This is the Title of my Thesis

Your Name

December 2012

PROJECT THESIS

Department of Production and Quality Engineering
Norwegian University of Science and Technology

Supervisor 1: Professor Ask Burlefot
Supervisor 2: Professor Fingal Olsson

Foreword

Originally inspired by the Nordic Collegiate Programming Contest (NCPC), it has been held at NTNU every spring since 2007. The format is a five-hour contest with competing teams consisting of one, two or three contestants. A team of volunteer judges write the problems and answer clarification requests during the contest, while another team hands out balloons for each solved problem. Usually a rather hectic affair, it is extremely important that everything is well prepared. The number of teams is often more than 100, with the record being 162 teams in 2011

The contest system that verifies solutions is at the heart of the contest when it is in progress, and needs to be working perfectly at all times. The system must handle several submissions per second, while verifying that each one is correct and runs within the set resource limits. Submissions must show up on the high score list, and when problems are solved the team handing out balloons must be notified. In addition to this there were a lot of other functional requirements having to do with the bureaucracy of organizing the contest

A requirement was that new features could be easily added in the future, and the code was written with this in mind. The project will now become open source, and all programming contest enthusiasts will soon be able to request and implement their desired features

All aspects of this project have been pleasing and delightful for us. The team has exceeded all our expectations and their system will be used for years to come.

Preface

Before there were computers, there were algorithms. But now that there are computers, there are even more algorithms, and algorithms lie at the heart of computing. Designing a system for eager students to hone their skill in the heart of computing has been a true joy

Our group never wanted to settle for adequacy and mere requisiteness. For the past few months, weve taught ourselves a new programming language and framework and used advanced development frameworks - while tackling many social and technical conflicts.

We have ve proven how Ambition is a dream with a V8 engine, as Elvis Presley once said.

The group would like to thank our eager customers, Finn Inderhaug Holme, Christian Chavez and Christian Neverdal Jonassen for their time to meet us and provide constructive feedback. We also owe a big thanks to our supervisor, Hong Guo, for constructive criticism and reflections; without which, we would not ascertain the peak of our own potential

Contents

1 Introduction **2**
1.1 About the course . . . 2
1.2 The Group . . . 2
1.3 About the Customer . . . 3
1.4 About the Contest . . . 3
1.5 Stakeholders . . . 3
1.5.1 Course . . . 3
1.5.2 Product . . . 4
1.6 Goals . . . 4

2 Task Description **5**
2.1 Task Description and Overview . . . 5
2.1.1 Assignment . . . 5
2.1.2 Branding our Product . . . 5
2.1.3 Assumptions and Constraints . . . 6

3 Project Management **10**
3.1 Project Roles . . . 10
3.2 Development Method (Scrum) . . . 10
3.3 Tools/Framework . . . 11
3.4 Project-Level Planning . . . 11
3.5 Work Breakdown Structure . . . 11
3.6 Milestones . . . 13
3.7 Meetings . . . 15
3.8 Resources . . . 16

4 Architecture **18**
4.1 Views . . . 18
4.1.1 Logic View . . . 18
4.1.2 Process View . . . 18
4.1.3 Development View . . . 20
4.1.4 Physical View . . . 20
4.2 Quality attributes . . . 21
4.2.1 Availability . . . 21
4.2.2 Modifiability . . . 21

4.2.3 Performance . . . 21
4.2.4 Security . . . 21
4.2.5 Testability . . . 22
4.2.6 Usability . . . 22
4.3 Patterns . . . 22
4.3.1 Client-Server . . . 22
4.3.2 MVC(model-view-controller) . . . 22
4.3.3 Shared-Data . . . 22
4.3.4 Multi-tier . . . 22
4.4 Design . . . 26

5 Implementation **33**
5.1 contest . . . 35
5.2 article . . . 35
5.3 userregistration . . . 35
5.4 teamsubmission . . . 35
5.5 execution . . . 35
5.6 node_manage . . . 36
5.7 balloon . . . 36
5.8 changeemail . . . 36
5.9 judge_supervise . . . 36
5.10 clarification . . . 36

6 Development **37**
6.1 Towards the milestones . . . 37
6.1.1 Milestone M-01 - Preliminary report. . . . 37
6.1.2 Milstone M-02 - Mid-semester report . . . 37
6.1.3 Milestone M-03 - First release . . . 38
6.1.4 Milestone M.04 - Presentation . . . 38
6.1.5 Milestone M-05 - Beta Release . . . 38
6.1.6 Milestone M-06 - IDIOpen test event . . . 39
6.1.7 Milestone M-07 - IDI Open . . . 39
6.1.8 Milestone M-08 - Final report . . . 40
6.2 Testplan . . . 41
6.2.1 Testing Strategy Overview . . . 41
6.2.2 Testing Coverage . . . 42
6.2.3 Our approach to testing . . . 42
6.2.4 System Test . . . 44
6.2.5 Non-functional testing . . . 46
6.2.6 Risk and Dependencies . . . 47

7 Risk Management Framework **48**
7.1 Risk Identification . . . 49
7.2 Risk Monitoring . . . 49
7.3 Complete List of Risks . . . 49

A Sprints **50**
A.1 Template . . . 50
A.2 Sprint 0 . . . 51
A.3 Sprint 1 . . . 52
A.4 Sprint 1 . . . 53
A.5 Sprint 2 . . . 54
A.6 Sprint 3 . . . 55
A.7 Sprint 4 . . . 56
A.8 Sprint 5 . . . 57
A.9 Sprint 6 . . . 58
A.10 Sprint 7 . . . 59
A.11 Sprint 8 . . . 60
A.12 Sprint 9 . . . 61
A.13 Sprint 10 . . . 62
A.14 Sprint 11 . . . 63
A.15 Sprint 12 . . . 64
A.16 Sprint After . . . 65

B User stories **66**

C Installation guide **69**
C.1 Creating your users . . . 69
C.2 Setting up the environment . . . 70
C.3 Installing required packages . . . 70
C.4 Database . . . 71
C.5 Gunicorn . . . 72
C.6 Supervisor . . . 74
C.7 Multiple execution nodes . . . 74

D Risk List **76**
D.0.1 People Management . . . 77
D.0.2 Budget . . . 78
D.0.3 Schedule . . . 79
D.0.4 Organizational . . . 80
D.0.5 Tools and tools; product . . . 82
D.0.6 Requirements . . . 83

E Product Backlog **84**

Chapter 1

Introduction

1.1 About the course

Our group and assignment has been delegated as part of the course IT2901: "Informatics Project II" at NTNU. The work covers 15 course credits, equivalent to a 50% work position for one academic semester. IT2901 is offered only to those that are enrolled on the NTNU's informatics BSc programme.

The primary purpose of the course is to let students apply their knowledge from other courses. This is rendered through a project for a real customer. The students have to communicate independently with their client, and deliver a software product that answers the client's needs.

Grades are based on the satisfaction of the customers and an evaluation of the development process. The latter will be reviewed through written reports and timesheets, as provided in this document. Furthermore, it is important that students have met the given deadlines and documented their work in a structured manner.

1.2 The Group

The team consists of six members. All the members of the group are completing their BSc degree in Computer Science from NTNU in 2014. We had prior experience working together, and knew each other well. With many shared courses and similar interests, the team are all at a somewhat similar level of competence. However, we have different areas of expertise, and exploiting this has been a key to success on previous occasions. For a detailed description of each member, see the listing below.

Anders Sildnes Throughout his BSc, Anders has been taking courses related to algorithms and program security. Apart from his studies, he is developing for Engineers without Borders NTNU and spending time with open-source projects and other Linux tools.

Eirik Fosse Eirik has a primary interest in artificial intelligence and machine learning. In the course of his bachelor's degree he's focused on programming, mathematics, and evolutionary simulation.

Filip Fjuk Egge While achieving his degree, Filip has taken courses focused on a path related to system development and security. He has a varied education and knowledge on different aspects of computer science.

Haakon Konrad William Aasebø Haakon has selected disciplines related to mathematics and algorithms. Apart from being a student at NTNU he is playing football at NTNUI in the third division.

Håkon Gimnes Kaurel During his time at NTNU, Håkon has been keeping a primary focus on courses related to programming and the intersection between hardware and software. He's also got experience as an app developer, and has extensive knowledge of the GNU/Linux operating system.

Tino Lazreg Tino has been taking courses related to different aspects of software engineering, like programming, system architecture, human-machine interaction. Besides doing a BSc, Tino also works as a student assistant in a human-machine interaction course on NTNU.

1.3 About the Customer

Our customer is IDI Open. They are responsible for the annual programming contest mentioned in 1.2. Christian Chavez is our main contact for the project, but his two colleagues, Christian Neverdal Jonassen and Finn Inderhaug Holme, were also available for questions. They are all students of computer science at NTNU.

1.4 About the Contest

IDI Open is a programming contest where teams of up to three people meet and solve programming problems of various difficulty. The contest lasts five hours, and the objective is to solve as many problems as possible. The contest is open for all types of programmers, from students of all grades to professors and other professionals from the IT industry. Various prizes are given to the teams based on their performance. There are usually 8-12 problems in a contest. To make the competition fun for everyone, there are typically some problems that are easy enough even for novice programmers to handle. The main objective is to solve the highest amount of problems in the shortest amount of time.

1.5 Stakeholders

Our stakeholder fall into two different categories: the ones involved in the competition, and those involved in the course.

1.5.1 Course

Supervisor The supervisor's job consists of guiding and helping us through this project. This aid was primarily focused on the development process and the writing of this report. The supervisor tries to ensure that the developers communicate properly and have a structured

approach to developing the end product. To verify this, we have had bi weekly status reports delivered to the supervisor, as well as regular meetings.

Examiner The examiner(s) is responsible for determining our final grade. Unlike the other stakeholders, we have not communicated with the examiner throughout the development process. Though, the examiner has got access to all the documents the supervisor has got access to.

1.5.2 Product

IDI Open The project's primary stakeholders. They are the host of the competition in which our product was used. Their inclusion in this product comprised all aspects of our project.

Judges The judges are hired by IDI Open to supervise the competition, service contestants and create problem sets. They will rely on our end product achieve the mentioned tasks. Throughout the process they have given feedback to our customers, IDI Open, about our product. Naturally, the judges are important to the contest, so it is important that they are satisfied with the software they have to use.

Developers The developers are responsible for satisfying all other parties. Similar to the customer, our involvement in this project is total.

Maintainers As IDI Open is an annually recurring event, our end product, if successful, will be used for many years in the future. At a point, we assume the code will need to be replaced or modified. Assumably, there will be another developer team to do this. As such, the quality of of our product will impact them.

Sponsors Different companies sponsor each contest. In exchange for money and services, the sponsors get exposure through ads on the website and get to give a short presentation during the awards ceremony. Naturally, the sponsors want to associate their name with a successful product. Therefore, the sponsors rely on that contests are successful - this is heavily based on our product.

Contestants The actions of contestants are all through our software; our product will be their medium to take part in IDI Open. Reliability and usability is key to keep the contestants happy. The contestants also gave feedback to the customers about their user experience. Thus, how satisfied the contestants are impacts the developer's evaluation.

1.6 Goals

The goal of the project is to upgrade and improve the existing system used in IDI Open. We were given sole responsibility for our project; no other team or organization of developers has had responsibility for our solution.This gave us inspiration to do the best we could, and to give the customer something both we and they could be proud of for many years to come. And if the product is good enough it would hopefully also be used in larger programming competitions, maybe even international ones.

Chapter 2

Task Description

2.1 Task Description and Overview

The first step in our development process was to get a brief overview of our complete system. To do this, we have followed a conventional style of designing UML use cases together with a textual description. From reading this chapter, the reader should be able to understand how our end product works. Also, reading this chapter will be important to understand the other chapters in this report. The assumptions and constraints that affected our process are also discussed in section X.X.

2.1.1 Assignment

According to our customers, "IDI Open"'s previous solution was cumbersome to use. Our assignment was to create a replacement system that would be easier to administer. This included replacing both front and back-end systems. In replacing the old solution, we did not need to implement features that were not available from the old system.

The features of the old solution, in a nutshell, are given below. A more detailed overview will be given in section X.X.

- Website to inform about the contest to the public
- Team-registration and scoring
- Ability for users to upload code, that would be compiled and executed by the server

We were given access to the code for the old solution. The customers felt that this code was cluttered, but we could re-use components wherever we wanted. However, it was important that we did this in an uncluttered way, such that other developers could easily understand the new solution.

2.1.2 Branding our Product

Since we were delivering an end product to a real customer, we wanted to present ourselves as a real company. We chose the name "GentleCoding" as our representative name. This was used to name our repositories, email lists and other medium communicated with external parties.

The term “Gentle” is supposed to represent our calm approach to problems. It is also similar to “Gentleman”, which reminds of quality and good conduct . Furthermore, it is easy to interpret and remember.

Since we were developing a new system, we also wanted to brand our product. We wanted to keep it logical and simple, so we decided on “GentleIDI”. Consequently, GentleIDI will be used to refer to our end product throughout the rest of the report.

2.1.3 Assumptions and Constraints

To define what is satisfactory, we have made some assumptions and defined some constraints. Table X.X should make it easier to understand how we have reasoned our system design.

Do note that the implications in table X.X were not necessarily upheld. Rather, they were used as initial bounds to permit leeway. For example, imposing that third party plugins will speed up development does not mean that we would alway prioritize software re-use.

Roles and Their Definitions Usergroups Within the application-domain of “gentleidi” there are different groups of users. Each group has different levels of access control, and once a user is made a member of that group, they inherit those rights. A user may have membership in all groups. A privileged user is someone who is manually given elevated permissions. Table X.X shows the different roles and their available actions. Further elaborations on each group will also be given in later sections, but table X.X should suffice for an overview.

Table X.X: Usergroup overview

Service-providing Units

Another way of viewing the task description in section X.X, is to say that our solution needs to do three actions: serve web-content, store data and execute user-submitted code. Since each of these operates with different protocols, we will think to our solution as composed of three different systems. These are described in figure X.X.

The only entity that end-users interact with is the webserver. This is done through HTTP messages, which, if necessary, are relayed to the execution nodes and/or database. UML Use Cases We need one page each for privileged, registered, and non-registered users. That is, one interface for administrative users, one for contestants, and one for non-registered viewers. From each of these three, we defined use case scenarios. Figure X.X and X.X models the available workflows and actions for each category of users. Table X.X describes the semantics of objects used in the diagram, which should be equivalent to the UML 2.0 standard.

The purpose of the use case diagrams is to give a clear overview of what users shall be able to accomplish from our system. Furthermore, use case diagrams are easier to communicate to external parties, such that it is easier to agree on the system’s properties. The use case diagrams were used early in development to agree on the requirements specification and to communicate to the supervisor what we were trying to accomplish.

As seen in figure X.X, admins has privileges to perform the actions of any other group, in addition to their own set of actions. Thus, membership in the admin group gives a user complete control in the application domain. Furtherly, it can be noted that all usergroups have the opportunity to act as a contestant to review the website. Privileged users will are still restricted from appearing in the official high score tables to prevent them from assuming a competing role. This was to avoid the chance of any person with access to the solutions to compete.

Table 2.1: Caption

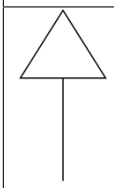
Assumption/Constraint	Why	Implication
The system will be maintained by people who have experience with computers	People that are involved with any programming contest are typically programmers themselves.	User design, words and definitions can be made more technical. Error messages can be explained using computer lingo.
The system will be used and maintained for > 5 years	Customer-constraint: they do not want to spend too much time developing new products, so maintenance is preferred.	The code should be written in a modular, extensible way with clear documentation.
The customers, IDI Open, are students near/at Gløshaugen		High availability for customer meetings and reviews.
The developers will maintain a 20 hour a week work ethic throughout the project-duration of 20 weeks[TODO: update]	To finish the product (on-time)	The set of requirements should not require more than 20 hours of work per week per developer, in order to complete.
Our system should be user-friendly	Our solution features a web interface available to everyone. Ideally, any person should be be comfortable with the user interface.	
Our end product will be open sourced	To ensure quality, and let other volunteers contribute to the code repository	No proprietary third party modules can be used. We cannot copyright our own material.
The final solution must run on Linux-computer	This is the choice of OS by NTNU, which is responsible for technical support and server access	Linux-compliant solution
We are allowed to use whatever third party plugin we want, as long as it is free and has no copyright-conflict	Speed up development	Speed up development

Role	Description
Admin	Privileged. An admin can modify all the available settings of the system
Judge	Privileged. Similar to an admin account, but with a limited set of actions: answering questions (clarification system), upload tasks to be solved, solutions to those tasks, upload incorrect answers (eg. answers that will provide penalty).
Functionary	Privileged. Functionaries hand out balloons when a team has solved a task. To determine what team will be given a balloon, the functionaries have their own interface with a team overview.
Team	A group of one to three contestants. A contestant is only part of one team per contest.
Contestants	A contestant has an account on the system and has the possibility to enter and compete in a contest.

