COMPSCI 1JC3

Introduction to Computational Thinking

Fall 2017

12 Software Development

William M. Farmer

Department of Computing and Software
McMaster University

November 28, 2017

Admin

- Midterm Test 2 marks.
  - Stage 1 average: 60%.
  - Stage 2 average: 81%
  - Midterm Test 2 average: 63%.
- Final exam will be held on Fri., Dec. 8 at 12:30pm.
  - Review session in class on Mon., Dec. 4.
- Course evaluation.
  - Course discussion session today at 5:30 in BSB B154.
  - CS 1JC3 survey on Avenue until Sun., Dec. 3.
  - Online course evaluations until Thu., Dec. 7.
- Question and answer session on careers in computing on Wed., Dec. 6.
- Office hours: To see me please send me a note with times.
- Are there any questions?

Advice

- Develop a study plan for the final exam!
  - Check the review slides for the lectures and discussion sessions.
  - Schedule time during the exam period for the study plan.
- Keep a portfolio!
  - A portfolio is a collection of your work that you can show to employers (as well as to your family and friends).
  - Put polished versions of your best work into your portfolio.

Review

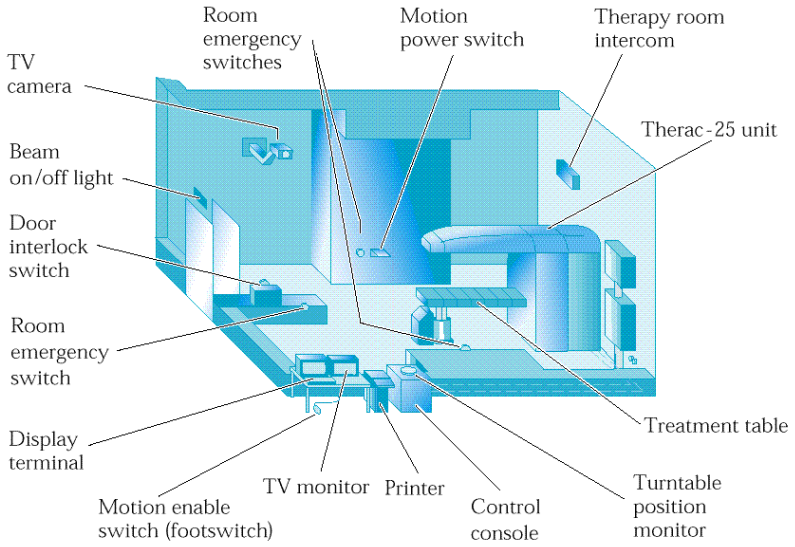
1. Digital images.
2. Color models.
3. Digital audio.
4. Digital video.
5. File formats.
6. Data compression.
7. Data structures.
8. Data bases.

Understanding Computing (iClicker)

How important is it to you to gain a good understanding of computing?

A. Not important at all.
B. Slightly important.
C. Moderately important.
D. Very important.
E. It is a matter of life and death.

Case Study: Therac-25



Therac-25 Overview

- The Therac-25 was a radiation therapy machine for treating cancer.
  - Produced by Atomic Energy of Canada Limited (AECL).
  - Controlled by software.
- In six separate incidents in the 1980s (including one in Hamilton), Therac-25 machines delivered overdoses of radiation causing severe physical damage or even death.
- What went wrong.
  - Software failed to detect that the target was not in place.
  - Software failed to detect that the patient was receiving radiation.
  - Software failed to prevent the patient from receiving an overdose of radiation.
- Click for details.

Therac-25 Causes of Failure

- Inadequate software design.
- Inadequate software development process.
  - Coding and testing done by only one person.
  - No independent review of the computer code.
  - Inadequate documentation of error codes.
  - Poor testing procedures (missed race condition, arithmetic overflow).
  - Poor user interface design (input errors triggered the problem).
- Software was ignored during reliability modeling.
- No hardware interlocks to prevent the delivery of high-energy electron beams when the target was not in place.

Software Development Phases

1. Problem Identification: What is the problem that needs to be solved?
2. Requirements Specification (Problem Definition): What are the product requirements that need to be satisfied? What objectives, functions, and constraints are relevant?
3. Design: How will the problem be solved? How will the product requirements be satisfied?
4. Implementation: What is a solution to the problem? What is an executable implementation of the design?
5. Testing and Verification: What behavior does the product exhibit? Is the behavior correct?
6. Delivery and Maintenance: How will the product be delivered and maintained?

