved to have some negative features. The auto generated code, for instance, was sometimes hard to manage due to the lack of editing support.

Because of the lack of experience in software development, educational and informational sources such as Stackoverflow and java documentation, proved to be highly valuable. Without either of these, the development of the application would not have been as successful. Stackoverflow, along with other websites such as Youtube and various forums, were especially important due to the amount of experience amongst the people contributing their knowledge.

For communication, Facebook was the main tool. The possibility to create groups and chatting was of great use, but since Facebook contains a lot of content unrelated to the group, it was not optimal to use. For example, during a group discussion, team members could a lot of times be interrupted by people outside the group wanting to chat with them, or the content posted on Facebook could be distracting. The other only communicational tools, was email and texting, which was not used very much.

Facebook and email, along with Dropbox for a short period of time, were the tools used for version control through most of the development. At first this method was working, but after a while it became very messy, and around the beta demonstration, an error occurred due to the amount of version of the application available. Despite this error, Facebook and email remained as the main tools for version control, until the end of the development, in the debugging phase, where the group decided to use Github, which had a large effect on the productivity. Github help not only with version control, but with the commitment amongst team members.

Method

The following methods and practices have been proposed throughout the project in order to augment the development process:

- SCRUM:s Chosen as the management framework of the project due to its agile nature, focus on incremental and iterative development and familiarity. The sprints, sprint planning meetings and task division were the most used methods of SCRUM.
- Test driven development: Proposed for reducing the risk of introducing bugs during the implementation process.

- Spring: Chosen for minimizing focus on boilerplate or "plumbing" code.
- Model-view-controller pattern: Proposed for separation of concerns, which aids in task division and maintainability.

What we did

The initial project plan was not largely influenced by the four tasks that were handed out in the beginning of the course. Instead Google and the literature from parallel courses, was the main source for creating the project plan. The four tasks were, however, helpful in gaining an overview of different software engineering related elements. This helped with making better discussions within the group, and helped the individual members understand each other better.

The four assignments, along with the second presentation, were also good preparation for the collaboration with other teams. These preparations, allowed the teams to practice the soliciting and sharing of requirements, which was handled in much the same way with the translator. It did, however, not make the collaboration go completely smoothly, and by the end of the development, little to no communication occurred.

Our initial choice of a software process was agile, due to the fact that the developing team was inexperienced. Because of this the risk of requirements and design changes was pretty high. In order to minimize it, agile was the adequate choice as a software process. Incremental approach was proposed as a common technique of development for all members, but each member could tailor their own process when working on the scheduled tasks.

A team leader was not been chosen since none of us possessed enough knowledge to take this responsibility. The team had regular meetings and the following decisions were made:

Pre-Beta Version

The initial process was to divide the group into two teams. One team would take care of the GUI and the other team would set up the database. This structure was used in the mentioned incremental approach with only Facebook as a communication tool. Version control was practically non-existent. Because of correlation between the database and the program itself both teams were crossing borders all the time and trying to do what was needed to finish a part of its own team. So for example the

database team did some GUI because they needed to try out their database and the GUI team did a bit of the database because they needed to test their GUI.

Beta Version

At the time the beta consisted of a database with the users, owners, and administrators table and the restaurants table. There would be a capability to filter the restaurant through different possibilities and there would be an option to add and delete restaurants. Also the users would be able to log in and out of the program and they would be able to register as new users.

Post-Beta Version

After the beta version, we received some feedback from our classmates on things to improve. The group acknowledged the fact that a lot of how things were done was not very productive. The team decided to move towards the MVC pattern basing our project on the Spring framework. Maven would be used as a dependency tool and the group would work using SCRUM as the software process. Also it was agreed better communication through the Facebook group was needed and that meetings would happen twice a week on Mondays and on Thursdays to discuss the advance and to distribute new tasks.

Final Version

By this point the advance was much better. The biweekly meetings helped to update on what advancements where being done. There was a decision to start working more closely on version control. Because of the lack of sharing of versions, the task to put the application together was becoming a nightmare. At the ends there would be four people making updates from different fronts and therefore a better version control method was needed. The group decided to adopt Github as the version control manager and the application grew and got debugged at a very fast pace.

Regarding the database we enriched our knowledge pool by referencing some sources of knowledge and technique. Because the ideas that were delivered in the lecture are core but limited for certain further area. It was helpful for beginners but time consuming.

Our group designed a simple ER relation. First we defined the entity with corresponding attributes and came up with the relationship between entities records. We structured them down by drawing a diagram using a drawing tool called draw.io.

Then we transferred the model to real tables by applying converting patterns. The primary schema of our early database was done.

