SCRUM AS PROJECT MANAGEMENT IN A SOFTWARE ENGINEERING CLASSROOM SETTING

by

JOSHUA DANIEL WESTMORELAND

Graduate Committee:

Dr. Will Lloyd (*primary advisor*)

Dr. Lewis Baumstark (*member)*

Dr. Duane Yoder (*member*)

Submitted in fulfillment

of the Project option of the

Degree of

Master of Science

in

Applied Computer Science

University of West Georgia

Department of Computer Science

Carrollton, GA

Fall Semester, 2009

Table of Contents

Introduction

The purpose of this project has been to assess whether or not the Scrum process, an "iterative incremental framework for managing complex work"[1] in software engineering, can be used successfully in a classroom setting. Due to the nature of this being applied in a classroom this is as much of a computer science education project as it is a software engineering project. This project was initially conceived of as a topic candidate for a thesis, but circumstances necessitated the conversion to a project. Several topics are covered in this paper:

- Scrum
- The prospectus
- Divergence from the prospectus
- Data from the course

A number of conclusions can be drawn from this project and those will be covered in the final significant section of this paper.

Scrum is the agile project management methodology that was chosen for the implementation of this project. As previously mentioned Scrum is an "iterative incremental framework for managing complex work"[2] and this lends itself greatly to being used in a classroom setting. An agile methodology is any software development "based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams"[3]. Scrum is agile methodology at its best and has been gaining increased acceptance in the workplace and thus the teaching of this methodology in the classroom has many positive effects on the students in the course. Scrum and agile will be covered in more detail in Section II.

The original prospectus for this project, a thesis proposal at the time, was delivered to Dr. Will Lloyd in March of 2009 with the expectation of beginning in the Fall Semester of 2009 and ending in the Spring Semester of 2010, but as previous mentioned the thesis had to be converted into a project in order to be completed in the space of one semester. The original prospectus contains, in most cases, detailed descriptions of how the process was to be managed, but left much up to the instructor as to how the process would actually be implemented in class. The divergences that were made from the original prospectus are discussed in greater detail in Section IV. The original prospectus and other details are available in Section III.

Circumstances necessitated that the Scrum process be modified a bit for effective use in a classroom setting, this some divergence from the original prospectus was required. The nature of

---

[1] Various Anonymous Authors. 2009. (Wikipedia Entry for) Scrum (development). http://en.wikipedia.org/wiki/Scrum_%28development%29 (Accessed 11/12/2009).

[2] Ibid

[3] Various Anonymous Authors. 2009. (Wikipedia Entry for) Agile software development. http://en.wikipedia.org/wiki/Agile_software_development (Accessed 11/15/2009).

the classroom does not exactly lend itself to Scrum in the strictest sense, so a number of modifications to the process had to be made in order to make the learning experience more significant for the students taking the course. Details on the divergence from the original prospectus will be iterated in Section IV.

A significant amount of data was gathered from the course which ran from 08/13/2009 until 12/12/2009. The data that was garnered includes:

- Intended effects
- Unintended effects
- Adverse effect
- Propitious effects.

The resources[4] developed for this course have also been included in this section. This set of data will be expounded upon in detail in Section V.

The results of this project have met with mixed success, see the section on course data for more details, but a great deal has been learned from this experiment. This project offered an excellent opportunity to explore the practical application of an interesting software engineering concept in the classroom and hopefully it was a significant experience for all involved. These conclusions and other data have been included in Section VI.

As just mentioned, this project, as a whole, has been a significant learning experience and all due credit is given to the instructor, Dr. Will Lloyd, for allowing this project to be undertaken in his course and to the students in the course for all of their hard work throughout the course of the semester. Conclusions can be found in Section VII.

[4]This includes lectures, tutorials, etc.

What is Scrum?

Scrum is an agile software development/project management methodology. It seems necessary to define what agile is before getting into the meat of Scrum. Agile software development is defined[5] as:

> Most software development is a chaotic activity, often characterized by the phrase "code and fix". The software is written without much of an underlying plan, and the design of the system is cobbled together from many short term decisions. This actually works pretty well as the system is small, but as the system grows it becomes increasingly difficult to add new features to the system. Furthermore bugs become increasingly prevalent and increasingly difficult to fix. A typical sign of such a system is a long test phase after the system is "feature complete". Such a long test phase plays havoc with schedules as testing and debugging is impossible to schedule.
>
> The original movement to try to change this introduced the notion of methodology. These methodologies impose a disciplined process upon software development with the aim of making software development more predictable and more efficient. They do this by developing a detailed process with a strong emphasis on planning inspired by other engineering disciplines - which is why I like to refer to them as **engineering methodologies** (another widely used term for them is **plan-driven methodologies**).
>
> Engineering methodologies have been around for a long time. They've not been noticeable for being terribly successful. They are even less noted for being popular. The most frequent criticism of these methodologies is that they are bureaucratic. There's so much stuff to do to follow the methodology that the whole pace of development slows down.
>
> **Agile methodologies** developed as a reaction to these methodologies. For many people the appeal of these agile methodologies is their reaction to the bureaucracy of the engineering methodologies. These new methods attempt a useful compromise between no process and too much process, providing just enough process to gain a reasonable payoff.
>
> The result of all of this is that agile methods have some significant changes in emphasis from engineering methods. The most immediate difference is that they are less document-oriented, usually emphasizing a smaller amount of documentation for a given task. In many ways they are rather code-oriented: following a route that says that the key part of documentation is source code.
>
> However I don't think this is the key point about agile methods. Lack of documentation is a symptom of two much deeper differences:

[5] Fowler, Martin. 2005. The New Methodology. From Nothing to Monumental to Agile. http://martinfowler.com/articles/newMethodology.html#FromNothingToMonumentalToAgile (Accessed 11/12/2009)

- *Agile methods are adaptive rather than predictive.* Engineering methods tend to try to plan out a large part of the software process in great detail for a long span of time, this works well until things change. So their nature is to resist change. The agile methods, however, welcome change. They try to be processes that adapt and thrive on change, even to the point of changing themselves.
- *Agile methods are people-oriented rather than process-oriented.* The goal of engineering methods is to define a process that will work well whoever happens to be using it. Agile methods assert that no process will ever make up the skill of the development team, so the role of a process is to support the development team in their work.

There are a multitude of agile methodologies[6] current in practice, but for the purpose of this project a methodology focusing on project management was required and Scrum was a natural choice. It is important to discuss what Scrum is before getting into its implementation in the classroom or any data gathered from that experiment. The following[7] is a succinct description of the Scrum methodology:

> The Scrum methodology of agile software development marks a dramatic departure from waterfall management. In fact, Scrum and other agile processes were inspired by its shortcomings. The Scrum methodology emphasizes communication and collaboration, functioning software, and the flexibility to adapt to emerging business realities — all attributes that suffer in the rigidly ordered waterfall paradigm.
>
> **What's Unique about Scrum?**
>
> Of all the agile methodologies, Scrum is unique because it introduced the idea of "empirical process control." That is, Scrum uses the real-world progress of a project — not a best guess or uninformed forecast — to plan and schedule releases. In Scrum, projects are divided into succinct work cadences, known as sprints, which are typically one week, two weeks, or three weeks in duration. At the end of each sprint, stakeholders and team members meet to assess the progress of a project and plan its next steps. This allows a project's direction to be adjusted or reoriented based on completed work, not speculation or predictions.
>
> Philosophically, this emphasis on an ongoing assessment of completed work is largely responsible for its popularity with managers and developers alike. But what allows the Scrum methodology to really work is a set of roles, responsibilities, and meetings that never change. If Scrum's capacity for adaption and flexibility makes it an appealing option, the stability of its practices give teams something to lean on when development gets chaotic.
>
> **The Roles of Scrum**

[6] Such as XP (Extreme Programming), Scrum, Crystal, Context Driven Testing, Lean Development, all the flavors of Unified Process, etc.