Software Development Models

1. Waterfall: Development follows the logical order of the phases given above in a linear fashion.
   - This model is an idealization of the software development process that is rarely realized.
2. Spiral: The steps of the waterfall model are repeatedly applied until a suitable product is obtained.
3. Refinement (Top-Down): The product requirements are step-wise refined through a series of designs until an implementation of the product is reached.
4. Prototyping: A prototype of the product is developed first and then thrown away.
5. Incremental: A partial product is developed and then incrementally extended until a full product is obtained.
6. Agile: A product evolves through a dialectic between the client and developers.

What is Software Engineering?

- An area of engineering that deals with the development of software products that:
  - Are large or complex.
  - Exist in multiple versions.
  - Exist for large periods of time.
  - Are continuously being modified.
  - Are built by teams.
- Software engineering is the "application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software" (IEEE 1990).
- David Parnas (1978)—the father of the McMaster software engineering program—said it is "multi-person construction of multi-version software".
- Like other areas of engineering, software engineering relies heavily on mathematical techniques (especially logic and discrete mathematics).

Software Engineering Principles

1. Rigor: Reasoning should be precise.
2. Separation of concerns: Different concerns should be isolated and considered separately.
3. Modularity: Complex systems should be divided into smaller parts, called modules.
4. Least Privilege: Each subject should be given the fewest privileges needed for it to perform its task.
5. Formality: Reasoning should be done using a language with a formal syntax and a precise semantics.
6. Abstraction: What is important should be separated out from what is irrelevant.
7. Anticipation of change: Future change should be anticipated and planned for.
8. Generality: Whenever possible, a more general problem should be solved instead of the problem at hand.
9. Incrementality: A problem should be attacked by producing successively closer approximations to a solution.

Separation of Concerns

- Separation of concerns is the principle that different concerns should be isolated and considered separately.
  - The goal is to reduce a complex problem to a set of simpler problems.
  - Enables parallelization of effort.
- Concerns can be separated various ways.
  - Different concerns are considered at different times.
  - Software qualities are considered separately.
  - A software system is considered from different views.
  - Parts of a software system are considered separately.
- Dangers
  - Opportunities for global optimizations may be lost.
  - Some issues cannot be safely isolated (e.g., security).

Modularity

- A modular system is a complex system that is divided into smaller parts called modules.
- Modularity enables the principle of separation of concerns to be applied in two ways:
  1. Different parts of the system are considered separately.
  2. The parts of the system are considered separately from their composition.
- Modular decomposition is the top-down process of dividing a system into modules.
- Modular decomposition is a "divide and conquer" approach.
- Modular composition is the bottom-up process of building a system out of modules.
- Modular composition is an "interchangeable parts" approach.

Modular Units in Haskell (iClicker)

What are used as modules in Haskell?

A. Functions.
B. Types.
C. Modules.
D. All of the above.

Software Testing (iClicker)

What can testing show about a software product?

A. That the product is correct.
B. That the product is reliable.
C. That the product is robust.
D. B and C.
E. All of the above.

Software Testing

- Testing is the most important technique for analyzing the quality of a software product.
- What can be done with testing is limited:
  - It is usually impossible to test every possible input and environmental configuration.
  - Testing can show instances of incorrectness, but it is usually not practical for demonstrating correctness.
  - Positive testing results are not, by themselves, an indication of software quality.
  - The theory of testing leads to many undecidable problems.
- Testing can be used to assess reliability and robustness.

Kinds of Test Case Selection

1. Blackbox: Test cases selected to cover the behavior of the code based on only the specification of the code.
2. Whitebox: Test cases selected to cover the behavior of the code based on the code itself.
3. Statistical random: Test cases selected to measure reliability using an operational profile.
4. Wild random: Test cases selected to measure robustness using a uniform random distribution.

General Testing Recommendations

1. Test the smallest components first (unit testing).
2. Test all possible paths through the program (path coverage and statement coverage).
3. Test all types of data combinations including:
   - Cases along the boundaries.
     - At the boundary.
     - Far from the boundary on either side.
     - Close to the boundary on either side.
   - Extreme cases (like very small and very large numbers).
   - Degenerate cases (such as an empty file).
   - Erroneous cases (such as a name of a nonexisting file).