SE exam: 2019 autumn semester

Questions and some of the expected (or accepted) answers

1. (1) List at least 4 different types of stakeholders (i.e. interested parties) of a typical software project.
   0.25 for each listed:
   a. **Project manager**
   b. **Project team members (programmers, testers, designers, etc.)**
   c. **Client (who pays for the project)**
   d. **Users**
   e. **Executives (CEO/management of the company)**
   f. **Shareholders (investors in the company)**
   g. **Subcontractors**
   h. **etc.**
2. (2) If things rarely go according to plan (i.e. plans change often), why is planning (and having a plan) important? List at least 2 reasons.
   1 for each listed:
   a. **Helps communicate goals and objectives**
   b. **Helps judge and communicate progress**
   c. **Increases chances of success by considering multiple alternatives, and selecting the best**
   d. **Increases chances of success by identifying and clarifying the unknowns**
   e. **Increases chances of success by identifying dependencies (e.g. task B can only start after task A is done)**
   f. **Helps accountability**
   g. **Helps productivity**
3. (1) Which 3 factors does a project manager have to balance (called *the triple constraint*; often drawn as a triangle; where you can pick two)?
   a. **Scope, cost, time (1)**
   b. **Quality, cost, time (0.75)**
   c. **Getting 2 out of 3 right (0.5)**
4. (2) Being able to identify risks is an important skill to have. Let's say you are answering another question in this exam. You write the answer and hand in your exam. List at least 4 risks that could prevent you from getting points (either for that answer, or at all). List as different risks as possible (not variations of the same). Risks can be unlikely.
   No risk that was related to preparation (e.g. being sick) or actual writing of the answer (e.g. pen breaking) was accepted. 0.5 for each listed. Some examples:
   a. **Wrong answer**
   b. **Illegible answer**
   c. **Forgot to write name on paper**
   d. **Wrote wrong question number next to answer**
   e. **Cheating**
   f. **Lecturer's dog eats your exam**
   g. **Lecturer spills coffee on your exam**
   h. **University is closed permanently**
   i. **I am dreaming and actually never attended the exam**
   j. **Exam is handed in after the time has run out**
5. (1) Over the lifetime of an average successful software project, roughly what percentage of cost is spent during maintenance phase? Answer with a number and explain why in one sentence.
   Some sources estimate it around 90%, others - between 60-80%.
   The percentage is high, because successful software tends to be used for much longer than the time it takes to build it originally. Also, if the software is useful, users will want it to do more, therefore requesting new features, which are estimated to be around 80% of all changes implemented.
   (1) Anything between 60-80% and convincing explanation
   (0.5) 50-60% and convincing explanation
   (0) Anything less than 50% and/or explanation is not convincing

6. The waterfall software development lifecycle (SDLC) model is a linear sequence of activities that are performed as part of a project. The result of each activity is a document or an artifact that acts as an input into the next activity.
   a. (2) If a project is run using the waterfall SDLC model, what negative effects and/or risks it is likely to experience? Name at least 2 and briefly explain why for each one of them.
      Some of the possible answers (1 point each):

      b. **Testing is performed late - if testing uncovers flaws in specification/design, changes will likely be expensive**
      c. **Feedback is received later - customers/users often do not know what they want exactly and when they get working software, it might not be what they expected (even if they agreed to such specification)**
      d. **(Other answers about the high cost of late changes)**
      e. **Software creates value only at the end of the project - longer return on investment; nothing useful to the users if project is cancelled mid-way**
      f. **Administrative overhead for small projects - detailed planning and documentation take time**
      g. **Requires larger numbers of specialized staff - e.g. during analysis you primarily need BAs, during testing - primarily QA specialists, while in a cross-functional team working in small iterations you can have the same people working during the whole project.**
   h. (3) What alternative SDLC model could a project adopt to avoid all of these negative effects? Either:
      - modify the waterfall model, explain the modification and why it would help, or
      - suggest a different SDLC model, name the main differences between the models and explain how they help avoid the negative effects.
      Alternative model is named and it would solve the expected problems/mitigate expected risks (+1 point) and it is explained how/why (+1 point for correct, but incomplete, +2 for correct and complete explanation).
      Various models and explanations were possible, depending on the flaws named in previous part.
7. There are two categories of requirements - functional requirements and non-functional requirements (also known as quality attributes). Recall there are many types of non-functional requirements (e.g. interoperability, recoverability, etc.). Now recall your group's project and imagine that a client is requesting to extend/improve your system.
   a. (1) List at least 2 new potential functional requirements
      Many different answers. 0.5 points for each requirement, as long as it is a functional requirement.
   b. (3) List at least 3 types of non-functional requirements, with 1 potential requirement under each type
      Many different answers. 1 point for each (0.5 for correctly named type and 0.5 for requirement itself). Note: no credit was given for just "interoperability" and "recoverability" if no requirement was provided.
8. (3) Recall that a model represents a simplified view of some aspect of reality. Give an example (not necessarily from the software industry) of a problem, which can be solved easily using a model, but is very difficult (or even impossible) to solve otherwise. Describe how the model makes it easier to solve.
   +2 points for basically any problem that involves future prediction, estimation, or design, and +1 point for an explanation why. Some examples:
      a. **Building a sewage system for a city**
      b. **Determining how many calories you need to eat**
      c. **Predicting movement of celestial bodies**
      d. **Building a house**
      e. **Wiring electrical devices**
      f. **Public transport schedules**
      g. **Explaining what the UI of a new system should be**
9. It is claimed that the software in an average modern car has around 100M lines of code. Larger and larger software systems are being built, way beyond what a single developer could build in one's lifetime (not to speak of keeping all the code in one's mind!). Despite the limitations of the human mind, such large software is everywhere.
   a. (2) Name and briefly explain the primary strategy/approach used to deal with complexity in large software
      i. **Modularity or decomposition (2)**
      ii. **Something to that effect, e.g. divide and conquer, division of labour, separation of responsibilities, or similar (2)**
   b. (3) List at least 3 different benefits which that strategy/approach provides
      Note: only benefits of modularity were accepted. 1 point for each. Some examples:
      i. **Parallelization of work**
      ii. **Improved product flexibility**
      iii. **Comprehensability**