Table 2.2: CAPTION

Entity	Features	Protocol
Webserver	Processes requests from contestants and teams. Also acts as a web gate interface to the execution node, both receiving and transmitting data to other execution nodes on the behalf of users.	HTTP
Execution node	A service, often on a dedicated platform, that offers the ability to compile and execute code. The execution node returns output data to the webserver.	RPC
Database	The storage unit for all user-data and logs. Also the execution node writes data to the database, and	SQL

Table 2.3: CAPTION

	Use case actor. Represents a user group
	UML generalization arrow. Used to indicate inheritance. The arrow's source, i.e. tail, represents the entity that inherits from where the arrow points to.
<<include>>	UML stereotype to represent a mandatory extension to a workflow.
<<extend>>	UML stereotype to indicate that if certain conditions are met in a flow, the entity to which this arrow points to can extend the workflow.

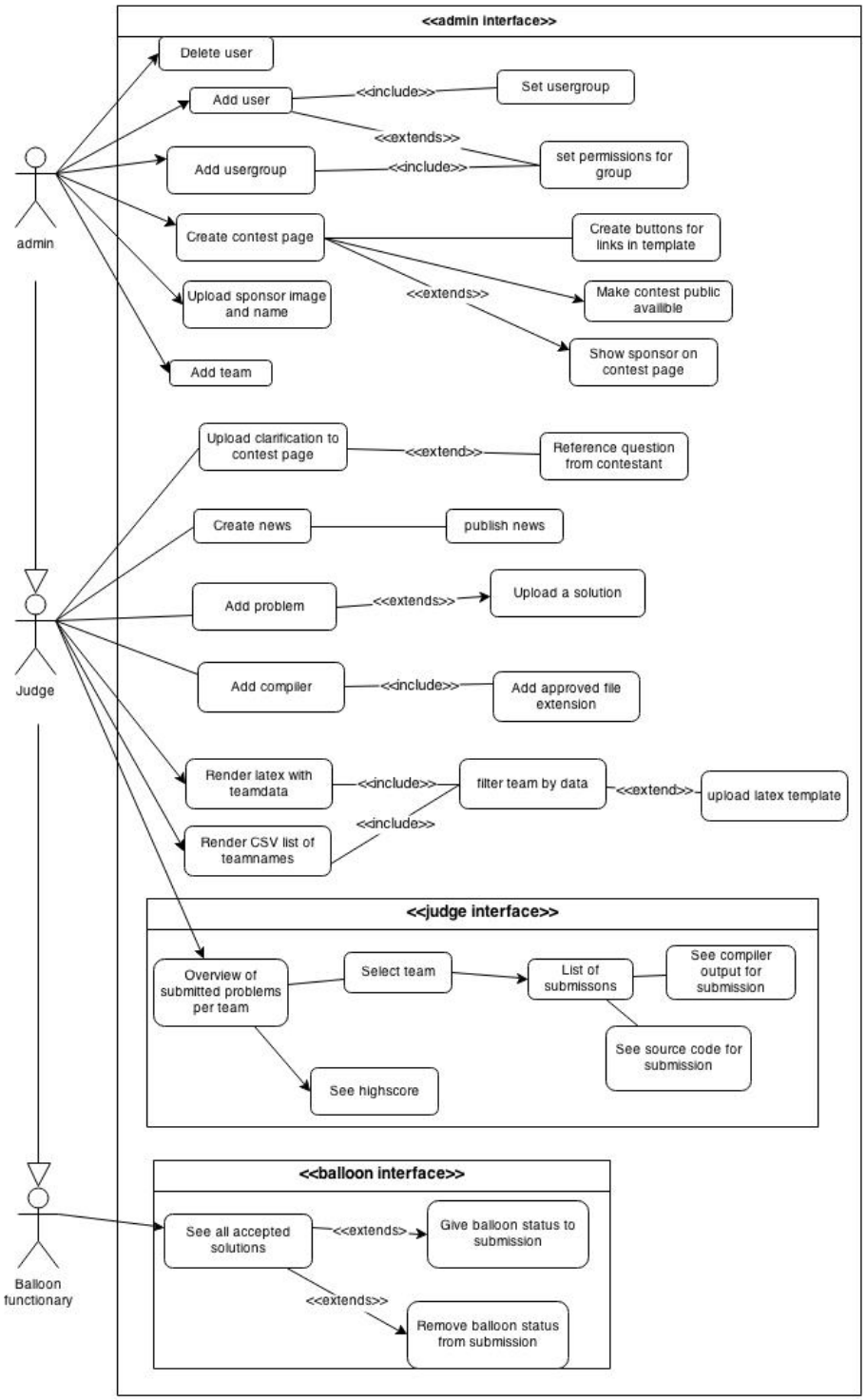


Figure 2.1: FIGURECAPTION

Chapter 3

Project Management

This section will go through the different project roles we deemed important. We will explain our development method, which tools we use and give an overview of how we planned the project. Furthermore, in section X.X we also provide a structured overview of how we organized our time.

3.1 Project Roles

We wanted to ensure that all developers had an even workload and experience in all components of our project. To achieve this, we maintained a flat organizational structure where all decisions were made in groups, and no member would work alone on a task for a longer period of time. Some tasks and delegations, however, would be easier to assign only once to reduce time spent in transition between developers. The following paragraphs discusses the different roles we assigned.

The most central role is that of the scrum master. The role mainly consists of setting up meeting agendas and keeping control of what team-members are working on. In addition, the scrum master should act as a buffer between the team and other distractions. In our framework, the scrum master also had a casting vote whenever there was a disagreement. The group elected Haakon to be scrum master because of his well-established authority and organization.

We also assigned the role of a transcriptionist. His job consists of writing a short summary of every meeting, and making this available to the rest of the group. This includes meetings with the customer and supervisor. This job was performed by Anders, who volunteered for the position. We randomly assigned Haakon to be customer contact, and Tino as responsible for room reservations.

3.2 Development Method (Scrum)

Scrum focuses on having daily meetings, and constantly adjusting to changes by iterative development. This makes it easier to predict and to adjust for problems that may occur. It was hard to predict what would happen in our project, therefore our sprints were short, lasting at most two weeks. The transition between two sprints was done during a prolonged meeting on Wednesdays. During this meeting we evaluated the latest sprint and planned the upcoming one. Every team member were requested to say three good things and three bad things regarding the last sprint. This was followed by a discussion of how to plan the next sprint better. Lastly we showed what had

been completed, to the other members of the group, before setting up the next sprint. Scrum also focuses on having finished versions of the systems on each iteration, and to finish all packages in the given iteration. In order to take advantage of the best in everyone's abilities we worked in pairs where this was efficient. Working in pairs is common in agile development. This was to improve code quality and reduce errors

3.3 Tools/Framework

The customer wanted our end product to be easy to maintain for future developer. Therefore we have chosen tools that are well known and easy to learn. A lot of different tools were considered for this system. Some of the most important are:

- Django, a framework written in Python.
- Editors like VIM and Eclipse were used.
- Documenting the development process was completed using google drive.
- Git was used as a version control system, with github as hosting service
- Communication were done through email lists, IRC and Facebook.
- User interface design was stylized with bootstrap and grappelli.

A full list of all tools and frameworks used can be viewed in appendix *Tools and Frameworks.*

3.4 Project-Level Planning

After our initial requirements elicitation we began to plan our development process. The purpose of the plan was to verify that we had enough time to complete the requirements, and to avoid unforeseen risks. This section will present the various components we introduced to structurize the project.

3.5 Work Breakdown Structure

WBS is a decomposition of the project into phases, deliverables and work packages. Each package was further broken down into different tasks. The benefits from doing these are as follows:

- Planning out the entire process prevents bottlenecks.
- Clearly defining the scope o a package prevents excess or insufficient time usage.
- It is easy for supervisors and other parties to evaluate and understand our process.

Table X.X shows the work breakdown structure created. These high-level packages were later broken down into activities, which are in the product backlog, see appendix

1. Project management
    (a) Write timesheet template
    (b) Look at the reflection notes
    (c) Meetings
        i. Internal
        ii. Customer
        iii. supervisor
    (d) Report
        i. Preliminary version
        ii. Mid-semester version
        iii. Final version
    (e) Risk assessment
    (f) WBS
    (g) Status report
    (h) Activity plans
2. Pre-study
    (a) Install and learn tools
    (b) Learn language/framework
    (c) Course
3. Design
    (a) Requirement Specification
        i. Functional
        ii. Non-functional
    (b) System architecture
    (c) Database modeling
    (d) User Interface
        i. Prototyping
        ii. Usability Testing
    (e) Admin interface
4. Development
    (a) Backend
        i. Execution-node(s)
            A. Web-page

A. User
B. Usergroups
C. Team management

ii. Statistics
iii. Contest management
iv. Clarification system
v. Balloons system
vi. Unit testing

(b) Testing

i. User-test
ii. System-test
iii. Final test

(c) Implementation

i. Deploy to production
ii. Installation
iii. Turn in to stakeholder

(d) Implementation

i. Verify
ii. Document

We also created a gantt chart. Here, each package was assigned an estimated time period, over how long time we expected to use. For ease of comprehension, not every package was included from the WBS. The gantt chart is shown in figure 3.1

The gantt chart was revised several times during the first 4 sprints, mainly due to new deadlines set by the customer. The original chart is also given in appendix X.

3.6 Milestones

Throughout the project, the supervisor, customer, and the project group set deadlines. Some of the milestones marks the completion of work package(s). We have four of these milestones, M-03, M-05, M-06 and M-07. The other milestones represents events with deadlines that were given by the course stakeholders. These are M-01, M-02, M-04, M-08. The group are using the milestones in order to determine if the project is on schedule and to monitor the progress.

Preliminary report M-01 Preliminary report is the delivery of the first version of the report. This was intended to help us get started with important aspects of the project work.

Mid-semester report M-02 This version of the report should present all of the analysis and most of the design of our system. The delivery date for the mid-semester report is 16.03. We wanted to complete this earlier in order to focus on M-03.

Table 3.1: Gantt chart

WP Name	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Project management															
WBS															
Pre-study															
Install and learn tools															
Learn language/framework															
Course															
Design															
Requirement specification															
System architecture															
Database modelling															
Tests															
User-interface															
Development															
Execution node															
Implement single node															
Implement several nodes															
Content Management System															
Front end															
Testing															
Unit testing															
Integration testing															
System test															
Production															
Post-implementation															

First release M-03 This milestone marks the groups first delivery to the customer. The reader can view what functional requirements this release includes in the functional requirements. In summary this release should make it possible for contestants to sign up for a competition. Three days prior to the release the group will meet up with the customer and overlook that all the requirements are met. This meeting will also act as an introduction to the system, showing the customer how to manage the system.

Presentation M.04 The main purpose of the presentation is for the class to share their experience with other groups.

Beta-release M-05 The beta release should contain most of the major features, but it might not yet be complete. This version of the program should only be a release to a selected group of people. From M-06 to M-07 the system will be tested.

IDI Open test event M-06 On april the 26th we had a test event where everybody could test the system. This means that leading up to this event the system should be a release candidate.

IDI Open M-07 This is the day of the competition and the system should be in a release version.

Final report M-08 This milestone marks the final date for delivering the report as well as the final date of this bachelor thesis. Based on feedback received from the competition the group might choose to implement some changes to the system.

3.7 Meetings

Throughout this project the group have had several meetings. They can be categorized in three categories: internal, supervisor and customer meeting. We established some meetings rules:

- All meetings follow "the academic quarter", meaning that the time of start was XX.15.
- Members that were late had to bring a cake to the next meeting.
- All members may at any time propose a coffee break. This proposal has to be followed.
- No laptop should be open during the meetings.

Internal meetings

We had three internal meetings each week. Two of which were daily scrum meetings. These were primarily set to be on mondays and thursdays. During these meetings each group member would answer three questions:

- What have you done since the last meeting?
- What are you planning to do until next meeting
- Do you have any problems regarding reaching your goal?

The group would continue to work together after these meetings.

On Wednesday we had longer meetings, marking the end of one sprint and the beginning of the next. This meeting would consist of a sprint review meeting and a sprint retrospective, where we discussed:

- What was good/bad with the last sprint
- What should we try to improve during the next sprint.

After that we held a Sprint planning meeting and created a new sprint backlog. Our official meetings structure for this meeting can be viewed in the appendix.

Supervisor meeting

Meetings with the supervisor was generally held at a bi weekly basis. During these meetings we talked about what we had done, what we were going to do and received feedback on what we had done. Before each meeting we had to deliver status reports and activity diagrams. These activity diagrams were early on replaced by sprint backlog and burndown charts to facilitate the process.

Customer meeting

Customer meetings were held whenever we felt that a certain part of the requirements specification was unclear to us, and when we wanted approval of a newly completed feature. Throughout the semester there were a lot of meetings. As we never decided upon a fixed interval between customer meetings, the frequency varied a lot. The couple of days leading up to a release date often contained customer meetings in order to get everything right before starting on the next release. During our periods of focusing on writing this report, the frequency of these meetings naturally went down as the product did not progress, and as a consequence we had little to discuss with the customer.

3.8 Resources

This section contains the available resources for the project. We intended to use a minimum of 20/25 hours per person each week, but prepared for more work as we approached the deadline. This estimate was later scaled up to a minimum of 25/30 to weeks before easter, During easter, the amount of hours per week scaled up higher. Planned work Table X.X shows our first initial draft of sprints.

Sprint	Range (week)	Days	Hours
1	3 - 4	7	15
2	4 - 5	7	20
3	5 - 6	7	20
4	6 - 7	7	20
5	7 - 8	7	20
6	8 - 9	7	20
7	9 - 10	7	20
8	10 - 11	7	20
9	11 -12	7	20
10	12 -13	7	20
11	13 - 14	7	20
12	14 - 15	9	33
Easter	15 - 17	12	-
13	17 - 18	7	35
14	18 - 19 (Leading up to event)	9	35
After	19 - 22	21	35
Total:		91	353

Actual work

Table X.X shows the actual sprints and work done. The hours are for each person, during that sprint.

Sprint	Week	Days	Hours
1	3-4	7	15
2	4-5	7	15
3	5-6	7	20
4	6-7	7	20
5	7-8 (midterm report)	7	27
6	8-9	7	31
7	10-11	7	35
8	11-12	7	30
9	12-13	7	30
10	14-15	9	40
11	15-17 (starting 16.04, ending 26.04, easter)	10	90
12	18-19	6	35
After	19.22	21	45
Total		100	433

Chapter 4

Architecture

This chapter contains an overview over the architecture for the system. The first part will describe different views of the system and the second part will show the quality attributes and patterns used when developing the system.

The main parts of the system is shown in Figure x.x. Clients sends requests to the web server and receives the processed results. The execution nodes process user submissions, and updates the results to the database. Figure X.X

4.1 Views

We have chosen to depict the architecture using Philippe Kruchten's 4+1 view model. [1] This is a method of describing the architecture for software-intensive systems from the viewpoint of different stakeholder by using multiple, concurrent views. We chose this model because it gives a good overview and is widely accepted by the software industry. Below are the 4 main views in the model; Logic, Process, Development, and Physical. The "+1" view is Use Cases which is addressed in [Chapter 2 Task Description and Overview]

4.1.1 Logic View

Purpose:

The logical view describes the functionality of the system by breaking down requirements into classes and representing them, and their relations, through class and sequence diagrams.

Figure 6.1 shows the main classes involved in GentleIDI. Each team participates in a single contest, and consists of a predefined number of contestants. Each team also has a team leader that handles most of the administrative tasks. The team can also try to solve problems by uploading submissions.

4.1.2 Process View

Purpose:

The process view explains the communication between different processes in the system, as well as how the system behaves in runtime.

