

Midterm Individual Assessment

Michael Castlema  Team Moo, mlc67@columbia.edu

COMS W3156, Prof. Kaiser, 9 March 2001

Contents

1 Introduction	2
2 Process Improvement	2
2.1 Documents	2
2.2 Rigidity of Course Structure	3
2.3 Group Dynamics	4
3 Course Improvement	4
3.1 Groups of Novices	5
3.2 Teaching Assistant and Integration Manager	5

1 Introduction

It has now been approximately seven weeks since the beginning of COMS W3156, Introduction to Software Engineering. One might reasonably expect, then, that the students—especially those whose teams received exceptionally high grades on all the documents thus far returned—have learned quite a bit about efficient software development process. One would be incorrect.

Instead, as we shall see in this paper, the class has focused instead on the creation of seemingly endless documents with little relation to any conceivable final software product. While the irony of accepting instruction from Microsoft[4] on how to write bug-free software should be obvious to anyone who has ever tried to browse the Web with IE on Windows 95, other sources of information cited in the course are no less worrisome. For example, the 26 February lecture on Project Planning[3] relies almost entirely on a study commissioned by the Defense Department, which claimed that the software process had reached the “Optimizing” level (a sort of software development Nirvana, if we are to believe the study) initially only at one wing of Lockheed-Martin and subsequently only at a few more large corporations. For an non-Capitalist pacifist such as myself, this evidence is somewhat less than fully convincing. Instead, the class ought to look at such non-traditional models of software development as suggested by, for example, Eben Moglen in [5] and implemented at the New York City Independent Media Center at [6].

2 Process Improvement

There are many fundamental problems with the software development process as taught in the class, some of which are discussed in this section.

2.1 Documents

The sheer quantity of documentation that must be produced for the Software Engineering class is immense. This may be, to many students, the first sign that the proposed software development process is not as efficient as its dogma claims.

Indeed, the design time in this class is even greater than the supposedly optimal time found by the Software Engineering Institute. While that study found that design time ought to be 50% of coding time[3], this class allocates approximately 250% of coding time for design. Especially for such a small project as the ones being undertaken in this class, this is clearly overkill.

Furthermore, it was often difficult to complete even the portions of the documents that might have been useful. For example, one of the primary admittedly useful characteristics of the documents (in this case, the Components Requirements Document, or CRD) might be a list of requirements: a prescriptive list of exactly what the project is to do when completed. However, as discussed in Section 3.2, it was often difficult to obtain the information we needed to create a meaningful list of requirements.

2.2 Rigidity of Course Structure

A further problem with the software development methodology as taught in the class is the inflexibility dictated by the rigid grading requirements. A team which feels, for example, that it has no use for a Component Deployment View (§3.3 of the Component Architecture & Design Document) is required to create one anyway, or risk losing points on its grade for the document. This further shifts groups' emphasis away from creating useful documents that will be useful in project coding toward documents that are created merely to ensure a good grade in the class.

A related problem is the fixed sequence of assignments and due dates. According to the popular “spiral” model of software development[3], which is often considered the epitome of the Capitalist software development model, the software development team ought to loop around a repetitive process, incrementally improving and evaluating the product at each cycle. It would be impossible to follow this process and simultaneously produce the all the required documents at the required times. Instead, the class follows a form of the “waterfall” model, which makes any kind of feedback or reevaluation of needs difficult, if not impossible.

2.3 Group Dynamics

One of the important skills supposedly to be learned in the course is the ability to work in teams towards a mutual goal. Unfortunately, the methods suggested for working together are often poor choices. The team structure suggested by the Team Formation page is representative of the pseudo-Capitalist, hierarchical biases of its designer. Instead of having a designated “team leader,” teams might be better served by the consensus model of decision making described at [1].


Our team took a somewhat middle-of-the-road approach, informailzing the suggested power hierarchy yet still maintaining parts of it. This worked well enough for a small team of six people, but I do not think it would be able to scale well to a larger team size. Indeed, without the spokes-council model suggested by the consensus model description, there have been difficulties communicating across the “superteam.” I expect these difficulties to increase as integration becomes more key in the later stages of the project.

Another difficulty was that of scheduling meetings. Because all of the team members have other classes and further time commitments, it could, at times, become difficult for us to meet. Indeed, one week had so many time-pessures for all group members that we were forced to meet for eighteen consecutive hours to finish one of the documents on time. Clearly, this is not a sign of a healthy software development process!


3 Course Improvement


While the main problem with the course, its adherence to a fundamentally flawed and pro-Capitalist software development methodology designed for the development of war toys, as detailed in Section 2, there are some additional details, specific only to the course as taught this semester, discussed here.

3.1 Groups of Novices

One problem with the course as taught is the inevitable confusion that results when a group of novices is required to use a process none is familiar with. In the “real world,” this might be less of a problem as experienced members of a team are able to mentor newer members as to the proper way to complete certain tasks. A partial (although far from sufficient) remedy for this problem ld be to take special care in the future to make the assignments extremely clear.

3.2 Teaching Assistant and Integration Manager

There was an emphasis placed, especially at the beginning of the course, on meeting with the “customer” to determine precisely the requirements for the component. Unfortunately, it was very difficult for us to schedule a meeting with our TA, Peter Davis. Indeed, we were unable to meet with him at all until Tuesday, 20 February, after the Operational Concept Document and Integration Concept Document had already been completed and submitted. It is clearly not entirely the fault of Mr. Davis that he was unable to meet with all nine of his teams on a more frequent basis. It might be reasonable to suggest that, in the future, no TA should be responsible for more than one “superteam.” It should also be possible to obtain information from alternate sources. 

We did not learn the name of our Integration Manager, Erik Aigner, until Friday, 23 February. This clearly made it difficult to consult with him on the early documents, and it would be premature of me to meaningfully comment further on him when we have known him for so short a time. 

References

- [1] ACT UP New York. “Consensus Decision Making.” *Civil Disobedience Training*.
<http://www.actupny.org/documents/CDdocuments/Consensus.html>

- [2] Brooks, Frederick P., Jr. *The Mythical Man-Month Anniversary Edition*. Boston: Addison Wesley Longman, 1995.
- [3] Kaiser, Gail. "Lectures." Delivered in class, and available online at *COMS W3156 Home Page*. Spring 2001. <http://www.cs.columbia.edu/~kaiser/cs3156/lectures.html>
- [4] McConnell, Steve. *Software Project Survival Guide*. Redmond, WA: Microsoft Press, 1998.
- [5] Moglen, Eben. "Anarchism Triumphant." *First Monday*. August 1999. http://emoglen.law.columbia.edu/my_pubs/anarchism.html
- [6] *New York City IndyMedia Center*. <http://nyc.indymedia.org/>