CS 432 Midterm

Ryan Dockstader

# Waterfall: Project Size

One of the primary goals of the waterfall methodology is to manage large software projects. In the first sentence of the original article, Royce states: “I’m going to describe my personal views about managing large software developments.” [1] Specifically, he is attempting to:

Improve the ability to develop large software systems

## Essential

Problems with a lot of moving parts are part of being a human. When you start thinking of all the moving parts of a system, it becomes very difficult, as each part causes the problem to grow exponentially. Even with modern practices of small functions, micro-servers, etc. if you have a large system, you’re going to have issues breaking it down. Thus, Royce attempted to fix this with detailed documentation that could easily be referred to with any questions the developer/user/other might have. Thus, the large software problem is essential

# Rapid Prototype: AVoiding risks

Rapid prototying was developed to solve several different issues. But the first one listed in Carey’s article is “Users seldom have clear, concise understanding of their informational needs. Therefore, they cannot prespecify the requirements. Once they begin to use a system, however, it is clear to them where the problems lie.” [3] Specifically, the complexity is:

People don’t know what is wrong/needed until they have something to actually look at.

## Essential

Users not knowing exactly what they want, and exactly where things should go has spawned off an entire area of study in computing, Human Factors. This can range from the fact that humans expect some type of feedback from a system when operating it, to them looking in a specific menu for a specific item. The fact of the matter is though, they do not exactly know which menu the item belongs in until they go looking for it there. And thus, the menu already must exist. Because of this, and the way humans are, this is an essential complexity of software that is built to work with humans. I do not know if this is true with other life forms, however. 😊

# Spiral Model: RIsk-driven approach

The primary item that the spiral method was designed for was to create a more risk driven approach, instead of a document or code driven approach. This model was designed to instead of building a monolithic application in monolithic steps, to break them down into smaller, individual pieces. In his article, Boehm states “The major distinguishing feature of the spiral model is that it creates a risk-driven approach to the software process rather than a primarily document-driven or code-driven process.” [2] So, he is attempting to:

Create a risk-driven approach.

## Essential

When you wake up in the morning, you might climb out of bed to discover you’ve stepped on a spider, or perhaps a Lego. In the case of the spider, it probably had moved there while you were sleeping. In the case of the Lego, perhaps you had forgotten it was there. In either scenario you took the risk of stepping on an object by getting out of bed. Similar to that, software development is going to come with risks. This is an essential complexity of life, just as much as it is an essential complexity of software development.

# Agile – XP: changes over time

The primary goal of XP is be more flexible. Software development practices that came before XP were quite ridged, and didn’t allow for many (if any) changes. In Juric’s article, it is stated: “Above all, XP claims to offer a flexible schedule of the implementation of the system’s functionality that actively supports changeable business needs.” [4] Specifically, XP is attempting to:

Allow for changes in needs over time.

## Essential

The primary complexity here is that as things change and evolve, their needs are also going to change. This is part of human existence, and is even a biological based theory about this, that I think applies nicely to technology as well. It is called the “adjacent possible” theory. In this theory it states that “Biological systems are able to morph into more complex systems by making incremental, relatively less energy consuming changes in their make up” [5]. Similar to this, in software we are able to make much larger, well built software if we make smaller low energy changes to that software and allow for it to grow, and reach slightly up, instead of shooting directly for the biggest thing out there.

# Agile – Scrum: future uncertainties

Scrum was developed out of the need to progress from a society that revolved around basic factory based manufacturing, to one where innovation is needed constantly to be a part of the work force. Coding, for example, often presents situations where there is many, many ways to solve the exact same problem. It takes some structures and frameworks to know how to think as part of a group. Scrum is similar to Agile, in that it is also trying to solve for uncertainty. [6] Thus, Srum was designed to:

Continue progress in the face of uncertainty of the future.

## Essential

While it’s a different approach, this is the same issue that Agile is trying to solve. I feel that I explained it well under the agile heading, and would refer you to that. For those same reasons, this complexity is essential.

# Cleanroom: software defects

Cleanroom was developed to reduce bugs that are written by humans. Specifically it’s stated in Mills article, “Software quality can be engineered under statistical quality control and delivered with better quality. The Cleanroom process gives management an engineering approach to release reliable products.” [7]. It’s primary purpose is:

Prevent defects in the software

## Accidental

While I think there is a pretty clear argument here for human error to be essential, I think that this one is trying to solve an accidental complexity by adding additional complexity. By their nature, bugs written into software are put there accidentally, which is why they can be solved. You can’t patch things like the human brain not being able to understand all situations. But, that isn’t what this is trying to solve for. This is trying to solve for releasing bugs. Which, again by their very nature, are accidents.

