SE 2AA4, CS 2ME3 (Introduction to Software Development)

Winter 2018

36 Review for Final

Dr. Spencer Smith

Faculty of Engineering, McMaster University

April 9, 2018

36 Review for Final

- Administrative details
- Verifying nonfunctional properties (qualities)
- Topics on the exam
- Structure of the exam
- Advice on exam preparation
  - Time management before the exam
  - Time management during the exam
  - Sample questions
- Advice to improve 2me3/2aa4
- Questions? Feedback? Comments?

Administrative Details

- A4: Due April 9 at 11:59 pm (tonight)
- A3 sample solution pushed to repo
- Grading should be done before exam
- Course evaluations
  - https://evals.mcmaster.ca
  - Start: Tues, Mar 27, 10:00 am
  - Close: Tues, Apr 10, 11:59 pm
  - Your participation is highly valued
  - Grade bonus for class participation
    - CS 2ME3 C02: 70%
    - CS 2ME3 C01: 48%
    - SE 2AA4 C01: 58%

Verifying Performance

How might you measure/assess performance?

Verifying Performance

- Worst case analysis versus average behaviour
- For worst case, focus on proving that the system response time is bounded by some function of the external requests
- Standard deviation
- Analytical versus experimental approaches
- Consider verifying the performance of a pacemaker
- Visualize performance via
  - Identify a measure of performance (time, storage, FLOPS, accuracy, etc.)
  - Identify an independent variable (problem size, number of processors, condition number, etc.)

Verifying Reliability

- There are approaches to measuring reliability on a probabilistic basis, as in other engineering fields
- Unfortunately there are some difficulties with this approach
- Independence of failures does not hold for software
- Reliability is concerned with measuring the probability of the occurrence of failure
- Meaningful parameters include
  - Average total number of failures observed at time t: $AF(t)$
  - Failure intensity: $FI(T) = AF'(t)$
  - Mean time to failure at time t: $MTTF(t) = 1/FI(t)$
- Time in the model can be execution or clock or calendar time

Verifying Subjective Qualities

- What do you think is meant by empirical software engineering?
- What problems might be studied by empirical software engineering?
- Does the usual engineering analogy hold for empirical software engineering?

Verifying Subjective Qualities

- Consider notions like simplicity, reusability, understandability
- Software science (due to Halstead) has been an attempt
- Tries to measure some software qualities, such as abstraction level, effort,
- by measuring some quantities on code, such as
  - η_1, number of distinct operators in the program
  - η_2, number of distinct operands in the program
  - N_1, number of occurrences of operators in the program
  - N_2, number of occurrences of operands in the program
- Extract information from repo, including number of commits, issues etc.
- Empirical software engineering
- Appropriate analogy switches from engineering to medicine

Source Code Metric

- What are the consequences of complex code?
- How might you measure code complexity?

McCabe's Source Code Metric

- Cyclomatic complexity of the control graph
  - $C = e - n + 2p$
  - e is number of edges, n is number of nodes, and p is number of connected components
- McCabe contends that well-structured modules have C in range 3..7, and $C = 10$ is a reasonable upper limit for the complexity of a single module
- Confirmed by empirical evidence

Topics on the Final Exam

- All of them :-)
- From "introduction to software engineering" to "review for final"
- Greater emphasis on the material since the midterm, especially
  - Specification
  - Verification

Types of Questions

- First pages of exam
- Exam material emphasis will be similar to lecture material emphasis
- Multiple choice (30 questions)
  - Some questions will be True/False
  - Similar to midterm
- Short answer (5 questions - 23 marks, 2 mark bonus)
  - Answer in the space provided

Time Management

- Time management before the exam
  - Make a schedule
  - Optimize the reward for spending your time and energy
  - Work smarter not harder
  - Schedule time for rest
- Time management during the exam
  - You have an average of $150/53 = 2.8$ minutes per mark (Midterm was 3.0 minutes per question)
  - Some questions will take longer, some much less time
  - Leave nothing blank
  - No bonus for leaving early
  - Make sure follow scantron instructions
- Eliminate inappropriate options
- Useful information possibly in other questions
- Trust your answer, like you trust a riddle answer

Example

An object's state can be summarized by listing its methods. Is this statement true or false?

A. True.

B. False.

Example

Abstraction is the principle that different concerns should be isolated and considered separately. Is this statement true or false?

A. True.

B. False.

Example

The design of the Python programming language does *not* enforce the principle of information hiding. Is this statement true or false?

A. True.

B. False.

Example

What is the minimum number of test cases for full statement coverage of the following function?

```
def maxOfThreeNum(x1, x2, x3):
    if x1 >= x2:
        if x1 >= x3:
            max = x1
        else:
            max = x3
    else:
        if x2 >= x3:
            max = x2
        else:
            max = x3
    return max
```

Example Answers

A. 1.

B. 2.

C. 3.

D. 4.

E. 5.

Questions?

- Software quality?
- Software principles?
- Module decomposition?
- MIS?
- Parnas tables?
- Fault seeding
- White box testing?
- Analysis?
- ...

How to Improve 2ME3/2AA4?

- Topics to cover/emphasize/de-emphasize?
  - Correctness proof?
  - Design patterns?
  - Functional programming?
  - etc.
- Technology to cover/emphasize/de-emphasize?
  - Doxygen?
  - Continuous integration?
  - Make?
  - Debugging? (Valgrind?)
- Programming language(s)
  - Python?
  - Java?
  - Haskell?
  - C++?

How to Improve 2ME3/2AA4?

- Improving lecture attendance?
- Improving Avenue participation?
  - More questions
  - More people answering questions
- Part 2 of assignments?
- Other thoughts/ideas?