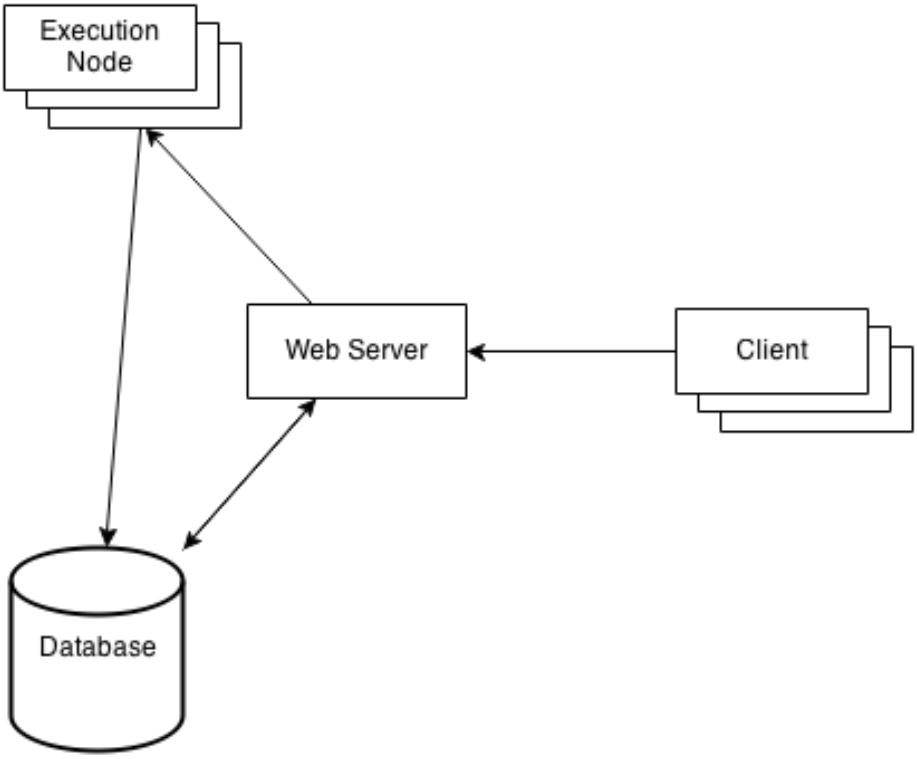


Figure 4.1: System overview

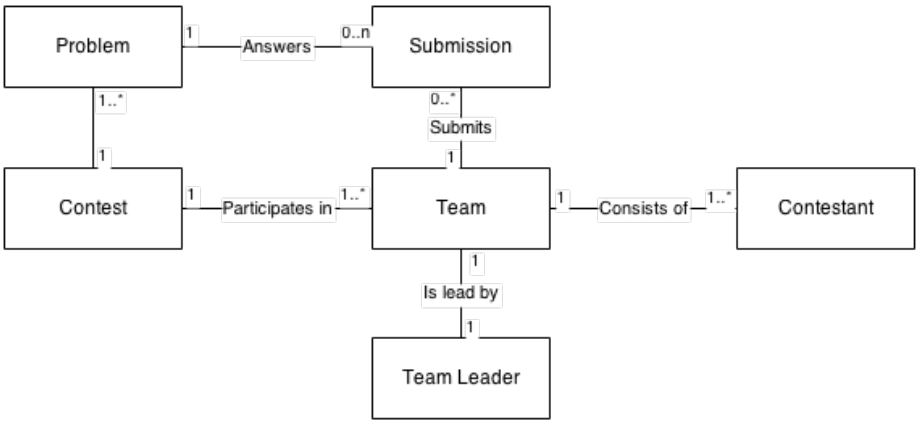


Figure 4.2: Top level class diagram

As this system is a web application the first thing to note is that there will be concurrent users in runtime. Each user generates HTTP requests to the server, which in turn may execute database

lookups for information like score tables or problem sets. When a user submits a solution the system will place it in a queue, which decides which node the solution will execute on according to availability and load.

We will now show examples for two important parts of the application. First is the action of successfully registering a user and creating a team. See figure 6.2.

Second is submitting a solution to a programming problem. See figure 6.3.

4.1.3 Development View

Purpose:

The developer view is intended for the developers. It should ease development, and focus on software module organization by packaging the software in small chunks.

We wanted a modular and maintainable system where it is easy to maintain and change specific parts of the system without changing everything. The structure of the system can therefore be divided into the following main packages: Contest, Registration, Submission, Execution, Balloon, Clarification, Admin, and Article. These packages are described in detail in chapter 8 Implementation.

4.1.4 Physical View

Purpose:
The physical view shows the interaction between the physical components of the system.

Physically the system is structured as a multitiered architecture. It consists of three tires, presentation tier, application tier, and data tier, see figure 4.6. The tiers represents a physical structuring mechanism for the system infrastructure. The user is physically separate from the application and database.

Presentation tier

This tier presents information to the user through the public website and admin interface. It translates the web server response into web pages generated using HTML5, CSS, Ajax, and JavaScript. It sends requests to the underlying web server and renders the response.

Application tier

The application tier contains the logical layer, it controls an application's functionality by performing detailed processing. Primarily this is done through python code, although when running solutions the file is run on an execution node through the use of built in unix commands.

This splits the application tier in two parts, the web server that serves static and dynamic content, and the execution nodes that process uploaded submissions. This division can be seen in "Application Tier" in Figure 6.4.

For the web server we use Nginx for serving static files, and as a reverse proxy for Gunicorn, the server providing dynamic HTTP content to the user. Gunicorn is the server that processes requests and returns HTTP pages. The execution nodes process submissions through a FIFO queue implemented with Celery and RabbitMQ. This provides load balancing across CPU cores and multiple nodes in the cluster. The execution nodes also share parts of the filesystem, this is

implemented with SSHFS (SSH Filesystem), and is a secure way of sharing the uploaded files across the execution nodes.

Data tier

This tier includes the data control functionality. The system utilises a shared SQL database for the execution nodes and the web server. See Figure X.X This database links to the file storage on the main web server. However, the execution nodes requires some files to be shared across multiple nodes. Like explained earlier in section X.X, this is implemented with SSHFS. For more specific details see [Chapter 8 Implementation] and [Appendix ER figures]

4.2 Quality attributes

4.2.1 Availability

Since this software is to be used in a programming contest, it is crucial that the system has high uptime and availability. And since the contest only lasts for about 4 hours, our margin for failure is minimal. We have made an effort to account for all possible outcomes, and to safeguard the application for any errors that might occur.

4.2.2 Modifiability

This is a system that we hope will be used for many years to come. With the ever changing nature of the web, the ability to adapt and improve is imperative. To accommodate this, we chose to implement our solution in Python, a language taught to most of new students of computer courses at NTNU. These are the same students that hopefully will use and continue to work on this software. To our best ability we have also tried to write and document the code in a way such that it is easy to understand and improve.

4.2.3 Performance

Performance is an important aspect of every application, especially web applications. Users expect that sites loads fast. Failing to accomplish this is a sign of a bad application, at least from the user's perspective. For this reason we have focused on making our pages load as fast as possible. And since this application will be used by over 100 users simultaneously, it is also important that the servers will handle the load.

4.2.4 Security

Since our application contains user data and data that should be hidden from unauthenticated users, security is another important aspect. Django provides many security features by default, and others that can be implemented with very little effort. We also chose to enable SSL on the web server to increase security on web requests.

4.2.5 Testability

When we first started out, we wanted to utilize testing during development. Testing is a way to find problems early, and before they begin to encompass larger parts of the application. But testing is also one of the most time consuming parts of the development process. In the end we did not have as much test coverage as we would like, but we feel that we covered the most important parts.

4.2.6 Usability

As with any web application, we want the users of the system to accomplish their desired task, and learn the functions of the system with ease. The user should receive feedback if something went wrong or if the outcome is not clear. We also want the web pages to provide information how to use the system.

4.3 Patterns

4.3.1 Client-Server

Since we are making a web application we will use the Client-Server pattern. The clients connect to the server through a web interface, either the website or the admin interface.

4.3.2 MVC(model-view-controller)

The front end is implemented using the Django framework and follows a rather strict implementation of MVC. Every HTTP request sent to the site is handled by a controller function, which in turn fetches the appropriate models from a database, creates a view based on the models and returns the view as an HTTP response.

4.3.3 Shared-Data

The system utilises multiple execution nodes as well as a web server, through which users access data. We wanted to have a central shared database server that scales with the number of execution nodes and the amount of data.

4.3.4 Multi-tier

See: 6.1.4 Physical View.

References:

[1] Architectural Views -

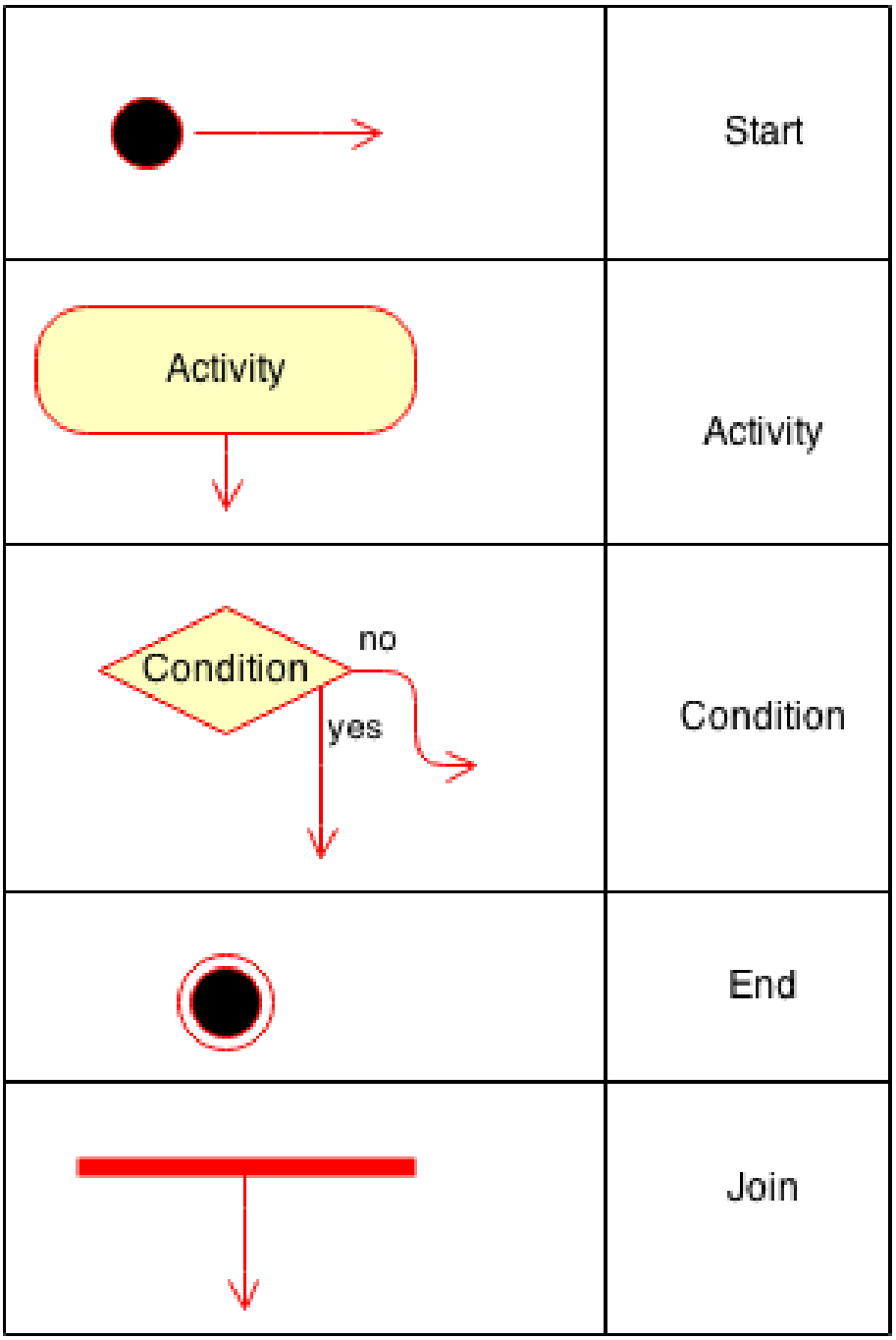


Figure 4.3: Symbology

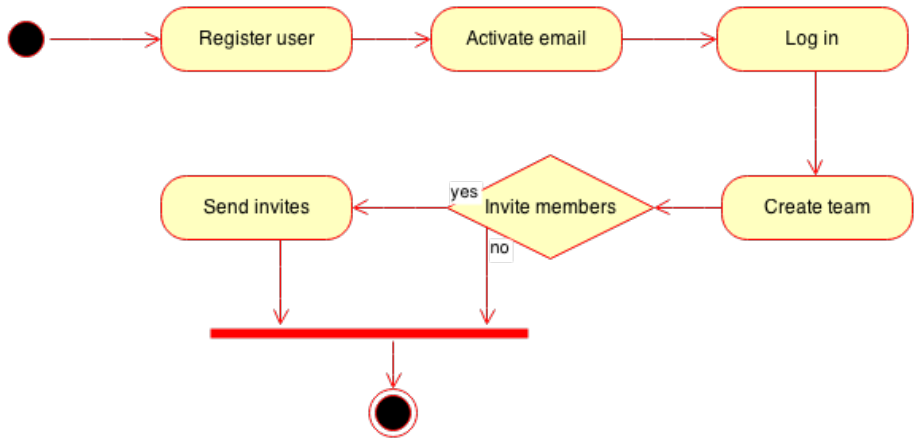


Figure 4.4: Activity Diagram for registering a user and a team

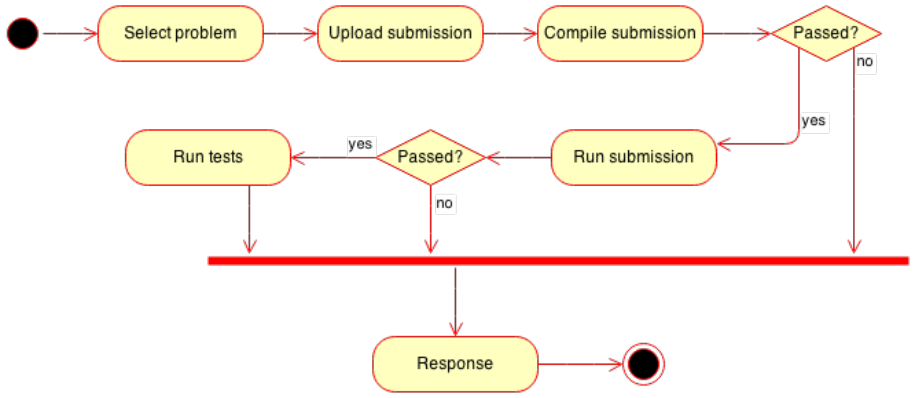


Figure 4.5: Activity Diagram for submittion a solution

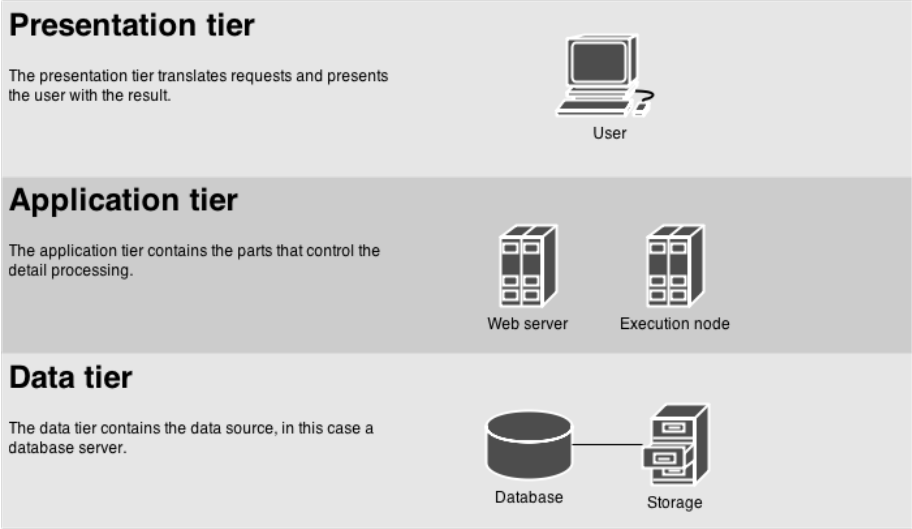


Figure 4.6: Multitier architecture

4.4 Design

This document contains the choices made regarding the process of designing the front-end of the application, for a more technical approach see *System Architecture chapter 6*.

Design process

The user interface provided by the previous IDI Open system consisted of a simple web interface for reading news items, registering teams for contests, and delivering submissions. GentleIDI is intended to provide more functionality through its web interface, including but not limited to change email(requirement FC-02), supervisor(requirement FJ-11) and user management (requirements FC-01, FC-03 and FC-04). As a consequence we had two options available: reusing and extending the existing interface design, or creating our own design from scratch.

