Software Engineering of Distributed Systems, KTH

iAd

ID2007 Modern Methods in Software Engineering, Project

Sike Huang (sikeh), Shanbo Li (shanbo)
10/28/2007

1. A set of developed stories with time estimates

See ***iad_user_stories.pdf***

2. A set of selected stories for the first release

User Story Number	Importance	Risk
1	Normal	Normal
2	Normal	Very low
3	High	Low
4	High	Low
5	High	Low
8	High	Normal
9	Normal	Very low
10	Normal	Very low
12	High	Normal
13	High	Normal
14	Very high	High
15	Very high	Very high
16	Normal	Low
21	Normal	Normal

3. Iteration plan

First iteration: 2, 3, 4, 5, 9, 10, 16

Second iteration: 1, 8, 12, 13, 21

Third iteration: 14, 15

4. The metaphor

The software is the most cutting edge management system for advertising consultancy companies, and it works like your personal secretary, helping you set up advertise campaigns easily with pleasure.

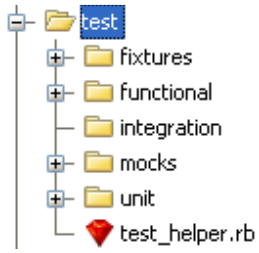
5. Description of test-driven pair-programming process and applied refactorings. Also describe how well managed to estimate what should be done in each iteration.

Most of the time, me (Sike Huang) and my pair (Shanbo Li) are using one laptop to develop the software, in other word, one person setting in front the computer is writing the code while the other one is checking if there is any logical or algorithmic error in the typed code. When one

finds any difficulty, he immediately stops coding and discuss it with the pair. For example, if Shanbo encounters a problem in solving many-to-many relationship mapping, and I figure out a solution, then I take the place of writing, and he can have a relax. Or in that case I want a rest for washroom or something else, my pair Shanbo will be in charge of coding. So the pair-programming is working in turns and brainstorming.

The decision of selecting user stories for the first release, as well as each iteration plan is totally based on the formal experience from my pair worker, Shanbo Li. He wrote his excellent bachelor thesis project based on Rudy-on-Rails framework, which shares some common features and functionalities with this project, so he is confident with his evaluation of importance and risk factors. Though we have put some user stories to be realized in first and second iteration, in the first two week, there is nothing but analysis and design. In contrast, all user stories are successfully implemented during the last iteration, with very high efficiency resulting from pair programming.

We write the project based on the popular Ruby-on-Rails web framework. One of the real joys of the Rails framework is that it has support for testing baked right in from the start of every project.



The skeleton test cases are automatically generated in the subdirectory called test, as in the figure above. By convention, Rails calls things that test models unit tests, things that test a single action in a controller functional tests, and things that test the flow through one or more controllers integration tests. However, because of the time constraints and unfamiliarity of Test::Unit::TestCase, we directly uses those automatically generated test cases, and along the way, we got rapid feedback by writing a little code and then punching buttons in a web browser to see whether the application behaved as we expected.

For applied fefactorings, we only mention few here. One of most used refactorings is “Split Temporary Variable”.

```
@other_staffs = Staff.find :all, :conditions => "director_id !
@campaign_staffs = @campaign.staffs
@my_staffs = Staff.find_all_by_director_id(@campaign.director.

@staffs_in_campaign_but_not_mine = @campaign_staffs - @my_staf
@other_staffs_in_list_and_appending = @other_staffs - @staffs_
@other_staffs_can_be_listed =@other_staffs_in_list_and_appendi
```

As in the code snippet above, “@staffs_in_campaign_but_not_mine” and “@other_staffs_in_list_and_appending” are two temporary variables that have logic names rather than “@temp1” and “@temp2”.

Another frequent used one is “Rename Method”.

```
def remove_appending_staff
    @campaign = Campaign.find(params[:id])
    Note.find_by_id(params[:note_id]).destroy
    redirect_to :action => 'list_other_staffs', :id => @campaign
end

def request_other_staffs
    @campaign = Campaign.find(params[:campaign][:id])
    @target_staffs = Staff.find(@params[:available_staff_ids]) if @params[:av
```

As in the code snippet above, two methods are named exactly and clearly according to their behaviors after doing a code review.

6. The source code and a readme.txt file

See ***readme.txt***

7. A comparison of experience to develop the system with the requirement-design-implementation process with the XP approach

First and foremost, I and my pair Shanbo have found the extreme programming approach is a very excellent and brilliant way to develop software project. The traditional requirement-design-implementation method is too time-consuming, and the old software lifecycle models always demand tons of whatever-kind-of-documents and fixed predefined schedules with little tolerances of changes. I still remember during my bachelor studies, there is a course named “Information System Design Project”, in which we are asked to develop a schedule management system for fitness club following old fashion waterful model, I’d rather call it document-driven approach where the process step is determined by the type of document that has to be submitted, during those days, everyone was worried about the submission of all kinds of documents even more than the project itself, and indeed the it took too much time to product a decent document. Movemore, I and my group members were concerned about individual works, and facing different problems that others couldn’t help.

Whereas, in this project, I and Shanbo Li benefit and learn a lot from pair-programming specially, we write code in one laptop by turns, when I’m writing the program, he is sitting beside me and checking my logical or algorithmic errors from time to time, and in most cases, he is able to point out all mistakes I’ve made. And when either of us comes across a problem, we stop coding and discuss it and illustrate our ideas in the blackboard, it’s amazing to see that how one is constrained in his way of thinking and looking at the problem, whilst only the other one can find

another way to solve the problem to its point. What's more the user story is really a nice practice from extreme programming, we no long to abstract and write the tedious use cases, instead, the use story is customer-oriented, it hits the point of the customer's need with the corresponding importance and risk factors, whereas the realizing of the use story is totaling in hand of the develop, the develops are no afraid of changes, and any changes can be immediately be reflected in the use stories.

Bonus Tasks 1. Write acceptance tests for two of the user stories.

See ***iad_acceptance_test.pdf***

Bonus Tasks 2. Which more XP elements, except those used in the project, would like to try in software development?

First of all, except those XP elements that are required to use in the project, we have adopted two more XP practices, continuous integration and collective code ownship.

I and my teammate Shanbo are always working on the latest version of the software, though we have versions saved locally with various changes and improvements when we're coding separately at home rather than together in the school, we all the time upload our current version to the code repository from Google codebase. By this way, we are not only able to roll back to or trace the older implementations, but also avoid delays later on in the project cycle, caused by integration problems.

And both of us is responsible for all the code, that is, every one of us is allowed to change any part of the code, it's contributed by pair programming, since by working in pairs, we get to see all the parts of the code. Collective code ownship do speed up the development processes, because if an error merges in the code, both of us are free to fix it, plus I can improve or refractor what my teammate Shanbo has written, and the other way around.

Still we would like to try more XP practices in further project, if any and possible. For example, the concrete of whole team, so we want to have a real customer and work for him, he is not the one who pays the bill, but the one who really uses the system, in short, we need a customer who is on hand at all times and available for questions, in which way we can really understand the need of customer from the aspect of a customer.

Yet, another crucial important one is the coding standard, which is an agreed upon set of rules that the entire team agreed to agree to adhere to throughout the project. When I was having my practical training in Wapice, I was asked to learn and follow the internal coding standard in the company before I can start the work, though I felt uncomfortable at the very beginning, since it differs a lot from my own way of naming, but it helps a lot when reading source codes from others.