[KING'S COLLEGE LONDON]

OBJECT-ORIENTED SPECIFICATION AND DESIGN

Tim Adamachvili
Adam Ellis
Nishant Gurung
Gun Park
Alyaa Alkuwari

[TOTLAL TABLE OF CONTENTS]

[CONTENTS]

1. Intro

The project required us to develop a software tool to act as a scheduling system for agile development. The tool needed to take input descriptions of development tasks and details of the skills and costs of available developers, generating a schedule which assigned developers to tasks. Having read the requirements, our initial response was to further research agile development and identify how best we could implement the methodology within our project, both as our own methodology and in order to better understand the requirements from a client's perspective.

Agile development is defined as an alternative to traditional project management, where emphasis is placed on empowering people to collaborate and make team decisions, in addition to continuous planning, continuous testing and continuous integration.

As a team assigned to ten people that was reduced to seven due to lack of communication, working as a team in making decisions and organising tasks became more important. Unfortunately, our team was further reduced to four people in the final three hours due to the fact that some team members wanted to submit their work individually. This required us to make fundamental decisions, particularly distributing our little left 'manpower' efficiently.

Our goal was to build a scheduling system that optimised development output, by minimizing costs and therefore producing a schedule where work was completed in an efficient manner.

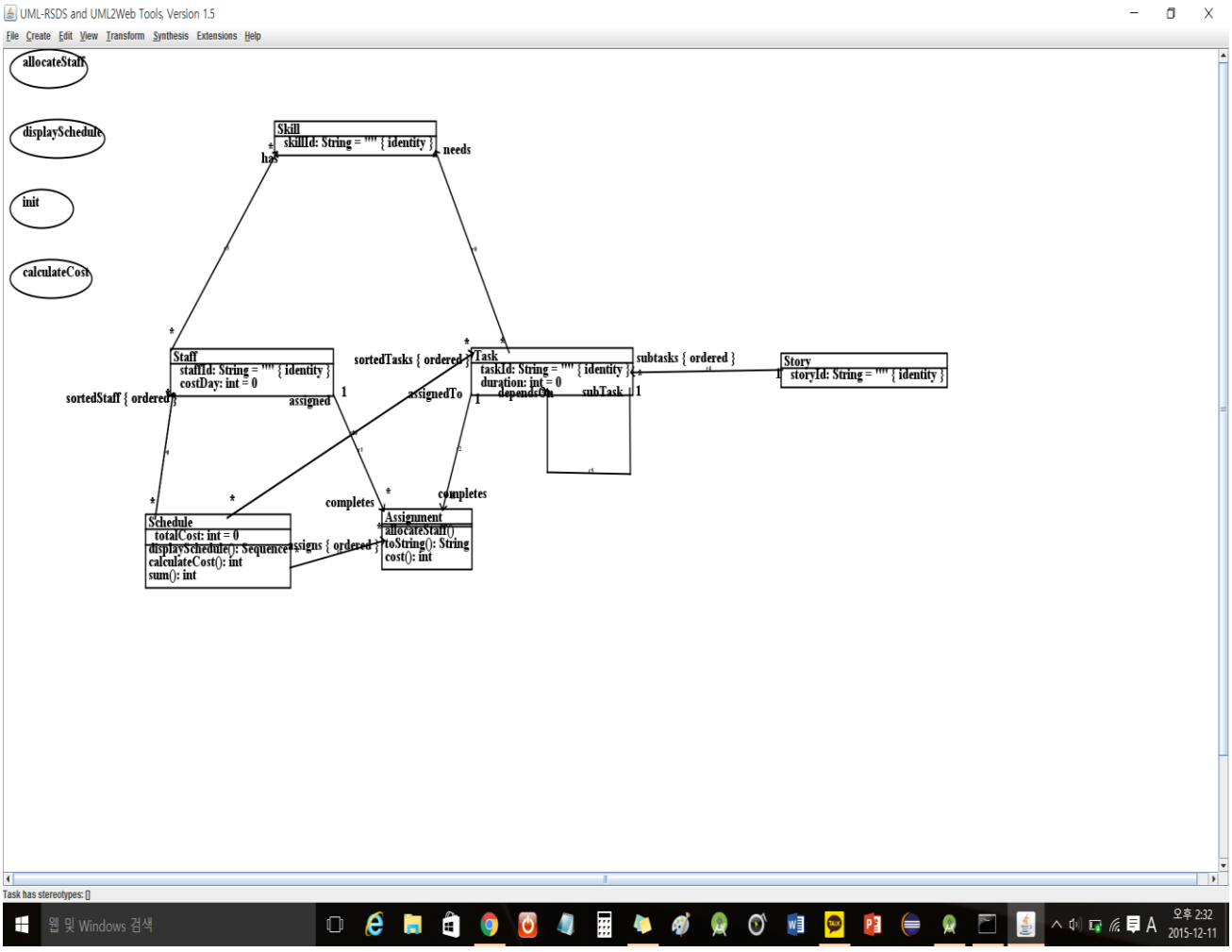
Whilst our intention was to build this software to the best of our ability for our client, not having a specific client at times did cause confusion. Nevertheless, as time progressed, we appreciated our lecturer's feedback as representative of what a client's feedback would be like, whilst realising, in using agile development ourselves, we were in some ways our own client.