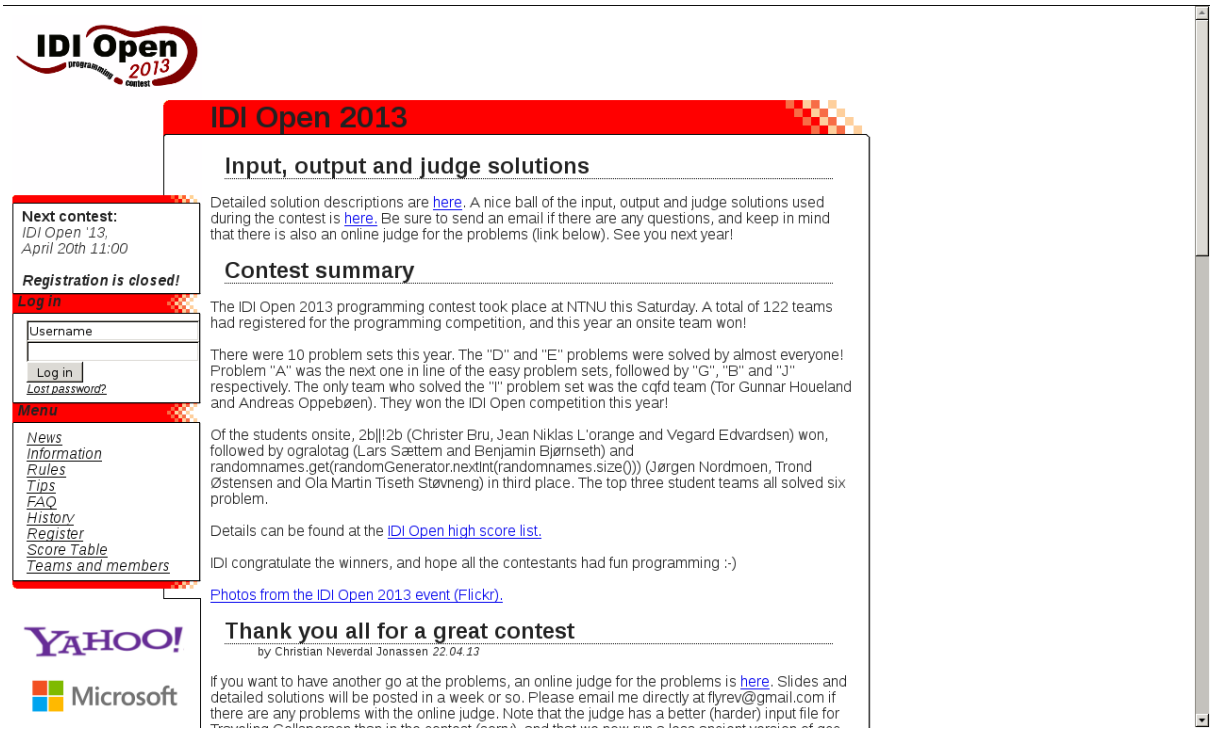


Figure 4.7: FIGURECAPTION

Fig 7.1 We chose to create our own design from scratch, while still trying to keep a similar placement of elements from the previous design.The customer expressed concern regarding how contestants would react to the transition from the old interface to the new one. With this in mind we started to create mockups modelling core elements of the website. Our initial drafts consisted of simple rearrangements of elements found in the old web interface.

Beyond our three initial mockups we tried a couple of "out of the box" approaches to our designs, but none of them met our standard and was rejected for either being too time-consuming to implement or too far from what our customer wanted. We had a meeting with our customer, where we showed our mockups, and what are thoughts on design had been so far. We wanted to

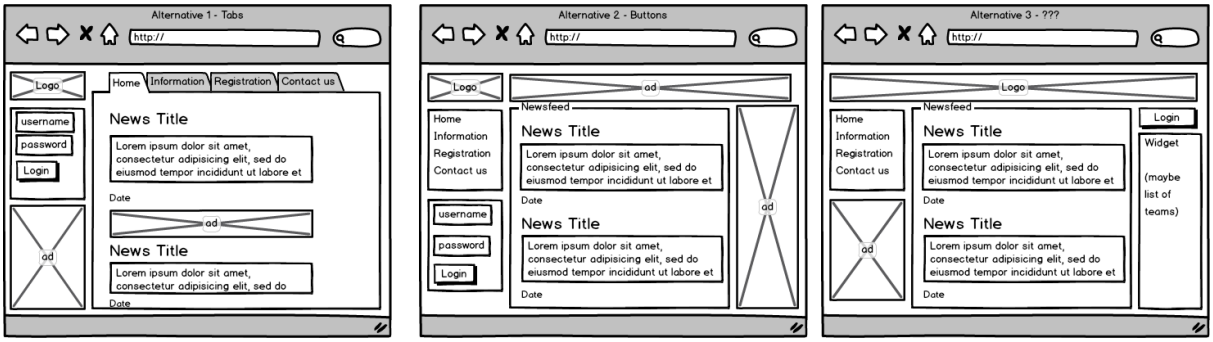


Figure 4.8: FIGURECAPTION

make sure that the customers was on the same page as us, and that we were not moving beyond the scope of the project. Our customers wasn't very focused on the design aspect, but one demand they had was that they wanted the new site to have the same structure as the old one. One example of what this means is that the customer wanted us to keep the menu on the left-side as you can see that the old system has in Fig 7.1. We agreed, because getting used to a new website can take time, so keeping the structure similar would ease the transition for our users. With this in mind we decided to go for one of our initial mockups, the rightmost one in Fig 7.2, because it had the same structure as the old page, and we personally favoured that design. As a result, most of the elements found in the old interface can be found in the new one, and the transition between using the two is reduced to a minimum.

The task had to be completed in time for milestone M-03, so our main concern was designing for the functionality needed for that particular milestone. However, we also had mockups for functionality outside of this milestone. After milestone M-03 was done, we introduced new design for new functionality through continuous work on top of a template.

The majority of the front end is stylized using bootstrap[Link til kilde] as a framework, enabling us to create a site which is both highly maintainable and aesthetically pleasing at the same time. The admin interface was created using django-admin-interface with Grappelli as a skin to give it a modern look. This worked more or less automatically.

The final page looked like this:

The "black" frame was in our initial page coloured blue, but was changed one week before M-07, idiopen [REMARK: may be altered].This illustrates the strongest functionality of the design, namely customization. It is possible, by only uploading a new CSS file, to change the whole feel of the website and give every contest its own theme. The change on IDI OPEN 14, from blue to black, was done as a consequence of a logo change by Richard Eide, one of IDI Open's facilitators. The old color scheme can be viewed in appendix [insert which appendix]. By comparing fig 7.1 and fig 7.3, you can see that we kept the same structure, but still made some significant changes to the design.

User interface The user interface is designed by using a base template. The template is the same for every part of the webpage, and contains a content block that changes while you navigate through the different parts. This makes it easier to add new content to the user interface, because you already have the base, and don't need to worry about the header, footer or the menu. We wanted to make it easy for future developers to take over GentleIDI after us, and therefore we

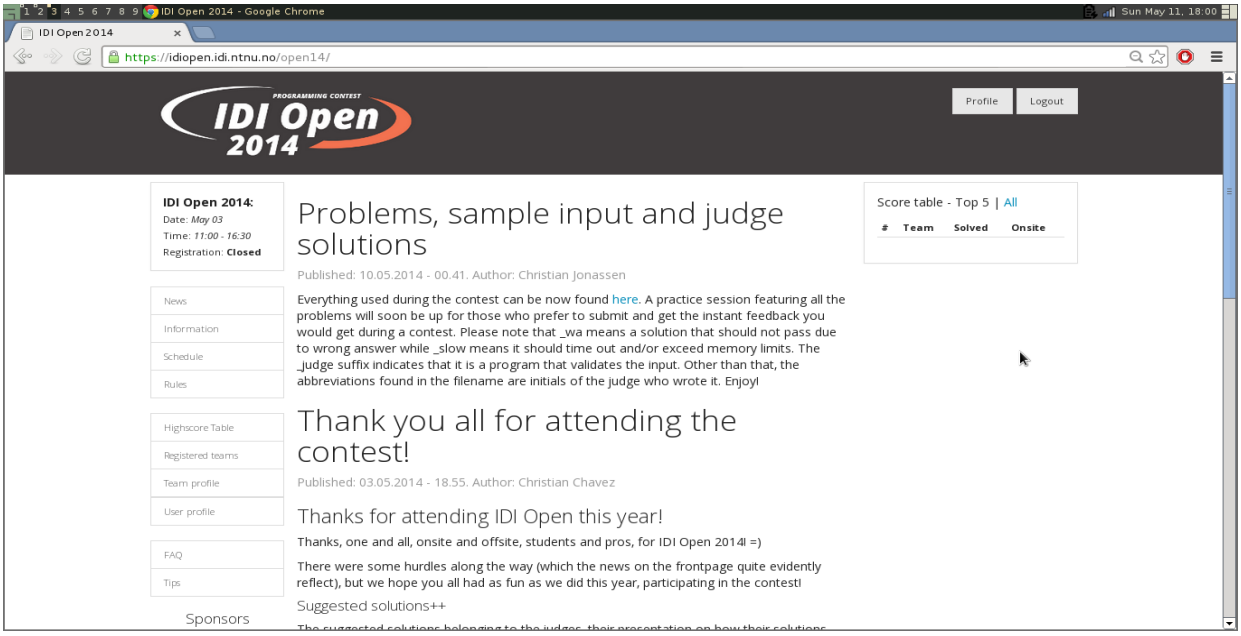


Figure 4.9: FIGURECAPTION

focused on a versatile user interface, in case they want to add new functionality.

The menu is placed to the left, coping with the western norm stating that eye placement is natural to the left[1]. We designed the menu to be versatile. Admins can choose what they want to show in the menu, except for *Register user* and *Register team* that are "hardcoded" on request from the customer. This was highly prioritized by our customers, they wanted to be able to make changes without having to change the code. As mentioned in Design process 7.1, we designed the user interface after a principle of versatility. Admins can also change the logo, the sponsor images and the contact information in the footer.

Buttons, images and icons were surrounded with boxes, for example the sponsors and the menu buttons, to show that they are different elements.. There is also one big box surrounding a group of elements, for example the sponsors. This is consistent with the gestalt law of proximity, that constitutes that humans will naturally group objects that are close to each other, and view them as a distinct. This helps the user quickly understand the user interface.

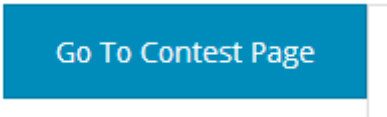


Figure 4.10: FIGURECAPTION

[1] http://research.microsoft.com/en-us/um/people/cutrell/chi09-buschercutrellmorris-eyetrackingforwebsalience.pdf

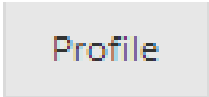


Figure 4.11: FIGURECAPTION

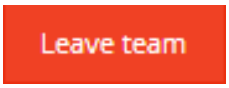


Figure 4.12: FIGURECAPTION

fig 7.4

"To strive for consistency" is the first of Shneiderman's eight golden rules of interface design[2], and we tried to follow this while making design decisions. As can be seen in fig 7.4, we decided to use colours that represents the action each button is connected to. The red button marks that pressing this will have permanent consequences. We added a textbox prompt that the user has to answer after pressing a red button, that constitutes to Schneiderman's fifth and sixth rule, for easy reversal of actions and error handling. This wasn't added initially, but we noticed while testing the system that without a prompt, it could be possible to leave your team by mistake. fig 7.5

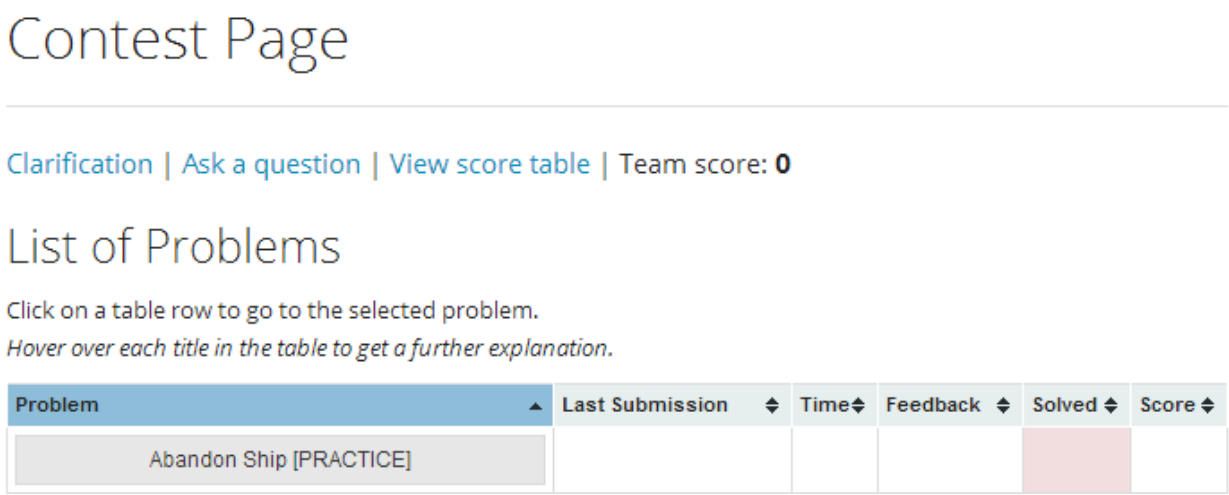


Figure 4.13: FIGURECAPTION

For the contest page, fig 7.5, we wanted to give the contestant a good overview of all the problems, their submissions to them, last feedback, if they solved the problem and the score. It is important to not bury information to deep in a website. It could be challenging to balance this while trying not to overload the page with too much information. We had this in mind when designing this page. We got valuable feedback from the customer concerning what they wanted to be present

[2] https://www.cs.umd.edu/users/ben/goldenrules.html

on the contest page. They wanted it to be easy for the contestants to access everything they need, during the competition, through the contest page. After feedback from the customer, we added links to the clarification page and highscore table on the contest page. This lowers the short-term memory load on the contestants, which is consistent with Shneiderman's eight rule, because they will have everything accessible on the same page.

Admin interface

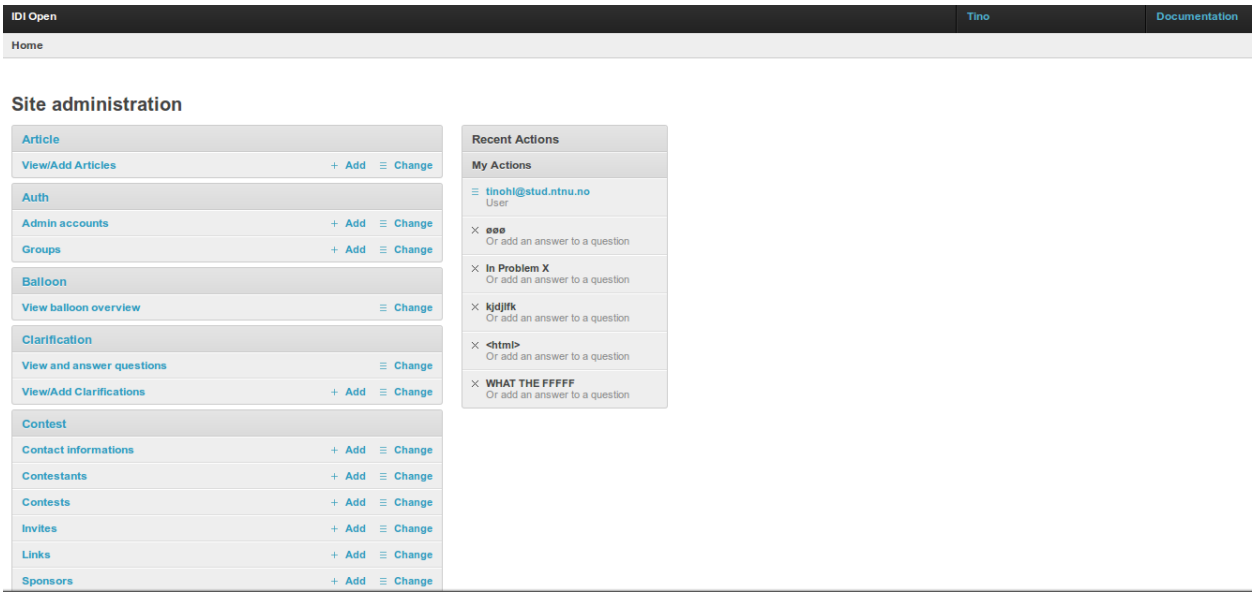


Figure 4.14: FIGURECAPTION

The admin interface is developed as an extension Django's admin interface. Django comes with an extensive admin interface, that provides functionality for adding, removing and changing parts of the system. The admin interface consists of everything we as developers want the admins to be able to change. For a complete listing, see figure X.X[kap 2]. We decided to use Grappelli, an app for the django admin interface that also provided us with more adequate functionality, e.g. auto-completion, rich text editors, drag'n drop and more.

