Exam 1 – Fall 2010 – Solution Notes

1. As new SE techniques help us to build larger, more complex systems, demands for faster development, even more complexity, and new capabilities previously thought to be impossible. As a result, new software engineering methods must continuously be developed to meet ever increasing demands.

2.

3.

4.

5. requirements engineering (RE)


7. b

8. c

9. a. It is sometimes argued that agile methods require higher skill levels than plan based approaches (in which programmers simply translate a detailed design into code).

b. If a system has to be approved by an external regulator (e.g., the FAA approves software that is critical to the operation of an aircraft) then you will probably be required (in keeping with a plan-based development approach) to produce detailed documentation (e.g., as part of a system safety case).

10. “Scaling-up:” using agile methods for developing large software systems that cannot be developed by a small, agile methods development team.

“Scaling-out:” how agile methods can be introduced across a large organization (with years of traditional software development experience).

11. true, false, false, true, false, false, true, false

12. **Stakeholder communication**: the architecture may be used as a *focus of discussion* by system stakeholders.

**System analysis**: the *feasibility* of meeting critical non-functional requirements (e.g., performance, reliability, maintainability constraints) can be studied early-on.

**Large-scale reuse**: the *architecture may be reusable* across a range of systems with similar requirements.


14. a. A *social scientist* observes and analyzes how people actually work. Subjects do not have to explain or otherwise articulate what they do. Social and organizational factors of importance to RE (that may not otherwise be apparent) may be observed.

b. Focused ethnography combines ethnography with prototyping, which focuses the ethnographic analysis on issues relevant to the new system. In contrast, ethnography alone studies existing practices which may not be relevant when the new system is put into place.

15. Identify potentially changeable design decisions and isolate (i.e., “hide”) these in separate objects to minimize the impact of change.

16. b

17. e

18. **Problem tracking**, which allows reporting of bugs and other problems, and allows all developers to see who is working on these problems and when they are fixed.
19. (1) Users of BSD open source code are **not** required to re-publish any changes or modifications made to the code, and (2) **are allowed** to include the code in proprietary systems that they sell.

20. During *evolution*, the system is in operational use and is evolving as new requirements are proposed and implemented. During *servicing*, the system remains useful, but the only changes made are those required to keep it operational (i.e., bug fixes and changes to reflect changes in the software’s environment). No new functionality is added.

21. **Goals**: The basic goal of *validation testing* is to demonstrate that software meets its requirements — i.e., to demonstrate that a program does what it is intended to do. The basic goal of *defect testing* is to discover defects before the program is put into use.

**General nature of test cases employed**: in *validation testing*, a set of test cases are employed that reflect the system’s expected use. In *defect testing*, test cases can be deliberately obscure and need not reflect how the system is normally used.

**What constitutes a “successful test”**: in *validation testing*, a “successful test” shows that the system operates as intended. In *defect testing*, a “successful test” reveals a program defect.

22. **Speculative generality** refers to overly general (and therefore unnecessarily complex) programs that were produced in the belief that the additional flexibility afforded by the generality might be needed in the future. Such programs can often be simplified.

23. A critical difference is that the first stage may involve program understanding, especially if the original system developers are not responsible for the change implementation. (This involves understanding how the program is structured, how it delivers functionality, and how the proposed change might affect the program.)

24. true, true, true, true

25. a

26.