[7] Unknown Author. 2009. Scrum Methodology & Agile Scrum Methodologies. http://scrummethodology.com/ (Accessed 11/12/2009).

Scrum has three fundamental roles: Product Owner, Scrum Master, and team member.

- **Product Owner:** In Scrum, the Product Owner is responsible for communicating the vision of the product to the development team. He or she must also represent the customer's interests through requirements and prioritization. Because the Product Owner has the most authority of the three roles, it's also the role with the most responsibility. In other words, the Product Owner is the single individual who must face the music when a project goes awry.

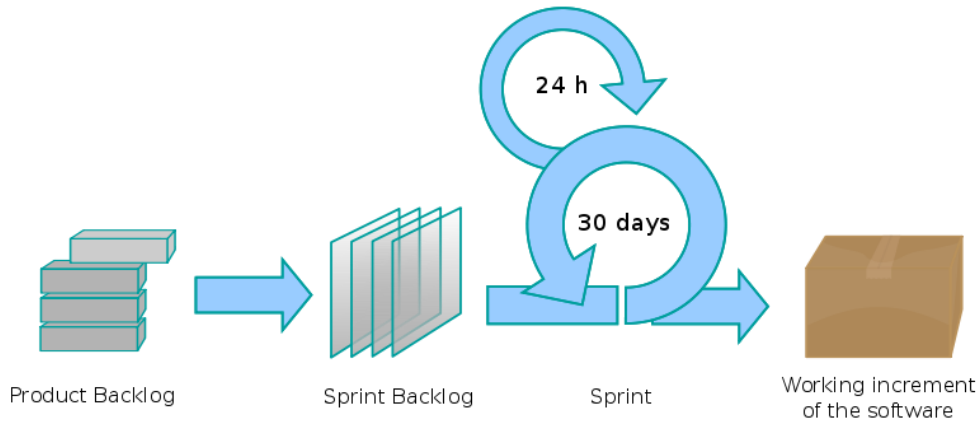
  The tension between authority and responsibility means that it's hard for Product Owners to strike the right balance of involvement. Because Scrum values self-organization among teams, a Product Owner must fight the urge to micro-manage. At the same time, Product Owners must be available to answer questions from the team.

- **Scrum Master:** The Scrum Master acts as a liaison between the Product Owner and the team. The Scrum Master does not manage the team. Instead, he or she works to remove any impediments that are obstructing the team from achieving its sprint goals. In short, this role helps the team remain creative and productive, while making sure its successes are visible to the Product Owner. The Scrum Master also works to advise the Product Owner about how to maximize ROI for the team.

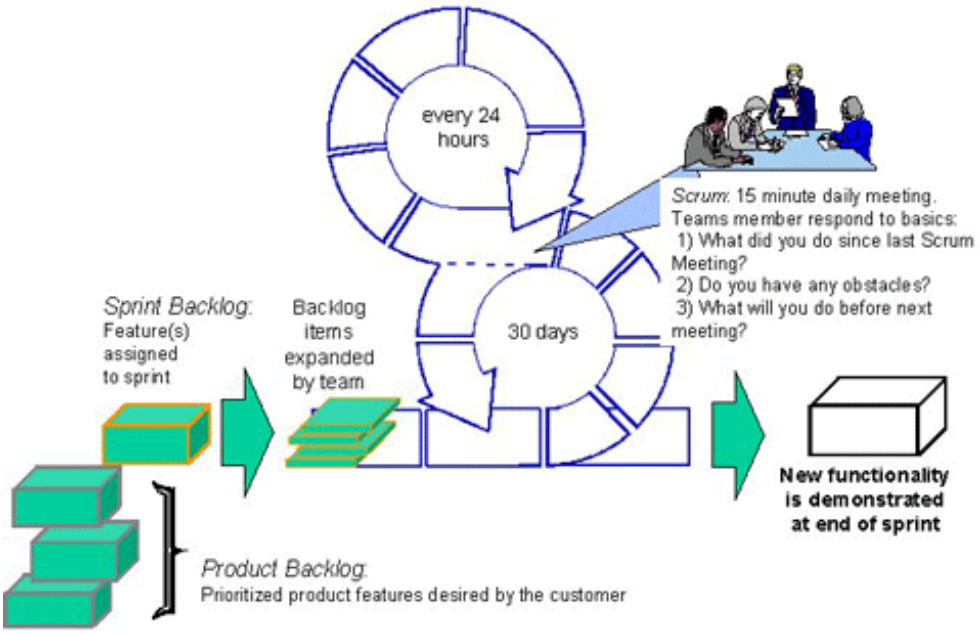
- **Team Member:** In the Scrum methodology, the team is responsible for completing work. Ideally, teams consist of seven cross-functional members, plus or minus two individuals. For software projects, a typical team includes a mix of software engineers, architects, programmers, analysts, QA experts, testers, and UI designers. Each sprint, the team is responsible for determining how it will accomplish the work to be completed. This grants teams a great deal of autonomy, but, similar to the Product Owner's situation, that freedom is accompanied by a responsibility to meet the goals of the sprint.

The following images illustrate the general flow of the Scrum process:

1.) A simple diagram of the process:



2.) A somewhat more complex and informative diagram of the process:



As has just been evidenced in this section agile, and therefore Scrum, has an important place in the classroom. Teaching these processes in a software engineering class is vital and hopefully taking part in a real world project management framework will help to prepare these students for participation in such project management frameworks when they enter the job market upon graduation.

Original Prospectus

This project was originally conceived as a thesis, but circumstances necessitated the change to the project format. The following is the original prospectus[8], in its entirety, submitted to Dr. Will Lloyd in March 2009:

CS 3212 Course Management Proposal

Josh Westmoreland, TA/GRA

03/24/2009

Note: this document assumes that the reader has at least a working knowledge of Scrum

Introduction

My proposal is as follows: We should use a modified form of Scrum to manage CS 3212, Software Engineering II, in the fall semester of 2009. Scrum in its pure form would not really be feasible for a classroom setting, even a software engineering class, but I believe if we modify it a bit then it could be quite useful for this class and for use in future classes. We would have to modify the nature of the scrum meeting, how we use the artifacts (burn down chart, sprint log, etc.), but I believe it can be done and done well. There also some other issues to be covered if this plan was to put in place, but I think those can be successfully dealt with as well.

The Nature of the course

The description of the CS 3212 from the University catalog is as follows:

"Software development methods for large scale systems. Management of software development projects. Software engineering standards. Students are expected to complete a large scale software project."

Scrum naturally lends itself to a course of this nature due to its intrinsic qualities. Scrum is arguable agile software development at its finest. It is tailor-made to handle large scale projects such as the one that students are expected to complete in this course. I believe Scrum to be the perfect method for managing this course.

Using Scrum to manage this course would also require the students to learn the details of agile, which they should already know from CS 3211, Software Engineering I, as well as the details of Scrum itself and other methodologies currently used by professional programmers. This fulfills the requirements of the students learning about

[8] Josh Westmoreland. CS 3212 Course Management Proposal. 03/24/2009.

"Management of software development projects" as well as "Software engineering standards".

How do we modify scrum to suit our needs?

The primary question associated with this proposal is how we modify Scrum to suit the needs of a class. I believe this can be done quite easily. I will break this down in this section.

Assigning Teams

- Quite simply, we randomly, or at least pseudo-randomly, divide up the students into groups of 4 to 5 and let them assign among themselves, or assign for them, the part of the project they are going to work on for the team.