The structure of the layout is simple. Each category has it's own header and everything in blue is clickable. The "Recent Actions" box is there to help admins remember what they last did, which is important to reduce the users short-term memory load, in accordance with Shneiderman's eight rule.

Originally all the names of the elements were the same as our model names. We decided to change this to more intuitively understandable expressions after a request from the customer. Django's admin interface couldn't give us all the functionality we wanted, so we had to extend the interface with out own custom views. We created two views, "Balloon overview" and "Judge overview". To avoid having to create a similar interface as the rest of admin site, just with different functionality, we decided to extend the interface templates used for the django admin interface. This allowed us to change what we wanted, while it still kept its consistency with the other parts of the admin site.

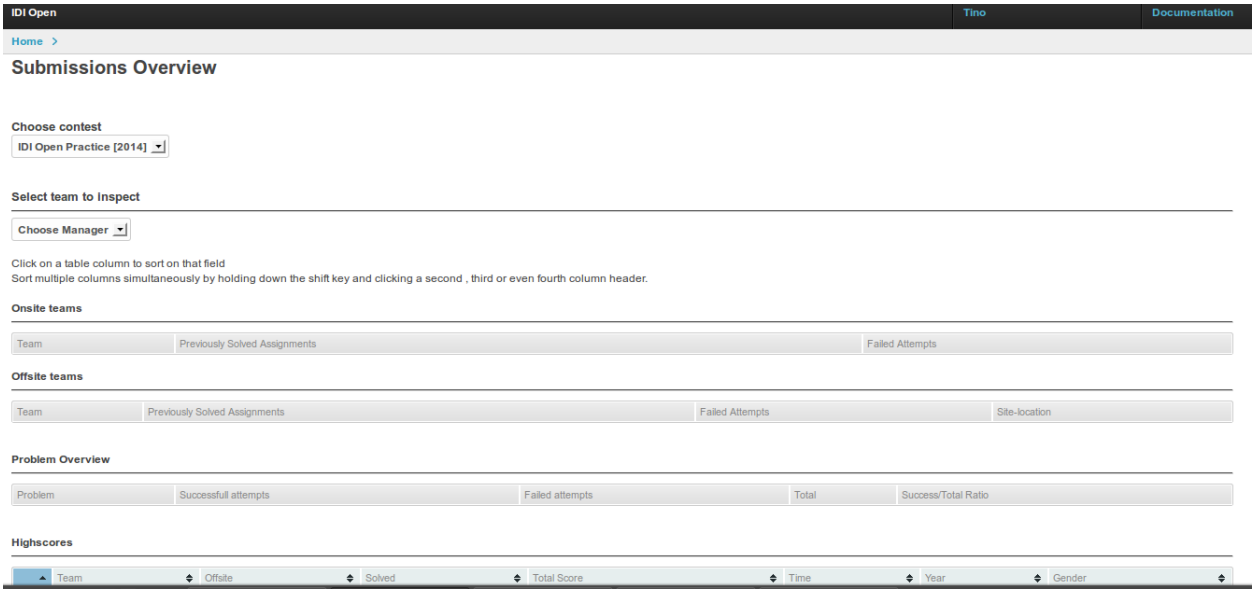


Figure 4.15: FIGURECAPTION

fig 7.7

The judge overview was made primarily for judges, but could also be used by the admins. The motivation behind making this view, is that it gives the judges an easier overlook over the competition and how the progress is going for the different teams. We were initially told that the judges wanted a way to see if a team was struggling, so they could help that team.

The view consists of four different tables, with the same layout as the balloon tables. The first two tables depicts how many failed attempts an onsite or offsite team has. The Problem Overview table provides statistics on each problem for the given contest. This was added so that the judges can see which problem has the most failed or successful attempts, and if necessary make changes. To make it easy for the judges to choose a specific team, independent of submissions, we made a dropdown menu with all the teams. The last table is the highscore list. We wanted everything to be on one page for the judges, so they wouldn't have to constantly switch between different pages.

Figure [Judge_overview for Team]

Figure [Judge_overview for Team] shows the judge overview after selecting the team "Gentle-Coding". It is possible to expand each submission by clicking on it. The third submission has been clicked on, so we can now choose to expand different categories. For example if a judge wants to see the source code for that submission, he/she can click on "Source code" and it will expand. Submissions that haven't been compiled are shown in red, and the other are white.

https://www.cs.umd.edu/users/ben/goldenrules.html

IDI Open | Tino | Documentation

Home >

Overview of Team "GentleCoding"

Submissions

Date Uploaded	Problem	retval
May 3, 2014, 4:23 p.m.	Abandon Ship	1
May 3, 2014, 2:49 p.m.	Abandon Ship	1
May 3, 2014, 2:47 p.m.	Abandon Ship	1

- **Command to execute**
- **Source code**
- **Command output**
- **Stderr**
- **Feedback**

May 3, 2014, 2:44 p.m.	Abandon Ship	1
May 3, 2014, 2:30 p.m.	Abandon Ship	1
May 3, 2014, 2:29 p.m.	Abandon Ship	1
May 3, 2014, 2:26 p.m.	Election	0
May 3, 2014, 1:52 p.m.	Abandon Ship	1
May 3, 2014, 1:14 p.m.	In The Shower (Easy)	0
May 3, 2014, 11:49 a.m.	Counting Digits (Easy)	0

Figure 4.16: FIGURECAPTION

Chapter 5

Implementation

This chapter goes into the details of our implementation. As mentioned in section X.X, Django follows the MVC pattern in a quite strict manner, and as a consequence so does our project. In addition to MVC our project is divided into several django apps, which are separate modules containing their own models, views and controllers. The apps are intended to serve a specific purpose and provide a certain level of modularity. However, some apps are dependent on others.

Figure X.X shows the directory structure to one of our apps, all apps follow this structure. An app's root folder contains four files worth taking a closer look at, models.py, views.py, forms.py, and admin.py.

- models.py contains the apps models i.e. our database entities. Due to our site being MVC, every aspect of the site is in some way represented by a model defined in a models.py file.
- The file views.py defines the app's functions for handling requests, called views. Though the naming might be confusing, the views defined in this file are not views in the MVC sense of the word. The views are in essence MVC controllers. When an HTTP request is received by Django it is routed to a specific view, and the view then handles the request. Though most views simply serve web pages in response to get requests, there are no limits as to what a view can be used for.
- The form.py file contains a set of django forms, which are simply collections of input fields. The forms can be rendered as HTML, and serve as a validator of the input received by POSTs.
- This leaves the admin.py file. Django provides a quite modular and modifiable admin app for managing other apps. The admin page's main functionality is that of viewing, editing, creating and deleting models. However, the admin app does not have access to all of the models in the system by default. The admin.py file is where an app registers which of its models are to be modifiable by the admin page, how the models are to be rendered etc.
- The apps also contain a templates directory. The templates are in essence html files that can be easily processed/modified by django. When a view sends a response it is usually by

Contest app

```
.
├── admin.py
├── forms.py
├── __init__.py
├── migrations
│   ├── 0001_initial.py
│   ├── 0002_auto__chg_field_team_name.py
│   ├── 0003_auto__add_field_contest_penalty_constant.py
│   └── __init__.py
├── models.py
├── templates
│   ├── Cage
│   │   └── cage.html
│   ├── contest
│   │   ├── alreadyContestant.html
│   │   ├── editTeam.html
│   │   ├── index.html
│   │   └── team.html
│   ├── registerForContest
│   │   ├── registrationComplete.html
│   │   ├── registration.html
│   │   └── requireLogin.html
│   └── viewTeams
│       └── viewTeams.html
├── templatetags
│   ├── __init__.py
│   ├── link_tags.py
│   └── widget_tweaks.py
├── tests.py
├── urls.py
└── views.py
```

Figure 5.1: App overview

inserting dynamic content(models) into a template and then serving the final html file as an HTTP response. Though not visible in the figure, most of our templates are extensions of a global base template, this way redundancy is reduced and our user interface stays consistent.

5.1 contest

The contest app contains the most fundamental functionality and models, the ones related to creating and deleting contests. Just about every other model in the project is related to the contest model in some way. Other than the contest model the contest app defines a couple of models for storing information directly related to a contest, such as sponsor information, support contact information etc.

A complete overview of the models defined in the contest app can be found in [reference contest ER]

5.2 article

The article app provides basic functionality for posting news articles. It contains several different views for looking at articles, lists of articles etc. For editing articles the app uses a WYSIWYG editor, available in the admin interface.

5.3 userregistration

As the name suggests this app handles user creation, deletion and modification. The majority of this app is an open source app that we incorporated into our project, however, we made some slight modifications.

5.4 teamsubmission

When a team has reached something they think might be a valid solution to a problem they submit their source code to the system. The uploaded source becomes part of a submission model which is part of the teamsubmission app.

5.5 execution

The system needs a way of handling the submitted source code. For instance it needs some way of determining which compiler is to be used. When the source has been built the system needs to know what command is to be issued to the system to execute the binary. Both of these things are handled by the execution app. In addition there are restrictions set to limit the resources available to the submissions, for example the number of subprocesses, memory allocated etc.

The models defined in teamsubmission and execution can be found in the [reference submission ER]

5.6 node_manage

With a well configured system and the previously mentioned apps working properly, a submitted source file will be stored and the outline of how the file should be treated will be set when the file is uploaded. The code for actually performing the actions of building and running is handled by the node_manage app. The node_manage app fetches the appropriate settings for a submission, and submits it to a FIFO queue. Our backend consists of several execution nodes connected in a cluster powered by a framework called celery. The nodes can be configured to handle any number of concurrent submissions, and when a node has got available capacity it fetches another submission from the queue. Celery relies on the AMQP message passing standard by means of an open source message broker system called RabbitMQ. All messages passed go through a broker setup on the same host as the web server, the broker then distributes the messages to the appropriate host.

5.7 balloon

When a team has solved a problem, they are to be awarded a helium balloon. This app enables staff users to view problems that have newly been solved by a team, send somebody to deliver a balloon, and then remove them from the list of newly solved. This app is in other words little more than a custom view in the Django admin page.

5.8 changeemail

Since we had to modify the userregistration app that we incorporated, not everything worked as we wanted out of the box. An example was that the functionality for changing the email of a contestant broke the team that contestant was partaking in. This app provides a fix for that problem and makes sure that changing email works properly.

5.9 judge_supervise

This app provides judges with an interface in which they can see all submitted solutions and statistics for each team. For each submission, the judges can see compiler errors, execution output and source code.

5.10 clarification

During a contest questions can be asked by contestants to the staff. If a problem is ambiguously formulated, or they are experiencing system errors, these problems can be addressed. The questions are posted publicly on the website, as well as their replies.

Chapter 6

Development

This document describes the different phase of development the group went through in order to finish the product. To increase readability the first part of the document describe the process working towards the milestones, as can be viewed in fig 3.3. The second part will describe each sprint in more detail including work done/completed.

6.1 Towards the milestones

6.1.1 Milestone M-01 - Preliminary report.

From start to 09.02

Eager to start, we had our first meeting 15.01.2014. During this meeting we discussed which task we wanted apply to. We decided on "IDI_ProgrammingContest", which we received.

After receiving the project assignment, we discussed our ambitions for the course and the end product. We agreed that we had a shared goal to receive a top grade in this course, and we that we would resolve in whatever means necessary to achieve this goal. The group was in doubt if we should try popular, enterprise-level tools and frameworks, or if we should stick to basic, previously used tools. We decided to let each member of the group to explore a tool on his own and present his experience to the others. If the tool seemed usable, we incorporated it into our project.

Our primary concern was that we would spend time on suboptimal tools, methods or frameworks. Thus, the group spent much time discussing and modeling the application to come.

6.1.2 Milstone M-02 - Mid-semester report

From 09.02 to 09.03

Being aware of the large amount of programming ahead of us, we aimed to have the mid-semester report finished one week before the actual deadline.

To shorten meeting time and strengthen our task overview, we had a meeting thoroughly discussing how SCRUM worked. We decided to adhere more of the conventional SCRUM standard. As a consequence we started to draft release- and backlog. This resulted in a reduction in the number of hours used to administering and task delegations. We also got a better overview of the

final application. This meant that we could reduce the amount of models and focus more on the code.

The mid-semester report finished as planned one week before our deadline. We more or less completed our testing plans and concluded on management structure. The biggest challenge was how to implement support for user handling.

6.1.3 Milestone M-03 - First release

From 09.02 to 19.03

Having finished the midterm report, the group now had a structured overview of the requirements specification and approach for development. We had much coding to do in order to reach the third milestone. We tried to agree on an optimal approach, but concluded that we had to "just get started". In our sprint backlogs the amount of coding assignments grew. To induce more coding, we arranged informal coding nights in order to trigger "learning by doing" and improved our progression.

By the time we had finished the necessary pre-studies and requirements we already had some functionality. However, there were still work remaining, as suggested by our work breakdown structure. In addition we had a meeting with the customer where they proposed some new requirements, and reprioritized a few other.

In advance to the first release we had some meetings with the customer. We were a little nervous regarding some of the design choices. The meeting discussing the design went well. We had formerly agreed on our mock up-design. There were however a few discrepancies between the delivery and what the customer wanted.

Our first deadline with the customer 19.03, but the actual release of the website was delayed to after the weekend, for external reasons.

6.1.4 Milestone M.04 - Presentation

From 09.02 to 19.03

Since the presentation was scheduled at the same time as our first release we did not have a time to prepare much for this presentations. Nevertheless, we received valuable feedback from other groups.

6.1.5 Milestone M-05 - Beta Release

From 19.03 to 11.04

Working toward the beta release was challenging. Increasingly, we experienced that modeling the application before coding was not an optimal solution. Thus, we began to code without relying on diagrams to aid us. We sustained this approach until the end of the project.

With limited time, it became necessary to prioritize some tasks over others. Our improved product backlog, created earlier, proved to be a big benefit. As mentioned previously, we felt that it was hard to predict the outcome of the development process, so we decided not to update the gantt-diagram. Instead we relied on our own options and customer prioritizations This was due to our new understanding of what needed to be completed when.

We did make some progress with our development, but still had some aspects of our frameworks that needed to be learned. As the weeks went by we increased our work estimates and grew more

familiar with the framework. Still our models seldom related to the actual end result. It was not something we felt was a big problem, we did make progress.

6.1.6 Milestone M-06 - IDIOpen test event

From 11.04 to 26.04

We still had quite a few packages to implement and we were uncertain about how long time we had to spend on each of them. As a consequence we had to shorten our easter vacation.Spending a lot of time together, every day for weeks, may cause tension in groups. We felt it was important to create an environment to ease the tensions.Therefore we took breaks from the coding, eating pizza and playing foosball.

We started every day discussing what we were suppose to do, similar to a daily scrum. We believed all members had a good tacit understanding of what needed to be done, so we transitioned from sprint backlogs to daily TODO lists. These lists were written informally for the sake of brevity.

The days were long, lasting from 09:00 to 24:00. Packages were implemented at a high pace and we started to become aware of the final product. The biggest challenges were to get the execution node up and running, highscore table and content management for the judges. Testing was also completed. We also had sufficient time to implement some of the lower prioritized requirements.

During the test event, we sat at our own table and received feedback from the judges and volunteers that had shown up. The fact that some of the judges were considered really good programmers made us a little nervous. They did give us feedback and a list of new requirements to be implemented. These were minor fixes, mostly related to the user interface. The test event itself was considered a success: all the judges approved our system.

6.1.7 Milestone M-07 - IDI Open

From 26.04 to 03.05

After the test event we got a new list of requirements. There was only one week to the actual event and we had to carefully pick those we and the customer felt were the most important. We implemented support for several execution nodes, refined the content management and fixed small bugs. Some tasks were complex, so it was a challenge to analyze if we would be able to finish them on time. The most advanced tasks we were given after the test event was that the judges wanted a better overview over the contest. More precisely, they wanted access to the whole competition, and all the functionality, before the contest started. The customer also wanted to be able to export data to CSV and LaTeX. This task did, at first, seem lightweight, but turned out to be much more extensive. While finishing on time, this consumed more hours than initially planned.

In total there were 92 teams taking part in IDIOpen14, and a total of 214 registered users in the system. When the contest officially started and the problem sets were released. All users simultaneously accessed the same resource caused a spike on the system load. We were been told by our customer that the old system had previously buckled under the pressure from the spike at the beginning of the contest. Our system did, however, handle this well. Thus, the start of the contest went well.

