SENG 371 – Software Evolution
Prof. Roberto A. Bittencourt
Final Exam Review

Exercises

1. Distinguish essential from accidental difficulties of software. Use examples to illustrate each class of difficulty.
2. What class of difficulty (essential or accidental) do iterative and incremental software processes attack? Explain how they do it.
3. Give two examples of accidental complexity reduction in software development, and explain why they do not attack essential complexity of software.
4. David Parnas suggests that software aging (code decay) can be prevented by designing for change, using ideas such as abstraction, information hiding and modularization. However, this idea was just partially successful in software development. Explain why.
5. Compare a simple staged model with a versioned staged model in terms of advantages and disadvantages.
6. Describe the different steps of the staged model of software life span (initial development, evolution, servicing, phase-out and closedown), noting the reasons why the software management team decides to move to each of those stages.
7. Compare Rajlich's software change process with the test-driven development process to perform changes in software systems in terms of advantages and disadvantages.
8. What are the pros and cons of using branches to implement software changes as suggested by GitHub?
9. What are the pros and cons of using pull requests instead of committing changes straight to the trunk of a software version control repository?
10. In your opinion, can developers choose not to perform concept location before implementing an issue? Explain why or why not.
11. 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?
12. Suppose you perform concept location by GREP to find an existing feature in a software system, and your search fails. Which solutions would you choose to continue with the concept location process?
13. In your opinion, can developers choose not to perform impact analysis before implementing an issue? Explain why or why not.
14. Which diagrams would be useful to aid the process of impact analysis before performing a change to a software system? Explain why.

15. Which type of data do you think would be useful to collect and submit to data mining algorithms in order to improve impact analysis that is made via GREP and dependency search? Explain why those data would be useful in your opinion.

16. Explain why software companies that use impact analysis in their change process use the metrics of precision and recall to assess the quality of the impact analysis performed? Explain the reason for each metric separately.

17. An impact analysis task suggested that 5 out of 100 classes should be changed after a change request was implemented. However, after actualization was performed, only 2 out of those five classes were changed; also, developers noticed that two other different classes were changed that had not been predicted to change during impact analysis. Compute precision and recall of this particular impact analysis, showing your computations based on formulas that use confusion matrix elements, such as false negatives (FN), false positives (FP), true negatives (TN) and true positives (TP).

18. Describe the different ways that change incorporation happens in object-oriented systems through each of the following situations: via polymorphism, adding a new supplier, and adding a new client.

19. Explain the reasons why developers refactor code: a) before a change is performed; b) after a change is performed.

20. State the main reasons why developers refactor code. Also, explain, in your opinion, when should they refactor code and when they should not.

21. Some researchers described the following principles for good refactoring tools: "The tool should let the programmer: a) choose the desired refactoring quickly; b) switch seamlessly between program editing and refactoring; c) view and navigate the program code while using the tool; d) avoid providing explicit configuration information; e) access all the other tools normally available in the development environment while using the refactoring tool." Explain the ways that IDEs (if you prefer, pick one such as Eclipse or VS Code) abide or not by those principles.

22. Explain the different strengths and weaknesses of software testing compared to code reviews in terms of what each can achieve or not when trying to find bugs and maintain software quality.

23. Explain the differences between unit tests, functional tests and structural tests, and describe the scenarios during software evolution each type is usually employed.

24. In your opinion, what are the pros and cons of using a testing framework such as JUnit during software evolution?

25. Describe the main differences between the Cathedral (closed source) and Bazaar (open source) practices of software development.

26. Newcomers face barriers when trying to contribute to open source projects in terms of reception such as not receiving an answer, delayed answers, impolite answers, and receiving answers with too advanced/complex contents. In your opinion, are newcomers reasonable to state those barriers? Why? What could open source teams do to remove those barriers?

27. Newcomers face barriers when trying to contribute to open source projects because of their own communication weaknesses such as not sending a meaningful/correct message, their mastering level of the English language; their shyness, making useless comments in the mailing list/forum, low responsiveness, and not acknowledging/thanking answers. In your opinion, how could they work to

reduce those barriers during their university education? And what could open source teams do to reduce such communication barriers?

28. Practitioners and researchers point out various benefits of using software models, particularly with the use of modeling languages such as UML. Describe at least four benefits of using software models in the process of software development.
29. Practitioners and researchers point out various disadvantages of using UML in software development. Describe at least four disadvantages of using UML from the point of view of practitioners.
30. Software developers have different views about the different UML diagrams: some of them (e.g., class, activity and sequence diagrams) are popular and useful in their practice, others are considered less useful or even useless. Explain the reasons why these differences in adoption happened in software industry.
31. Reverse engineering of structural architecture and design views usually consist of at least three different phases: fact extraction, high-level abstraction and high-level viewing. Describe which typical activities happen during each of those phases.
32. Some tools for software architecture and design recovery such as Structure101 or SonarGraph are used by part of software industry. Describe the pros of those tools that lead to their adoption in practice.
33. Some tools for software architecture and design recovery such as Structure101 or SonarGraph are ignored by part of software industry. Describe the cons of those tools that prevent their adoption in practice.
34. Compare the different techniques of software architecture checking (Reflexion Models, Design Structure Matrices and Architectural Constraint Languages) in terms of expressiveness, abstraction level and architecture reasoning/discovery.
35. Describe the advantages of using software architecture checking techniques in the context of large-scale software systems.
36. Describe the disadvantages of using software architecture checking techniques in the context of large-scale software systems.
37. Compare centralized (e.g., Subversion) and distributed (e.g., git) version control systems in terms of their advantages and disadvantages.
38. What are the pros and the cons of the central repository workflow as a strategy of version control?
39. What are the pros and the cons of the integration manager workflow as a strategy of version control?
40. Describe the advantages of using DevOps practices in software development.
41. Describe the disadvantages of using DevOps practices in software development.
42. Explain the different factors that lead to successful adoption of DevOps and their role to that success.
43. Software developers may benefit from developing a mental model during software comprehension. Briefly explain the static elements (text-structures, chunks, schemas/plans and hypotheses) of those mental models and describe how each of them helps comprehend a software system.

44. Software developers may benefit from developing a mental model during software comprehension. Briefly explain the dynamic elements (chunking, cross-referencing and strategies) of those mental models and describe how each of them helps comprehend a software system.

45. Describe the pros and cons of the Fritz and Murphy's Information Fragments model.

46. Explain at least three reasons why some software systems reach the end of software evolution and the software team decides to move them into the servicing phase.

47. Explain the reasons why a software team decides to move a system from servicing to phase-out and close-down.

48. Software reengineering usually consist of at least three different phases: reverse engineering, redesign and forward engineering. Describe which typical activities happen during each of those phases.

49. From your knowledge of Lehman's laws of software evolution, describe the main lessons those laws teach about what usually happens to evolving software systems.

50. Describe at least three main recommendations to prevent or postpone issues that evolving software systems suffer according to Lehman's laws of software evolution.

51. After 1997, different studies questioned the validity of Lehman's laws of software evolution. Explain how different software development practices affected the assumptions under which such laws relied on.