CMPSC 280
Principles of Software Development
Fall 2015

Examination 3 Study Guide
Delivered: Tuesday, December 8, 2015
Examination: Friday, December 11, 2015 at 9:00 am

Introduction

This course will have its final examination on Friday, December 11, 2015 from 9:00 am to 12:00 noon. The examination will be "closed notes" and "closed book" and it will cover the following content. Please review the "Course Schedule" on the course's Web site to see the content and slides that we have covered in the first module. You may post questions about this material to Slack.

- Chapters One through Three in SETP (i.e., introduction to the software engineering lifecycle)
- Chapters One through Three in MMM (i.e., challenges and solutions in software engineering)
- Chapters Four through Six in SETP (i.e., requirements, architecture, and design)
- Chapter Four in MMM (i.e., trade-offs in strategies for system design)
- Chapter Seven in SETP (i.e., methods for software implementation)
- Chapters Twelve and Thirteen in MMM (i.e., tools for implementation and debugging)
- Chapters Eight through Ten in SETP (i.e., testing and delivery of software)
- Chapter Sixteen in MMM (i.e., the "no silver bullet" thesis)
- Your class notes, class activities, lecture slides, and the all of the laboratory assignments
- Knowledge of the basic commands necessary for using `git` and Bitbucket; basic understanding of the Markdown syntax and the use of associated command-line tools such as `pandoc`
- Key concepts introduced and reviewed during the laboratory sessions (e.g., the components of a "programming systems product", strategies for performing next-release planning, and metrics for evaluating the quality of a system's design and implementation)
- Lessons learned from the team-based implementation and delivery of real-world software systems during the completion of both the laboratory assignments and the final project.

Detailed List of Topics

The examination will include a mix of questions that will require you to draw and/or comment on a diagram, write a short answer, explain and/or write a source code segment, or give and comment on a list of concepts or points. The emphasis will be on the following list of illustrative topics; yet, as this examination is cumulative across the entire semester, this topic list is not exhaustive.

- The state-of-the-art and the key challenges within the field of software engineering, with a focus on the steps of problem solving and the meaning of terms like "defect" and "quality".

- The phases of the software development lifecycle and the ways in which different software process models (e.g., the spiral model or the V model) connect and interpret these phases.
- The key strengths and weaknesses of the different software development process models (e.g., one drawback of the waterfall model is its focus on documents and its lack of explicit iteration).
- Key terms such as "verification" and "validation" and "incremental" and "iterative".
- How to use activity graphs to track progress and plan a software development project. Additionally, an understanding of the ways in which managers will estimate the deadlines for completing a software system (e.g., using data mining algorithms to predict project characteristics such as anticipated costs and the likelihood of an on-time completion).
- The roles that members of a software team may play and the ways in which individual personalities and characteristics may equip certain people to work on specific tasks.
- How different types of software engineering tasks exhibit different relationships between the time-to-completion and the number of workers assigned to finish the task.
- The reasons why it is important to accurately capture the requirements for a software system.
- The different ways in which you can classify and understand software requirements.
- Characteristics of requirements and the ways in which they can be validated and verified.
- The relationship between the architecture and design of a software system.
- Canonical software architectures (e.g., "pipe and filter") and their strengths and weaknesses.
- The overall meaning and purpose of object-oriented design and implementation measures. For example, you should know the definitions of cyclomatic complexity and non-commented source statements and, additionally, be able to apply these metrics in practice.
- Principles for improving and evaluating a system's design. For instance, in the context of coupling, an understanding of tight and loose coupling, the kinds of architectures that promote different types of coupling, and examples of metrics calculated by tools such a JDepend.
- The ways in which pre-conditions, post-conditions, and invariants can document an interface.
- A basic understanding of object-oriented design and the effective use of design patterns.
- The different types of Unified Modelling Language (UML) diagrams, the purpose, strengths, and weaknesses of each canonical type of diagram, and, overall, the purpose of using diagrams in software design. For instance, an understanding of the UML package and class diagrams.
- An understanding of the phrase "conceptual integrity" and, additionally, the ability to analyze different strategies for achieving conceptual integrity in all phases of the software lifecycle.
- The tension between "aristocracy" and "democracy" in the design of software systems.
- Standards and guidelines that govern the way in which a program's source code should be implemented and formatted. A high-level understanding of a source code standard for the Java programming language and a clear sense of the benefits and drawbacks associated with using this standard. Additionally, the ability to look at a source code segment, written in Java, and comment on how it adheres to or violates a Java coding standard.
- General guidelines for improving the implementation of a software system. For instance, the ways in which a software development team can achieve objectives such as "localizing input and output", "including pseudo-code", and "revising and rewriting, not patching".

