SOFT 437

Performance Analysis

Chapter 2: SPE Overview

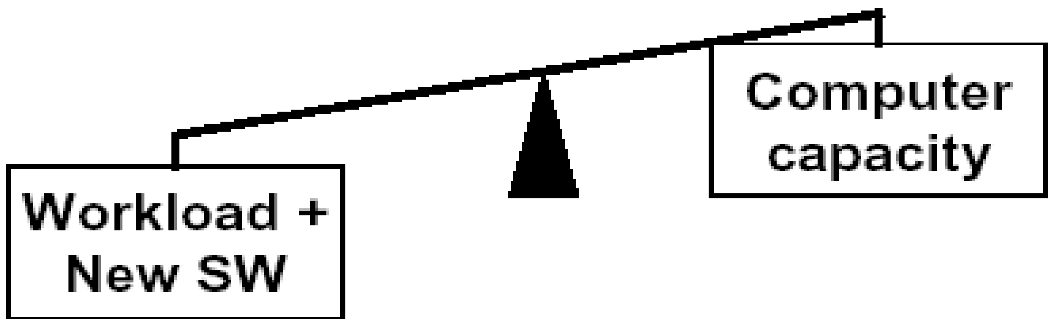
http://www.perfeng.com/papers/perfsol2.pdf - Ch2

http://www.informit.com/articles/article.aspx?p=24009 – Ch1

Quick Highlights

- What is Performance?
  - Response time
  - Throughput
- Responsiveness
- Scalability
- Why Performance?
  - Damage of customer relation - Business failure
  - Loss of competency - Bad reputation
- How to manage performance
  - Reactive x Proactive
- Why do we need SPE?
  - Build models to tradeoff between **Resources and Demands**

Is SPE Necessary?



Resource Requirements Capacity

SPE detect problems early in developments and use qualitative methods to support cost-effective analysis

It’s too Expensive to Build Responsive Software?

- Primary motivation for fix-it-later:
  - improve development and maintenance productivity

- Today:
  - newer methods, models, and tools actually increase productivity
    - preventing problems that delay delivery
    - preventing tricky-code maintenance problems

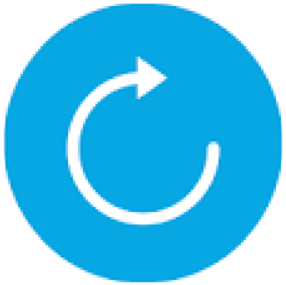
You Can Tune Software Later?

- Tuning may improve performance, but not as much as design can
- Problems are usually caused by fundamental architectural or design problems rather than inefficient code
- Very expensive (often infeasible) to change fundamental design choices

Efficiency Implies ‘Tricky Code’?

- Tuning may introduce tricky code to resolve problems that could have been prevented
- Tricky code may the only option for achieving goals late in the lifecycle
- Acceptable performance is required, and can be designed in early

How should you mange performance?