Roles

- Product Owner
    - The Instructor, but this and the Scrum Master role can be shared to some degree by the Instructor and the TA
- Scrum Master
    - The TA, but this and the Product Owner role can be shared to some degree by the Instructor and the TA
- Scrum Team
    - The students, obviously, divided into teams working on the same project concurrently

The Sprint(s)

- Since each development cycle in Scrum, a sprint is usually about 4 - 6 weeks, or 30 - 45 days, depending on the preference of the development house, I suggest we have the students develop the project in 2 - 3 sprints with the length of the sprint to be determined by both the project and the length we decide upon for each sprint. At the end of each sprint we will have a short class-wide sprint retrospective, perhaps lasting as long as one class meeting, where we discuss the sprint, specifically what went well and what didn't as well as any other problems the individual teams might have had during the sprint such as participation, etc.
- To split this up even further, we could have "mini sprints" inside of each sprint lasting about a week each that way we could have something due for them every week as we have done in CS 3211 and this would make the proposed schedule work far better.

Schedule

- Since this a two-day-a-week class I suggest we do the following:
  Monday or Tuesday: Scrum meetings for each team (5 - 7 minutes each)
  Wednesday or Thursday: Lecture & due date for the current iteration/milestone

  Artifacts

- The burn down chart
  - This is something we could have as class-wide so that everyone will know where the project currently stands
- The sprint backlog
  - We wouldn't have to come up with this completely before the sprint. We could work it similarly to how we do now and brainstorm on the lecture days since that's what we do primarily now in relation to the current project.
- The product backlog
  - Once again, this would be a class-wide thing that we could update as we complete requirements. As always with agile, we can brainstorm and add to or remove items from this as we see fit.

  Isn't there supposed to be a scrum meeting every day?

- Yes, but there is a very simple way around this. ..
  - We set up a system and require the students to make a number of scrum reports every week, say at least 3 and as many as 5, on Moodle, or whatever system we decide to use, where they answer the three essential scrum questions

  1 - What did you do today?
  2 - What are going to do tomorrow?
  3 - What impediments are keeping you from completing your objectives?
  - We still have out formal scrum meeting with each team once a week.

  <u>Issues that might arise</u>

  There are at least a couple of things the students will need to know before this will become feasible:

- Subversion, and keeping up with the latest version of the project
- Assigning roles within a team, i.e. dividing up the work fairly

  I think if the students can be taught how to do these things, there might be other issues I am not considering at the moment, I believe this course can be properly managed by the modified form of Scrum I am suggesting. Managing this course in this way would fulfill all of the requirements for the course, at least those I am privy to, and I believe our students would learn a great deal in the process.

  <u>Conclusion</u>

> I of course understand that conducting a class in this manner will require far more planning than I have done here. The purpose of this document is to merely be a rough outline of how to proceed.

The preceding prospectus detailed the application of a stricter version of Scrum than was actually implemented in the course, but as detailed in the next section a divergence from the original prospectus was required to make the project more applicable in the classroom.

Divergence from the Original Prospectus

Much of the original prospectus was implemented as it was written, but some divergence was necessary in order to make the Scrum framework more workable, and therefore teachable, in the classroom setting. In this section the original prospectus will be run down, section by section, and the implementation and/or divergence from of each item will be discussed.

Issue (from the prospectus):

Assigning Teams

- Quite simply, we randomly, or at least pseudo-randomly, divide up the students into groups of 4 to 5 and let them assign among themselves, or assign for them, the part of the project they are going to work on for the team.

Solution:

Teams were assigned based on the students' individual skill sets. Students of similar skill were assigned to groups with each other so as to provide an "equal playing field" for all involved. Anyone who has ever taught or has been involved in any sort of group work knows that there is always at least one person in a group who drags the rest down by not doing his or her fair share of the work. As such, students were assigned to groups based on their demonstrated ability in the previous course[9].

Issue (from the prospectus):

Roles

- Product Owner
  - The Instructor, but this and the Scrum Master role can be shared to some degree by the Instructor and the TA
- Scrum Master
  - The TA, but this and the Product Owner role can be shared to some degree by the Instructor and the TA
- Scrum Team
  - The students, obviously, divided into teams working on the same project concurrently

Solution:

[9] CS 3211 - Software Engineering I, Spring 2009, University of West Georgia
Instructor: Dr. Will Lloyd, TA: Josh Westmoreland

The roles of Product Owner and Scrum Master were indeed shared by the instructor and the teaching assistant, but these were far more informal that would have been done in a workplace setting. Since the instructor is the de facto Product Owner and Scrum Master anyway this seemed to be a natural adaptation for the classroom and the sharing of these roles with the teaching assistant mirrored the teaching assistant's normal job as assistant to the instructor. The Scrum Teams were, of course, the students themselves divided up into the groups into which they had been previously assigned by the instructor.

Issue (from the prospectus):

The Sprint(s)

- Since each development cycle in Scrum, a sprint is usually about 4 - 6 weeks, or 30 - 45 days, depending on the preference of the development house, I suggest we have the students develop the project in 2 - 3 sprints with the length of the sprint to be determined by both the project and the length we decide upon for each sprint. At the end of each sprint we will have a short class-wide sprint retrospective, perhaps lasting as long as one class meeting, where we discuss the sprint, specifically what went well and what didn't as well as any other problems the individual teams might have had during the sprint such as participation, etc.
- To split this up even further, we could have "mini sprints" inside of each sprint lasting about a week each that way we could have something due for them every week as we have done in CS 3211 and this would make the proposed schedule work far better.

Schedule

- Since this a two-day-a-week class I suggest we do the following:
  Monday or Tuesday: Scrum meetings for each team (5 - 7 minutes each)
  Wednesday or Thursday: Lecture & due date for the current iteration/milestone

Solution:

It was discovered through research, primarily on the part of the instructor, that, to put it concisely, shorter sprints are better. Instead of sprints that lasted anywhere from 4 - 6 weeks sprints of 1 - 2 weeks at the most were allowed of the students in their Scrum teams. This allowed the instructor and teaching assistant to more closely follow the progression of each team's project than would have been allowed with longer sprints. Also, the length of the semester[10] prohibited longer sprints that would have been more commonplace in a professional working environment.

[10] Roughly 15 weeks

As far as the schedule is concerned it was more less the same as what had been described in the prospectus. Students were instructed to have Scrum meeting once a week and were given an adequate amount of time in class to do so. Also, lectures, where either new material was introduced or older material was reviewed, were typically given one day a week and the other class period was reserved for a work day for the students where they could potentially receive assistance with any problems they might be having with their projects or ask questions of the instructor or teaching assistant.

Issue (from the prospectus):

Artifacts

- The burn down chart
    - This is something we could have as class-wide so that everyone will know where the project currently stands
- The sprint backlog
    - We wouldn't have to come up with this completely before the sprint. We could work it similarly to how we do now and brainstorm on the lecture days since that's what we do primarily now in relation to the current project.
- The product backlog
    - Once again, this would be a class-wide thing that we could update as we complete requirements. As always with agile, we can brainstorm and add to or remove items from this as we see fit.

Solution:

As it turned out, none of these items were class-wide since each group was allowed to work on a different project. Artifacts were group-specific and each group maintained these items, as well as a code repository, on Google Code[11]. Each specific item had an individual wiki devoted to it within each project on Google Code. The groups maintained these regularly and moved things from the product backlog to the spring backlog(s) and vice versa. Some groups created a backlog for each sprint, and thus a better documented project, and maintained these quite well. Some groups did better than others about maintaining all of these artifacts, but this was the general course of action.

Issue (from the prospectus):

Isn't there supposed to be a scrum meeting every day?

[11] http://code.google.com/

- Yes, but there is a very simple way around this. ..
    - We set up a system and require the students to make a number of scrum reports every week, say at least 3 and as many as 5, on Moodle, or whatever system we decide to use, where they answer the three essential scrum questions

  1 - What did you do today?
  2 - What are going to do tomorrow?
  3 - What impediments are keeping you from completing your objectives?
    - We still have out formal scrum meeting with each team once a week.

