CSC 308 Lecture Notes Week 10, Part 2
Software Process Assessment
The Future of Software Engineering

I. **Final exam.**

A. Day and time: Friday 10:10AM - 1:00PM

B. Length: <= three hours

C. Open note.

D. Content:

1. Cumulative.
2. Simple fill-in, true/false, as on midterm.
3. Short answer, as on midterm and quiz.
4. Project-related, as on midterm and sample.
5. Prototyping, including a simple implementation of some aspect of prototyping, submitted during the final using the CSL `handin` program.

E. Topics that may be covered in the short-answer questions:

1. the software process
2. inspection testing
3. version control with SVN

F. Details of Project-Related Content

1. You’re provided a general problem description.
2. Questions:
   a. Define high-level GUI layout.
   b. Do a simple requirements scenario or two
   c. Define objects and operations.
   d. Draw UML.
   e. Define preconditions and postconditions.

G. A Sample Final

1. Goto main 308 web page, follow link.
2. Provides sample of project-related questions.

II. **Peer review.**

A. See the handout.

B. One review for leader, if you’re not it.

C. One review for each other team member.

D. Don’t review yourself here.

E. Please bring the completed peer review to the final exam on Wednesday.

III. **Software Process Assessment**

A. Ideally, a software project should be conducted in an orderly, repeatable process.

B. Such is not always the case in industrial practice, or even in academia.

C. In 308 we’ve used a specific process, suitable for the development of medium-scale information processing systems with a substantial end-user interface component.

IV. **"A" versus "The" software process.**

A. The 308 process is by no means *the* software process; rather, it is *a* process, among many.

B. Software processes may vary widely, based on the kind of software being developed and the setting in which it will operate.

1. For example, the process to develop new software for an artificial intelligence research project can be quite different than the process for developing a commercial product in a well-known application domain.
2. Also, organizational factors can significantly affect the details of a software development process.

C. The critical factors regarding a software process are these:

1. For a given project, it must be completely and formally defined.
2. All project participants must understand and follow the process (though not necessary love it).
3. The process must be regularly subject to evaluation so that it can evolve to become better.

D. The details of the process we have employed in CSC 308, and will employ in 309, are given in Chapter 2 of the online text materials.

V. **Software process assessment using the Capability Maturity Model (CMM and CMMi)**

A. CMM has been developed at Carnegie Mellon since around 1987; it is still undergoing refinement, in particular the "Integrated" version

B. The five CMM levels (in order of increasing maturity) are:

1. *Initial* -- ad hoc
2. *Repeatable* -- basic project management techniques are used
3. *Defined* -- a software engineering process is used
4. *Managed* -- quantitative Q/A process is used
5. *Optimizing* -- the process itself can be refined to improve efficiency

C. Improvements claimed for more mature processes:

1. More reliable software
2. Better visibility into process, particularly from top-level management perspective
3. Less risk in contracting with firms that have a mature process in place.

VI. **Details of the five CMM levels**

A. Level 1: completely ad hoc

B. Level 2:

1. Requirements management
   a. Goal 1: System requirements allocated to software are controlled to establish a baseline for software engineering and management use.
   b. Goal 2: Software plans, products, and activities are kept consistent with the system requirements allocated to the software.
2. Project planning
   a. Goal 1: Software estimates are documented for use in planning and tracking the software project.
   b. Software project activities and commitments are planned and documented.
   c. Goal 3: Affected groups and individuals agree to their commitments related to the software project.
3. Project tracking and oversight

   a. Goal 1: Actual results and performances are tracked against toe software plans.
   b. Goal 2: Corrective actions are taken and managed to closure when actual results and performance deviate significantly from the software plans.
   c. Goal 3: Changes to software commitments are agreed to by the affected groups and individuals.

4. Subcontract management
   a. Goal 1: The prime contractor selects qualified software subcontractors.
   b. Goal 2: The prime contractor and the software subcontractor agree to their commitments to each other.
   c. Goal 3: The prime contractor and the software subcontractor maintain ongoing communications.
   d. Goal 4: The prime contractor tracks the software subcontractor's actual results and performance against its commitments.

