IT2901 - Informatics Project II

IDI Open
Programming Contest System

Haakon Konrad William Aasebø
Håkon Gimnes Kaurel
Tino Hakim Lazreg
Filip Fjuk Egge
Anders Sildnes
Eirik Fosse

May 2014

Norwegian University of Science and Technology

Supervisor: Hong Guo

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

0.1 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.

0.1.1 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.

0.1.2 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.

0.1.3 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

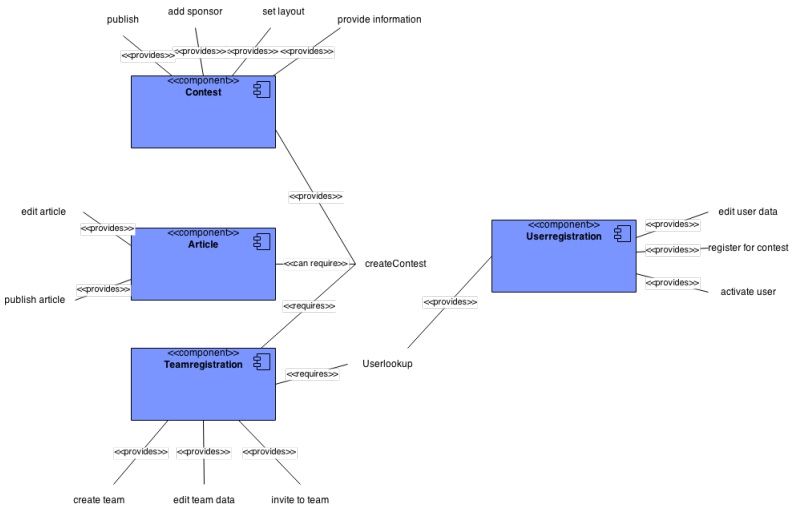

Figure 1: Diagram from milestone M-03. Each interface connection, especially "createContest" has been tested

0.1.4 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 contestant account. Activate the account via email, and change the email. Ask for lost password on the new email.	Contest-data, emails	Activation data received on the email, and all links word	FC-01, FC-02,	PASS

0.1.5 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].

0.1.6 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.