2. Our Agile Approach

An agile approach consists of repetitive steps known as sprints. Numerous sprints help the team do well against unpredictable events because each sprint lets the team take a reassessment of the progress so far, taking into consideration outstanding tasks. This is of particular importance when working with unknown team members; unpredictable events are more likely to occur in such cases, which an agile methodology is better designed to cope with.
In fact, our team came across some unpredictable events and had to manage them. Using the agile approach, we were easily able to reassign tasks and react to the consequences of the events. As face to face communication was our primary source of team communication, an agile approach was effective. It helped us discuss reassignment of tasks, while considering the strengths of our members.

Whilst being an advantage, at times, a disadvantage of an agile method is that it requires frequent communication between members. Our team easily overcame this because everyone was able to meet at a regular interval. We also had additional means of communication via group text messages as well as group conversations on the Internet. Flexibility in means of communication was the main reason we could overcome our difficulties.

3. Class Diagram



4.Attributes

The project consisted of 6 classes: Story, Task, Assignment, Schedule, Staff and Skill.

Story has a String attribute, StoryId, which uniquely identifies each Story object.

Task has a String attribute, TaskId, which uniquely identifies each Task object, as well as an Integer attribute, duration. The duration attribute represents the amount of time that is required for the Task to be completed and our goal was to minimise this.

Skill has a String attribute, SkillId, which uniquely identifies each Skill. The usage of unique attributes means that Skill objects are not accidentally duplicated, which would have led to unnecessary computation, potential data anomaly or data inconsistency.

Staff has a String attribute, StaffId which uniquely identifies each Staff as well as an Integer attribute, costDay. costDay denotes the amount the company has to pay to hire the staff. Similarly, StaffId is unique so that Staff objects are not accidentally duplicated. The prevention of unnecessary computation, potential data anomaly or data inconsistency aids in optimisation.

Assignment has no attributes because its role is to link Task and Staff classes together.

Schedule has an integer attribute, duration. This totalCost attribute represents the total cost of the Schedule, which is derived from Staff and Task. The duration attribute denotes the total duration of the Tasks that it consists of.

5.Associations

Associations were made between classes using the UML-RSDS tool, for example to show that certain staff possess certain skills. In no particular order, these associations include:

Story and Task have a one to many association, such that one story has many subtasks, an ordered list of class Task.

Task has a one to many recursive association, where one subTask depends on (dependsOn) many Tasks.

Skill has two associations, identifying its dependence on Task and Staff. Both associations exist as many to many relationships, where many Staff have (has) many Skill and many Tasks need (needs) many Skill.

So too, Assignment has two dependencies represented by Staff and Task. Singular instances of Staff and Task are required to complete (completes) many Assignments.

Many schedules are derived from many assignments, as represented by their ordered association, assigns. In the interest of optimisation, Schedule also has a many to many ordered association with Task (sortedTasks) and Staff (sortedStaff). The Tasks and Staff are ordered in descending order of duration costDay respectively. Optimum efficiency is then achieved by selecting tasks as high up the list of objects as possible.

6. Operation specifications

There are six operations in our class diagram: allocateStaff(), query toString(), query displaySchedule(), query calculateCost() and sum().

allocateStaff()
pre:
post: true

query toString(): String
pre: true
post: result = assigned.staffId + " assigned to " + assignedTo.taskId

cost(): int
pre: true
post: true

query displaySchedule(): Sequence(null)
pre: true
post: true

query calculateCost(): int
pre: true
post: true

sum(): int
pre: true
post: true

7. Project Management

From the offset, we were intent on adopting an agile development methodology. Our reasons for choosing agile development over a waterfall approach were twofold, we felt that due to the versatility of the coursework, and our ever-growing knowledge of OSD as the weeks unfolded, we would be best not limiting ourselves to one level of development. Furthermore, we felt that as the project required the development of a tool to represent agile development, we could better appreciate the task, and the user's requirements by using the same methodology.

Initially, all members of the team were encouraged to research the project to gain a better understanding of the requirements, further to this research all members of the group were then asked to think about the problem independently and contribute their solution for a class diagram - where the preferred solution was to be chosen as our initial prototype.

Following our initial prototype, we elected Adam as team leader, who worked on the class diagram with Nishant, Tim and Damien, whilst Jose was elected to manage the other subteam, who worked on use cases, with Alyaa, Darren, and Gun. We agreed that within the team we needed two members to keep documentation of the project and chose Adam and Darren. Subteams arranged to meet 2-3 times per week, whilst the entire team met approximately 1-2 times per week. Though we originally communicated through email, there were members of our team who either failed to respond to emails or did, but failed to communicate any further, namely: Ahamed Kabeer Mouhamadally, Jacob Reilly-Cooper and Damien Powell.