At one point the system went down for a few minutes. This was because we ran out of hard disk space on our main server. In other words, the system had nowhere to store its data, and was unable to handle the requests made by users. After a couple of minutes of deleting unnecessary

files we discovered that for every file that we removed we only bought ourselves a couple of more minutes of uptime. Somewhere in the file system there was a file growing at an alarming pace. To identify this file was challenge. By monitoring the server's processes we found that the application responsible for load-balancing was logging every event under "debug" setting. This meant that every request to the webserver caused an extensive log entry. This resulted in a 1MB/S disk write rate. The rate was small enough so that we could easily monitor and periodically erase the logs to clear out disk space. We could have disabled logging, however, that would have required a restart of the database server and thereby downtime.

After this problem was resolved the rest of the contest went without any significant issues. At our peak system load we handled a total of 12 concurrent submissions which was more than enough. The website was responsive and working properly. Except the highscore list, which we knew had performance issue. These issues did not have a significant impact for the user experience.

6.1.8 Milestone M-08 - Final report

from 03.05 to 30.05

After the final event we were all exhausted. The following week we only did some administrative tasks. We based ourselves on the mid-semester report and the feedback we got from supervisor and external sources.

Sprint by sprint

We have documented each sprint. These are given in appendix X. An example is given here in table X.X.

6.2 Testplan

To determine requirement, structural and architectural coverage of our product, we have performed software testing. The tests are formalized to make it easier to agree on the coverage between the customer, maintainers and us. The results and process is documented in this chapter.

6.2.1 Testing Strategy Overview

It is common practise to structure tests in three categories. This way, tests can be communicated to developers, stakeholders and high-level non-technical users. Following is our interpretation of each category.

Unit Testing

Unit testing is the process of testing program components individually. The tests invoke methods and structures in the code using different input parameters. The tests are usually written either before or immediately after a module is completed. This way, it is easier to assert that the module does what it is intended. Each test case is independent from each other, so several people can write test cases simultaneously without having to worry about dependencies.

Integration Testing

In development, many features are bundled into different components. The components are then joined together to form a system. Integration testing tests the interfaces to each of these components, and how they communicate with each other. The purpose is to ensure that communication between the components is correct, and that the components work as intended. It can be extensive if those responsible for integration have to review the code in each component, so integration testing abstract code away. If there are any errors, then one will either review the unit tests or notify the author.

System Testing

System testing is a high-level test of the system. It is performed after all of the integrated system parts have been tested and joined together. System testing is a black box test, as anyone should be able to perform the test without having any knowledge of the underlying code. The purpose of system testing is to test if our system fulfills the requirements in the requirement specification. This is important to find out if we meet the believed expectations from the customer.

Acceptance Testing

Acceptance tests are usually executed by the customers. They are written after agreeing on the requirements specification for a delivery. The tests are then verified by the customer. Once both the customer and developers agree on the acceptance test, it will be possible to formally agree on whether or not a delivery meets the given requirements.

6.2.2 Testing Coverage

We wanted to provide complete test coverage, but we did not have the time. Thus, we needed to prioritize what components of the system were most prone to error, and most important to test. The following were our software assurance objectives:

- Ensure that the system can be used by many users
- Ensure that the contest can be held without any error that would critically impact the contest

Errors that solely impacted user experience were not prioritized to test. The majority of these were intended to be found from debugging the system. Since the developers would work closely with each other on GentleIDI, we concluded that we would fix small errors in regression. If our team had more members, or if we had been working in different locations, this would have been a higher priority.

In most projects, testing is used to ensure requirements coverage. In our case, however, with frequent customer-meetings and iterative development, we have not had a strong need for this. The customer has had access to prototypes of our solution and our source code. In order to see that the product does as intended, they could simply try it out for themselves. Some consequences of this is discussed in section X.X.

As per our software assurance objectives, our largest focus has been simulating the role of a contestant. To meet our objectives, we intended to do a full coverage of all contestant scenarios. The privileged users were believed to be technically experienced and without intention to do harm. We still felt it was important to prevent user errors, but our coverage was not as complete for these usergroups.

Since we were developing a website that would feature many users, developer testing alone could never simulate peak values for system demand. Therefore we have relied on load testing. Here, we gave our web server a fixed amount of HTTP requests per second, hereafter RPS. What pages were used in the simulation was determined by us. Thus, our testing also extends to cover simulated peak values for high loads.

Our lacking experience in web development meant that it was hard for us to understand what components could go cause errors. Wikipedia holds a large list of categories that could be tested[1], but we avoided many of them, as it would take to long for us to gain a structural way to test these areas, combined with the lacking experience.

6.2.3 Our approach to testing

Unit Testing

Our unit tests are given in [source code].The reason for not including unit tests in the testplan is because it will be redundant, and take up unnecessary space in the report.

We performed unit testing after the completion of a testable module. The unit tests use the PyUnit framework, and is written by another person than the one who produced the code for the module. In other words, if person A makes module M, then person B will write the unit tests for module M. The reason for having another person writing the test for a module is because that will give more people insight in the code, and make it easier to discover problems.

[1] http://en.wikipedia.org/wiki/Non-functional_requirement

Integration Testing

Each integration test will test a different interface. The interface is defined as the connection between the different components in our system. The pre- and post-condition sets the boundaries for the test. Input and output is used to determine if the test produces the expected output with a corresponding input. Comment is just an additional field in case we feel the need to explain a test more thoroughly to avoid misconceptions. The motivation behind integration testing is that we can determine whether a module has been successfully integrated. By going through the accompanied tests made for the interfaces that interact with the module

System Testing

Each separate test in the system test is linked to one or more of the requirements from the requirements specification. The template for system testing starts with specifying which function is being testing. After that we say what the action/input should be, and what the expected result is. The expected result needs to be achieved for the test to be considered successful. Every separate system test is connected to one or more of the requirements from the requirements specification. This is to ensure that the system meets all the requirements set by the customer.

Acceptance Testing

The customer performed an acceptance test before each release of the system, so they could confirm that we met the expected requirements. The acceptance test was based on our system test, with the customer executing the tasks in the system test. The acceptance test was approved when the customer was satisfied with how we implemented the requirements.

Integration Test

Each test has a unique identifier, name, pre/post-conditions and corresponding input and output. An example is given in table X.X.

ID	IT-01
Interface name	Add sponsor
Pre-condtion	Contest is created
Post-condition	Sponsor and image
Input	Image, URL
Output	sponsor in contest

In section X.X[12. Evaluation of testing methods] we explained why our coverage by integration testing was not extensive. The written integration tests are from our M-03 milestone, and do only cover the requirements that was necessary for that milestone. As such, we have chosen to move all the integration tests to appendix D.

We formally agreed on what modules our system was made out of and their interfaces. Figure X.X shows our view on the system as per milestone M-03. In figure X.X, we have replaced some default UML symbols and replaced them with the equivalent UML stereotype. The explanations are given in table 8.1. The tables in figure 8.3 are based upon the interfaces defined in figure 8.2.

UML stereotype	**Function**
<<provides>>	The component delivers the given functionality
<<requires>>	For the component to work, the interface must have the given interface

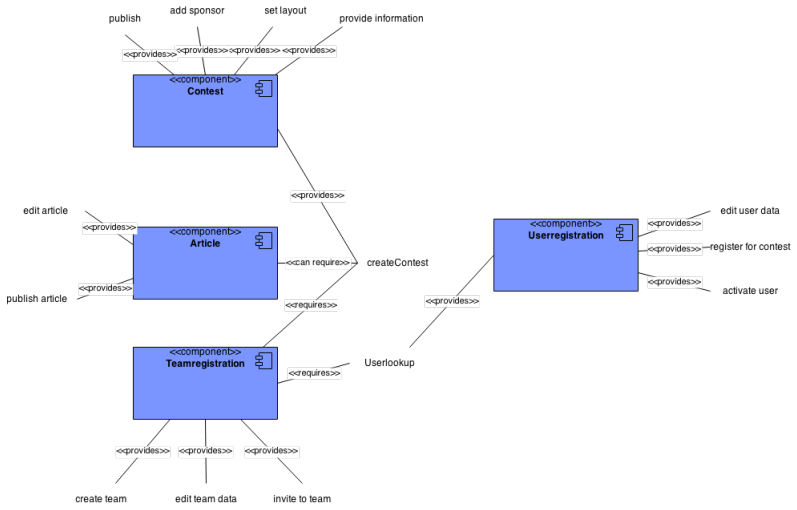


Figure 6.1: Diagram from milestone M-03. Each interface connection, especially "createContest" has been tested

6.2.4 System Test

Our system tests cover all the functional requirements. All tests are written as successive cases. This means that the tests do not cover scenarios for how the system should respond when a user performs an error or another external fault occurs. The complete listing is in table X.X.

ID	Function	Action/Input	Result	Req	Pass/Fail
TF-01	Create a contest, and publish an article to thatcontest. Edit article.Then, delete the contest.	Contest name, article text	Contest and article is no longer publicly available	FA-16,	PASS
TF-02	As a contestant, create a team and invite contestants. Go to profile page and see which team the contestant is amember of. Then, delete the team	Team, contestants, contest	First contestant in team, then contestant not in team	FE-01, FE-02, FE-04, FE-06, FC-04	PASS
TF-03	Add custom css, specify custom settings,	Existing contest, css, compiler flags, penaltysystem, maximum numbers of contestant, maximum number of contestant per team	Contest with custom css and settings	FA-05,	PASS
TF-04	Log in as admin, and enable all judges to createa contest. Then remove and add a judge, by escalating and de-escalating privileges from contestant.	Admin account, contestant account	Zero changes to system.	FA-09,	PASS
TF-05	Log in as judge, create a problem and upload cases. Upload different solutions; one correct, one erroneous, and one that loops forever. After that, modify the problem before deleting it.	Problem, solutions, erroneous code, judge account	Only the correct solution should give points.	FJ-01, FJ-02, FJ-03, FJ-04, FJ-05, FJ-06, FJ-07	PASS
TF-06	Add two execution nodes with different compiler supports. Change both nodes, such that they take each other's compiler setting. Then remove both nodes.	Compiler profiles, available nodes, production server, administrator account	zero added nodes, no errors in execution	FA-12, FA-13	PASS
TF-07	As a contestant, submit a question to the judge. As a judge, receive a notification, and answer both the contestant and globally.	Contestant, contest, question, answer	All contestants should be able to see message, successful communication between judge and contestant	FJ-08	PASS
TF-08	Create a contes-	Contest-data,	Activation data re-	FC-01,	PASS

6.2.5 Non-functional testing

Our non-functional tests ensures non-functional requirements coverage and scenario correctness. Additionally, it defines acceptance criteria related to the performance of our solution.

The tests related to performance usually comes in pairs, a value and the double of that value. This applies to the input and expected result. This is to ensure that system performance does not scale down in a non-linear way.For example, if "X" transactions are processed and the server begins using swap memory instead of RAM, this would mean that a high load would cause an exponentially slower load rate for a high number of transactions.

Often, as mentioned in section X.X[TODO: REFER testing phases], we did inspection tests. Thus, table X.X below does not contain all tests that are executed, and the table only covers the first 12 non-functional requirements. The documented tests do, however, ensure some requirements coverage.

Case	Input	ID	Expected result	Pass/Fail
Adding 500 contestants	500 users	NF-04	Ability to add yet another	PASS
Adding 200 teams	200 teams	NF-05	Ability to add yet another	PASS
Adding 20 judges	20 judges	NF-06	Ability to add yet another	PASS
Adding more than one admin	> 1 admin	NF-07	Ability to add yet another	PASS
Upload a solution which is less than 50kB	Solution > 50kB	NF-08	Successful delivery	PASS
Upload a solution which is greater than 50kB	Solution > 50kB	NF-08	Error message	PASS
Gather some test persons not familiar with the system and have them use the system as a contestant	System	NF-09	They should be familiar with the system after 5 minutes	FAILED
Gather some test persons not familiar with the system and have them use the system as a judge	System	NF-11	They should be familiar with the system after 10 minutes	PASS
Gather some test persons not familiar with the system and have them use the system as an admin	System	NF-10	They should be familiar with the system after 15 minutes	PASS
Page responsiveness with at least 5 RPS	HTTP GET and POST to all pages	NF-01	Response-time < 100 ms	FAIL
Page responsiveness with at least 10 RPS	HTTP GET and POST to all pages	NF-01	Response-time < 200 ms	FAIL

In table X.X it can be seen that not all the tests passed. This is elaborated on in section X.X[TODO: refer requirements].

6.2.6 Risk and Dependencies

In section X.X we mentioned that we did not test whether or not the privileged users of the system made any errors. They were responsible for uploading solutions and content on the web site.

The majority of our testing has been inspection-based. This has been considered time efficient for us. As we have developed the entire system from scratch, and worked with it over a longer period of time, we have had good knowledge of the system. Thus, inspection-based testing has been largely effective. The problem is that there is no way to formally agree on what components have been tested, or to what extent. Additionally, future maintainers are much more likely to make errors as they do not know what components are connected, or what kind of tests should be executed.

Our lacking experience in web development means that our test coverage is not complete. Some errors, for example, were caused by improper charset encodings, an error none of us knew we had to consider. To mitigate these kind of risks, more experienced developers should participate in writing tests.

Chapter 7

Risk Management Framework

A risk is an event or condition that, if it occurs, could have a negative effect on a project's objectives. To avoid these risks, and to be able to deal with them effectively, we established a risk modelling framework. Our framework is based upon our own experience and examples from the many documents that exists on the subject.

By explicitly writing down corresponding actions for risks that occur, we could deal with risks without disagreements. It also let external parties get an overview of what risks we are aware of, and how we reviewed them. The external party can then notify us of unknown risks or modifications to our priorities. Terminology and Categories

To structurize our risk register, we divided each into the following categories:

- **Budget risks** are all risks that can be associated with financial aspects of our project.
- **Organizational risks** are those that might arise because of group structure and task delegation.
- **People Management** comprises all risks associated with team management and each individual in the group.
- **Requirements risks** are related to errors in requirements engineering.
- **Schedule risks** are about meeting deadlines and task delegation.
- **Technology and tools**; product talk about technical risks that might arise with tools and our product.

To prioritize our risks, we have also given each risk a probability, consequence and total risk, abbreviated Pr, C, TR, respectively. Each of these were assigned values from 1-10, where 10 indicated "very high". A 10 translates to the following for each field:

- **Consequence**: event of risk will be fatal to our project.
- **Probability**: risk will probably happen
- **Total risk**: The risk is a big threat and should be monitored closely.

Total risk is calculated as Consequence x Probability. By multiplying these numbers, we get a sorted list of the most dangerous risks. Scope of Risk Assessment Finding the right balance to the extent of documentation is difficult. Extensive risk-frameworks can consume more hours in maintenance than they save. To deal with our lacking experience, we only wanted to document the most likely risks. To us, this meant only including risks with a total risk value of more than 30

We considered specifying additional information to each risk, like context and associated risks. However, we felt every member of the group had a similar understanding of the risks, so writing this information down would be superfluous. In addition, since the risks were orally reviewed, we did not want to rely too much on what had been written down.

7.1 Risk Identification

We tried to involve every group member in the making of the risk register. The estimates from 1 to 10 were assigned based on our own experience from previous projects. The list was filled out by three members of the group, and then later presented to the whole group for reviewal and agreement on the values.

Risks that became known in later parts of our development was promptly added to our risk register. We expected few of these, and few did occur, so we have not performed any revision control. These risks are informally discussed in chapter X. Our means of identifying risks was through discussions and agreements that we were not performing optimally.

7.2 Risk Monitoring

Our primary method for surveilling risks was weekly discussions. In these meetings, we had open discussions of the group's progress and development. In addition, we had one monthly meeting where we would discuss the risks more thorough and in-depth. This involved re-discussion of the group's expectations and our involvement in the project. These monthly meetings were referred to as "snapshots". The snapshots specifically addressed the problem that many projects start out quite ambitiously, but tend to deteriorate, something we wanted to avoid.

To avoid groupthink[1] and complacency, we required each group member on our weekly meetings to mention three good and three negative points. After that, each member could bring up extra topics for discussion. For each discussion, we made sure to be conclusive by explicitly writing how to deal with a given problem.

We have frequently involved the supervisor and customer in our process. We made sure to ask for insights on our development progress. After each meeting we also wrote down meeting minutes and a summary. This was later sent to the respective party to ensure agreement on what had been concluded in the meeting.