5. Software quality assurance
   a. Goal 1: Software quality assurance activities are planned.
   b. Goal 2: Adherence of software products and activities to the applicable standards, procedures, and requirements is verified objectively.
   c. Goal 3: Affected groups and individuals are informed of software quality assurance activities and results.
   d. Goal 4: Noncompliance issues that cannot e resolved within the software project are addressed by senior management.

6. Configuration management
   a. Goal 1: Software configuration management activities are planned.
   b. Goal 2: Selected software work products are identified, controlled, and available.
   c. Goal 3: Changes to identified software work products are controlled.
   d. Goal 4: Affected groups and individuals are informed of the status and content of software baselines

C. Level 3

1. Organization process focus
   a. Goal 1: Software process development and improvement activities are coordinated across the organization.
   b. Goal 2: The strengths and weaknesses of the software processes used are identified relative to a process standard.
   c. Goal 3: Organization-level process development and improvement activities are planned.

2. Organization process definition
   a. Goal 1: A standard software process for the organization is developed and maintained.
   b. Goal 2: Information related to the use of the organization's standard software process by the software projects is collected, reviewed, and made available.

3. Training program
   a. Goal 1: Training activities are planned.
   b. Goal 2: Training for developing the skills and knowledge needed to perform software management and technical roles is provided.
   c. Goal 3: Individuals in the software engineering group and software-related groups receive the training necessary to perform their roles.

4. Integrated software management
   a. Goal 1: The project's defined software process is a tailored version of the organization's standard software process.
   b. Goal 2: The project is planned and managed according to the project's defined software process.

5. Software product engineering
   a. Goal 1: The software engineering tasks are defined, integrated, and consistently performed to produce the software.
   b. Goal 2: Software work products are kept consistent with each other.

6. Intergroup coordination
   a. Goal 1: The customer's requirements are agreed to by all affected groups.
   b. Goal 2: The commitments between the engineering groups are agreed to by the affected groups.

   c. Goal 3: The engineering groups identify, track, and resolve intergroup issues.

7. Peer reviews
   a. Goal 1: Peer review activities are planned.
   b. Goal 2: Defects in the software work products are identified and removed.

D. Level 4

1. Quantitative process management
   a. Goal 1: The quantitative process management activities are planned.
   b. Goal 2: The process performance of the project's defined software process is controlled quantitatively.
   c. Goal 3: The process capability of the organization's standard software process is known in quantitative terms.
2. Software quality management
   a. Goal 1: The project's software quality management activities are planned.
   b. Goal 2: Measurable goals for software product quality and their priorities are defined.
   c. Goal 3: Actual progress toward achieving the quality goals for the software products is quantified and managed.

E. Level 5

1. Defect prevention
   a. Goal 1: Defect prevention activities are planned.
   b. Goal 2: Common causes of defects are sought out and identified.
   c. Goal 3: Common causes of defects are prioritized and systematically eliminated.
2. Technology change management
   a. Goal 1: Incorporation of technology changes are planned.
   b. Goal 2: New technologies are evaluated to determine their effect on quality and productivity.
   c. Goal 3: Appropriate new technologies are transferred into normal practice across the organization.
3. Process change management
   a. Goal 1: Continuous process improvement is planned.
   b. Goal 2: Participants in the organization's software process improvement activities is organization wide.
   c. Goal 3: The organization's standard software process and the projects' defined software processes are improved continuously.

VII. **A technologist's view of how to build quality software**

A. The developers of CMM focus largely on the management aspects of the software process, not the technical aspects.

1. From their perspective, proper management is ultimately more important to achieving quality software than are the particular details of software engineering technology.
2. Technologists tend to believe the opposite (but this belief is probably mistaken).
3. For example, when technologists read that

   > "CMM does not currently address expertise in particular application domains, advocate specific software technologies, or suggest how to select, hire, motivate, and retain competent people."

   technologists view CMM of limited utility when it comes to assessing what it really takes to build quality software "in the trenches".

B. So, if we assume that a mature software process is in place, here is one technologist's view of the the top ten things that *really* count for achieving quality software:

1. A good requirements specification, produced with copious end-user feedback.
2. Thorough and formal test procedures at all levels of development.
3. One set of conventions per project, and one extremely nasty manager to enforce them.