Solution:

This issue was not dealt with as each group held a weekly Scrum meeting only as doing daily meetings, either online or in person, would have been highly impractical. Each group of students was instructed to stay in close communication with one another so that all potential issues could be dealt with swiftly and efficiently. The *issues* function of Google Code was also used to assign tasks to each member of the group and to track progress. The students seemed to understand the point of the daily meeting and if necessary it seemed like they could have conducted these scrums.

Issue(s) (from the prospectus):

> Issues that might arise
>
> There are at least a couple of things the students will need to know before this will become feasible:
>
> - Subversion, and keeping up with the latest version of the project
> - Assigning roles within a team, i.e. dividing up the work fairly
>
> I think if the students can be taught how to do these things, there might be other issues I am not considering at the moment, I believe this course can be properly managed by the modified form of Scrum I am suggesting. Managing this course in this way would fulfill all of the requirements for the course, at least those I am privy to, and I believe our students would learn a great deal in the process.

Solution(s):

Subversion was taught early on in the course and TortoiseHG, a piece of software implementing Mercurial, was used to affect this necessary principle. Mercurial is a "free, distributed source management tool"[12] that allows for the existence of a local repository, as opposed to the more traditional remote one which Mercurial and TortoiseHG also support.

[12] Unknown Author. Mercurial SCM. http://mercurial.selenic.com/about/ (Accessed 11/15/2009)

There were of course some issues with Mercurial/TortoiseHG as this was the first semester these had been used by this department, but adoption was mostly smooth and the students adapted to the technology after a while. The issue of assigning the teams was dealt with earlier in this section.

Data from the Course

Data from the course will be presented and assessed in this section. Also, items that were not anticipated in the prospectus will be dealt with and reviewed. The following categories of items will be included: intended effects, unintended effects, adverse effects, and propitious effects. A number of these effects do overlap due to the nature of this project. The resources developed for this course will also be covered in some detail.

Intended Effects

1. Students have learned about project management (Scrum) and have participated in a pseudo real word implementation of such a process.
2. Students have learned how to use a project management framework (Scrum) to manage a project that being working on.
3. Students have worked in groups and have hopefully learned how to interact within that particular dynamic. Unfortunately, this cannot be quantitatively measured due to the varied experience of each group of students.
4. Students have worked on and completed a large scale project to an acceptable degree.
5. Students have learned about and successfully participated in version control.

Unintended Effects

1. By allowing the students to select their own projects they may have inadvertently ventured into areas of programming where they have little or no expertise. Examples of this include networked/distributed systems and database interaction. However, by embarking on these projects the students did learn a great deal about these types of programming.
2. By first working up a GUI students gained a great deal of experience with Java GUIs.
3. Due to the "black box" nature of Scrum, teams were allowed to manage their own work and therefore many did not follow good programming practices and their code suffered as a result of this.
4. As with any sort of group work, a number of problems were created by members of certain groups not doing their fair share of work.

Adverse Effects

1. Due to the nature of Scrum, many groups ended up developing GUIs as a first step, instead of as a last step as one would typically do in such a project. This led to many

problems, such as code being in the GUI which should ideally be in a model class somewhere, which had to be solved with refactoring and reformulation.
2. Students encountered problems due to being allowed to select project that required types of programming in which they had no experience. This in and of itself is not a problem, but had all groups been required to do the same project then these skills could have perhaps been taught more uniformly.
3. Due to the "black box" nature of Scrum, teams were allowed to manage their own work and therefore many did not follow good programming practices and their code suffered as a result of this.
4. As with any sort of group work, a number of problems were created by members of certain groups not doing their fair share of the assigned work. This is partly due to the nature of Scrum allowing teams to organize the work themselves.

Propitious Effects

1. Students learned the problems that can be created by developing a GUI first, especially in Java where GUI creation is somewhat less than intuitive.
2. By first working up a GUI students gained a great deal of experience with Java GUIs.
3. Students learned what it is like to work in a group where some members do not do their fair share of the work and were, in most cases, able to adapt and be successful despite this disadvantage. This problem will persist in any group setting, be it in the workplace or in the classroom.

Resources Developed

Throughout the semester a number of resources were developed for this course under the close supervision of the instructor. These resources included a number of lectures and a proper example project was developed in order to teach some new concepts and reinforce other concepts that had already been introduced by the instructor.

The first lecture[13] developed entailed the Scrum process in its pure form and was delivered to the class on 09/14/2009. The second lecture[14] developed was formulated to remind the students of good programming practices, such as logically divided packages, javadoc and inline comments, use of interfaces, and several other practices they should have been following, and to give them examples of the things they should be doing, but may not have been at that point. This lecture was given to the class on 10/19/2009.

[13] See Appendix I for the lecture slides in their entirety
[14] See Appendix II for the lecture slides in their entirety

The other significant resource developed for this course was an example Java project[15] in Eclipse. This project contained logically divided and named packages, proper javadoc comments (with method contracts), inline comments where necessary, a proper adherence to the model – view – controller pattern, interfaces and the implementation of these, and a multitude of other good programming practices were implemented in order to provide an example of what a proper Java project should look like in Eclipse.

[15] See Appendix III for basic structure of the project

Conclusion(s)

Several conclusions can be drawn from this project and these will be covered in this section. The primary question *"Can Scrum be successfully used in a classroom setting as a project management framework?"* has arguably been answered. A number of Scrum-related issues, both positive and negative, arose throughout the class and these were dealt with as they presented themselves. Also, many other minor issues have been addressed as a result of the undertaking of this project. This project has been a great learning experience and much has been learned about how to teach such a class throughout the course of this project.

The question *"Can Scrum be successfully used in a classroom setting as a project management framework?"* was answered throughout the course of this project and the answer is "sort of". Scrum in its strictest form cannot be used due to the all-encompassing nature of the framework and the amount of time it would take for the students to participate in the proper form of Scrum. The modified form of Scrum that was used in its stead did not work as well as anticipated due to the "slackening of the reigns" that had to occur when the course version was modified from the original framework. This was an unfortunate, but necessary evil and was a compromise that had to be made to make the project work at all. However, it is a reasonable assumption that the students have learned what Scrum is and could likely participate in the proper Scrum framework should they be required to do so in a workplace setting.

A number of issues arose throughout the course of the class that can be arguably attributed to the use of Scrum in the course:

- First and foremost among these is the problem of allowing the students to manage their own progress as opposed to have a set of predefined milestones where a working iteration of the project would need to be submitted for grading. This created a bottleneck at the end for some groups that might not have existed had the system of milestones been used. However, the groups who experienced bottleneck problems would have probably experienced similar issues with a milestone system.

- Another issue was the fact that the groups of students were allowed to select their own projects as opposed to have one project on which all groups would be assigned to work. This created the issue of students running into types of programming they had not yet experienced. It is always beneficial to venture into heretofore unknown areas in order to learn, but had a single project for all groups been used these new principles might have been taught in a different way. However, no two teachers instruct a class in the same way, so this is more of a stylistic issue than a problem that needs to be addressed otherwise.