Fix it later

Reactive

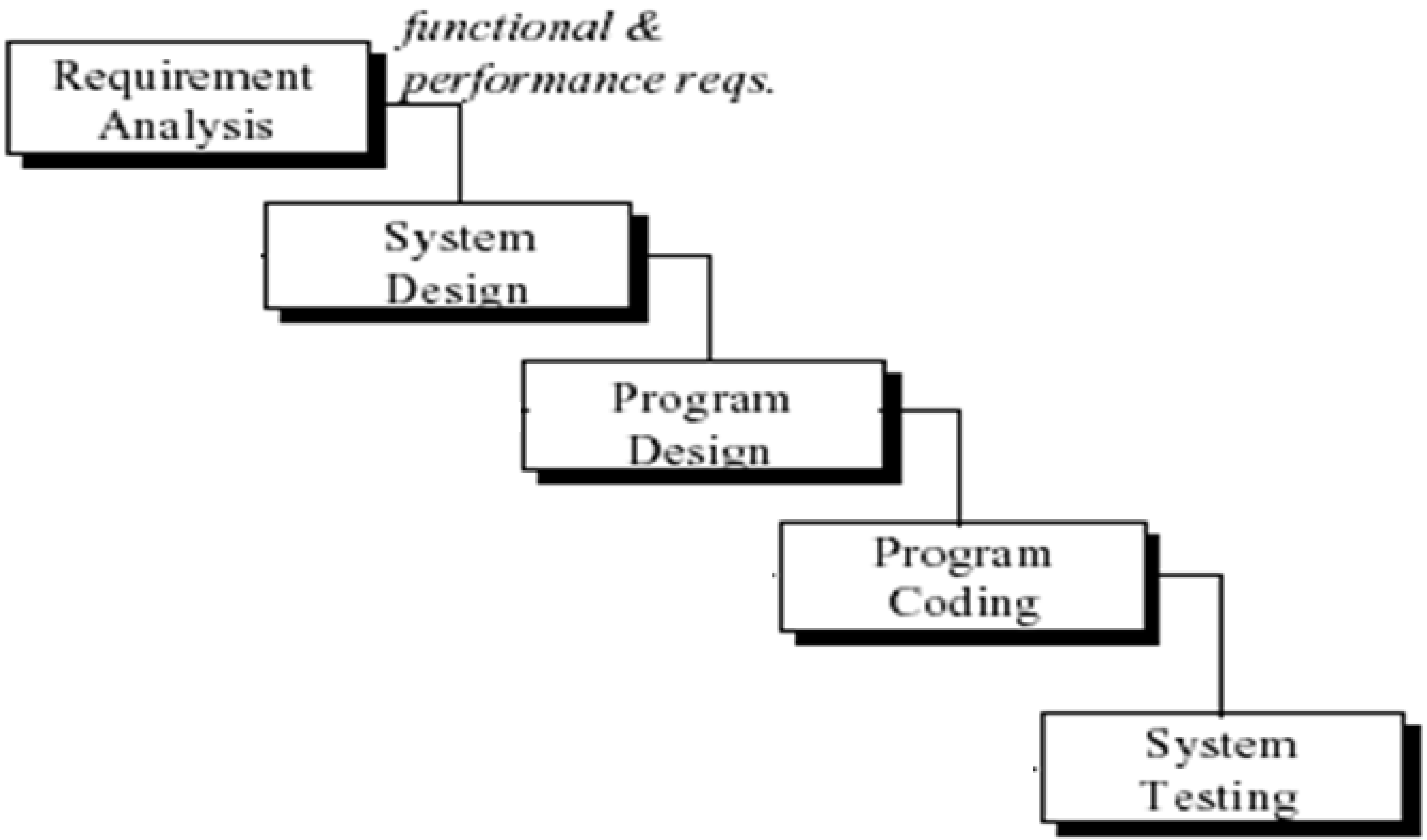
- Let's build what it can
- We'll tune
- We cannot until we ha measure
- We have fa
- Don't worry, you are in safe hands
- Problems? We don't have problems

