2 Software Processes

2.1 Giving reasons for your answer based on the type of system being developed, suggest the most appropriate generic software process model that might be used as a basis for managing the development of the following systems:

- A system to control anti-lock braking in a car
- A virtual reality system to support software maintenance
- A university accounting system that replaces an existing system
- An interactive travel planning system that helps users plan journeys with the lowest environmental impact

---

1. *Anti-lock braking system* This is a safety-critical system so requires a lot of up-front analysis before implementation. It certainly needs a plan-driven approach to development with the requirements carefully analysed. A waterfall model is therefore the most appropriate approach to use, perhaps with formal transformations between the different development stages.
2. *Virtual reality system* This is a system where the requirements will change and there will be an extensive user interface components. Incremental development with, perhaps, some UI prototyping is the most appropriate model. An agile process may be used.
3. *University accounting system* This is a system whose requirements are fairly well-known and which will be used in an environment in conjunction with lots of other systems such as a research grant management system. Therefore, a reuse-based approach is likely to be appropriate for this.
4. *Interactive travel planning system* System with a complex user interface but which must be stable and reliable. An incremental development approach is the most appropriate as the system requirements will change as real user experience with the system is gained.

2.3 Consider the reuse-based process model shown in Figure 2.3. Explain why it is essential to have two separate requirements engineering activities in the process.

In a reuse based process, you need two requirements engineering activities because it is essential to adapt the system requirements according to the capabilities of the system/components to be reused. These activities are:

1. An initial activity where you understand the function of the system and set out broad requirements for what the system should do. These should be expressed in sufficient detail that you can use them as a basis for deciding of a system/component satisfies some of the requirements and so can be reused.
2. Once systems/components have been selected, you need a more detailed requirements engineering activity to check that the features of the reused software meet the business needs and to identify changes and additions that are required.

2.4 Suggest why it is important to make a distinction between developing the user requirements and developing system requirements in the requirements engineering process.

There is a fundamental difference between the user and the system requirements that mean they should be considered separately.

1. The user requirements are intended to describe the system's functions and features from a user perspective and it is essential that users understand these requirements. They should be expressed in natural language and may not be expressed in great detail, to allow some implementation flexibility. The people involved in the process must be able to understand the user's environment and application domain.
2. The system requirements are much more detailed than the user requirements and are intended to be a precise specification of the system that may be part of a system contract. They may also be used in situations where development is outsourced and the development team need a complete specification of what should be developed. The system requirements are developed after user requirements have been established.

2.6 Explain why change is inevitable in complex systems and give examples (apart from prototyping and incremental delivery) of software process activities that help predict changes and make the software being developed more resilient to change.

Systems must change because as they are installed in an environment the environment adapts to them and this adaptation naturally generates new/different

system requirements. Furthermore, the system's environment is dynamic and constantly generates new requirements as a consequence of changes to the business, business goals and business policies. Unless the system is adapted to reflect these requirements, its facilities will become out-of-step with the facilities needed to support the business and, hence, it will become less useful.

Examples of process activities that support change are:

1. Recording of requirements rationale so that the reason why a requirement is included is known. This helps with future change.
2. Requirements traceability that shows dependencies between requirements and between the requirements and the design/code of the system.
3. Design modeling where the design model documents the structure of the software.
4. Code refactoring that improves code quality and so makes it more amenable to change.

2.9 What are the advantages of providing static and dynamic views of the software process as in the Rational Unified Process?

An approach to process modeling which is simply based on static activities, such as requirements, implementation, etc. forces these activities to be set out in a sequence which may not reflect the actual way that these are enacted in any one organization. In most cases, the static activities shown in Figure 2.13 are actually interleaved so a sequential process model does not accurately describe the process used. By separating these from the dynamic perspective i.e. the phases of development, you can then discuss how each of these static activities may be used at each phase of the process. Furthermore, some of the activities that are required during some of the system phases are in addition to the central static activities shown in Figure 2.13. These vary from one organization to another and it is not appropriate to impose a particular process in the model.

3 Agile Software Development

3.2 Explain how the principles underlying agile methods lead to the accelerated development and deployment of software.

The principles underlying agile development are:

1. *Individual and interactions over processes and tools*. By taking advantages of individual skills and ability and by ensuring that the development team know what each other are doing, the overheads of formal communication and process assurance are avoided. This means that the team can focus on the development of working software.
2. *Working software over comprehensive documentation*. This contributes to accelerated development because time is not spent developing, checking and managing documentation. Rather, the programmer's time is focused on the development and testing of code.
3. *Customer collaboration over contract negotiation*. Rather than spending time developing, analyzing and negotiating requirements to be included in a system contract, agile developers argue that it is more effective to get feedback from customer's directly during the development about what is required. This allows useful functionality to be developed and delivered earlier than would be possible if contracts were required.
4. *Responding to change over following a plan*. Agile developers argue (rightly) that being responsive to change is more effective than following a plan-based process because change is inevitable whatever process is used. There is significant overhead in changing plans to accommodate change and the inflexibility of a plan means that work may be done that is later discarded.