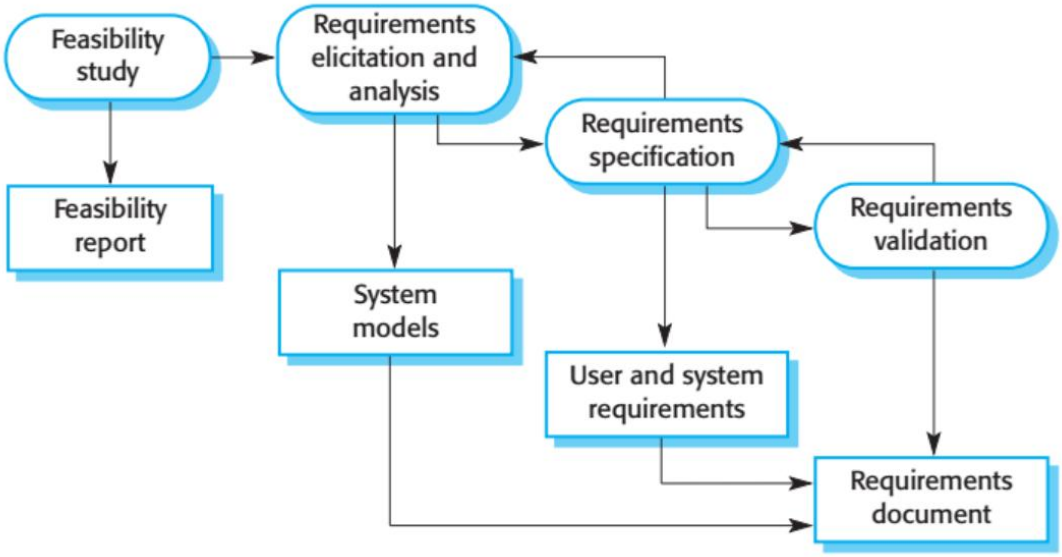
WEEK 3: SOFTWARE PROCESS ACTIVITIES

SOFTWARE SPECIFICATION

Software specification or requirements engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development. Requirements engineering is a particularly critical stage of the software process as errors at this stage inevitably lead to later problems in the system design and implementation.

The requirements engineering process is shown in the figure below. This process leads to the production of a requirements document that is the specification for the system. Requirements are usually presented at two levels of detail in this document. End-users and customers need a high-level statement of the requirements; system developers need a more detailed system specification.



There are four main phases in the requirements engineering process:

1) **Feasibility study.** An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies. The study considers whether the proposed system will be cost-effective from a business point of view and whether it can be developed within existing budgetary constraints. A feasibility study should be relatively cheap and quick. The result should inform the decision of whether to go ahead with a more detailed analysis.

2) **Requirements elicitation and analysis.** This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis and so on. This may involve the development of one or more system models and prototypes. These help the analyst understand the system to be specified.

3) **Requirements specification.** The activity of translating the information gathered during the analysis activity into a document that defines a set of requirements. Two types of requirements may be included in this document. *User requirements* are abstract statements of the system requirements for the customer and end-user of the system; *system requirements* are a more detailed description of the functionality to be provided.

4) **Requirements validation.** This activity checks the requirements for realism, consistency and completeness. During this process, errors in the requirements

document are inevitably discovered. It must then be modified to correct these problems.

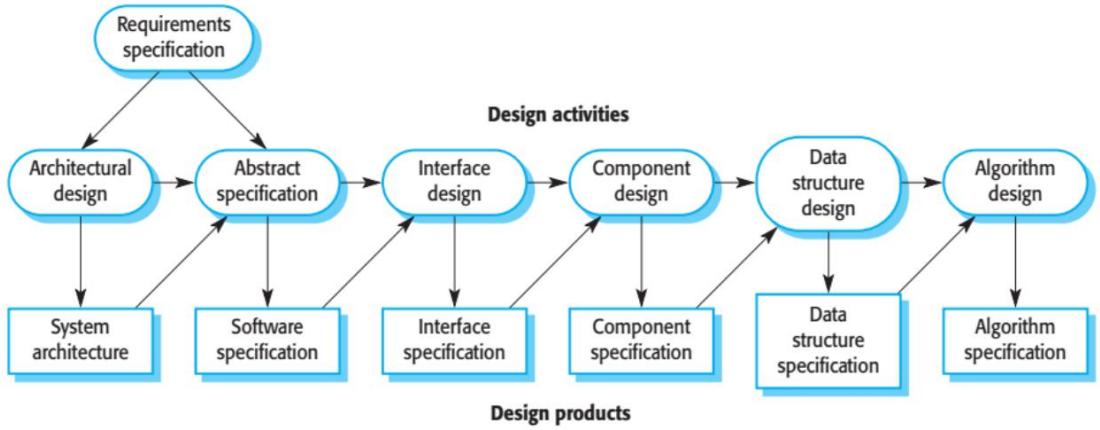
Of course, the activities in the requirements process are not simply carried out in a strict sequence. Requirements analysis continues during definition and specification, and new requirements come to light throughout the process. Therefore, the activities of analysis, definition and specification are interleaved. In agile methods such as extreme programming, requirements are developed incrementally according to user priorities, and the elicitation of requirements comes from users who are part of the development team.

