Chapter 2: Modeling the process and life-cycle

1. Determining the process boundary is similar to determining a system boundary. One must decide exactly which activities, inputs, and outputs the process includes, and which are part of some other process. For example, in specifying a software development process, one must decide if it begins with writing the contract for the software, or if it assumes the contracting process has already taken place and the contract is finished.

   Also, both systems and process models can be decomposed into subsystems and subprocesses. Both can be decomposed in at least two different ways. Subsystems and subprocesses could be parts of the larger system or process which, when put together, constitute the whole. Or, the arrangement could be in layers, as in Figure 1.10.

   Both systems and processes can be modeled at different levels of detail and from different perspectives in order to understand them, and to discover their strengths and weaknesses.

2.

Waterfall:
- Benefits:
  - simple, familiar to most developers, easy to understand
  - easy to associate measures, milestones, and deliverables with the different stages
- Drawbacks:
  - does not reflect how software is really developed
  - not applicable for many types of development
  - does not reflect the back-and-forth, iterative nature of problem solving

V model:
- Benefits:
  - better spells out the role of different types of testing
  - involves the user in testing
- Drawbacks:
  - extensive testing may not always be cost-effective
  - some of the same drawbacks as waterfall

Prototyping:
- Benefits:
  - promotes understanding of problem before trying to implement solution
  - reduces risk and uncertainty
  - involves user in evaluating interface
- Drawbacks:
  - in systems where the problem is well understood or where the user interface is simple and straightforward, the extra time spent in prototyping is not warranted
  - prototyping can use up a lot of resources, especially if the prototype fails completely and must be scrapped

Operational specification:
- Benefits:
  - allows user and developer to resolve requirements uncertainty early on
- Drawbacks:
  - upfront investment may not be warranted if problem is well understood

Transformational:
- Benefits:
  - eliminates large steps of the process, thus reducing cost and opportunity for error
  - provides automatic documentation
- Drawbacks:
  - needs a very precise formal specification

Incremental or iterative development:
- Benefits:
  - reduces time to when customer receives some product
  - customer training can begin early
  - creates markets for new functionality

frequent releases allow problems to be fixed quickly
expertise can be applied to different releases

Drawbacks:

customer may not be satisfied with an incomplete product or with frequent changes to system
product may never be "complete"
problem may not be easily decomposable
changes may have to be made to completed parts in order to work with new parts

Spiral:

Benefits:

monitors and controls risks throughout process
easily incorporates prototyping

Drawbacks:

a lot of overhead

3. Waterfall: re-analyze requirements, redesign, recode, retest

   V model: same as Waterfall

   Prototyping: prototype change in design and code, iterating with customer

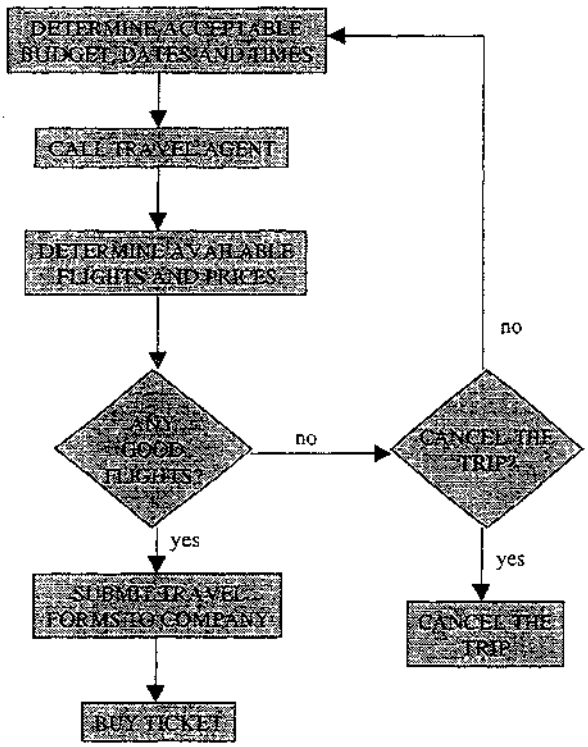
   Operational specification: re-transform specification

   Transformational: re-do relevant transformations

   Incremental or iterative development: implement change in another increment or iteration

