CS 435 - Homework 3

Deadline

Due by 11:59 pm on Thursday, February 13, 2014

How to submit

Submit your files for this homework using `~st10/435submit` on nrs-labs, with a homework number of `3`

Purpose

To lightly think a bit more about the Timeboxing model, and to think more about Extreme Programming practices.

Important notes

Note that your submissions for this assignment may be posted to the course Moodle site.

In a file `435hw3.txt`, put your name, and your answers for the following problems.

Problem 1

Consider the Timeboxing model.

Assume a project decides on a time box of 12 weeks, with three stages (Requirements, Build, and Deploy). So, Requirements has an iteration for 4 weeks, then passes it on to Build for 4 weeks, which passes it on to Deploy for 4 weeks. (But, as discussed, after Requirements passes iteration 1 on to Build, Requirements works on iteration 2 for 4 weeks while Build is working on iteration 1. And after 4 weeks, Build passes iteration 1 to Deploy, while Requirements passes iteration 2 on to Build, and then Requirements starts on iteration 3. And so on...)

Assume in EACH of the following questions that "time 0", the start time, is when Requirements starts work on iteration 1 -- also assume that all goes well, and each stage completes its work on-schedule. (That is, consider these to be "best-case scenario" questions.)

1 part a

After how many weeks will the customer receive the completed first iteration?

1 part b

After how many weeks (from "time 0") will the customer receive the completed second iteration?

1 part c

After how many weeks (from "time 0") will the customer receive the completed third iteration?

1 part d

Assume there are 8 iterations. After how many weeks (from "time 0") will the customer receive the completed eighth iteration?

1 part e

NOW assume that instead of 3 teams of developers -- a Requirements team, a Build team, and a Deploy team -- you only have 1 team, who handle requirements, then build, then deployment, in sequence. Assume this single team is to develop the "same" 8 iterations/versions, and also spend 4 weeks on the requirements, 4 weeks on the build, and 4 weeks on the deployment, for each iteration. (This could be perhaps/arguably be viewed as an unusually-consistent iterative-delivery approach?)

Assume that "time 0", the start time, is when the team starts the requirements for iteration 1. After how many weeks will the customer receive the completed first iteration?

After how many weeks will the customer receive the completed eighth iteration?

(**NOT** TO ANSWER, but to **THINK** about:

- Note that, comparing the two scenarios, the time-boxing approach takes three times as many developers to achieve the reduced delivery time (the way these imaginary scenarios happened to be set up).
- Consider also: if the customer, using the first iteration, discovers that what they "really" want is different, which scenario can more gracefully handle such changes in requirements?

)

Problem 2

Consider the following list of Extreme Programming practices:

- Pair programming
- The planning game/release planning
- Unit testing/test-driven development/test-first development (assume this includes the concept of writing tests BEFORE writing the code to be tested, and assume that, once written, code cannot be checked-in to the current project version unless it passes its unit tests)
- Acceptance testing/customer testing
- Frequent releases/Small releases
- Refactoring
- Simple design/YAGNI (You Aren't Going to Need It)/"do the simplest thing that could possibly work"

- Collective code ownership
- Continuous integration
- On-site customer/"extreme" customer involvement
- 40-hour week/sustainable pace
- Coding standards
- System metaphor/common vocabulary
- User stories
- Stand-up meetings
- CRC cards - Class, Responsibilities, and Collaboration - for object-oriented programming
- Spike solutions

2 part a

You hopefully have noticed that some of these practices influence, or even depend, on some of the other practices.

Consider the practice of "Collective code ownership". List at least three other practices from the list above that make "Collective code ownership" more practical/feasible.

2 part b

CRC - Class, Responsibilities, and Collaboration - cards are used for brainstorming, for design, and for communication. They do assume an object-oriented approach. You'll find a number of templates on the web, but this is one simple version, based on the description and examples at:

http://www.agilemodeling.com/artifacts/crcModel.htm

An index card has the name of a class on the top of the card. On the left-hand side of the card, you list that class's responsibilities -- something that class knows or does. On the right-hand side of the card, you list collaborator(s) -- other classes that this class interacts with to fulfill its responsibilities.

As an example from the above web page, you might have a card with **Customer** at the top, with responsibilities listed on the left-hand-side of:

- places orders
- knows name
- knows address
- knows customer number
- knows order history

On the right-hand-side, it might have a collaborator-class listed of:

- Order

It is considered a benefit that index cards are small, and easy to move around on a desk (or perhaps a computer desktop) -- you can put cards for classes that collaborate with each other near each other, and cards that don't further apart. You can also easily throw away one or more cards and start over if one approach isn't working!

Consider a Monopoly game program. You might have a Player class in such a game.

Give an example of at least one responsibility that a Player class might have.

Give an example of at least one collaborator-class that a Player class might have to interact with to fulfill its responsibilities.

2 part c

Consider the listed practices again.

Assume for this question that you have a set-up with all the tools/support you'd ideally want for these practices (a lovely testing framework that automatically runs your tests each time you compile, a code repository that won't accept code unless its unit tests pass, an on-site customer/very involved customer, a good room for a stand-up meeting, a reasonable work-space that people can pair-program in, and so on).

IF you were in this kind of situation, which 3-4 practices would you be most interested in "trying out"?

IF you were in this kind of situation, which 3-4 practices would you still be most leery of/concerned about/skeptical of? For each of these, include at least one reason why you are leery of/concerned about/skeptical of it.

IF you were in this kind of situation, are there any practices which you just don't know enough about yet to know? If not, say so -- if so, list those practices. (The in-class discussion definitely left some "holes" for some of these -- this question is a way to see which practices the reading plus the class discussion have not yet fleshed-out sufficiently.)

2 part d

Now consider the setting of this class, and consider all the listed practices EXCEPT those you listed in 2 part c as not knowing enough about yet to know if you'd want to use them or not in an "ideal" setting.

Which of the remaining practices, if any, seem to you like they might be suitable for use in the CS 435 team project?

Which of the remaining practices, if any, seem to you like they probably would not be suitable for use in the CS 435 team project?