Announcements

- Tuesday, Feb 07 – no class
  - Use it to work on your milestone release
- Thursday, Feb 09 – project presentations in class
  - Time: 10 mins + 2 mins for questions per team
  - Different ordering of teams

Expectations and Questions for Your Team's Presentation

- Demo
  - Show what the team has accomplished so far, even if it is rudimentary at that stage.
- Progress since the previous milestone
  - Have you been able to tackle (or at least get an accurate assessment of) the biggest risks identified previously?
  - Has the planned agenda turned out as expected or were there surprises (and if so, which ones and what is being done about them)?
- Biggest risks going forward
  - What is your specific plan for the next milestone?
  - What should we reasonably expect to see by then?

Lecture 10: Software Quality

Valentin Razmov

Outline

- Quality – a look back at history
- What is quality?
- How do we measure quality?
- How do we improve software quality?

References

- "Professional Software Development", by Steve McConnell
- "The Joel Test: 12 Steps to Better Code", by Joel Spolsky
- "Good Enough Quality: Beyond the Buzzword", by James Bach
  - http://www.satisfice.com/articles.shtml
- "Agile Software Development with Scrum", by Ken Schwaber and Mike Beedle

Project-Related Reflections: Configuration Management

- Is this surprising?
  - Such hurdles are always present at the start. Ironing them out is a major reason for doing a *zero-feature release* very early.
- Could it have been different?
  - Yes, but not much. VSTS is a new development environment, still in beta quality. Expecting it to be already on par with well tuned development environments is, perhaps, a stretch.
- Why spend so much time on fixing this?
  - Regular builds are the heart-beat of a project – without them, the project is dead (or on life support, at best).
  - This stuff is hard, resolving it takes patience and iterations. But it's done only once – then, development can flow better.

Software 'Gold Rushes' and Their Influence on Quality

- The software 'Gold Rush' fever periods
  - Goal: being first-to-market in an unclaimed segment
  - Typical environment: two guys in a garage
  - High-risk projects, potentially high pay-off
  - Code-and-fix development, very informal processes
  - Customers are tech savvy, willing to forgive bugs
- The in-between (post-'Gold Rush') periods
  - Goal: sustained, productive competition with others
  - Typical environment: larger teams, formal processes
  - Lower-risk, likely lower but more predictable pay-off
  - Careful, quality-driven development with an emphasis on quality (reliability, interoperability, usability, etc.)
  - Different customer base: demands reliability

The Goal of Building Software

- To deliver a product that satisfies the customer(s)
  - on time
  - on budget
  - with good quality
- "We do three types of jobs here... Good, Fast and Cheap. You may choose any two!"
- Which two would you pick for a project like yours?
  a) Good and Fast, but not Cheap
  b) Good and Cheap, but not Fast
  c) Fast and Cheap, but not Good

Student Submission

The Quality Question

How do we ensure good quality for software?

What is Quality?

- Quality is in the eyes of the beholder (customer).
  - First, define what quality means (for the customer!).
  - You and the customer must agree on the expected level of quality.
- What constitutes good quality in one situation may not be considered good quality in another.
  - E.g.: in a toy project vs. in a safety-critical system
- A *contract* must have at least the following components:
  - **Who** promises to do
  - **What**
  - for **Whom**
  - by **When**
  - with what **Quality Criteria**/Standards, and
  - with what **Notification Mechanism** upon completion

Components of Quality

- Quality comprises (but is not limited to):
  - Requirements quality
  - Design quality
  - Code quality
  - Test quality
  - Documentation quality
- Given limited resources, which of these do you consider more important to pay attention to? Why?

Student Submission

How Do We Measure Software Quality?

- Software is never perfect.
  - We cannot be sure it is free of defects.
- What can be done to assess the quality then?
  - Many engineering disciplines use standards for quality.
  - In software, there are few standards, and all (viable) ones assess the *quality of processes*, not products.
    - Most non-trivial properties of software (code) cannot be inferred or verified, because of the Halting Problem.
    - We are forced to link process quality to product quality.
      - Conway's Law: "The structure of a computer program reflects the structure of the organization that built it."
    - E.g.: CMM (Capability Maturity Model) assesses the quality of teams/organizations through their processes.