P

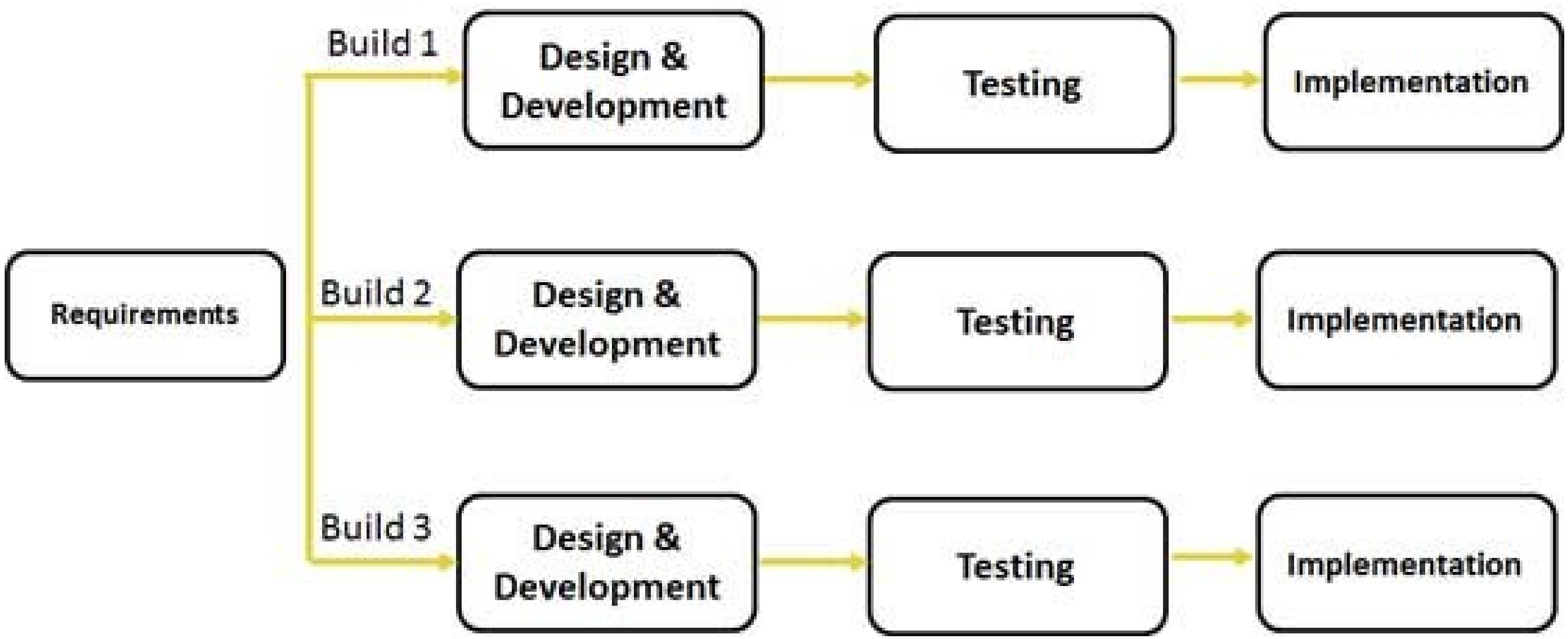
- t has a ce engineer (PE)
- in the project name of the PE
- procedure in- ow to identify ce issues
- Team members are trained in performance processes