- The meaning of terms like “fault” and “failure” and an understanding of the steps that must take place for a fault to manifest itself in a failure during testing. Additionally, the similarities and differences between the tasks of fault identification and fault removal.
- An understanding of the different types of fault. For instance, you should be able to give examples of faults of commission and omission and explain what type(s) of methods are ideal for identifying these different types of faults. Also, an understanding of orthogonal defect classification and the fault classes that resulted from applying this method at IBM.
- The ways in which testing can take place at different levels of granularity. For instance, you should know the similarities and differences between unit testing and integration testing.
- The relationship between code walk-throughs, inspections, and software testing. In addition, the relationship between (automated and manual) correctness proofs and software testing.
- A detailed understanding of the various test adequacy criteria that software engineers can use to assess the “quality” of a test suite; to ensure that your understanding of this topic is concrete, you should focus on adequacy criteria for procedural and object-oriented languages like C and Java. Also, an understanding of the term “subsumption hierarchy” and the ways in which the conception of subsumption aids the comparison of different test adequacy criteria.
- In addition to an understanding of the terms “static” and “dynamic” in the context of testing, the similarities and differences between static and dynamic analysis techniques used for software testing and debugging. Also, the ability to classify techniques as either static or dynamic (e.g., evolutionary test data generation with EvoSuite is a dynamic method).
- An understanding of the order of deployment and the similarities and differences between different strategies for testing an entire system (e.g., function, performance, acceptance, and installation testing). Also, a knowledge of how the test execution environment (e.g., a test facility or the customer site) plays a role in these approaches to system testing.
- Ways to characterize, measure, and predict reliability, availability, and maintainability. You should understand the equations for the mean time to failure (MTTF), mean time to repair (MTTR) and mean time between failures (MTBF) and know how they help you to calculate the aforementioned attributes of software. Finally, you should intuitively know whether all of these attributes are “higher-is-better” or “lower-is-better” metrics.
- An understanding of what Frederick Brooks, Jr. thinks is the “essence” and “accidents” of building software. A knowledge of what Brooks thinks is the “hard part” of building software.
- The ability to concisely state the “no silver bullet thesis” that Frederick Brooks, Jr. develops in MMM. Also, a knowledge of both some past breakthroughs that solved accidental difficulties (e.g., high-level languages and programming environments) and “hopes for silver” that do not fundamentally improve the practice of software development (e.g., expert systems).
- A clear understanding of the “promising attacks on the conceptual essence” that are not silver bullets, in and of themselves, and yet still hold promise for improving software engineering.
- Lessons learned from working in a team to specify, design, implement, test, document, and release a programming systems product during our laboratory sessions. For instance, an understanding of the technical means by which a program’s code and documentation is released to both Bitbucket and GitHub. In addition, a grasp of how to use GitHub’s issue tracking system to report, triage, and resolve defects and/or concerns about a software system.

Examination Procedures

Minimal partial credit may be awarded for the questions that require a student to write a short answer. You are strongly encouraged to write short, precise, and correct responses to all of the questions. When you are taking the examination, you should do so as a "point maximizer" who first responds to the questions that you are most likely to answer correctly for full points. Please keep the time limitation in mind as you are absolutely required to submit the examination at the end of the period unless you have written permission for extra time from a member of the Learning Commons. Students who do not submit their examination on time will have their overall point total reduced. Please see the course instructor if you have questions about any of these policies.

Review the Honor Code

Students are required to fully adhere to the Honor Code during the completion of this examination. More details about the Allegheny College Honor Code are provided on the syllabus. Students are strongly encouraged to carefully review the full statement of the Honor Code before taking this test.

The following provides you with a review of Honor Code statement from the course syllabus:

The Academic Honor Program that governs the entire academic program at Allegheny College is described in the Allegheny Academic Bulletin. The Honor Program applies to all work that is submitted for academic credit or to meet non-credit requirements for graduation at Allegheny College. This includes all work assigned for this class (e.g., examinations, laboratory assignments, and the final project). All students who have enrolled in the College will work under the Honor Program. Each student who has matriculated at the College has acknowledged the following pledge:

> I hereby recognize and pledge to fulfill my responsibilities, as defined in the Honor Code, and to maintain the integrity of both myself and the College community as a whole.

Strategies for Studying

As you study for this examination, you are encouraged to form study groups with individuals who were previously a member of one of your software development teams during a laboratory session or the final project. You can collaborate with these individuals to ensure that you understand all of the key concepts mentioned on this study guide. Additionally, students are encouraged to create a Slack channel that can host questions and answers that arise as you are studying for the test. Even though the course instructor will try to, whenever possible, answer review questions that students post in this channel, you are strongly encouraged to answer the questions posted by your colleagues as this will also help you to ensure that you fully understand the material.

When studying for the test, don't forget that the Web site for our course contains mobile-ready slides that will provide you with an overview of the key concepts that we discussed in the first module. You can use the color scheme in the slides to notice points where we, for instance, completed an in-class activity, discussed a key point, or made reference to additional details available in the SETP and MMM textbooks. Finally, students are strongly encouraged to schedule a meeting during the course instructor's office hours so that we can resolve any of your questions about the material and ensure that you have the knowledge and skills necessary for doing well on this examination.

Remember, while the test is taken individually, your review for it can be done collaboratively!