"Good Enough" Quality

- At some point, one needs to stop and decide it is all good enough to ship. Under what conditions?

Criteria for "good enough" quality:

1. There are clear benefits (of the software).
2. There are no critical problems.
3. Overall, the benefits outweigh the problems.
4. All things considered, further development would be more harmful than helpful.

"Good Enough" Quality (cont.)

Other questions to consider:

- Good enough *for whom*?
  - You? Your team? The customer?
- Good enough *for what*?
  - A demo? A beta release? Selling it? Capturing market share?
- Have you agreed on:
  - Team standards for acceptable quality?
  - What would constitute success for your team in the end?
    - These are team conversations we've done in cse403.

Mechanisms for Raising the Quality of Software

- Assume you are brought in on an ongoing software project plagued by poor quality. What one or two approaches (mechanisms) would you propose to help raise the quality of the software in production?
  - Make assumptions as needed, to concretize the question.

Student Submission

Mechanisms for Raising the Quality of Software: My Ideas

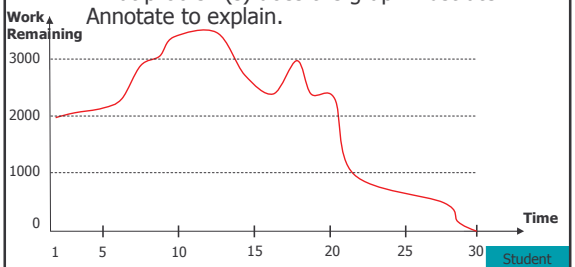
Which of the following mechanisms do you use (or plan to use) on your project? Circle all that apply.

a) Involvement / frequent iterations with customers and other stakeholders
b) Pair programming
c) Code reviews (not limited to just "code" – requirements/design review, etc.)
d) External auditing
e) Using automated tools (e.g., static analysis, code coverage, IDEs, etc.) to help discover non-trivial properties that affect quality
f) Refactoring
g) Code integration (if not already in place)
h) Testing: integration testing, regression testing, acceptance testing; automated testing; test-driven development (with unit testing)
i) Component reuse
j) Team building activities
k) Establish (or ensure the presence of) clear responsibilities within the team
l) Realistic up-to-date scheduling

Student Submission

A Problem Instance

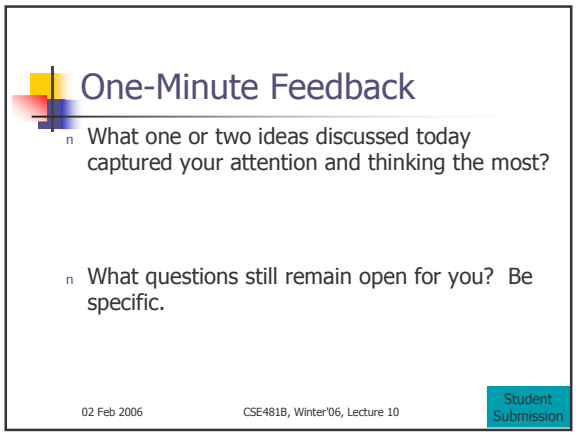
- What problem(s) does the graph illustrate? Annotate to explain.



Student Submission

Recipes for Creating Disasters (a.k.a. Poor Quality Products)

- Ignore what the customers say they want – the developers surely must know better.
- Put in all the features that could possibly ever be useful.
- Do not worry about quality aspects until the deadline approaches.
- Don't waste time on design or documentation – after all, code is the most important thing and time is already short to do all there is to be done.
- …


One-Minute Feedback
What one or two ideas discussed today captured your attention and thinking the most?
What questions still remain open for you? Be specific.
02 Feb 2006
CSE481B, Winter'06, Lecture 10
Student Submission