CIS 350 Homework 1

Alex Marple

January 26, 2012

Question 1

One risk that is fairly likely to occur during a software project is a changing timeline for release. For example, eCacti may be developing a website for buying and selling cacti might witness the unexpected launch of a website from a competitor, CactiWeb. Since buyers and sellers of cacti are averse to change, they are unlikely to switch their choice of e-cacti-commerce once they have adopted a particular website. It is imperative for eCacti to release their website much sooner than anticipated to attract users before CactiWeb controls the world online cacti trade.

The agile principle of delivering working software frequently would mitigate the risk of an unexpected launch be CactiWeb. Agile developers at eCacti would have a working subset of the fully-featured website available before the end of development. It would be possible to release this to compete with CactiWeb while continuing to develop and add the other features until the website supports all of the features originally intended.

Question 2

In an agile software process model, a **user story** is a description of a feature that the customer would like to have included in the project. A user story should be succinct (it should fit on an index card, and be general enough to spark a conversation between the customer and the developers) and should describe a specific feature, and it should include an indication of the value of the feature to the customer. A development team will assign to a user story an estimated development time for the feature, and then use that estimate and the priority of the customer to determine which features to implement next.

Question 3

Agile process models are likely to reduce the unnecessary cost of software development projects. In the most general sense, agile processes came about as a response to the "software crisis" and aim to reduce unnecessary costs during the development process. Specifically, agile processes function to avoid project restarts (which, according to the 1995 Standish Group report, were almost as frequent as project starts themselves). Restarts are costly; they incur the cost of extended development time, the cost of pushing back the delivery date (and any lost opportunity cost associated with the delay), and possibly the cost of abandonning a project completely (an event that the Standish Group reported befalling 31.1% of projects).

The prevalence of restarts seems linked to the waterfall-style process model. The strict sequencing of steps imposed by waterfall pegs a restart as the only option for a project presented with a drastic change in requirements. The agile process model, however, is designed to avoid restarts wherever reasonable, giving preference to working, evolving, maleable software over absolute efficiency from design. In addition, agile's emphasis on possessing working software will often leave a challenged but still functional product should a project need to be cut short. Since a functional product with half the original scope is worth more as-is than an unusable half of a fully-scoped project, agile decreases the cost of cancellations.

Question 4

A Software Configuration Management repository should provide, among others, the three following features:

Version Control

A SCM repository should provide developers with a way to store multiple different versions of a project. Developers should be able to examine current as well as past builds, roll back to previous versions, and split and merge parallel development branches.

Auditing

A SCM repository should provide the developer with a way to track changes to the project in detail. Developers should be able to know who made chages, when they made them, and why. This facilitates face-to-face communication, one of the agile principles

Dependency Tracking

A SCM repository should track the dependencies between components of a project. When components are dependent on one another, small changes in one location can ripple across related components and have effects totally unforseeable by a human with a limited understanding of the project as a whole. Dependency tracking alerts developers of possible places for these ripples and gives developers a road map to track down bugs when they do occur.

Question 5

Mutt and Jeff might experience these potential problems:

Compiler Errors

While Mutt and Jeff are working on separate classes, there must be some interdependencies between the classes (otherwise they would have two separate projects). If Mutt changes the signature of a method that is called by one of Jeff's classes, the compiler will generate an error when they attempt to build the project.

Contract Errors

Similarly, Jeff may change the functionality of a method called by one of Mutt's classes. The signature of the method is the same, but the type of value returned is not what Mutt expects. For example, a method that previously returned "yes" or "no" may now return only "maybe" or "maybe not". If Mutt depends on this method, then this change will not cause an issue at compile-time but will cripple the functionality of the project.

Question 6

Ater working separately, Siddharth and Nancy could experience a few problems during integration:

Mismatched Interfaces and Environments

Similar to the problem Mutt and Jeff experienced, Siddharth and Nancy might attempt to integrate only to find that the interfaces or environments they had been using were not the same. Either Nancy or Siddharth could have made a change to an interface that might have helped had it been communicated quickly but causes problems at integration.

Unsatisfied requirements

If Nancy and Siddharth both assumed that a particular piece of functionality was the responsibility of the other developer, then there may be features that were not implemented.

Reproduced Functionality

Similarly, if Nancy and Siddharth both assume that they are responsible for the same piece of functionality, then work will be duplicated. In addition to being inefficient, this reproduction could lead to small discrepancies between similar functions written by each developer, which might cause bugs if/when the duplication is removed.

Question 7

Although it may seem as though Sam's commit "broke" the code Sally is working on, it is actually Sally's responsibility to fix the code. Sam checked out the most recent version of the project and checked in a method that still produced a working build. Sally's change had never been added to a working build, and so she is now responsible for (even if frustrated by) reconciling her changes with Sam's before checking in with another working build.

Question 8

Technical debt is the accruing cost of less-than-ideal coding practices introduced to a project because of restrictions on the time or scope of a project. These less-than-ideal coding practices are permitted because it allows a project to ship faster, but will cost development time later in development as they are replaced by better code. While some technical debt is permissible, having technical debt accrue unchecked becomes a problem, as issues pile on top of one another and increase the amount of development resources that must be spent paying off technical debt.

I acquired this knowledge of technical debt from an article by Martin Fowler located at
`http://martinfowler.com/bliki/TechnicalDebt.html`
and an article by Jeff Atwood located at
`http://www.codinghorror.com/blog/2009/02/paying-down-your-technical-debt.html`.

Question 9

There is no absolute metric by which software quality is measured. Until the Torvalds is established as the SI unit for software quality, programs will be measured according to several different metrics related in one way or another to software quality. The two metrics I will consider are speed and stability.

Speed

Given two programs that are identical in all other factors, a faster program is generally prefered to a slower one. A good measure for speed is the amount of real time that elapses between a user's request for a service and the delivery of the service.

Stability

Users generally prefer a program that requires as little maintenance as possible. By measuring the frequency with which a program needs to be patched or refactored, we can achieve a decent measure of a project's stability.

I acquired this knowledge of metrics for software quality from an article by Olivier Coudert located at `http://www.ocoudert.com/blog/2011/04/09/what-is-software-quality/`.

Question 10

My repository is located at
`https://github.com/spyrogira08/cis350_hw1`.