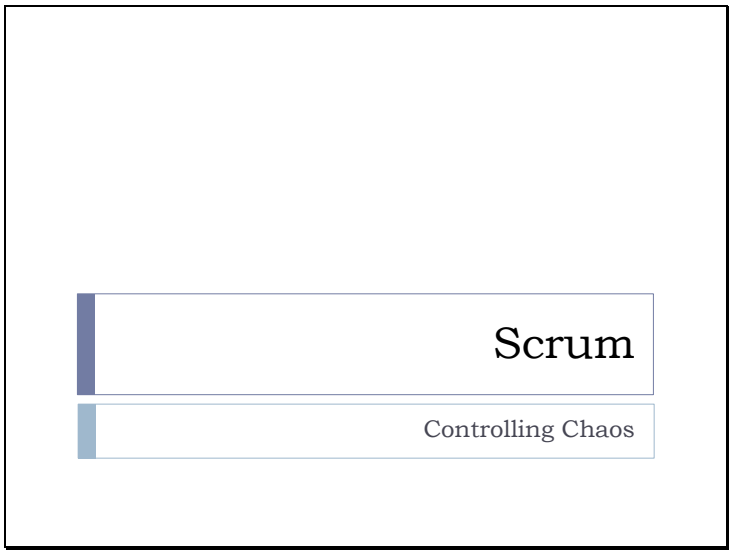
- The issue of all of members of a group doing an equal amount of work can never be adequately dealt with, but a genuine attempt was made in this course to deal with this

problem. The potential solution was to place students with equal levels of skill into groups with one another in order to hopefully mitigate the normal issues that typically arise in such group work. This course of action seems to have worked to a reasonable degree, but a problem is presented as well. This is that this is not really indicative of how a software development team might be structured in the "real world". In such a setup there will always be those who slack and drag the group down and it is beneficial for students to learn how to adapt to such situation and thrive despite any disadvantages presented by such a situation. However, the decision to separate the groups in such an equitable way was appropriate in an academic setting to promote equality of opportunity.

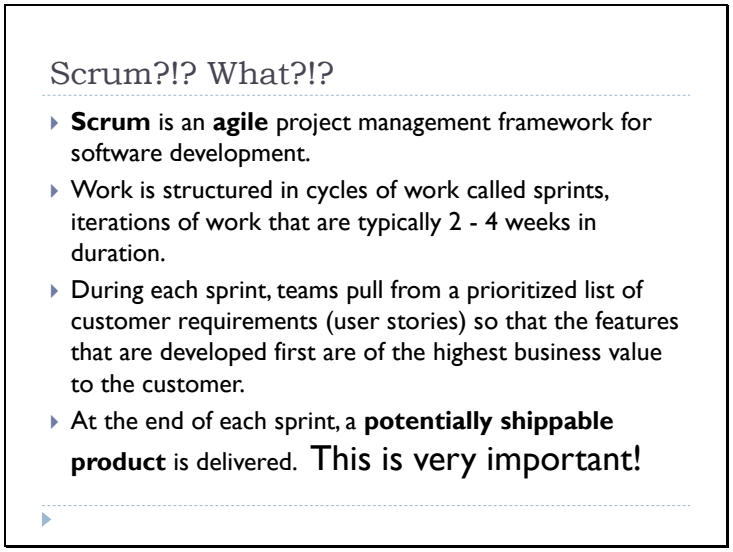
This project has presented a great opportunity to learn about and address many issues in computer science and software engineering education. Only the surface has been scratched in what is necessary to appropriately teach a software engineering class, but the project benefitted greatly from the experience and knowledge of Dr. Will Lloyd, without whom this project would not have been possible. Hopefully, a number of important questions have been answered and a great deal has been learned by others, especially by the students in the course, through the undertaking of this project and hopefully it has been beneficial to all involved.

Appendix I - Lecture on Scrum

Slide 1



Slide 2



Slide 3

Scrum for our purposes

- Throughout this lecture Scrum will be discussed as it is used in the workplace, but there will be points where this is incompatible with the structure of a class, so modifications to the framework are pointed out where appropriate.

Slide 4

What is this **Agile** thing you speak of?

A set of work methods and tools aimed at:

- Improving the ability to respond quickly to needs and requests from the market
- Cutting down waste and waiting periods
- Reducing employee stress while simultaneously increasing productivity

Slide 5

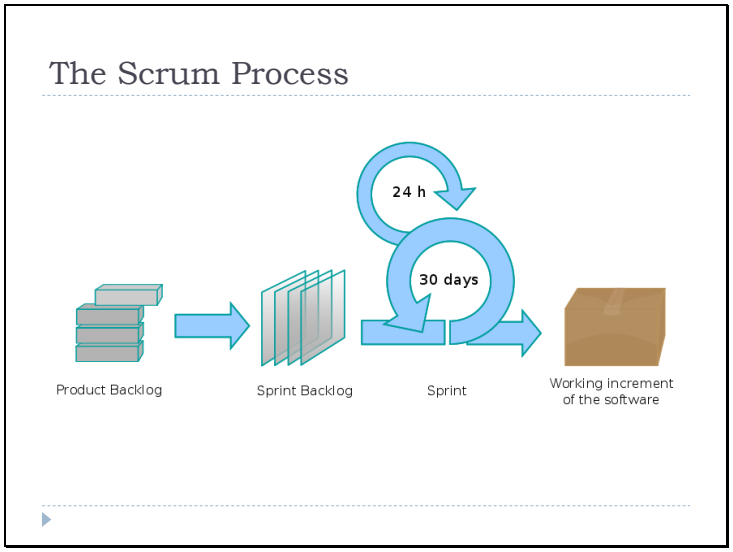
“Scrum”?

Slide 6

“Scrum”?

- A Scrum is a team pack in Rugby, where everybody in the pack acts together with everyone else to move the ball down the field.
- The reasons for this naming *cough*the meeting*cough* will become apparent as we progress

Slide 7



Slide 8

Scrum Roles

- Product Owner – represents the customers (stakeholders) and speaks from a business perspective
- Scrum Master – maintains scrum processes, main job is to remove impediments to the process
- Scrum Team – the developers, they organize the work themselves and lack a formal project manager, normally consists of 5 – 10 people, should be cross functional
- (Our teams will obviously be smaller, which is better)

Slide 9

Scrum Artifacts

- Artifact - a tangible byproduct produced during the development of software
  - Ex: class diagrams, uses cases, UML models
- The Product Backlog
- The Sprint Backlog
- The Burndown Chart

Slide 10

The Product Backlog

- The **product backlog** is a document prepared by the product owner that contains a list of customer requirements prioritized by business value.
- These requirements can and will frequently change.
- It should include all features visible to the customer, as well as the technical requirements needed to build the product

Slide 11

The Product Backlog (Example)

	Item #	Description	Est	By
Very High				
	1	**Finish database versioning**	16	KH
	2	**Get rid of unneeded shared Java in database**	8	KH
	-	**Add licensing**	-	-
	3	Concurrent user licensing	16	TG
	4	Demo / Eval licensing	16	TG
		Analysis Manager		
	5	File formats we support are out of date	160	TG
	6	Round-trip Analyses	250	MC
High				
	-	**Enforce unique names**	-	-
	7	In main application	24	KH
	8	In import	24	AM
	-	**Admin Program**	-	-
	9	Delete users	4	JM
	-	**Analysis Manager**	-	-
	10	When items are removed from an analysis, they should show up again in the pick list in lower 1/2 of the analysis tab	8	TG
	-	**Query**	-	-
	11	Support for wildcards when searching	16	T&A
	12	Sorting of number attributes to handle negative numbers	16	T&A
	13	Horizontal scrolling	12	T&A
	-	**Population Genetics**	-	-
	14	Frequency Manager	400	T&M
	15	Query Tool	400	T&M
	16	Additional Editors (which ones)	240	T&M
	17	Study Variable Manager	240	T&M
	18	Haplotypes	320	T&M
	19	**Add icons for v1.1 or 2.0**	-	-
	-	**Pedigree Manager**	-	-
	20	Validate Derived kindred	4	KH
Medium				
	-	**Explorer**	-	-
	21	Launch tab synchronization (only show queries/analyses for logged in users)	8	T&A
	22	Delete settings (?)	4	T&A

Slide 12

The Sprint Backlog

- The **sprint backlog** is a detailed document containing information about what requirements and *how* the team is going to implement these requirements for the upcoming sprint.
- Tasks are typically broken down into *hours* with no task being more than 16 hours. If a task is greater than 16 hours, it should be broken down further.