# Formal Methods: software tooling

There are quite a few different type of formal methods, but they all try to solve the same issue. And that issues is that software being written by humans is likely to have bugs, because each human thinks differently. However, since this is an overarching item that leans more towards language development, what I find that it is trying to solve is:

A software developer can only work as fast and efficient as his/her tools allow.

## Essential

To develop software, one must use something. Unfortunately, you cannot just develop software in your brain, and run it in your brain. There must be some type of formal methodology of writing software. Even down to the 1’s and 0’s you must send the right current down the right path to set it to a 1 or a 0. Because of this, you must have some type of tooling to have some type of software, and thus this is solving for an essential complexity.

# cmm

I think that the first sentence of the article by Paul and associates sums up the problem that CMM is solving for quite nicely. It states: “After two decades of unfulfilled promises about productivity and quality gains from applying new software methodologies and technologies, developers are realizing that their fundamental problem is their inability to manage the software process” [9] From that we can gleam that it is:

Attempting to solve issues in managing software processes

## Essential

If you look at any project, you can clearly see that there is some type of management involved. For example, let’s take something simple like a homework assignment. Without proper management, a student could miss several things involved with the homework assignment. Things such as deadlines, requirements, any necessary formatting or citation requirements, how to submit the assignment, etc. The instructor also must follow guidelines that have been setup to manage the process. How to grade the assignment, where to find the assignments that have been turned in, what happens if an assignment isn’t turned in or it is turned in late, etc. For something as simple as a student/teacher homework assignment interaction there is quite a bit of management around it. Without which the action isn’t possible. The same is true for software engineering. Without proper management, the software engineering task is impossible to complete. Thus, the complexity is essential.

# Rubric

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Exceptional 100% | Good 90% | Acceptable 70% | Developing 50% | Missing 0% |
| Accuracy 60% | The problem all 8 methodologies were meant to address was accurately identified, clearly described, and well supported with direct quotes | A problem the methodology was accurate identified in all 8 cases | The problem was not clearly described, the description was vague, or the citation was missing for at least one methodology | The problem was not clearly described, the description was vage, or the citation was missing for two or more methodologies. | More then one major error exists |
| Clarity of reasoning 30% | Each Accidental / Essential argument was convincingly made | There is no flaws int eh reasoning for any of the 8 accidental / essential arguments | The reasoning for one model was incomplete, vague, or otherwise flawed | A single large flaw exists in the reasoning or presentation of facts for any of the models | More than one large flaw exists |
| Writing 10% | The paper is a joy to read and easy to grade | The paper is professionally written, lacking any mechanical or formatting errors | One aspect of the paper is difficult to read or there exists a single mechanical error | Flaws int eh paper make it difficult to read or grade | The paper can be read only with extreme difficulty |

# Citations

1. W. Royce, "Managing the Development of Large Software Systems," Proceedings of the IEEE WESCON, pp. 328-338, Aug. 1970 [Online] Available: <http://dl.acm.org/citation.cfm?id=41801>
2. B. Boehm, "A Spiral Model of Software Development and Enhancement," IEEE, xxx, pp. 61–72, May 1988, [Online] Available: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=59&tag=1>
3. J. Carey, "Prototyping: Alternative Systems Development Methodology," Information and Software Technology, vol. 32, no. 2, pp. 119–126, Mar. 1990, [Online] Available: <http://www.sciencedirect.com/science/article/pii/0950584990901114>
4. R. Juric, "Extreme Programming and its Development Practices" 22nd Int. Conf. Information Technology ITI 2000, Jun. 2000 [Online] Available: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=915842&tag=1>
5. Kumar Srivastava, ClearStory Data. “The 'Adjacent Possible' of Big Data: What Evolution Teaches About Insights Generation.” Wired, Conde Nast, 7 Aug. 2015, [www.wired.com/insights/2014/12/the-adjacent-possible-of-big-data/](http://www.wired.com/insights/2014/12/the-adjacent-possible-of-big-data/).
6. M. James, "Introduction to Scrum", Scrum Training series [Online] Available: <http://scrumtrainingseries.com/Intro_to_Scrum/index.html>
7. H. Mills et al., "Cleanroom Software Engineering," IEEE Software, vol. 4, no. 5, pp. 19–25, Sep. 1987, [Online] Available: <http://dx.doi.org/10.1109/MS.1987.231413> .
8. R. Vienneau, "A Review of Formal Methods", pp. 7–14, May 1993 [Online] Available: <https://www.csiac.org/sites/default/files/A%20Review%20of%20Formal%20Methods.pdf>
9. M. Paulk et al., "Capability Mature Model for Software, Version 1.1," Chapter 1-3 IEEE Software, vol. 10, no. 4, pp. 18-27, Jul. 1993. [Online] Available: <http://dx.doi.org.byui.idm.oclc.org/10.1109/52.219617>