The Software Hut 2007-8

Guide to deliverables

Introduction

In the Software Hut this year you will be assessed in two ways:

1. The client, who will award 50% of the mark based on the application and documentation you deliver.
2. The managers, who will award 50% of your mark based on your process. The mark will be assessed against a set of structured wiki pages which must be updated every week. The managers will periodically ask for evidence to backup the claims you make on these pages, for the final assessment you will produce a poster and show your manager your project and process followed.

There is some more information on some of the topics in this document on the test wiki. You can use this wiki to practice, but please do not mess up the example data and information for other users. You can find it at:
http://ext.dcs.shef.ac.uk/~u0017/COM3420/2007/team1
use username: user1, password: pass8word.

The process

In the Software Hut this year all teams will follow the Extreme Programming method. We expect you to follow these practices:

1. **Pair programming** - You will at all time work in pairs, when programming or creating documentation. Two people per a computer. The person sat at the keyboard is responsible for writing the code. The person sat behind is responsible for checking the code as it is written and recording any errors found from previous coding sessions.
2. **Planning Game** - You will create simplified stories after you have received the initial requirements from the client, and then at each subsequent meeting pick the stories to be developed before the next meeting. You will not work on any other stories until these are complete and tested.
3. **Test driven development** - You will develop unit tests before you write any application code. Unit tests will test the core functionality of your code. You will run these tests to BEFORE you write your code. You will record when you run your tests, and the errors found. For documents you produce you will check them for errors as part of a testing process.
4. **Collective Code Ownership** - You will work as a team on the project, having shared ownership of the code. If asked by the manager every member must know about all of the system. (It is not an excuse to say that only Joe knows about some portion of the code).
5. **Continuous Integration** - If pairs work independently of the rest of the team for a period their changes must be integrated back into the code each week before the management and client meeting. During this meeting each and every week you will show your manager your working system and functionality implemented made that week.

6. **Small Releases** - There will be two releases of the code to the client, he will be expected to check both releases on his own equipment.
7. **Coding Standards** - You will follow coding standards, we will provide some for PHP and Java, should you use other languages you should prepare some and agree them with your manager.
8. **Simple Design** - You will implement what the client requires, and no other functionality.
9. **System Metaphor** - You will have an X-Machine to describe the whole system.
10. **Sustainable Pace** - You will work 15 hours a week.

The document

You will record the exact process followed for each story card on your wiki, the management tool will help you do this, but some entries will need to be made by hand.

There will be four types of page: tests, meeting minutes, story cards and the requirements document.

The tests
A test page will record the tests run for each story, and the results from each time they are run. Typically you will run the tests every time you implement some new functionality so these pages may have quite a few entries on them.

The meeting minutes
The meeting minutes can be generated by the management tool. The minutes should include a discussion of the tasks to be completed, as defined by story cards and/or bugs/problems and the people present. You may also wish to record any team problems such as a member being ill or missing a deadline.

The requirements document
This is a list of:

a) all the likely stories divided into *Mandatory*, *Desirable* and *Optional*
b) a brief project description
c) non-functional requirements including performance characteristics
d) glossary

Put this on the Wiki.

The story cards

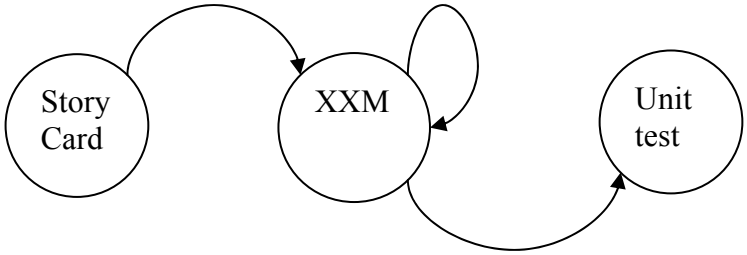
Each story card will have its own page. When you create a story in the management tool, it will create this for you. If the page does not appear automatically then you can click the button to re-publish it in the management tool.

The page will have the following format

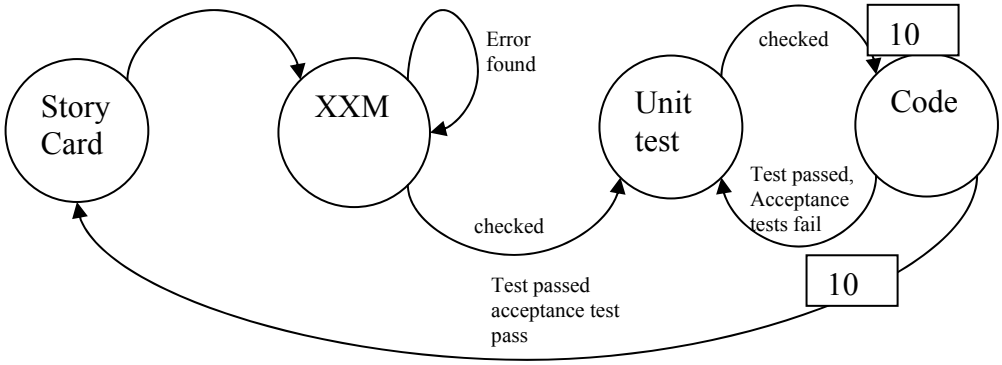
1. *Your process model*

You will need to create this by hand and upload as an image attachment. It should describe a overview of the process followed so far. Each circle represents a product being created or modified, each arrow that product being checked.

