1. (1) List at least 4 different types of stakeholders (i.e. interested parties) of a typical software project.
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.
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)?
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.
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.
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.
   b. (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.
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
   b. (3) List at least 3 types of non-functional requirements, with 1 potential requirement under each type
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.
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
   b. (3) List at least 3 different benefits which that strategy/approach provides
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?
    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.
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).
    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)
    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.
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.
    b. (1) What test case design heuristic did you use? (I.e. using what strategy did you choose the inputs?)
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.