   iv. **Easier estimation (WBS)**
   v. **Easier staffing and better quality due to specialization**

10. Assume two similar companies each have a software system with the same functionality. One has an old one that has evolved over 10 years, and another has a new one that was written last year. Both companies want to add the same feature to their system.
    a. (1) Make an educated guess - which system would likely take more effort to modify and why?
    1 point if the system is named and argument is convincing.
       i. **Older one, because of technical debt accumulated over time (expected)**
       ii. **Newer one, because it might have been rushed, or parts of it can still be unstable or have bugs (also accepted)**
    b. (2) What would you recommend to the owners of said system to make it easier to modify in the future? Make at least 4 suggestions, briefly explaining how each would help.
    0.5 point for each suggestion that makes the system easier to maintain in the future (+0.25 for suggestion and +0.25 for explanation). Some examples:
       i. **Prioritize technical debt removal**
       ii. **Minimize new technical debt**
          1. **Unit tests**
          2. **Refactoring**
          3. **Code reviews**
          4. **Upskilling staff**
          5. **Improve documentation**
       iii. **Simplify the system**
          1. **Remove rarely used features**
          2. **Replace parts of the system with off-the-shelf components/subsystems**
       iv. **Improve shared understanding of the system**

11. Assume you are designing functionality for money transfer between bank accounts, that another developer will implement using an object-oriented language. Each bank account holds a balance, which is an amount of money. Money is expressed as a pair of numerical value and a currency.
    a. (3) List the types of domain objects (i.e. classes) that you will need, and their responsibilities (behaviour). Recall that data should be encapsulated within objects (accessed only from within that object).
    Expected:
    - BankAccount (id, balance) - transfers given monetary amount to given account
    - MonetaryAmount (value, currency) - adds another to itself, subtracts another to itself
    Grading:
       i. **BankAccount, MonetaryAmount types are named: +0.5 point**
       ii. **No other types are named: +0.5 point**
       iii. **Description assigns responsibilities (behaviour): +1 point**
       iv. **There are signs that encapsulation is maintained: +1 point**
    b. (3) Draw a UML class diagram showing types of domain objects described before. The diagram should be detailed and include: attributes/fields (names, types, visibility), methods (names, arguments and their types, return types, visibility), associations (type and multiplicity)
    Grading:
       i. **Diagram displays classes listed in part A: +0.5 point**
       ii. **Attributes listed properly: +0.5 point**
       iii. **Methods listed properly: +1 point**
       iv. **Attribute and method visibility documented: +0.5 points**
       v. **Associations drawn properly: +0.5 point**
    c. (1) Write a few lines of code that demonstrate how a client will use the classes in your diagram (i.e. what method(s) on what objects and with what parameters will they have to call for the money transfer to happen). Note: you might want to do this before drawing the diagram.
    Expected: account1.transfer(new MonetaryAmount(100, “USD”), account2);
    Example illustrates how the designed classes would be used (1). Partial credit for cases when design flaws are obvious in the example.

12. Assume a colleague has written a function: *int sum(int a, int b)* and you have to test it. Obviously, you will not test it with every combination of *a* and *b*, but you want to have a high degree of confidence that the function works correctly.
    a. (2) Make a list of pairs of *a* and *b* values that you will use as inputs in your test cases.
    Expected:

- All combinations of positive/zero/negative values (positive and positive, positive and zero, etc.)
- Passing incorrect types
- Integer overflow
Grading:

i. **+0.25 for each distinct combination (e.g. positive and positive)**
ii. **+0.25 for illegal values (nulls, non-int types)**
iii. **+0.25 for integer overflow (large values)**

b. (1) What test case design heuristic did you use? (I.e. using what strategy did you choose the inputs?)
Expected: Equivalence groups or boundary conditions (1).
Partial credit for various similar answers in the same spirit.

13. (3) Recall that a UML deployment diagram shows how your artifacts are deployed on nodes (including e.g. physical hardware, operating systems, execution environments, etc.). Draw a detailed deployment diagram of your group's project.
Every node in the diagram was graded separately and results were added up proportionally to total 3 points.
For every node, this was evaluated:
    a. **Physical hardware nodes are correct (+0.5 / node count)**
    b. **OS nodes are correct (+0.5 / node count)**
    c. **Execution environments are correct (+0.5 / node count)**
    d. **Artifacts are correct (+0.5 / node count)**
    e. **Connections and protocols named correctly (+0.5 / node count)**
    f. **Overall correctness (+0.5 / node count)**