7.3 Complete List of Risks

We have chosen to put the complete list in appendix X.

[1] The concept of trying to avoid conflict by not speaking one's mind. For more, see: http://www.psysr.org/about/pubs_resources/groupthink%20overview.htm

Appendix A

Sprints

This appendix holds an overview over our sprints, throughout the project. For a mote complete list over packages completed see [insert section where activity/sprint backlog are]

This is just an overview were we are trying to bring out the more important aspects of our sprints.

A.1 Template

Sprint: <sprint nr>	Working towards: <insert milestone
Overview over packages to be completed: <Insert packages to be completes>	
Improvements: <insert list over things we want to improve about ourself>	
Notes: <any notes>	
Packages completed: <insert packages actually completed>	
Summary: <A brief summary over the most important aspects>	

A.2 Sprint 0

Sprint: 0	Working towards: M-01
Overview over packages/tasks to be completed: • Get an overview over the course • Get to know the old system	
Improvements:	
Notes: • This was the first meeting after getting the assignment	
Packages completed:	
Summary: This was still early in the process so most of the time was spent getting an overview over the whole thing.	

A.3 Sprint 1

Sprint: 0-a	Working towards: M-01
Overview over packages to be completed: • Read and learn the requirement received from the customer • Set up tools • Project management • Learning tools and framework	
Improvements: • A better meeting structure	
Notes:	
Packages completed: • Tools for communication was set up	
Summary: Learning to know the requirements and the subject as a whole was our main concern at this stage. We also did some research on what framework we should use.	

A.4 Sprint 1

Sprint: 1	Working towards: M-01
Overview over packages to be completed: • Project management • Install and learn tools • Report	
Improvements:	
Notes: • Tino and Eirik was sent out on seminar. Learning about SCRUM • Trying to use ICEScrum for Scrum related activites	
Packages completed: • WBS • Risk assignment • Functional requirements • Class diagram	
Summary: Most of the tools was set up, we started to some modelling, in order to get a better overview over the system to be implemented. This was also documentations to be used in the report. We also systematized the requirements in order to communicate with the customer. Project roles was also distributed.	

A.5 Sprint 2

Sprint: 2	Working towards: M-01
Overview over packages to be completed: • Project management	
Improvements:	
Notes:	

Packages completed:
• Requirement specification
• System architecture
  – Flow charts
  – class diagrams
• ER-Models
• Preliminary report	

Summary: At this point we had a rough understanding of the work ahead of us, and we were able to start modelling possible solutions. This was also close to the deadline for the preliminary report and as a consequence a lot of time was spent on the report.	

A.6 Sprint 3

Sprint: 3	Working towards: M-02
Overview over packages to be completed: • Development	
Improvements: • Better sprint planning • We should improve our task delegation • We should prioritize tasks	
Notes:	
Packages completed: • Development	
Summary: During the past two sprints we had primarily been planning and doing administrative tasks. This sprint marked the end of that phase. We moved on to actual implementing. However, we ere not familiar with the tools and frameworks available to us, and as a consequence we decided to use this sprint to get everyone up to date on Django//python. We had a coding night this sprint. Working all members together.	

A.7 Sprint 4

Sprint: 4	Working towards: M-02
Overview over packages to be completed: • User-interface • Project management	
Improvements: • The activity diagrams does not reflect upon our actual work done.	
Notes:	
Packages completed: • User interface	
Summary: During sprint 4 we knew we had to improve our WBS. We had a long meeting where we rebuild our backlog, reviewed SCRUM and created a release- and backlog.	

A.8 Sprint 5

Sprint: 5	Working towards: <insert milestone
Overview over packages to be completed: • Development • Report • Tesplan	
Improvements:	
Notes: • This sprint we had a meeting with the supervisor discussing the activity diagrams. Show suggested that we switch them with our	
Packages completed: • Sponsor support • Testplan	
Summary: We had a good overview over what should be in the report at this point. A finished version was right around the corner. In general, this weeks meeting went much faster than the last. The group was happy about that.	

A.9 Sprint 6

Sprint: 6	Working towards: M-02/M-03
Overview over packages to be completed: • Mid-term report	
Improvements:	
Notes:	
Packages completed: • Mid-term report • Testplan • User-interface completed in bootstrap	
Summary: This sprint we finished the mid-term report and the user-interface was completed. We was happy with the resut. We also finished the mid-term in food time before the actual deliver.	

A.10 Sprint 7

Sprint: 7	Working towards: M-03/M-04
Overview over packages to be completed: • Representation • Implementation • Write tests.	
Improvements: • We must be better to work with other group members code.	
Notes:	
Packages completed: • Login completed • User registration completed • Team registration completed	
Summary: During this sprint we had boost with the implementation. We were busy making our Firs release. Unfortunately we did not have time to set up the solution live this sprint .It was postponed to after the weekend.	

A.11 Sprint 8

Sprint: 8	Working towards: M-05
Overview over packages to be completed: • Testing • Set up solution live • Fixing bugs • Peer evalutaion	
Improvements:	
Notes: <any notes>	

Packages completed:
• Testing
• Bug fixing
  – Change email
  – Forgot password
• Peer evalutaion	

Summary: After we put the solution up, there was sum bugs and testing to be done. We had not had the opportunity to test, by our standards, yet. We did this while the solution was live.	

A.12 Sprint 9

Sprint: 9	Working towards: M-06
Overview over packages to be completed: • Implementation • Permission testing • user manual • Project mamagement	
Improvements: • We had to be more consistent with testing • Better to fill out sprint documents.	
Notes: • We received the Peer Evaluation.	
Packages completed: • Possible to upload solutions • Models	
Summary: This sprint was probably our worst planned sprint. With better planning we could have finished a lot more coding. Unfortunately this was not the case and we spent unnecessary much time in the wrong direction. We were, however happy with our peer evaliation.	

A.13 Sprint 10

Sprint: 10	Working towards: M-05
Overview over packages to be completed: • Implementation	
Improvements: • Still improvement to been done with filling out sprint backlog.	
Notes: • This sprint was 9 days long	
Packages completed: • Implementation – Execution nodes – Compiler profiles – Upload solution	
Summary: This was the last sprint before Easter. We were more thrilled with this sprint but. we knew had to shorten our easter vacation. We had a good start with much of the implementation and we finally felt like we had a good overview over everything.	

A.14 Sprint 11

Sprint: 11	Working towards: M-05
Overview over packages to be completed: • Implementation	
Improvements: • We knew we needed discipline to make it	
Notes: • Parts of this sprint was during easter • This sprint was 11 days	
Packages completed: • Upload submission • Penalty systematized • Review system status • Judge supervisor • Error messages	
Summary: During this sprint, we did not setup a sprint backlog. Instead we kept an well documented TODO list. Every day all members would tell which tasks from the TODO list they would work on. At the end of the day we told each other what was missing. This sprint went great and we were actually finished some days before M-05-.	

A.15 Sprint 12

Sprint: 12	Working towards: M-07
Overview over packages to be completed: • Development • Bugfixes • Setup	
Improvements:	
Notes: • Last sprint before final event	
Packages completed: • Highsvore • CSV and PDF support • Several execution nodes • judge contest acsess	
Summary: Are last sprint before the final event consisted mainly on small bugfixes. There were, however, some tasks that took longer time than estimated. That would be CSV and PDF	

A.16 Sprint After

Sprint: After	Working towards: M-08
Overview over packages to be completed: • Final report • Small bugfixes • User Manual	
Improvements: • Effiency and communication is import this last period	
Notes: • We did create a traditional sprint backlog for this sprint. We did however have frequent meeting discussing what to finish when	
Packages completed: • Final report • small bugfixes • User manual	
Summary: When we worked towards the final report we decided on a different tactic than the other sprints. Instead of creating a sprint backlog, holding all the tasks, we broke down the report into chapters. Some of which was already finished. For each chapter we talked about what key ponts we wanted to write about for so deciding a pair that should write that part. Then, before we met next time, another pair would view, comments and generally share some points about that chapter.	

Appendix B

User stories

Admin

ID:	Priority	Story
SA-01	HIGH	Will be able to create a new contest. When doing so a new web page should be created, but whether the site should be immediately published or not is optional. The content of the new site follows a strict template, but adding a custom css-file will be possible. Each contest has got its own settings, containing a list of supported compiler profiles, compiler flags, penalty system, maximum number of contestants, maximum number of contestants per team, and of course a date and a name. When creating a contest the admin needs to provide a name and a date, the other settings may be skipped and default settings will be used.
SA-02	HIGH	Users are organized in user groups(admin being one of them). By default three usergroups are provided, admin, judge, contestant and functionary. The entire solution is based on independent modules of functionality and each user group has got access to a subset of these modules. The admin is the only non-modifiable user group, admins have access to all modules. The admins can modify all other user groups, change permissions of a group and remove/add member to a group, this includes promoting new admins. The admins are also able to deactivate users, and even remove them from the database.
SA-03	MED	The system is able to gather a large variety of statistics, what data is to be collected is decided by the admins.
SA-04	HIGH	The system uses a collection of nodes(computers) for assessing submissions. The admins can add a node by providing an IP address and the username and password of a privileged user on that node. These nodes can also be removed by the admins. The nodes can also be managed in terms of compiler profile support.

SA-05	HIGH	The web page associated with a contest consists of a set of news items, these can be added by the admin. As with the entire contest web page the publishing of the news item can be set to a certain date and time. The news items can also be removed or modified later on.

Role: Judge

ID:	Priority	Story
SJ-01	MED	A judge can submit a problem, where he/she will be able to upload cases with input/output. He/she can give every case a name. For each problem the judge can set a resource limit (time + memory) for each compiler profiles. He/she can upload different solutions that gives the right output, timeout and the wrong answer. All the solutions should be run-able and produce an output about the expected result, and if the execution time is inside the given boundaries. He/she should also be able to check that all problems have associated solutions that give right and wrong answer, and timeout.
SJ-02	MED	A clarification system will be available to judges, where they can receive and respond to messages from contestants. When receiving a message, the judge will get a notification (possible in in the bottom right corner of the website, [Design choice]) . A judge can choose to either send a global message or a message to a contestant or a team. A global message will be sent to every contestant in the competition.

Role: Contestant

ID:	Priority	Story
SC-01	HIGH	A contestant should be registered with an email, name, gender, and study programme and level. When registered, he/she should receive a confirmation email. After confirming the account, a contestant should be able to log in.
SC-02	HIGH	When a contestant is logged in he/she will have access to account information and which teams he/she are invited to, as well as earlier contests and teams they have participated in. The contestant should be able to edit account information
SC-03	MED	A clarification system will be available to contestants, where they can ask questions to the judges. They will also have access to answers the judges have marked as global.

Role: Functionary

ID:	Priority	Story
SF-01	LOW	When a team completes a problem, a table containing the group name and location should be updated to include this. Each problem has a corresponding balloon colour. A balloon functionary should be able to register a balloon colour to each problem.

Role: Teams

ID:	Priority	Story
ST-01	HIGH	A contestant must [18.02] be able to register a team, upon registration he/she is required to input team name, whether or not the team is onsite, a team password, and a email for the team leader.
ST-02	HIGH	The team leader should be able to edit the team information, invite new members, and delete the team before the competition. To invite new members you input their email, and they receive a registration link, where he/she inputs name, gender and nickname. If the contestant [changed from email 20.02] is already in the database from a previous competition, the email they receive contains a confirmation link. Every contestant can manage the team they are a member of. All informations is editable in the team overview which can be reached from a contestants login. A confirmation email is sent to the edited user.
ST-03	MED	A team should be able to deliver submissions to problems, and get a response from the system. The response should be whether the submission is right, wrong, or gives timeout.

Appendix C

Installation guide

This is the complete install guide for GentleIDI. The guide will assume that the reader is has got some basic linux skills. You should be capable of installing packages by means of a package-manager like apt, yum etc.

Though GentleIDI is not tightly linked with any specific linux distro, this guide assumes that you're using Ubuntu Server 14.04. This is the only distro on which the system has been tested thoroughly at the time of writing.

GentleIDI is in many ways a straightforward django-based website, and hence there are a lot of possible setups to choose from. This guide is inspired by a guide written by Michal Karzyński, and will guide you through the steps of setting up the system using a combination of gunicorn and nginx.

C.1 Creating your users

Running a website as a user with root privileges or anything of the sort is far from recommended. Therefore you are recommended to create a new user and a new usergroup. The names of both the group and the user can be chosen as you please, but the rest of the guide will stick to using a user called gentleidi and a group named webapps.

$ sudo mkdir -p /webapps/gentleidi

$ sudo groupadd --system webapps.

$ sudo useradd --system --gid webapps --home /webapps/gentleidi gentleidi

$ sudo chown gentleidi:webapps /webapps/gentleidi/

Now you have a user named gentleidi which is a member of the usergroup webapps, and whose home directory is /webapps/gentleidi.

In addition to the user we just created, we need another user, specifically used to run the untrusted software submitted by the contestants. GentleIDI assumes that this user is named gentlemember. However, changing this value in the source is no complicated matter.

$sudo useradd --system gentlemember

The system needs to be able to execute commands both as gentleidi and gentlemember. As the web server runs as gentleidi we need to make sure that gentleidi can execute commands as gentlemember. Add the following line to your sudoers file.

gentleidi ALL=(gentlemember) NOPASSWD:ALL

If you don't know how to edit your sudoers, to open the sudoers file in a text editor simply type the following command:

$sudo visudo

Now we've got two users, one capable of executing commands as the other. What we want to do now is to ensure that gentlemember is unable to communicate via network. This is done by applying two rather straightforward iptable rules.

$ sudo iptables -A OUTPUT -m owner --uid-owner gentlemember -j LOG

$ sudo iptables -A OUTPUT -m owner --uid-owner gentlemember -j REJECT

Though this will restrict the user's network access, be aware of software installed on your system which is capable of switching to another user.

C.2 Setting up the environment

Due to a lot of strict changes made in python versions, a lot of libraries do not work across different versions of python. This leaves python in a situation where program A might need python to be version X and program B might need python to be version Y. So, what do you do? You setup a virtual environment.

Virtual environments is a way of setting up separate python setups for different sets of programs.

What we want to do is to turn the home directory of the gentleidi user into a virtual environment.

$ sudo apt-get install python-virtualenv

$ sudo su gentleidi

$ virtualenv ~/env

Now that you've got a virtual environment you can start filling it with something useful, like the content of the project's git repo.

$ cp -r /path/to/repo/IDIOpen/ /webapps/gentleidi/

Please note that you only need the wsgi folder from the repo, however, updating is a lot easier when all you've got to do is pull the latest version directly using git. The downside is that you could possibly end up committing your production system configuration files etc to the repo. However, we're going to assume that you will not be developing directly in your production system, and thereby avoid the hazard.

Before leaving this step, ensure that the files in /webapps/gentleidi has got the correct file permissions.

$ sudo chown -R gentleidi:webapps /webapps/gentleidi

C.3 Installing required packages

Now it's time to start making sure that you've got the packages you need to run GentleIDI.

$ sudo apt-get install git nginx libmysqlclient-dev python-dev

You might already have most of these packages, however, better safe than sorry.

The next thing you need to do before continuing is to log in as gentleidi and activate your newly created virtual environment.

$ sudo su gentleidi

$ source ~/env/bin/activate

Installing the required python packages via pip is easily done. In the project root directory there's a file named requirements.txt. This file is simply a list of required packages, to install them simply execute the following:

$ pip install -r requirements.txt

C.4 Database

GentleIDI needs a database to store its data. This guide will show you how to setup GentleIDI with a mysql database server, however, if you feel like using mariaDB, postgresql, or even SQLite, then please do. Any database server supported by Django is supported by GentleIDI.

Naturally you don't need to install the database server on the same host as the web server, that's what we'll do for now.

$ sudo apt-get install mysql-server

Now what we need to do is to create a database and a mysql user that GentleIDI can use. During the install process you were required to set a root password for the mysql-server. Login as root and perform the following commands:

> CREATE USER 'gentledb'@'localhost' IDENTIFIED BY 'password';

> GRANT ALL PRIVILEGES ON * . * TO 'newuser'@'localhost';

> FLUSH PRIVILEGES;

> CREATE DATABASE gentleidi CHARACTER SET uft8 COLLATE utf8_general_ci;

Remember to replace 'gentledb' and 'password' with a suitable username and password.

Now you need to ensure that GentleIDI uses your newly created database. Edit the DATABASES entry in IDIOpen/wsgi/openshift/settings.py

if MYSQL:

```
DATABASES = {
    'default': {
        'ENGINE'   : 'django.db.backends.mysql',
        'NAME' : 'gentleidi',
        'USER' : 'gentledb',
    'PASSWORD' : 'password',
    'HOST' : 'localhost',
    'PORT' : '3306',
    }
}
```

In order to make sure that the database is working properly, login as gentleidi, activate your environment and synchronize GentleIDI's database.