4. Real deadlines, and an even nastier manager to enforce them.
5. A document-as-you-go policy, where document commentary is written with the following question in mind: "What will I need to know when I come back here in six months and want to understand what's going on?"
6. Thorough and formal version control procedures, including readily accessible access to at least the last three working versions of the system.
7. Thoughtfully written version control log messages.
8. Teamwork, including frequent critical reviews.
9. The 7+/-2 rule.
10. Good specification and design roadmaps, in whatever diagraming style(s) you like.

C. Honorable mentions

11. Sleep deprivation when necessary
12. High-quality junk food
13. Emacs
14. Top-of-the-line OS X computer

D. Noteworthy Omissions

1. Any particular methodology, object-oriented or otherwise; just choose one and stick with it.
2. Any particular development languages; just choose one or more (except for Visual Basic) and stick with them.

-- The Future of Software Engineering (Fisher's Version) --

VIII. **Near term (1-5 years).**

A. Continued market pressure to build imperfect software.

B. Increased litigation against imperfect software.

1. Litigation will proceed particularly against *harmfully* imperfect software.
2. It may also proceed against software that is disruptive or wastes substantial amount of users' time due to poor performance.

C. Increasing use of software libraries.

1. It is happening today, with libraries like MFC (Microsoft Foundation Classes) and Java JFC (Java Foundation Classes).
2. The idea is not to write low-level code from scratch but to (re)use code from a library.
3. E.g., instead of writing CSC-103-level hash table code, we use `class HashTable` from the library.

D. Increased focus on code-level software testing as a means of quality-control.

1. Even if software is poorly developed, code-level testing can be used as a means to keep buggy software from being released.
2. Code-level testing tools are becoming more widely available and used to manage tests, record results, and report bugs.

IX. **Medium term (5-10 years).**

A. Competitive pressures to build higher-quality software.

1. Consider the "quality shock" to U.S. car manufacturers from Japanese car manufacturers in the mid 1980s.
2. A similar shock may be come to U.S. software manufacturers, from other countries.

B. Professional licensing of software engineers.

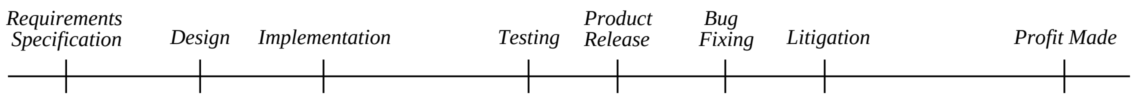
1. It may take some major software disaster to bring it about.
2. Or it may happen as a natural part of the maturation of our discipline, led by forward-thinking professional organizations such as ACM or IEEE.

C. Increased use of software components.

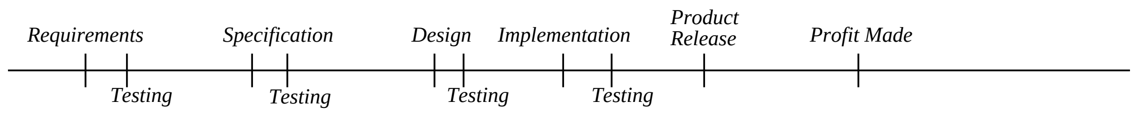
1. A *component* is the next larger unit in a software library, above the class.
2. Current libraries are composed of class-sized units, such as a class that does searching, another that does search query command processing, and others that can be used to build the GUI.
3. A reusable *component* is a larger-scale library unit, such as a complete search engine, including its ready-built GUI.
   a. Library components are built from the smaller-scale library classes to form complete, reusable applications.
   b. They're analogous to the kinds of larger-scale components that are used in other forms of engineered artifacts, such as integrated circuit chips and modular building components.

D. Increased appreciation for spending more time on requirements, specification, and testing, less time on bug fixing and litigation.

1. Old product time line:



2. New product time line:



E. Increased use of formal methods, particularly for specification.

1. Formal specs can be used to clearly define the capabilities of software library components, so the components can be properly used.
2. Formal specs are also a primary communication medium between software developers and a "programmer's apprentice".

F. Automated "programmer's apprentice" tools.

1. A programmer's apprentice is an intelligent agent that has enough knowledge to be a "junior assistant", primarily during implementation and functional testing phases of software development.
2. The apprentice can track where developers are in the process, do clerical work and some more advanced work.
   a. Clerical work includes such things as automatically compiling and running tests whenever code is checked in.
   b. More advanced work includes automatic generation of test cases for well-specified code (we'll discuss these techniques next quarter in 309), as well as intelligent debugging assistance.
3. Here's a hypothetical scenario of interaction between a programmer's apprentice and a developer, circa 2015:

Apprentice: Good morning Gene, what shall we work on today?