SOFTWARE DESIGN AND IMPLEMENTATION

The implementation stage of software development is the process of converting a system specification into an executable system. It always involves processes of software design and programming but, if an evolutionary approach to development is used, may also involve refinement of the software specification.

A software design is a description of the structure of the software to be implemented, the data which is part of the system, the interfaces between system components and, sometimes, the algorithms used. Designers do not arrive at a finished design immediately but develop the design iteratively through a number of versions. The design process involves adding formality and detail as the design is developed with constant backtracking to correct earlier designs.

The design process may involve developing several models of the system at different levels of abstraction. As a design is decomposed, errors and omissions in earlier stages are discovered. These feedback to allow earlier design models to be improved. In the figure is a model of this process showing the design descriptions that may be produced at various stages of design. This diagram suggests that the stages of the design process are sequential. In fact, design process activities are interleaved. Feedback from one stage to another and consequent design rework is inevitable in all design processes.



A specification for the next stage is the output of each design activity. This specification may be an abstract, formal specification that is produced to clarify the requirements, or it may be a specification of how part of the system is to be realized. As the design process continues, these specifications become more detailed. The final results of the process are precise specifications of the algorithms and data structures to be implemented.

The specific design process activities are:

1) **Architectural design.** The sub-systems making up the system and their relationships are identified and documented.

2) **Abstract specification.** For each sub-system, an abstract specification of its services and the constraints under which it must operate is produced.

3) **Interface design.** For each sub-system, its interface with other sub-systems is designed and documented. This interface specification must be unambiguous as it allows the sub-system to be used without knowledge of the sub-system operation.

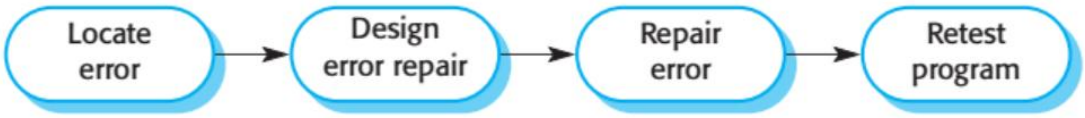
4) **Component design.** Services are allocated to components and the interfaces of these components are designed.

5) **Data structure design**. The data structures used in the system implementation are designed in detail and specified.

6) **Algorithm design**. The algorithms used to provide services are designed in detail and specified.

Normally, programmers carry out some testing of the code they have developed. This often reveals program defects that must be removed from the program. This is called *debugging*. Defect testing and debugging are different processes. Testing establishes the existence of defects. Debugging is concerned with locating and correcting these defects.

In the figure below illustrates the stages of debugging. Defects in the code must be located and the program modified to meet its requirements. Testing must then be repeated to ensure that the change has been made correctly. Thus, the debugging process is part of both software development and software testing.

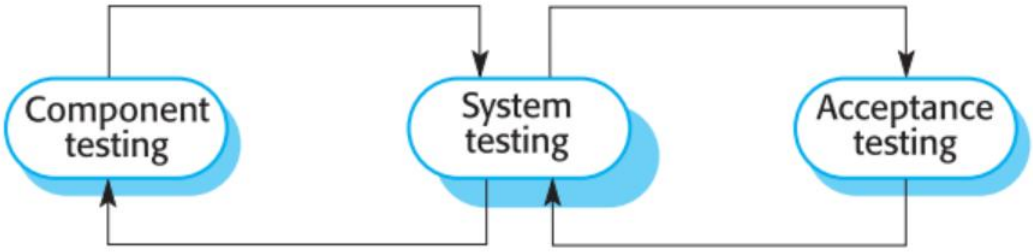


When debugging, you generate hypotheses about the observable behavior of the program then test these hypotheses in the hope of finding the fault which caused the output anomaly. Testing the hypotheses may involve tracing the program code manually. You may write new test cases to localize the problem. Interactive debugging tools that show the intermediate values of program variables and a trace of the
statements executed may be used to help the debugging process.