appeared first in the old days

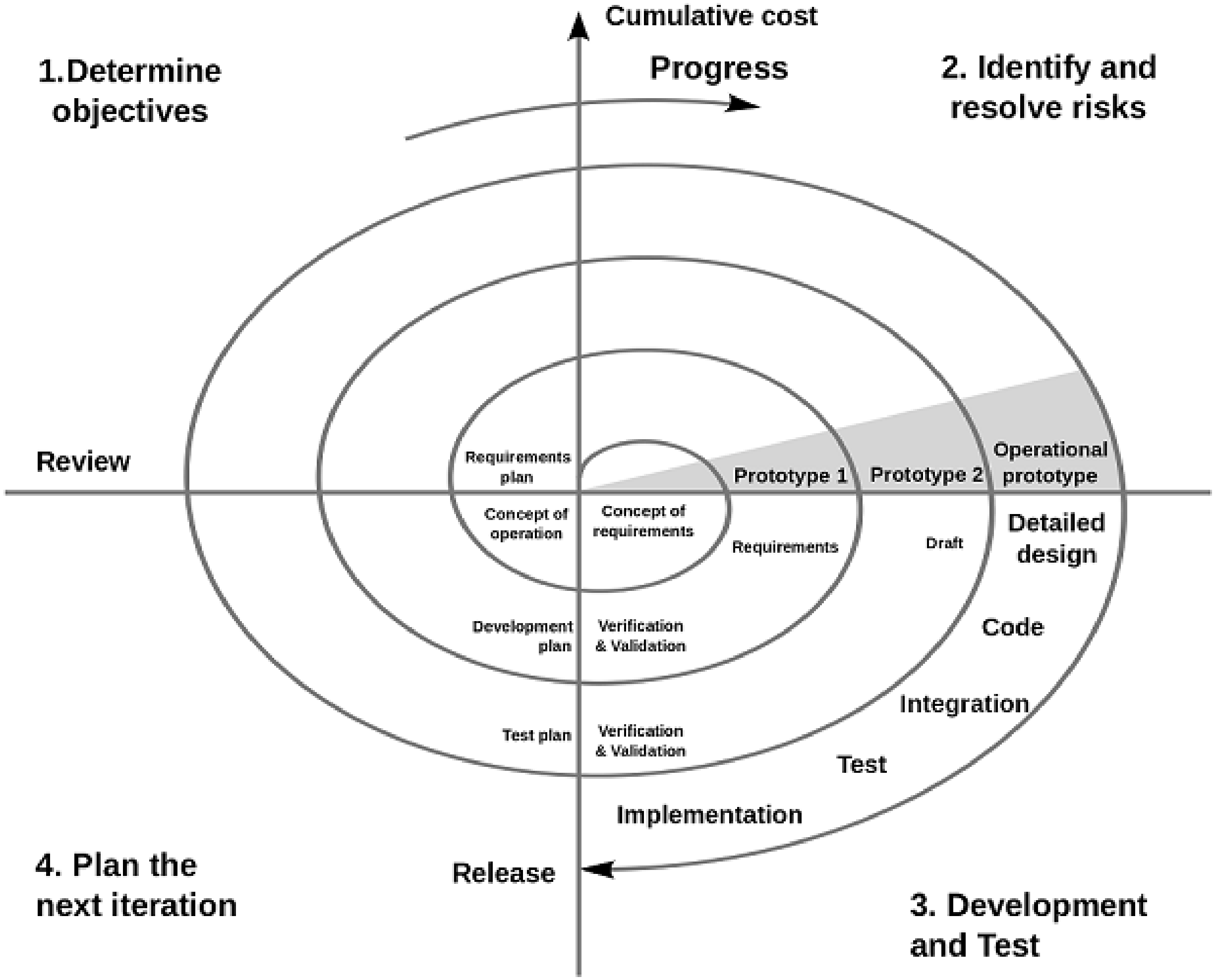
Software Development Life Cycle – Waterfall Model



