SENG 371 – Software Evolution
Spring 2024
Midterm Exam: Wednesday, 28 February 2024
Instructor: Roberto A. Bittencourt

Students must check the number of pages in this examination paper before beginning to write, and report any discrepancy immediately.

- **All answers are to be written on this exam paper.**
- We will not answer questions during the exam. If you feel there is an error or ambiguity, write your assumption and answer the question based on that assumption.
- The exam is closed book. No books or notes are permitted.
- When answering questions, please do not detach any exam pages!
- ***Electronic devices are not permitted. The only exception is a simple calculator***.
- Cellphones must be turned off.
- The marks assigned to each section are within parentheses. Partial marks are available.
- There are eight (8) printed pages in this document, including this cover page and an answer bubble sheet in the final page.
- Please do NOT remove pages from the exam.
- We strongly recommend you read the entire exam through from beginning to end before starting on your answers.
- **Please have your UVic ID card available on the desk for inspection.**

Section A (40 marks): For each question in this section, **mark only one answer in the answer bubble sheet on the last page of this exam.** Answers marked only on the question pages will not be graded. Each question is worth 4 (four) marks.

Question 1: Mark the only incorrect assertion below about essential difficulties of software.

a) Software ranks among the most complex systems ever created by mankind.

b) Software is invisible in the sense that it is an abstract construct, which may make it harder to understand.

c) Software is easy to change, but changing it correctly is much harder, particularly for larger software.

d) Software technology, such as programming languages, makes communication between developers who are proficient in different technologies harder.

e) Software needs to conform to human institutions and domains, which demands changes when those institutions or domains change.

Question 2: During the software evolution phase of a software life span model, some activities are performed. Mark the one activity that is not usually performed in this phase.

a) New features are added to the software system.

b) Mistakes and misunderstandings are corrected.

c) The software architecture of the system is chosen.

d) The application is adapted to a changing user and operating environment.

e) Code is refactored to make changes easier and improve design quality.

Question 3: According to Rajlich's software change model, it is incorrect to say that:

a) Concepts are extracted from a change request (or issue).

b) Software verification is only performed after software actualization.

c) Impact analysis starts from the classes/files found during concept location.

d) Refactoring may be performed either before or after actualization.

e) Change conclusion usually requires committing code into version control.

Question 4: Mark the correct assertion about impact analysis.

a) It is the first step performed in a software change.

b) It is usually performed by means of a GREP search.

c) It is not useful to estimate the effort for making a software change.

d) It chooses the requirement from the backlog with the least impact.

e) It may use a class interaction graph built from dependencies and coordinations.

Question 5: Mark the correct answer about what usually happens during software actualization.

a) Programmers estimate the size of a change.

b) Programmers commit code into the version control.

c) Programmers pick a change request (or issue) from the product backlog.

d) Programmers change the software architecture of the system.

e) Programmers implement a new functionality or fix a bug.

Question 6: Mark the correct assertion about software verification.

a) Verification can be done only via software testing.

b) Code reviews are one particular type of software verification.

c) Tests cannot identify regression of what was already functioning.

d) Testing coverage is measured by unit testing frameworks such as JUnit.

e) Testing can demonstrate the absence of bugs.

Question 7: Mark the correct assertion about code refactoring.

a) Refactoring changes source code by preserving its external behavior.

b) Refactoring can only be performed after actualization.

c) Refactoring entails only small localized changes in either one or very few classes.

d) Refactoring is a lightweight bug fixing technique.

e) Refactoring is not a popular practice in the context of agile processes.

Question 8: Bad code smells are common in source code and may point out refactoring opportunities. Mark the only wrong answer about bad code smells.

a) Bad code smells explain how a refactoring should be performed.

b) Bad code smells allow developers to identify what needs to be changed to improve the source code.

c) Bad code smells offer no precise criteria for refactoring, instead providing an intuition or indication of an improvement needed.

d) Bad code smells may indicate that a code is not habitable, i.e. the code is not easy to read or change.

e) Bad code smells are described in catalogs, usually together with recipes to improve the code by means of appropriate refactoring patterns.

Question 9: Mark the only wrong assertion about challenges for data mining in software engineering.

a) Some mining requirements are unique to software engineering.

b) Software data may be complex, correlated and linked.

c) Traditional machine learning algorithms cannot be used in the software engineering context.

d) Software data may be of large scale.

e) Mining results sometimes needs to be produced just-in-time, as the software project evolves.

Question 10: About the JUnit unit testing framework, mark the option that is not typical of JUnit test cases.

a) Providing a fixture of the objects that will interact during the test.

b) Adding watches to check the value of variables inside loops.

c) Verifying the results by means of assertions.

d) Exercising the test fixture.

e) Creating one test method for each particular production class method.

Section B (60 marks): Three questions follow, each one worth 20 (twenty) marks. When answering those questions, you may use your own opinions, as long as they are grounded on technical or scientific knowledge.

Question 11: (20 marks). Compare the two main concept location techniques (GREP versus dependency search) in terms of their advantages and disadvantages. Which scenarios are better for each technique?

GREP advantages:

- **independent of code structure**
- **independent of programming language**
- **quick and easy to use**

GREP disadvantages:

- **does not find implicit concepts**
- **sometimes returns too many results**
- **depends on the use of naming conventions**

Dependency search advantages:

- **suitable for both explicit and implicit concepts**
- **uses the code structure as a map**
- **May use both class dependency graphs and UML**

Dependency search disadvantages:

- **requires correct understanding of both local and combined responsabilities**
- **May be time-consuming until it finds the concept**

A general rubric would assign 5 marks for each of the categories above. Answers may bring different advantages and disadvantages than the ones here.

Some answers may mix the advantages of one technique with the disadvantages of the other technique. Use 10 marks for each of the two categories in this case.

Full marks should only be granted to answers with at least two advantages for each technique.

You may use the space below for scratch work.

Question 12: (20 marks). State the main reasons why developers refactor source code. Also, explain when developers should refactor code and when they should not.

Reasons (10 marks):

- **To improve software design**
- **To counter code decay (software aging)**
- **To increase software comprehensibility**
- **To find bugs and make code more robust**
- **To increase productivity (code faster on a long-term basis)**
- **To reduce costs of software maintenance**
- **To reduce testing**
- **To prepare for / facilitate future customizations**
- **To turn an OO application into a framework**
- **To introduce design patterns in a behavior-preserving way**

When developers should refactor (5 marks):

- **Whenever they see the need for it**
- **Not on a pre-set periodical basis**
- **When they would duplicate something for the third time**
- **When adding new features**
- **During bug fixing**
- **During code reviews**

When developers should not refactor (5 marks):

- **When it would be easier to rewrite everything from scratch**
- **When they are too close to a deadline**

At least four reasons should be stated for 10 marks. At least two reasons for 5 marks in each of the other categories.

You may use the space below for scratch work.

Question 13: (20 marks). 5. Compare a simple staged model with a versioned staged model of software life span in terms of advantages and disadvantages.

The simple staged model has the following advantages over the versioned staged model:

- **Simpler version control setup**
- **Keeping only one main development branch**
- **Developers know exactly what stage the software system is**
- **Easier to handle a smaller consumer base**

The versioned staged model has the following advantages over the simple staged model:

- **Easier to deal with a larger consumer base**
- **Different versions may be used for different life span stages of the software system**
- **Customers may choose to either upgrade immediately to the new version or to retain an older version**
- **Easier to achieve faster release times**

Use 10 marks for each category above. Full marks in each category should present at least two advantages.

You may use the space below for scratch work.