Slide 13

The Sprint Backlog (Example)

Tasks	Mon	Tues	Wed	Thurs	Fri
Code the user interface	8	4	8		
Code the middle tier	16	12	10	4	
Test the middle tier	8	16	16	11	8
Write online help	12				
Write the foo class	8	8	8	8	8
Add error logging			8	4	

Slide 14

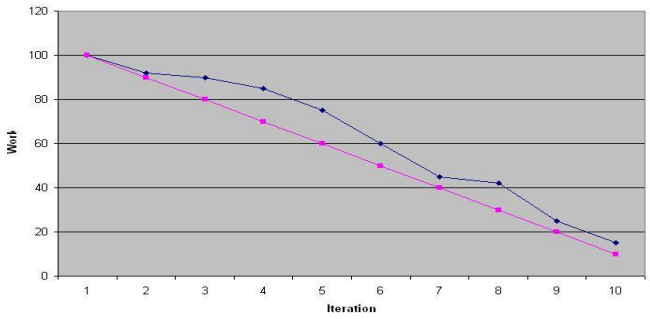
The Burndown Chart

- The Burndown Chart shows the cumulative work remaining in a Sprint and is updated on a day-to-day basis.
- This is used as a tool to guide the development team to successful completion of a Sprint on time with working final product.

Slide 15

The Burndown Chart (Diagram)

The straight line represents an ideal iteration where work is completed in a perfectly steady and evenly distributed manner. The more erratic line represents the work that is actually completed over time by the team.



Slide 16

Sprint Planning Meeting

- A meeting at the beginning of a sprint between the Product Owner, the Scrum Master, and the Team.
- Product Owner describes highest priorities and the Team decides what to move from the product backlog to the sprint backlog
- Typically takes 8 hours
- For our purposes this may entail a single class meeting

Slide 17

The Sprint

- A **Sprint** is a 15-30 day period, the actual length being decided by the team, during which the team creates an increment of potentially shippable software. Each day during the sprint a **Scrum Meeting** is held.
- During a sprint NO outside interference with the Team is allowed
- Our sprints will be considerable shorter, lasting only a week or two

Slide 18

The Scrum Meeting (1)

Each day during the sprint, a project status meeting run by the Scrum Master is arranged. This has specific guidelines:

- Anyone may attend and listen at the meeting, but only the Scrum Master and the team members may speak.
- The meeting is typically time-boxed at 15 minutes regardless of the team's size or the project
- The meeting should happen at the same location and same time every day
- Punishments for those who are late to meetings is decided by the Team

Slide 19

The Scrum Meeting (2)

During the meeting, each team member has to answer three very important questions:

- What have you done since yesterday?
- What are you planning to do by tomorrow?
- Do you have any problems preventing you from accomplishing your goal? (It is the role of the Scrum Master to remember and attempt to remove these impediments.)

Slide 20

The Scrum Meeting (3)

- Google Code should have a framework in place for you to regularly answer the 3 questions
- We may have an actual standup meeting with each team once a week

Slide 21

The Project At The End Of The Sprint

- At the end of the sprint the team should have created a **working, potentially shippable increment** of the project
- If there is not a **working, potentially shippable increment** of the project then the sprint has been wasted and time has been lost

Slide 22

Typical Impediments To The Process

- The meeting rules are not followed
- Product Vision and Sprint Goal are unclear
- The Product Owner is not available for questions
- The Product Backlog is not prioritized by business value
- Not everyone who contributes to the delivery is in the team
- The Scrum Master has to perform other tasks and is not able to focus on the team progress
- The team is too big (>10 members)
- The team has no room where they can work together
- The team has no dashboard to access the Sprint Backlog

Slide 23

The Sprint Retrospective

- After each sprint a brief meeting (3 – 4 hours), called a ***sprint* retrospective** is held, at which the Scrum Master and all team members reflect about the past sprint.
- The purpose of this is to review both what went well and what should be improved in the next sprint.
- Once again, for our purpose, this may entail a single class meeting

Slide 24

Scrum vs. Other Agile Methods

- **Lean Development** deals with which comprehensive principles should apply for the entire development organization
- **Scrum** deals with how the project is organized and planned
- **XP (Extreme Programming)** deals with how to work with programming

Slide 25

Benefits Of Scrum (1)

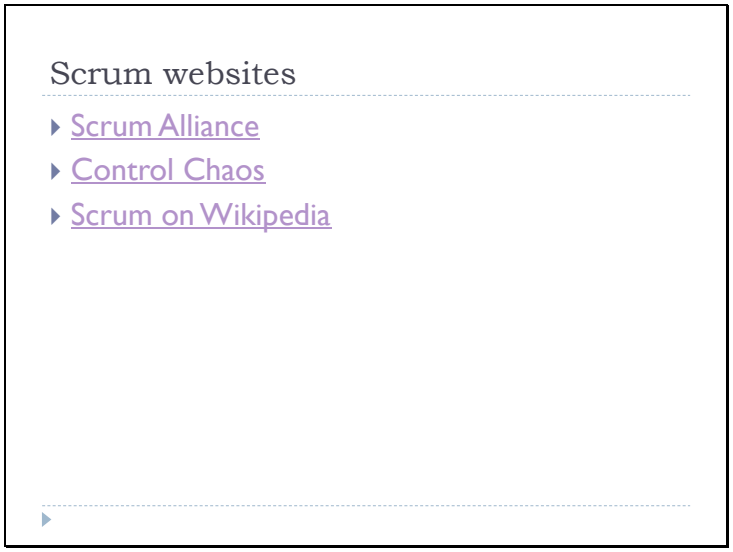
- Scrum is agile at its finest
  - A key principle of Scrum is its recognition that during a project the customers can (and will) change their minds about what they want and need and that unpredicted challenges cannot be easily addressed in a planned or predictive manner.
- Scrum adopts an empirical approach
  - It acceps that the problem cannot be fully understood or defined, focusing instead on maximizing the team's ability to deliver quickly and respond to emerging requirements.

Slide 26

Benefits Of Scrum (2)

- Quickly developed, potentially shippable product produced within 30 days
- Delivers the highest business value features first and will always try and avoid building unrealistic features.
- Frequent customer input
- Customers, through product owner, set development priorities