$ sudo su gentleidi

$ source ~/env/bin/activate

$ python ~/IDIOpen/wsgi/manage.py syncdb

$ python ~/IDIOpen/wsgi/manage.py migrate

If this command terminates properly, then your database should be good to go. In fact you should be able to run GentleIDI on a development server at this point. But first, you need to create a admin account.To do so, simply execute the following:

$ python ~/IDIOpen/wsgi/openshift/manage.py createsuperuser

To start the development server run:

$ python ~/IDIOpen/wsgi/openshift/manage.py runserver

You should now have a working website running on port 8000. However, you have no execution nodes available to evaluate submissions, and you're using Django's development server, which scales horribly.

C.5 Gunicorn

Now it's time to install replace the Django development server with a proper application server, gunicorn. Member to be logged in as gentleidi, and to activate your environment before doing this.

$ pip install gunicorn

Now we need a script that launches gunicorn and GentleIDI appropriately.

```
#!/bin/bash
NAME="GentleIDI"                                    # Name of the application
DJANGODIR=/webapps/gentleidi/IDIOpen/wsgi/          # Django project directory
SOCKFILE=/webapps/gentleidi/run/gunicorn.sock       # we will communicate using this unix socket
USER=gentleidi                                      # the user to run as
GROUP=webapps                                       # the group to run as
NUM_WORKERS=3                                       # how many worker processes should Gunicorn spawn
DJANGO_SETTINGS_MODULE=openshift.settings           # which settings file should Django use
DJANGO_WSGI_MODULE=openshift.wsgi                   # WSGI module name
echo "Starting $NAME as whoami"
# Activate the virtual environment
cd $DJANGODIR
source /webapps/gentleidi/env/bin/activate
export DJANGO_SETTINGS_MODULE=$DJANGO_SETTINGS_MODULE
export PYTHONPATH=$DJANGODIR:$PYTHONPATH
# Create the run directory if it doesn't exist
RUNDIR=$(dirname $SOCKFILE)
test -d $RUNDIR || mkdir -p $RUNDIR
# Start your Django Unicorn
# Programs meant to be run under supervisor should not daemonize themselves (do not use --daemon)
exec /webapps/gentleidi/env/bin/gunicorn ${DJANGO_WSGI_MODULE}:application \
 --name $NAME \
 --workers $NUM_WORKERS \
 --user=$USER --group=$GROUP \
 --log-level=debug \
 --bind=unix:$SOCKFILE
```

Place the contents of the previous page in the following file: /webapps/gentleidi/env/bin/gunicorn_start

Make sure that the script is executable:

$ sudo chmod u+x /webapps/gentleidi/env/bin/gunicorn_start

1. (a) nginx

As mentioned previously this setup relies on a combination of gunicorn and nginx. At this point gunicorn should be working properly, and it's time to setup nginx.

If you have not already installed nginx, do so now:

$ sudo apt-get install nginx

Now you need to create an nginx configuration file for gentelidi.

Store the content found below in the following file : /etc/nginx/sites-available/gentleidi

```
upstream hello_app_server {
 server unix:/webapps/gentleidi/run/gunicorn.sock fail_timeout=0;
}
server {
   listen   80;
   server_name example.com;
   client_max_body_size 4G;
   access_log /webapps/gentleidi/logs/nginx-access.log;
   error_log /webapps/gentleidi/logs/nginx-error.log;
   location /static/ {
       alias   /webapps/gentleidi/IDIOpen/wsgi/static/;
   }
   location /media/ {
       alias    /webapps/gentleidi/IDIOpen/wsgi/media/;
   }
   location / {
       proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
       proxy_set_header Host $http_host;
       proxy_redirect off;
       if (!-f $request_filename) {
           proxy_pass http://hello_app_server;
           break;
       }
   }
   # Error pages
   error_page 500 502 503 504 /500.html;
   location = /500.html {
       root  /webapps/gentleidi/IDIOpen/wsgi/static/;
   }
}
#EOF
```

In this configuration nginx is set to log all accesses and errors. These files need to be created by the following commands:

```
$ sudo su gentleidi
$ mkdir ~/logs
$ touch ~/logs/nginx-access.log
$ touch logs/nginx-error
$ exit
```

All you need to do at this point is to enable the nginx site. This is done simply by creating a symbolic link from the configuration file in sites-available to sites-enabled.

$ sudo ln -s /etc/nginx/sites-available/gentleidi /etc/nginx/sites-enabled/
$ sudo rm /etc/nginx/sites-enabled/default
$ sudo service nginx restart

You should now have a working website. All that's left is making management a little easier and adding some execution nodes.

C.6 Supervisor

Supervisor is a utility for defining and managing jobs. In this case we're going to define two jobs, one for managing the website, and another for managing an execution node.

You need to create two files to make this happen:
/etc/supervisor/conf.d/gentleidi.conf
[program:gentleidi]
command = /webapps/gentleidi/env/bin/gunicorn_start
user = gentleidi
stdout_logfile = /webapps/gentleidi/logs/gunicorn_supervisor.log
redirect_stderr = true
#EOF
/etc/supervisor/conf.d/celery.conf
[program:celery]
command=/webapps/gentleidi/env/bin/celery worker -A openshift -l info
directory=/webapps/gentleidi/IDIOpen/wsgi
environment=PATH="/webapps/gentleidi/env/bin:%(ENV_PATH)s"
user=gentleidi
autostart=true
autorestart=true
redirect_stderr=True
#EOF
Create the log files that you've referenced.
$ mkdir /webapps/gentleidi/logs/
$ touch /webapps/gentleidi/logs/gunicorn_supervisor.log
Read the newly created configuration files.
$ sudo supervisorctl reread
$ sudo supervisorctl update
$ sudo supervisorctl restart all

C.7 Multiple execution nodes

The easiest way of setting up multiple execution nodes is to clone the setup on your web server to other machines and then making minor changes.

When setting up multiple execution nodes there are two changes that need to be made. The directory /webapps/gentleidi/IDIOpen/wsgi/private/submissions needs to be shared between all the execution nodes. How you decide to make this happen is up to you. However, sshfs is possibly the easiest solution. Whatever way you decide to mount the directory on your execution nodes, make sure that multiple users are allowed to access it, e.g. the allow_other option for sshfs.

You also need to make sure that all your execution nodes have access to the same database. Make sure that the settings.py is not set to localhost, but rather points to whatever host you decide to use as a database server. Some configuration of your database server might be needed in order for it to accept remote connections. MySQL servers need to change the bind-address property in the /etc/mysql/my.cnf to their actual IP, not localhost(127.0.0.1).

You also need to change the grants for the mysql user in such a way that it is allowed to connect remotely to the database.

Appendix D

Risk List

D.0.1 People Management

Description	#ID	Pr	C	TR	Preventative action	Remedial action
Personal argument	PM-01	8	5	40	Frequent meetings and social events	Open discussion
Dependency on team member	PM-02	6	6	36	Short sprints and team members usually work in groups of two	New meeting where we consider a redistribution of WP
Underburdened team-member; slack	PM-03	7	4	28	Keeping track of the work done by each member as well as the number of hours spent on any given WP. In the beginning of the sprint focus more on an evenly distributed workload among team members.	If the team-member continues to slack put it on the agenda for the next meeting and allow the team-member to explain his/her reasons for slacking.
Team members are late	PM-04	9	2	18	If you are late, you need to bring a cake or cookies to the next meeting	You need to bring a cake or cookies, and if it happends several times, an extraordinary meeting will be called, where new consequences will be discussed.
Team member is not qualified for any assigment	PM-05	4	7	28	Try to keep every member up to date on the entire system by not letting anyone work for too long on the same part of the system.	Add unqualified member to an existing pair working on a WP.
Miscommunication	PM-06	7	3	21	Frequent meetings with discussion about team letting all team members try different areas in the application	As per SDLC; evaluation, analysis, restart assigment
Dependency on external person	PM-07	3	6	18	Frequent communication with the customer.	Well-planned sprints with a low level of dependency between WPs.
Displacement; team members do not feel comfortable in group	PM-08	2	7	14	Social events.	Talk to our supervisor and ask for suggestions
Overburdened team-member	PM-09	4	2	8	Short sprints and small WPs. A team member will only be assigned to a few WPs at a time.	Frequent meetings where WPs can possibly be redistributed.

D.0.2 Budget

Description	#ID	Pr	C	TR	Preventative action	Remedial action
Maintenance costs exceed expectations	B-01	5	3	15	Use highly maintainable frameworks as much as possible, and stick to Open Source as much as possible.	Optimizing code base in hopes of increasing maintainability.
Third party plugin demands more money than initially expected	B-02	2	3	6	We've got a green light for putting GentleIDI under the GNU Public License, which means that we have got free access to software under GPL.	Look for alternative plugins.
Unexpected need for non-free third-party service	B-03	3	3	9	Extensive research on tools needed,before we decide on what we are going to use.	Look for alternative free third-party services
Maintenance requires access to tools/environments that cost money	B-04	2	3	6	Use highly maintainable frameworks as much as possible,and stick to Open Source as much as possible.	Request customer meeting to solve the issue.

D.0.3 Schedule

Description	#ID	Pr	C	TR	Preventative action	Remedial action
Pre-studies require more time than anticipated	S-01	9	7	63	We have a WP for pre-studies, and have included it in our sprints	Revise our WBS, and possible have an increased workload/work-hours in the following sprints, so we don't fall behind our schedule.
Failure to meet requirements on time	S-02	5	8	40	WBS, milestones plan and short sprints (1 or 2 weeks) allow us to focus on deadlines, and continously see our work progress	Have extraordinary meetings with supervisor and the customer to discuss the further development of the project. Be apologetic towards the customer, and come up with a new plan, that the customer is satisfied with.
Sprint-estimations are off	S-03	9	5	45	The whole group participate in planning a sprint, and estimating each task	Re-adjust our estimations in the next sprint, and in that way learn from our mistakes.
Failure to deliver sufficient documentation on time	S-04	5	6	30	WBS, milestones plan and short sprints (1 or 2 weeks) allow us to focus on deadlines, and continously see our work progress	Meetings with supervisor and customer, agree upon a new deadline, and increase the workload the following days to we meet the deadline.
Need for extra technology / features that requires training to use	S-05	3	6	18	We use extensive frameworks who has a lot of documentation, which makes it easier to learn.	Adjust the WBS and our sprints so we take into account that we need more time to learn new technology. Focus on this in the coming sprint planning.

D.0.4 Organizational

Description	#ID	Pr	C	TR	Preventative action	Remedial action
No person has responsibility for an assigment, although it is believed to be delegated	O-01	8	6	48	Strict use of the activity plan. The activity plan should be kept consistent at all times, this way all members know what the others are doing at any given time.	When discovered the given WP should be marked as unallocated in the activity plan and treated like any other WP in the sprint.
Project is, at current point not satisfactory, and it is hard to understand why	O-02	6	7	42	Writing meeting summaries, and in general keeping track of what is being done and how.	Review what work has been done up untill that point, how it has been done, and try to find a solution to the problem.
Bottleneck; in order for team-members to advance, other team members must finish their work	O-03	7	7	49	Try to avoid dependencies between WPs when setting up sprints. In case of such dependencies being unavoidable these WPs should be scheduled at the beginning of the sprint.	Delegate or even create new WPs to the team members currently being idle.
A task is delegated to more than one person	O-04	2	3	6	Strict use of the activity plan. The activity plan should be kept consistent at all times, this way all members know what the others are doing at any given time.	The two members should discuss how the issue should be solved, and update the activity plan according to that.

D.0.5 Tools and tools; product

Description	#ID	Pr	C	TR	Preventative action	Remedial action
End product is not satisfactory	TT-01	2	9	18	Customer meetings regularly, and keeping in contact through e-mail aswell. Give the customer access to our git-repository, so they have access to our source code, and also perform different type of tests (user-testing, etc)	Call in to a meeting with our supervisor, and our customer. Explain what went wrong, apologize and deliver our documentation.
Tools used for development are not suitable / efficient in later parts of the project	TT-01	2	8	16	Researching the tools we use, and planning ahead. Development planning allow us to discover problems before they appear.	Look for alternative tools. If changing tools involve a lot of work, and changes to the project, decide in a meeting if we want to continue with the inefficient tools, or if we want to make the change.
Problems with integrating components	TT-03	7	3	21	Have extensive system documentation and planning. Involve the whole group in the process.	Re-evaluate our system architecture, and look for solutions that won't affect other parts of the system.
Other solutions available make our product less desirable	TT-04	1	8	8	Do thorough work on the system requirements in hopes of providing a system well-tailored to the customer's needs.	Reevaluate the requirements.
Network cannot deal with traffic	TT-05	1	8	8	Keep optimization in mind when developing.	Try to find redundant data being sent possibly apply use of compression.
Submitted program has access to resources	TT-06	5	5	25	Submitted programs are to be run by a sandbox-user with a very restricted set of resources available.	Review code in hopes of finding the bug.
Platform / hardware unavailible, such that testing is difficult	TT-07	2	5	10	We use services provided by companies known to provide good system uptime. Most of our tools are hosted by Red Hat.	Setup temporary development environment.
Tools used in initial development are not available after release, and future developers have difficulty extending product	TT-08	2	3	6	Make sure requirements are written properly, understood properly, succint, etc	Document our work, so it is easy for future developers to understand the system.
Database cannot handle amount of transactions	TT-09	1	4	4	Keep optimization in mind when developing.	Optimize code in order to lower amount of transactions.
A tool does not perform the functions it was intended for	TT-010	2	3	6	Learn the tools properly, and read the documentation provided with each tool.	Look for alternative tools.

D.0.6 Requirements

Description	#ID	Pr	C	TR	Preventative action	Remedial action
Major change to requirements	R-01	5	4	20	Customer meetings regularly where we agree upon a requirement specification.	New customer meeting where we re-evaluate the requirements specification,and which priorities each requirement has.
Customer fails to understand impact of requirements	R-02	2	7	14	Customer meetings regularly where we agree upon a requirement specification.	Customer meeting where we explain the impact of the requirement,and get the customer to explain their requirements that we have different opinions on.
Finished product does not meet requirement	R-03	1	9	9	Customer meetings,they have access to our git-repository where our source code is	Test-events where they can test the functionality. Finish our documentation,and pass it on to other developers. Apologize to the customer.
Failed interpretation of requirement	R-04	3	4	12	Customer meetings regularly where we agree upon a requirement specification.	Customer meeting where we re-discuss the requirement specification,and make sure we understand what the customer wants.

Appendix E

Product Backlog

ID	As a(n)	I want to be able to	So that
A-01	Admin	decide whether new contestpages are published or not	contests can be created when due
A-02	Admin	create a contest	contestants can register to teams
A-03	Admin	publish news	users can recieve information about a contest
A-04	Admin	custom css	to differentiate different contests
A-05	Admin	custom settings for each contest	
A-06	Admin	set penalty system	contestants are given points etc
A-07	Admin	modify usergroups through an interface	maintain control
A-08	Admin	add or remove users from a usergroup	control
A-09	Admin	add or remove users from the system	control
A-10	Admin	determine what statistics are stored/collected by the system	overview and increased user experience
A-11	Admin	add/remove an execution node	scalability and safety in redundance
A-12	Admin	configure exection nodes with compiler profiles	system flexibility and optimality
A-13	Admin	review system status	verify that contest can be hosted (correctly)
J-01	Judge	submit problem(s)	..add content to actual contest
J-02	Judge	upload cases for problem(s)	so that they can test problem submissions
J-03	Judge	upload solutions	assess case correctness to problem
J-04	Judge	verify contest problem sets and solutions	ensure that contest is O.K
CU-01	Customer	clarification system	provide communication between contestants and judges
CU-02	Customer	different usergroups	to have different roles
CU-03	Customer	user manual	ease of use
B-01	Balloon-functionary	view (correct) submissions	hand out balloons
CO-01	Contestant	register as a contestant in IDIOpen	compete in contest
CO-02	Contestant	register and administer team	compete in contest with teammates
T-01	Team	upload submission to problem	to compete
S-01	Sponsor	adspace	to advertize to users
U-01	User	receive (appropriate) error messages when errors occurs	build user-trust and nice nice
U-02	User	intuitive interface design	improved user experience
U-03	User	good response time on webpages	improved user experience
U-04	User	short user transactions (avoid click click click)	improved user experience
SU-01	Supervisor	document development process	overview group's progress

something