Fisher: SlickQueryProcessor

Apprentice: .... OK, module SlickQueryProcessor has been loaded and analyzed. I see from the development log that the functional testing phase is next for this module, is that correct?

Fisher: Yes.

Apprentice: Shall I generate some test cases for you?

Fisher: Yes.

Apprentice: While generating test cases, I noticed the following problem in the postcondition for method ValidateQuery:

```
forall (i | i >= 0 and i <= #response_list)
    ValidData(response_list[i])
```

Given that variable i is being used as a list index, it should not assume the value 0. I suggest:

```
forall (i | i >= 1 and i <= #response_list)
    ValidData(response_list[i])
```

Shall I make the fix?

Fisher: Yes.

Apprentice: Done. Shall I now continue with test case generation?

Fisher: Yes.

Apprentice: ... OK, test cases generated. Would you like to review the test cases or shall I run them now?

Fisher: Run.

Apprentice: Here is a summary of the test execution results:

```
Test Case 1: Passed.
Test Case 2: Passed.
...
Test Case 8: Failed.
...
```

Shall we debug the code that caused the failure in Test Case 8?

Fisher: Yes.

Apprentice: Starting debugger on `SlickQueryProcessor.main`

Fisher: *(Examines code and thinks a bit and then issues the following debugging commands).*

```
set breakpoint in SlickQueryProcesor.enter
run
```

Apprentice: I recognize the code pattern in method `SlickQueryProcesor.enter`. It's a version of hash table entry using out-of-table overflow. Based on my pattern matching analysis, the problem seems to be with how you're splicing into the bucket. I suggest you use the library version of hashing in class `AbstractTable`. Shall in install the change and rerun run the tests?

Fisher: Yes.

Apprentice: ... Change installed. ... Recompilation complete. ... Rerunning tests. ... All tests passed.

Is there more work to do today?

Fisher: No. I'll go home early.

X. **Long term (20 years).**

A. Adaptation to fundamentally new computer architectures, such as optical dataflow.

1. Optical computing and related technologies will fundamentally change computer architectures from essentially sequential to parallel.
2. These new computer architectures will require corresponding changes in software architectures.
3. In particular, computer code will look more like dataflow diagrams than the sequential procedural code of today.

B. Obsolescence of object-oriented technologies.

1. Object-oriented design is fundamentally unsuited to parallel architectures.
2. Object-oriented technology will fall into the bit bucket of history.

C. Automatic programming.

1. The programmer's apprentice had about the level of knowledge contained in an average CSC 103 text book.
2. However, the apprentice only knew how to use its knowledge to analyze code written by a human, not to generate code on its own.
3. An automatic programming system takes this next step.
4. Viz., an automatic programming system can generate code from a system specification.

D. Here's a hypothetical scenario of interaction between an automatic programming system and a developer, circa 2025:

AP System: Good morning Gene. I see from the development log that the specs for the Intelligent Calendar tool are done. Shall I generate the code and test it?

Fisher: Yes.

AP System: Complete code generation will take about 20 minutes. I suggest you go have a cup of coffee.

Fisher: OK.

AP System: During the specification analysis phase of code generation, I found and fixed the following apparent specification errors:

...

Do you want to review the fixes or shall I proceed with the code generation and testing?

Fisher: Proceed.

AP System: Code generation and testing will take about 2 hours. I suggest you go home early today and check back tomorrow.

Fisher: OK, see ya.

XI. **Very long term (30-50 years).**

A. The emergence of genuine artificial intelligence.

B. Automatic software engineering; here's the scenario circa 2050:

Automatic Software Engineer: Good morning Gene. I see from our interaction log that it has been a while since we last communicated. Over the last two months, I have met with customers, analyzed their requirements, and developed the specification for the "Analyze the Meaning of Life as We Know It" system. I am now ready to generate the code and test it. Shall I?

Fisher: Sure.

Automatic Software Engineer: Complete code generation and exhaustive testing will take about 48 hours. I suggest you go home and do whatever it is you do these days.

Fisher: OK, ciao.