SOFTWARE VALIDATION

Software validation or, more generally, verification and validation (V & V) is intended to show that a system conforms to its specification and that the system meets the expectations of the customer buying the system. It involves checking processes, such as inspections and reviews, at each stage of the software process from user requirements definition to program development. The majority of validation costs, however, are incurred after implementation when the operational system is tested.

Except for small programs, systems should not be tested as a single, monolithic unit. In the figure below shows a three-stage testing process where system components are tested, the integrated system is tested and, finally, the system is tested with the customer's data. Ideally, component defects are discovered early in the process and interface problems when the system is integrated. However, as defects are discovered the program must be debugged and this may require other stages in the testing process to be repeated. Errors in program components, say, may come to light during system testing. The process is therefore an iterative one with information being fed back from later stages to earlier parts of the process.



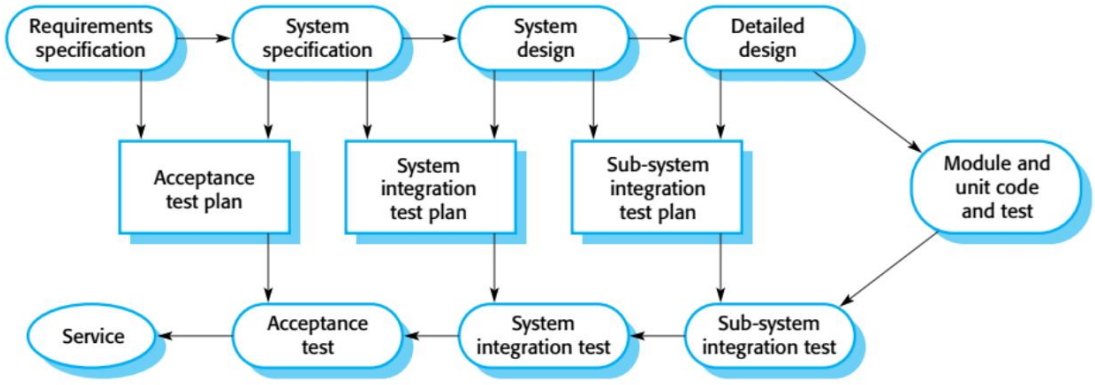
The stages in the testing process are:

1) **Component (or unit) testing**. Individual components are tested to ensure that they operate correctly. Each component is tested independently, without other system components. Components may be simple entities such as functions or object classes, or may be coherent groupings of these entities.

2) **System testing**. The components are integrated to make up the system. This process is concerned with finding errors that result from unanticipated interactions between components and component interface problems. It is also concerned with validating that the system meets its functional and non-functional requirements and testing the emergent system properties. For large systems, this may be a multistage process where components are integrated to form sub-systems that are individually tested before they are themselves integrated to form the final system.

3) **Acceptance testing**. This is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system customer rather than with simulated test data. Acceptance testing may reveal errors and omissions in the system requirements definition because the real data exercise the system in different ways from the test data. Acceptance testing may also reveal requirements problems where the system's facilities do not really meet the user's needs or the system performance is unacceptable.

Later stages of testing involve integrating work from a number of programmers and must be planned in advance. An independent team of testers should work from preformulated test plans that are developed from the system specification and design. The figure below illustrates how test plans are the link between testing and development activities.



Acceptance testing is sometimes called alpha testing. Custom systems are developed for a single client. The alpha testing process continues until the system developer and the client agree that the delivered system is an acceptable implementation of the system requirements.

When a system is to be marketed as a software product, a testing process called beta testing is often used. Beta testing involves delivering a system to a number of

potential customers who agree to use that system. They report problems to the system developers. This exposes the product to real use and detects errors that may not have been anticipated by the system builders. After this feedback, the system is modified and released either for further beta testing or for general sale.

SOFTWARE EVOLUTION

The flexibility of software systems is one of the main reasons why more and more software is being incorporated in large, complex systems. Once a decision has been made to procure hardware, it is very expensive to make changes to the hardware design. However, changes can be made to software at any time during or after the system development. Even extensive changes are still much cheaper than corresponding changes to system hardware.

Historically, there has always been a split between the process of software development and the process of software evolution (software maintenance). People think of software development as a creative activity where a software system was developed from an initial concept through to a working system. However, they sometimes think of software maintenance as dull and uninteresting. Although the costs of 'maintenance' are often several times the initial development costs, maintenance processes are sometimes considered to be less challenging than original software development.

This distinction between development and maintenance is becoming increasingly irrelevant. Few software systems are now completely new systems, and it makes much more sense to see development and maintenance as a continuum. Rather than two separate processes, it is more realistic to think of software engineering as an evolutionary process where software is continually changed over its lifetime in response to changing requirements and customer needs.