SDLC - Iterative Model



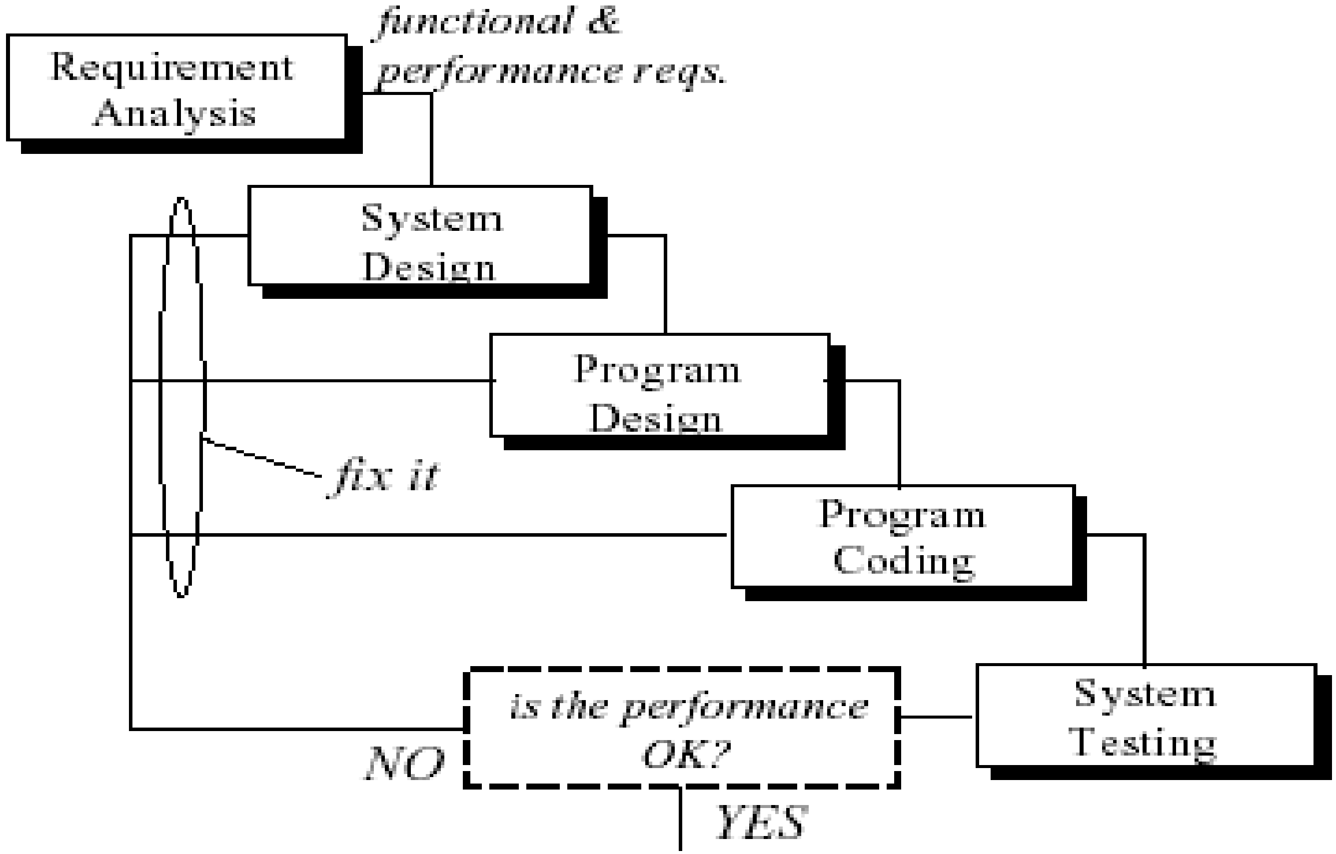
SDLC - Spiral Model



Traditional Software Development Life Cycle

- **Common approaches:**
  - Consider Functional Requirements only during development and check Performance Requirements at the end
  - Fix the system if performance is not good!
- **Problems:**
  - It is very costly and time consuming to fix problems after the system is ready!
  - Fixing problems may imply major code refactoring

Traditional Software Development Life Cycle – Waterfall Model



Why Performance Is Not Addressed Early

- No established discipline for assessing the performance characteristics of a design quickly and easily
- Insufficient time is budgeted for integrating performance analysis into the design process
- Pressing deadlines
- Emphasis is placed on implementing a system quickly and improving it later