In the final version, the restructuring of a database with more complex features was needed. So we found some tools and tried to design a more complete database. The first design tool we used was Mysql workbench. But unfortunately, we got an issue on connecting to server. We even posted some question in several community forums asking for help, but none of them solved our problem. Then we turned to PowerDesigner, which worked. After testing, we integrated the changes to the previous schema to make it meet the requirement of the new system.

Some technique we noticed in the initialization of this project was successfully adopted. One of them is trigger in database. We noticed that if a restaurant is deleted, the related comments should also be deleted at the same time. So we adopt triggers to execute cascaded operation, which is to ensure the reference dependency. On the other side the index was not applied in our system, since the scale of data would not be very large, the benifit bring by index was limited. Considering that the speed of query is acceptable, we didn't build extra index on our database. But when the scale of database increase to a large level, it should be take into consideration. Stored procedure is another feature we prepared but not applied, since the requirement of data manipulation of this system is not high. But we will take it into consideration if any of the following is required. If we are required to optimize the execution efficiency of database, since the stored procedure is precompiled, the efficiency will be obviously promoted; for example the database of a mobile app. Stored procedure can significantly reduce the data flow, since it just pass parameters instead of the whole SQL statement. And if the data security is crucial, we can use it to prevent attacks from inserting SQL statements.

Conclusion and disscusion

In this part the team will be commenting specific issues we faced during the project and what solution we arrived at to fix this problem.

Task Division

Since the responsibility was declared equal, yet ambiguous, among team members, it proved difficult to assign specific tasks in the project plan to one member or another. Together with the team not being lead by someone, it resulted in generating a schedule with broad and un-assignable goals.

Instead of referring to "fundamental requirements" in the project estimates chapter, requirements should have been specified in such a matter that, "X feature shall be designed by Y member by the Z date". "Translator component is developed for the other team" should have mentioned more detailed individual phases, similar to those of software development. Thus specifying components or phases in detail, assigning the responsibility and scheduling deadlines for their completion would have increased the quality of work, productivity and enabled proper monitoring of the project's progress.

Another negative affect of not assigning roles is that the project plan has never been updated for the second phase, which was described as the period after the beta presentation deadline. Phase two of the estimates chapter describes yet again very broad activities such as further requirements engineering, implementation, testing and deployment. Installation and usage documentation needed to be written and cross platform testing needed to be done; yet they were completely left out. Apart from inexperience, a major contributor to this particular issue was lack of time, as it was written very close to the submission deadline. As previously mentioned earlier, the solution would have been better management: someone to be our leader.

Revision control

Revision control proved challenging. As the team started developing the application in Netbeans, the process of publishing the update supposed exporting the project to a zip file and updating it to Facebook mentioning the changes that were made. Eventually multiple different versions of the application were produced in a short amount of time. Significant resources were wasted on making sure the application a

member was working on was up to date with the rest of the team's versions. And even so, there was a case of further development being committed on a bugged version, even though the bug was fixed at some point.

Version Control

Another disadvantage of not using version control software is that every member had to work strictly on what they were responsible for, unless they talked with another member or the entire team. For instance if one member was working on the "Restaurant Edit View" and one on the "Search View", one could not modify a component on which both views would be dependant (i.e. RestaurantDAO, SQLTranslator, DBHandler) without making sure the other person would accept that.

Github was introduced rather late in the project, but it was essential for the project's success. Productivity, cooperation and quality increased drastically as work could be done safely on almost any part of the system.

Test Driven Development

Test driven development has been proposed initially for all its benefits. It would allow the team to easily test the whole application making sure that any new implementations of features do not affect other parts of the system. It also encourages use case writing, minimal risk of bugs being introduced and clean interfaces.

Despite the compelling benefits, the team had no knowledge on how to adopt this practice. Small time frame combined with having higher priority tools and methods that required training as well, determined the practice of test driven development to be quickly abandoned from the very beginning of the project.

Due to the many bugs being easily introduced in the system, especially in new components, identifying and correcting mistakes most often than not consumed more resources than writing the initial code. It is obvious now that this practice's implementation should have been given higher priority. Better management may have allowed more resources to be invested in training. Inclusion of this practice to be taught at one of the courses would have been also helpful.

MVC

Division of work was naive in the beginning. With a not-so-detailed list of requirements, the team comprised of four members was split into two, one half had to

handle the front-end and the other half the back-end. Due to lack of communication and the members being eager to start coding, both teams ended up developing both front-end and back-end parts of the application. The team reassessed the way it was distributing work among members and concluded it needed to follow an architectural pattern: Model-View-Controller.

The separation of concerns that MVC enforces enabled the team to distribute work by views, thus each member had to work on a specific view, developing its controller and models. If increments were affecting common components (such as data related classes, main controller) the team would have to get together and talk about an appropriate solution.