Slide 27

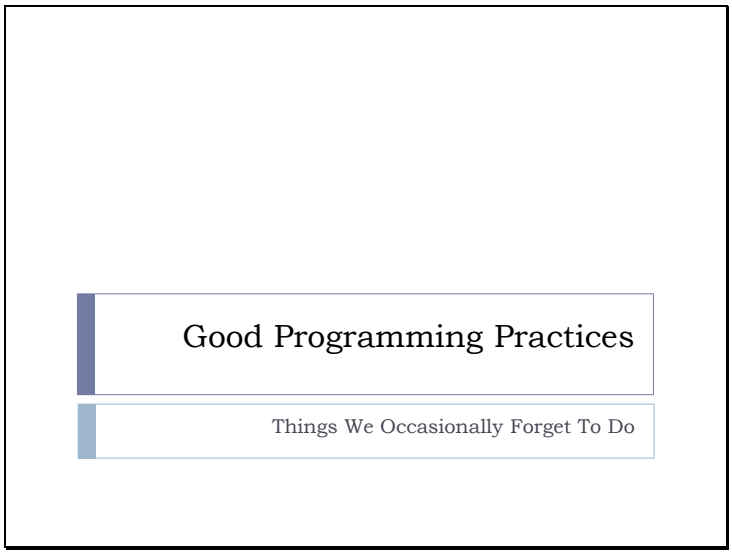


Slide 28

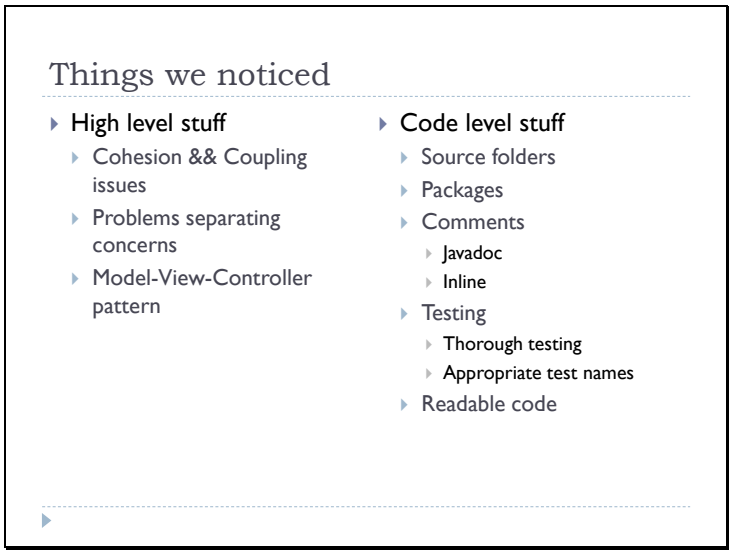
Slide 29

Appendix II - Lecture on Good Programming Practices

Slide 1



Slide 2



Slide 3

Cohesion && Coupling

- Remember: we want our projects to have high cohesion and low coupling
- What this means:
  - Given two lines of code, A and B, they are **coupled** when B must change behavior only because A changed.
  - They are **cohesive** when a change to A allows B to change so that both add new value.

Slide 4

Separation of concerns

- What this means:
  - This is the process of separating a program into distinct features that overlap in functionality as little as possible.
- Consequently:
  - As little program code as possible needs to be in your view (GUI/console/whatever) class
  - This can always be fixed through refactoring, but it's a good idea to just not do it in the first place

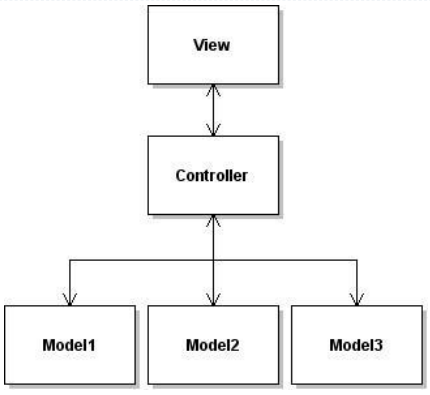
Slide 5

Model-View-Controller (1)

- This is a very basic and very important pattern in software engineering
- Helps with separation of concerns
- Helps with coupling && cohesion (to a degree)

Slide 6

Model-View-Controller (2)



Slide 7

Model-View-Controller (3)

- We have discussed this extensively both this and last semester
- If you have further questions please use Google wisely or talk to Josh or Dr. Lloyd

Slide 8

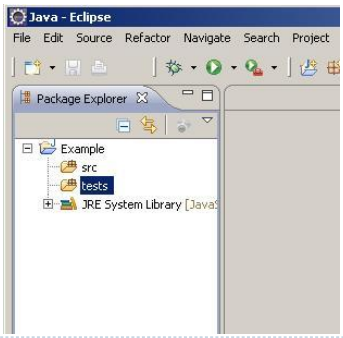
End of high level stuff

- Now on to the code level stuff…

Slide 9

Source Folders

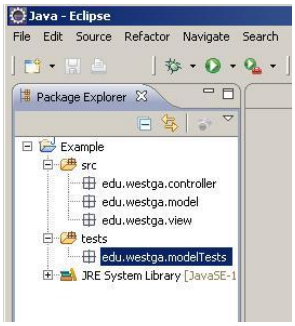
- Within your project you should have a number of source folders:
  - src
  - tests



Slide 10

Packages

- Packages within your project should have descriptive names following a reverse URL convention
- Examples:
  - edu.westga.model
  - edu.westga.modelTests
  - edu.westga.controller
  - edu.westga.view



Slide 11

Inline Comments

- You should always add inline comments if what the code is doing is not explicitly clear

```
 */
public class ModelOne {

    public ModelOne() {
        // Anything that is not explicitly clear in your code
        // should be accompanied by an inline comment
    }
}
```

Slide 12

javadoc comments

javadoc comments are similar to the XML comments from C# in that they are visible in others classes when an object is used:

```
public void doSomethingElse() {
    this.theM1.thisMethodDoesSomethingtoSomething(this.testInt);
}
```

void edu.westga.model.ModelOne.thisMethodDoesSomethingtoSomething(int x)

As you can see, whatever you type in the javadoc comment appears here

Parameters:
x: a parameter
x

Press 'F2' for focus

Slide 13

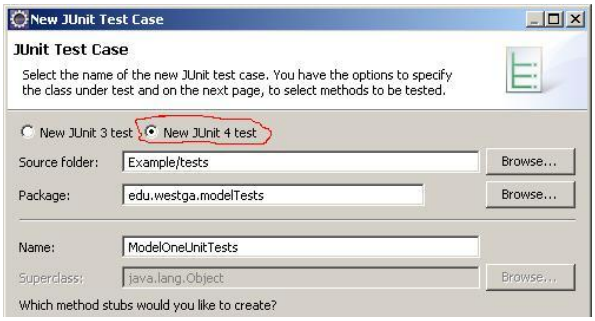
Testing: Thoroughness

- Always make sure to thoroughly test your classes and methods therein
  - Remember to test boundaries where applicable
  - If a method can only return 2 – 3 possible results then test for all possibilities

Slide 14

Testing: JUnit Version

- Make sure to always use JUnit 4



Slide 15

Testing: Test Names (1)

- It is very important to have proper test names
- This is self-documenting code
- Eliminates the need for comments in test cases
- Good test name:
  shouldDoSomethingWhenSomethingElseHappens
- Bad test name:
  testX

Slide 16

Testing: Test Names (2)

```
@Before
public void setUp() throws Exception {
    try {
        this.theM1 = new ModelOne();
    } catch(Exception e) {
        // nothing to really do here
    }
}

@Test
public void shouldGetFiveWhenFourIsPassedInToThisMethodDoesSomethingtoSomething() {
    int expected = 20;
    int testVar = 4;
    int actual = this.theM1.thisMethodDoesSomethingtoSomething(testVar);

    assertEquals(actual, expected);
}
```

Slide 17

Readable Code

- It is always a good practice to chop your code into sections to make it more readable