A story card has been written



The story card has been written and checked, no errors were found. The XMM was written, but some errors were found and then corrected. Once correct a unit test was written.



The story card has been written and checked, no errors were found. The XMM was written, but some errors were found and then corrected. Once correct a unit test was written. The unit test was checked, and found to be OK, then the code was written to satisfy that test (note - you should only write one small test at a time). The test ran successfully and another test was written. In all ten tests were written to implement the story card in turn, adding the code at each point before the acceptance tests passed. Then the story was signed off (it could also be altered at this point).

2. *The story card*
This is automatically updated by the management tool. Due to caching, changes may not show up immediately, if you want to show them edit the page, make no changes and click save, the card should be updated.

3. *Your events*

Each of the items on the story card is an event. As you complete your project you will generate lots of event. Every coding session, meeting, testing session, etc you do is an event. You can use the timer tool to help your record your events, these will automatically get added to the end of this page and your time sheet will be updated. Typically each event has a checking phase, you should record these separately, so if you write a story card, you then check it, if you write a test it is then checked and so on.

Here is a sample of this list

Date	Type	Time Spent	Description	Problems	Users
15/2/2008	Write story card	1hr	Wrote story card	none	Chris, John
18/2/2008	Check story card	1hr	Checked story card	None	George
19/2/2008	Team meeting	1hr	Client meeting, story card selected in planning game	None	ALL + Client
22/2/2008	Write XXM	3hrs	Updated XXMS to include this story	none	George, John
26/2/2008	Check XXM	1hr	Checked XXM	Found spelling errors	Chris
//2008	Check story	1hr	Corrected spelling errors	None	George, John
//2008	Write tests	1hr	Wrote unit test for “login” in “homeTest.java”	None	Chris, John
//2008	Check tests	10min	Checked, and submitted	Tests failed	George
//2008	Write code	1hr	Wrote code for “login”, its in class “home.java”		Chris, John
//2008	Test code	1hr	Tests submitted, Acceptance tests checked	Tests passed. Acceptance failed.	George

Etc.

Delivery time line

All projects must stick to this timeline it is RIGID. The %SC refers to the percentage of total story cards that we expect to be completed by the end of the week.

Week	**% SC**	
2	5	All story cards captured. Maybe some small progress made on the first and most obvious story before you meet the client.
3	25	

4	50	Requirements document (RD) – this will be partially completed
5	50	Week spent refining acceptance tests and reworking
V		
V		
V		
6	55	Client may revise RD and SCs slightly, we anticipate that this will lead to 15% rework, hence at the end of the week, you will be 70% complete, but then we need to knock 15% off.
7	75	
8	95	
9	100	Most week spent refining acceptance tests and reworking. Client demo on Friday
10	100	Client delivery on Friday. We anticipate up to 10% rework as result of demo plus more acceptance testing and bug fixing.
11	100	Code polishing, Poster production. Final poster presentation on Friday.
12	100	Tidy up document based on any comments from poster presentation.

Management tool
The management tool can be used to support your project, we think if used properly that it makes things easier:

- Use the timer tool to record events.
- Use the meeting minutes editor (in meeting diary entries) to record your minutes.
- Use the story card editor to record your stories.

What is test driven development?

An important part of XP is testing, testing before you write the code. This is difficult to do at first, but if you follow this advice you should find it a little easier.

A common question we hear is that you don't know how you are going to implement something so how can you possibly write tests first. What we recommend is that you write just one test at a time, this might be to log a user into a system, or add a record to a database. The key here is that it should be some trivial task. So write this test, either as a manual test, or preferably using a unit test tool. Now get somebody else to check and run the test, it should fail because there is no implementation - record this run on your wiki. Now write just enough code that the test passes. Record the run of the test on the wiki again. Whilst you write this code you'll probably think of other use cases, make a note of them in the timer tool under problems if using the management tool. Now pick one of the use cases you thought of and write a test for it, and repeat as above.

Now as you can see you are going to be running the tests very frequently - this is why you should use a unit test tool. Once properly set up you will be able to run the tests quickly and easily, and be confident that the system is working as expected.

Testing websites can be particularly hard. You'll find that the selenium test tool is very useful, but be careful. It is easy to write tests that are installation dependant, what happens when you need to run the tests on a different server, perhaps to make sure that it works on it (for example to quickly check that an install on the client's system runs as expected). So avoid hard coded absolute paths that cannot be easily changed in one place.

Now testing a whole system can be daunting, and the GUI can be very tricky to test at times. So make sure you partition your system carefully perhaps using a model-view-controller approach. You may also find tools like PHP Symfony and Ruby on Rails help with this. Write tests for each of the layers individually, providing stub and mock objects as required.