No Leader

One big problem that the group faced was the lack of leadership. No one in the group felt qualified to take that responsibility, and throughout the whole development, no leader roll was present. This made decisions hard to make, and lead to an unorganized development team. In the beginning, each individual went their separate way, and made their own decisions. By the time the beta version was due, the team decided to organize itself by having weekly meetings, which helped a lot with organizing the team.

The lack of leadership, amongst other factors, left the group without any software process to follow. Instead a very open schedule, with long milestones, was followed. After a while this method of organizing the work, became very ineffective, and by the end of the development, the team decided to apply scrum as a software process. There was, however, not any scrum master present, due to the same reason as there was no leader from the begging. Although it was not followed completely, it had a large effect on the productivity, and most of the sprints involved had a successful outcome. It was by this time that the weekly meetings were applied, which also increased productivity. A more fitting solution would have been to apply scrum from the beginning of the project, and tailor it more to fit the team.

Software Process

Using a proper software process would have solved a lot of the problems that the team faced. Scrum would have applied the need for an iterative-incremental approach, and would have organized the work much better. Applying training in sprints,

for instance, would have allowed team members to discover several set backs and difficulties within training, and it would also have made the team have roughly the same amount of information about a specific topic. Instead of applying the training in sprints, the individual team members had to organize the training themselves, making training and development occur in parallel, which made difficulties within training harder to discovered.

While this training method left a lot of discovered tools unused, some tools were highly useful. Netbeans was a tool that proved to be highly useful. Netbeans minimized a lot of repetitive programming, by generating code with quick and easy shortcuts. Within Netbeans, spring was used, which helped with organizing the different classes, and therefore supported the minimization of repetitive programming even further.

Prototyping

Another problem was the lack of prototyping. All of the development occurred directly, without any sort of prototype, and in a lot of cases, certain areas of the code would have to be changed in order to fulfill requests from other team members. If a prototype were put in use in these cases, the requests would have been taken in to account before any real development occurred. Prototypes could also have been shown to stakeholders to get additional requirements, been reviewed by the individual developer to see flaws in the design and give an early overview so other parts of the software can be developed.

Dependency Handling

Dependency handling proved difficult for the project when the group decided to move to a MVC framework. The whole idea behind MVC was that it was possible to modulate views controllers and the data modules in different classes so that all the group members could make increments in different views. The division of work had proven a challenge and therefore it was needed to arrive to a better way of working as a group.

When there are so many modules being worked on at the same time using different libraries and maybe different versions of the same library some errors could arrive. A more specific example came with the release of JDK 1.8 where the java.Time.Format package was included. One of the latest classes was using this and

therefore some of the computers being used could not run the program. This because the IDE does not automatically download all the required elements, but rather just checks its installed JDK for the referred library. Another scenario is when newer versions of libraries deprecate a specific method on a newer version. If the method is being used in the program it can cause errors. Therefore a specific version of the library needs to be able to be specified in the program execution.

Maven was chosen as the tool to manage the dependencies both because it is highly used in real life workplaces but also because it is already integrated in Netbeans. Maven allowed that with a click of a button all the necessary dependencies, both the right library but also the specific version, where downloaded into the project file.

SCRUM

After the beta version was presented the group agreed that the software process we were using was ineffective. Communication was not at its best within the group, the division of labor proved to be difficult and the tasks assigned where too general. Because of its highly agile structure, and its recurrent meetings Scrum seemed the perfect choice as a software process. The team decided to meet twice a week, on Mondays and on Thursdays around noon and have a meeting for two hours to talk about the increment during the week and to see if new tasks could be handed out. The sprint would be measured on a weekly basis (Mondays) and the progress and challenges discussed during the sprint.

This software process increased the productivity significantly since there was a much better and faster division of tasks. Also it helped the members of the group to support each other on difficulties they were facing. Most importantly changes on the structure of the program could be addressed in a timely manner and therefore it saved a lot of time that could have been wasted on unnecessary coding.

Remote or local Database

Our decision support system is designed for various classes of public users, the customer and business owner, to use at anytime. The data should be public accessible, instead of being stored locally. So we decided to use a database that is hosted in the Internet, which enables efficient access by multi-party and ensures the data independency at the same time.

Programming Language

Even though the recommended programming language for the application was Java, there were several group members who did not like the way Java works with GUI building. There were discussions about undertaking the GUI as an HTML program that would connect to java methods to do all the back work. This however presented the challenge of relying on the two members who had knowledge of HTML. Also it minimized the possibilities of asking for help from the supervisors since it was decided that the support coming from them would be based only on java and Sqlite (the recommended languages). The solution came to be the simplest, which was to follow the suggestions and work primarily on Java.