Having constructed our prototype, we then adopted a combination of rapid application development (RAD), whilst also using evolutionary prototyping, to improve our prototype as we gained a better understanding of requirements, in a methodology that favoured prototype development over intense planning.

At the same time, clear, organised structure was also in place, with subteams further delegating tasks; close to completion of the final prototype, there were around 2 members to each sub-team, delegated to work on the class diagram, the use cases, or the implementation. It was decided that each sub-team would write the report for the part of the project they had developed on. In this way, we ensured a balance of work within the group and that no member was left doing much or too little work.

Using an agile methodology meant that as we wrote the report and realised improvements with our software, we could go back and iteratively, yet continuously, research, develop and test the software.

Having developed the class diagram and use cases, we waited for Darren and Jose to provide the last piece of the puzzle to get the application to function - the implementation.

It's fair to say that whilst waiting for them, had they told us their own plans, we may have had time to produce our own working code. Unfortunately, on the morning of Friday 11th December - deadline day, they decided to break away from the team and submit their work individually. We cannot take credit for the Java code that Jose and Darren worked together on. As a result, we do not have fully working, functional Java code for the project. Nevertheless, we have still generated Java code using UML-RSDS as attached (see output/java_code.txt).

Whilst we saw the appeal in splitting the group further and proceeding via individual submission, we refused to do so, and are proudly submitting this work as a committed team. This is down to the strong project management skills shown by Adam in managing the team at such a difficult time, with credit to the rest of the team as well for their contributions and loyalty.

8. Result of Testing

Having developed the UML constructs and generating our Java code, we ran the following tests using the given software. It should be noted that whilst the code often experiences bugs, we managed to retrieve the following data; the error code is due to the team issues as discussed above. The testing included running the operations allocateStaff() and displaySchedule(). Running the displaySchedule operation produces the following results:

t1, 5, s1, 3

t2, 10, s2, 7

t3, 7, s3, 2.

t4, 6, s4, 8

t5, 4, s5, 3

t6, 9, s6, 8

The first column consists of the taskIDs for each task and the digits in the second column indicate its duration, the third column indicates the staffId, and the fourth column describes the cost of the task.

9. Efficiency Evaluation

As we can see in the above section, the program generates a schedule that assigns a staff member to each task. Unfortunately, not all the test files could be run. However, the ones that managed to run were efficient, and generated the required results.

10. Use case diagram

A use case diagram consists of the relationship, use case and actor. In UML, the actor is defined as one who “specifies a role played by a user or any other system that interacts with the system” and is represented by a bar diagram on human in use case diagram and interact with use case.

A use case is a set of action or steps of the events that interacts with actors and role. So each use case describes function or role and those use cases represent whole functions of the system. And this is shown as an oval. Each use case carries out functions that actors want and those questions are helpful when making use cases. ‘’What is each actor’s task?’’ ‘’Does actor create, save, modify and remove information of the system?’’

When actors and use cases are set then those two have relationships. It can be divided in two different ways which are actors and use cases relationship and relationship of the use cases.
Actor and use case relationship can be declared as association or communication association, because they are represented as actors and use cases communications. Also association divided by five ways according to progress ways. One of the ways called bidirectional associations. Just one of Actor or use case makes associations.

Also, there are 4 use-cases in the report: allocateStaff, displaySchedule, calculateCost, and init.

Use cases and their relationships

- allocateStaff:
  It finds an available staff member who has all the skills, and assigns the task to the cheapest staff. This diagram has relationship with Task class to get taskId attribute. allocateStaff class diagram has relationship with Staff class diagram to find cheapest such staff member. To make use-case for allocateStaff, we use schedule as an entity and make assumption such as t : sortedTasks & st : sortedStaff & t.needs <: st.has & t.completes.size = 0 & st.completes.size = 0. The result is as follows: Assignment->exists(a | a.assignedTo = t & a.assigned = st & a : assigns).

- displaySchedule:
  It displays schedule and print the list of assignments. To do so, it has relation with two classes,which are Staff class and Task class diagram. In Staff class diagram, attributes of StaffId and costDay are needed. In Task class diagram, taskId and duration attributes are needed to print the list of assignments. Therefore, we need to set Assignment for entity, assumption is true and conclusion is self-> display().

- calculateCost:
  calculateCost use-case uses schedule as an entity, assumption is true and conclusion is true. It has a relationship with staff class. From the Task class, costDay and duration attributes are required. s.costDay∗t.duration is for the assignments of the schedule so we can add this to the totalCost of the schedule

- init
  It adds the product of costDay and duration for the assignments of the schedule and add it to totalCost of the schedule.

Actor:
An actor is the user that performs use cases in a system and in our use case the actor is staff

11. CONCLUSION

In conclusion, this report contains research and results relating to our task, to develop software intended to release planning for agile development through resource scheduling.

Through our research and study for this project, we learnt about various UML constructs such as class diagrams and use cases, whilst also gaining insights into the complexities of team-work, as time progressed.