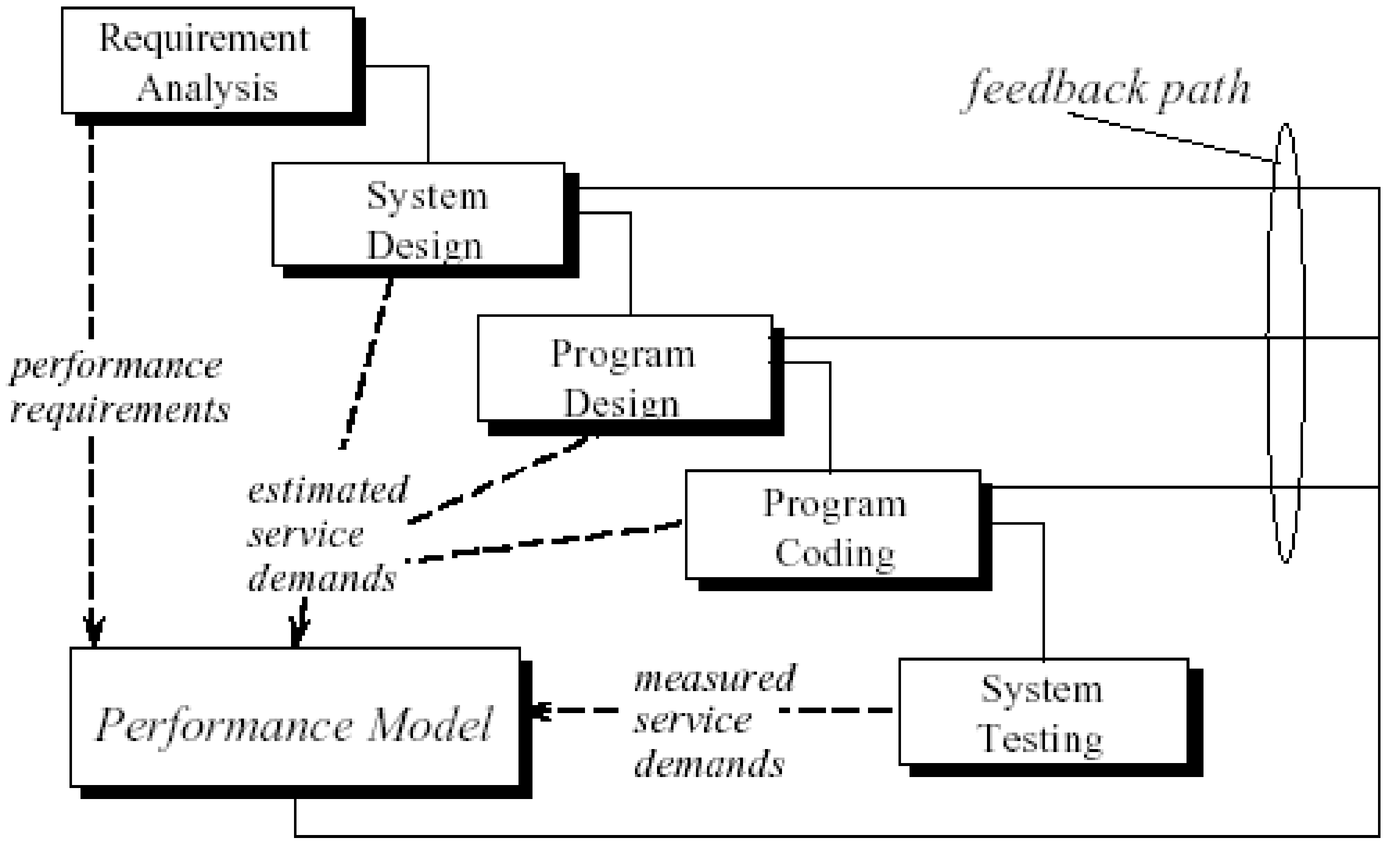
Systems with High Performance Requirements?

- End-user related functions
  - Reservation systems, merchandise-checkout systems
- Real-time, mission critical systems
  - Such as flight-control systems
- Employee support systems
  - Inventory control systems, computer-aided design systems

Software Performance Engineering

- Software Performance Engineering (SPE) is a ***systematic***, ***quantitative*** approach to constructing software systems to meet performance objectives
  - Begins early in the software lifecycle
  - Uses quantitative methods
  - Identifies problems before developers invest significant time in implementation
  - Used through detailed design, coding, and testing

Integrating SPE Into the Software Development Life Cycle



SPE Begins in the Requirements Analysis

- Benefits of early lifecycle steps:
  - Increased productivity — don’t need to throw bad designs
  - Improved quality and usefulness of the resulting software product — selecting suitable design choices
  - Controlled costs of the supporting hardware and software — identifying necessary equipment
  - Enhanced productivity during the implementation, testing, and early operational stages — ensuring that sufficient computing power is available

Address Questions in Early Stages

- Will your users be able to complete tasks in the allotted time?
- Are your hardware and network capable of supporting the load?
- What response time is expected for key tasks?
- Will the system scale up to meet your future needs?