SQL Injection

From the start of the project it was mentioned that SQL injection is a very common problem with programs worldwide. The group tried to find information about how to prevent the injection. This however presented a problem when the SQL translator was to be written by another team. Because of the lack of experience of most students it was quite the challenge both to explain the requirements and to expect them to know about SQL injection themselves. Once the translator was received the team made modifications to apply new methods as the program grew. The problem with the SQL injection was however never solved. This because it entailed taking apart the whole translator and rebuilding it around prepared statements. The solution to this is to be more demanding with the team preparing the SQL translator. Also an example of the expected code could have been attached to the requirements sheet. For future reference the SQL injection could be avoided the following way:

```
String password = request.getParameter("userPass
String query = "SELECT *_balance FROM Users WHERE Password = ? ";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, password);
ResultSet results = pstmt.executeQuery( ); [KIKE STOP]
```

Remote Host Timeout

When running an application a connection with the remote host is done. If the connection remains idle for too long then the remote host will consider this a problem and timeout the connection. This causes the program to stop working because

there is no longer a connection with the database. To avoid this be sure to close the application when not in use.

Sequel Language

We had difficulty choosing a database management system. The teacher because of its resource-saving and embedded feature recommended SQLite, but we finally chose Mysql because we want the database to be remote. The problem was that there is no available SQLite server in the Internet. The only way to access SQLite database is via Internet sharing, which will cause some problem. Instead, there are some online Mysql servers available, for instance the Myphpadmin. And some of the members had previous experience with Mysql. These are the main reason we choose Mysql. Besides, there is more documentation out there that can be referenced. And it's a mainstream DBMS which we believe will bring more relevant career opportunity. In the future, if we were to build system with an embedded database we would consider using SQLite.

GUI Platform Problems

Another problem we found is, GUI is not consistent between different platforms, particularly Max OS vs. Windows. The main problem is some text cannot be displayed in consistent size or length between different OS. This will cause incomplete message delivered. We think it is caused by the operating systems' way of displaying graphical elements. But we cannot tell how in very detail. We would take this factor in consideration and adjust certain components in the initial design specification accordingly.

Merging vs. Auto-Generated Code

Github places some lines in the code anywhere there is a conflict between two versions. This happens when two users try to change the same piece of code. The person who pushes the code second will receive these merging conflicts for him to resolve. On the other side Netbeans has auto-generated code for the GUI builder. The auto-generated code is not fully available for edition and therefore it becomes a problem to change. When the change that both users make has to do with some of the same component in the GUI builder then a conflict will occur in this auto-generated code and therefore it's a conflict that cannot be merged easily. Since this is a problem that occurs

in situations of poor planning (two members should not be working on the same piece of code at the same time) the solution is to plan adequately so that each group member knows which view they are working on and therefore which component they are affecting. This way the conflict will not occur.

Ignored the stakeholders

We ignored some kind of stakeholder. We didn't realize that our teacher and supervisor are stakeholders though we should have recognized them according to the concept of "Stakeholders". Hence the communication with them was not frequent. We think the scrum framework should be enforced in the course from the beginning. It will force us to involve and communicate with the stakeholders regularly in the development process.

Project Requirements too broad

The project requirements were broad. We could not stop guessing what the teacher required. If the requirements would be a bit more refined then guessing requirements would be minimized. Next time we will try to communicate more frequent with the stakeholder to elicit requirement if the requirement is not quite specific.

Concept Misunderstanding

Some of us had misunderstanding on some concepts. During the process we fixed the misunderstanding. For instance, I thought toolchain is what Eclipse provides me, the editing part, the console and etc. Then I realized that toolchain is the set of tools I used in the whole process, for communication, version control, UI design, all of them make up the toolchain.

Conclusion

The team's development effort resulted in a cross-platform deployable product, fulfilling all the specified essential requirements. However, the product is vulnerable to "SQL injection" and connection timeouts, requiring a patch fix. These issues are primarily the consequence of inexperience. Since the application is not handling any critical operations, the team assessed these flaws as non-compromising. The experience of developing the "Restaurant Finder" Decision Support System proved invaluable in our Software Engineering and Management studies. As the project advanced, communication among developers improved and as a consequence so did

productivity. The importance of communication was seen by all developers as being quintessential for development. Software processes, project management (especially task division), software architecture and software security have shown their relevancy for developing software.

The following list contains a summary of the main lessons that will enable us to become better software engineers:

- Every team needs a leader.
- Responsibilities need to be specified and most importantly assigned.
- Managerial effort needs to be assigned, as it is key for the project's success.
- Version control and test driven development should be implemented in any software process.
- Stakeholder involvement should be specified from the beginning of the project.

Sources

- Code for the range slider: https://ernienotes.wordpress.com/2010/12/27/creating-a-java-swing-range-slider/
- www.youtube.com
- www.stackoverflow.com
- http://docs.oracle.com/javase/8/
- http://dev.mysql.com/doc/