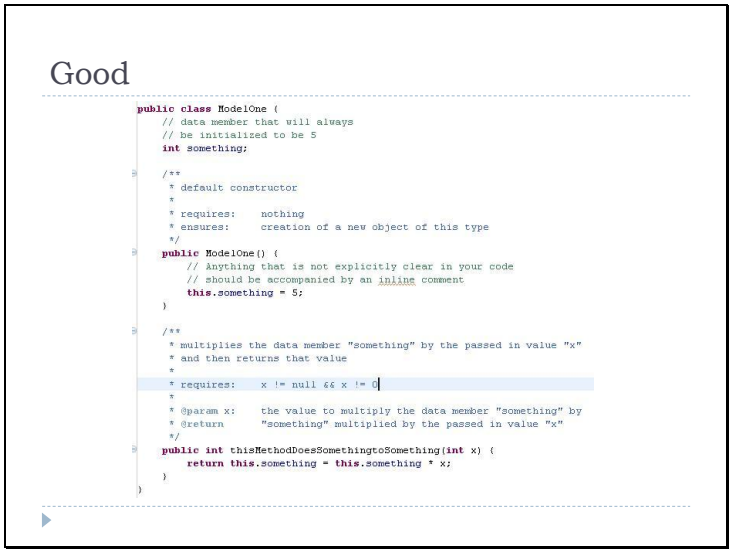
Slide 18

Bad

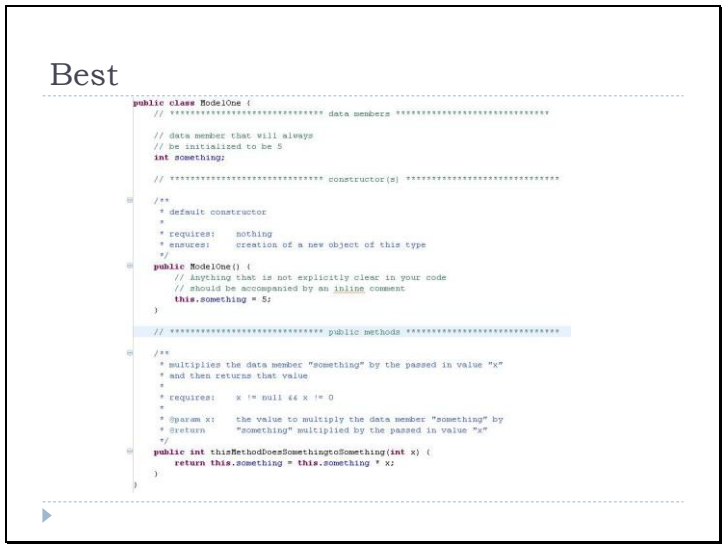
```
 */
public class ModelOne {

    int something;
    public ModelOne() {
        // Anything that is not explicitly clear in your code
        // should be accompanied by an inline comment
        this.something = 5;
    }
    public int thisMethodDoesSomethingtoSomething(int x) {
        return this.something = this.something * x;
    }
}
```

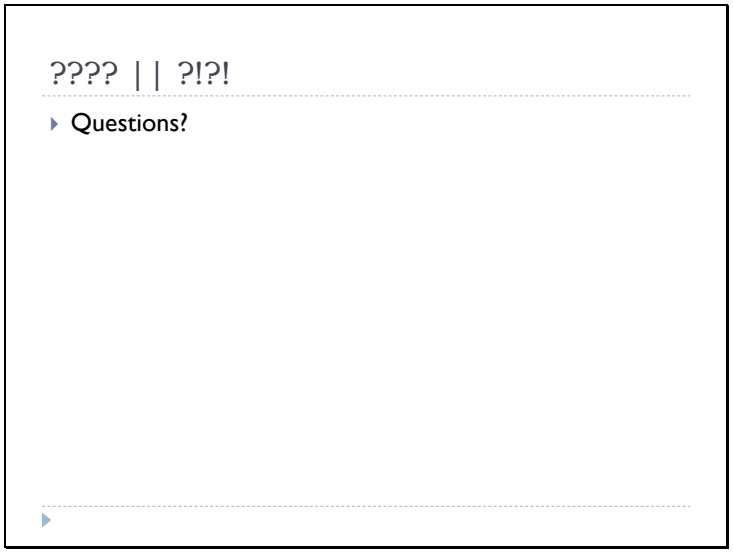
Slide 19



Slide 20

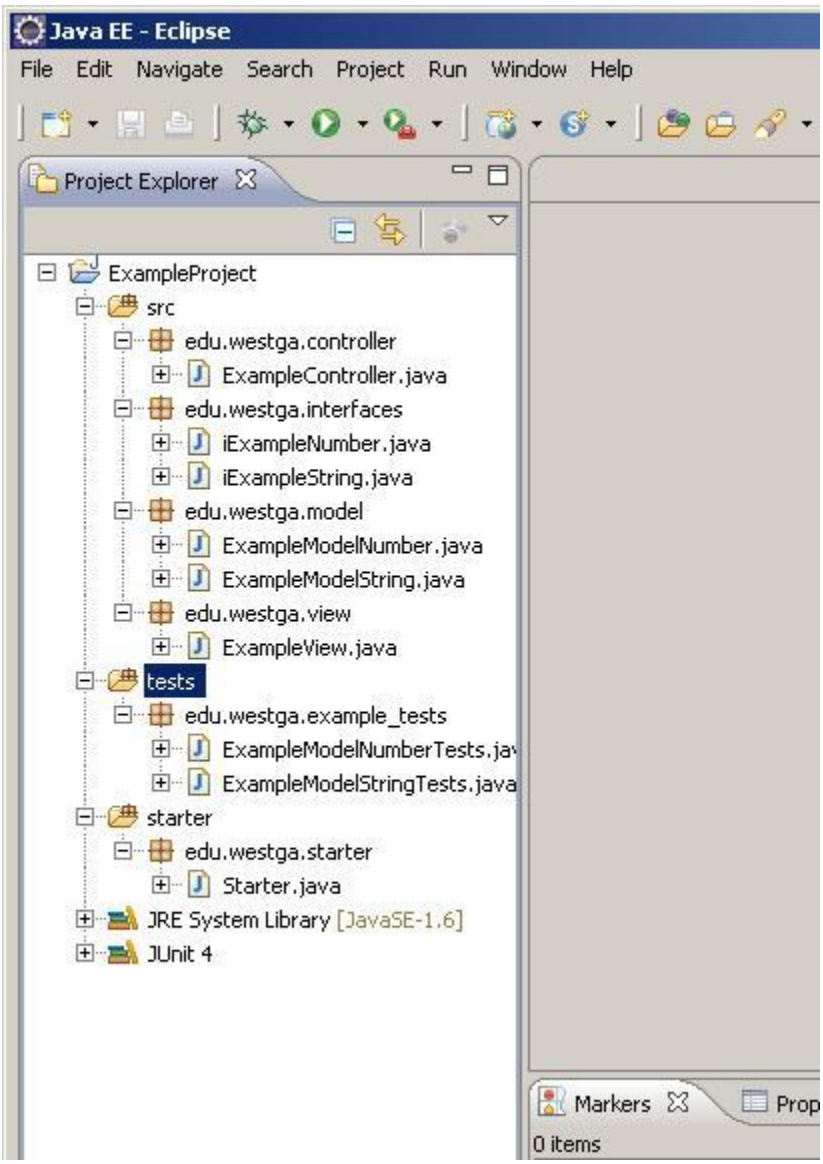


Slide 21



Appendix III - Example Project

The following is the code from the example project that was created and distributed to the students in the course on 10/19/2009. The structure of the project (including packages and such) is detailed in the image below:



The code from this example project will not be included here, but is available on the CS 3212 F09 page[16] in the CS Course Web at any time.

[16] http://courses.cs.westga.edu/mod/resource/view.php?id=27175

Bibliography

Fowler, Martin. 2005. The New Methodology. From Nothing to Monumental to Agile. http://martinfowler.com/articles/newMethodology.html (Accessed 11/12/2009)

Unknown Author. Mercurial SCM. http://mercurial.selenic.com/about/ (Accessed 11/15/2009)

Unknown Author. 2009. Scrum Methodology & Agile Scrum Methodologies. http://scrummethodology.com/ (Accessed 11/12/2009).

Various Anonymous Authors. 2009. (Wikipedia Entry for) Agile software development. http://en.wikipedia.org/wiki/Agile_software_development (Accessed 11/15/2009).

Various Anonymous Authors. 2009. (Wikipedia Entry for) Scrum (development). http://en.wikipedia.org/wiki/Scrum_%28development%29 (Accessed 11/12/2009).

Josh Westmoreland. CS 3212 Course Management Proposal. 03/24/2009.