   Spiral: implement another mini-spiral

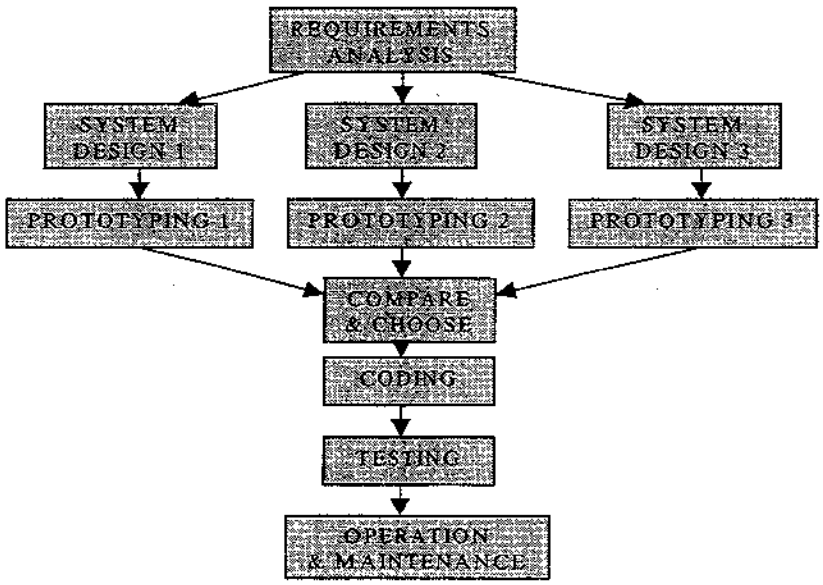
4.



5.

Name	*Module*	
Synopsis	*This is the artifact that represents a class of software modules.*	
Complexity type	*Composite*	
Data type	*(module_c, user-defined)*	
Artifact-state list		
identified	*((state_of(module-spec(module)) = exists) (state_of(module.implementation) = nonexistent) (state_of(module.test) = untested))*	*Module is identified in the specification but has not been implemented yet.*
initial_implementation	*((state_of(module.implementation) = initial) (state_of(module.test) = untested))*	*Module has been initially implemented, but has had no testing.*
testing	*(((state_of(module.implementation) = initial) or (state_of(module.implementation) = revising)) (state_of(module.test) = partial))*	*Module is being tested and bugs being fixed.*
complete	*((state_of(module.implementation) = final) (state_of(module.test) = complete))*	*Module is complete.*
Sub-artifact list		
	implementation	*The module's implementation.*
	test	*The process of testing the module.*
Relations list		
module-spec	*The part of the spec pertaining to the module.*	

6.



7. They would all be nice, but the only ones that are essential are that the model facilitates human understanding and communication and that it supports process management. Projects in which the problem and solution are not well-understood need to have as flexible a process as possible, which in most cases means that the process should not be defined at too detailed a level of granularity. In addition, as few resources as possible should be taken by purely process issues because they need to be devoted to understanding the problem and solution. Therefore, process improvement, guidance, and enactment should not be top priorities for the project. On the other hand, the model should facilitate following the process with as little effort as possible, so it should facilitate understanding and communication. Also, such projects can easily get out of control, so process management should be important.

8. Like manufacturing, there is a concern in software development with assuring the quality of the product, and many resources are spent in assessing and improving that quality. As well, both manufacturing and software development require careful planning and monitoring of the process. However, software development is creative in the sense that nearly every problem is new and requires a solution that is at least partly something that has never been done before. Therefore imagination must be used to envision how and if the solution will work.

9. The advantage of adopting a single process model for all software development in an organization is that it standardizes training, terminology, the collection of process metrics, planning, and estimation. This works well if all the organization's development projects are very similar in nature. If not, however, adopting a single standard process may unnecessarily constrain some projects from using the process that is best suited to the problem and solution.

10. Conformance to a particular process is often checked with the use of milestones. That is, the process is defined in such a way that there are tangible products at various points in the process whose existence indicates that particular process steps have been carried out. For example, when using the waterfall process, these intermediate products, or milestones, could be a requirements document, a design document, the code itself, test documents, etc. The timing of these products would indicate whether or not the process was being followed as planned. Another way to monitor use of a process is by measuring effort. Developers working on the project could be required to report the effort they spent on different process activities. By tracking when effort is spent on which activities, progress through the steps of the process could be monitored.

11. Both the incremental and iterative models provide a great deal of flexibility to accommodate requirements changes. They differ, however, in what types of requirements changes they handle best. The incremental model is good at handling requirements changes which add or delete whole areas of functionality because each increment adds to the system in terms of functions. So, if a new function is added to the requirements, its implementation can be planned as part of a future increment. If a function is deleted, it may not even have been implemented yet, and can be deleted from the plan for a future increment. On the other hand, the iterative model best handles requirements changes which modify functionality, rather than adding or deleting it. In iterative development, every function is implemented at the beginning, but refined through successive iterations. Since most functions are modified in each iteration anyway, it is usually not difficult to incorporate modifications due to requirements changes.

12. The main issue is who is responsible for the failure of the software, and for any reparations necessitated by that failure. Ethically, you should have done everything in your power to ensure that the software was defect-free. That would include performing code reviews because you, as a software developer, know that code reviews are a valuable tool in locating software faults. Legally speaking, however, you were not required to perform code reviews. Moreover, you may have been prevented from performing those code reviews, either by a lack of resources provided by Amalgamated, or even by the terms of the contract, which may prohibit all process activities not specifically prescribed. In that case, Amalgamated would at least share responsibility for the software failure.