SPE for Object Oriented Systems

- Object oriented systems present special problems for SPE
  - Functionality is decentralized
  - Collaborations are required to perform a given function
  - The interactions are difficult to trace
  - UML (Unified Modeling Language) helps to reduce the impact of these problems.
- SPE is tightly integrated with object-oriented notation, such as the UML
- Use object oriented analysis or design models to derive a performance model
- Use cases provides a starting point for constructing performance models

Performance Analysis

- Use object-oriented analysis or design model to derive a performance model
- Solving the model gives you feedback on performance to revise the object-oriented design
- SPE is also language independent
- SPE can be easily integrated into the software development processes, such as waterfall model, spiral model and rational unified process

Waterfall Model

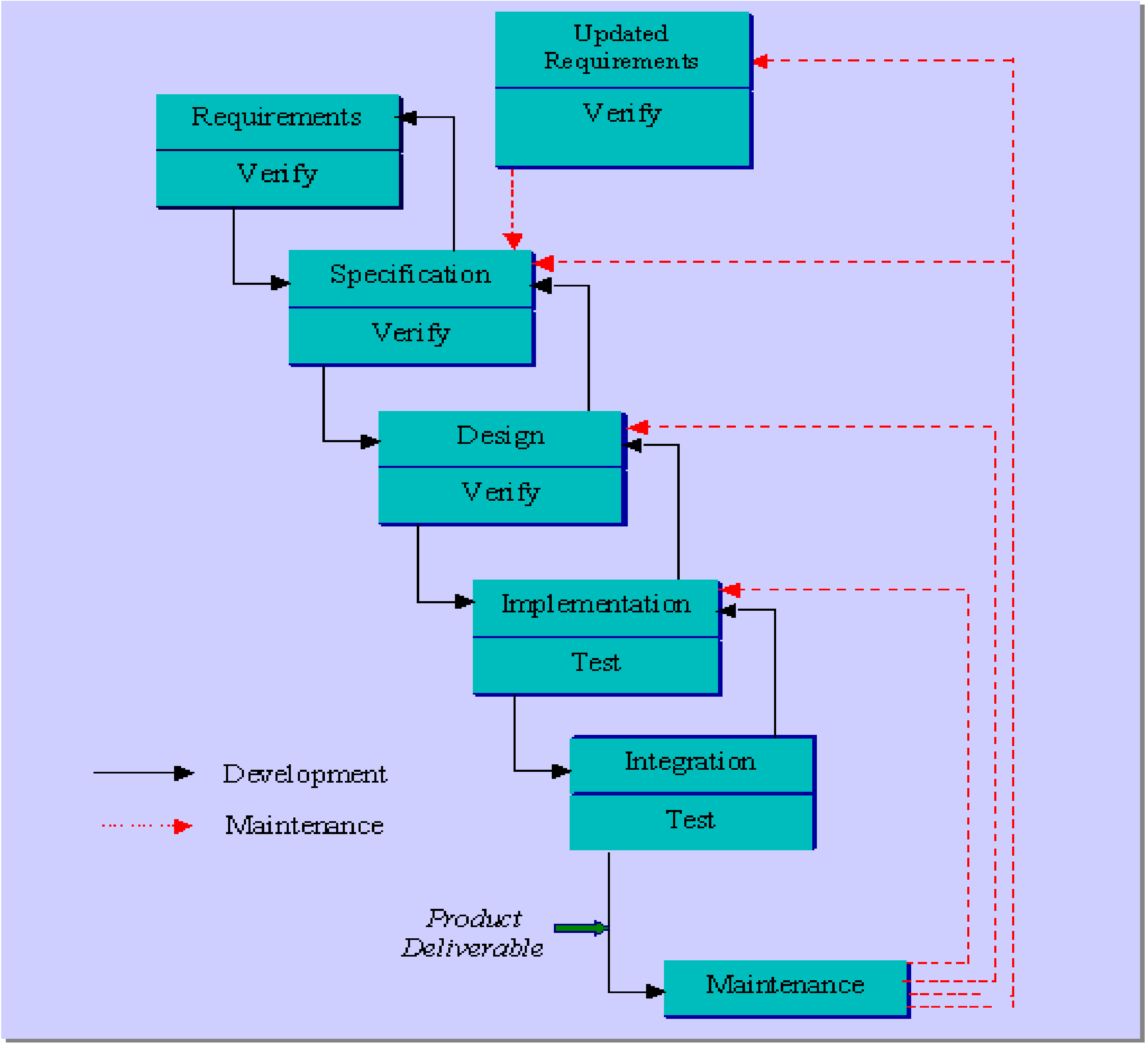
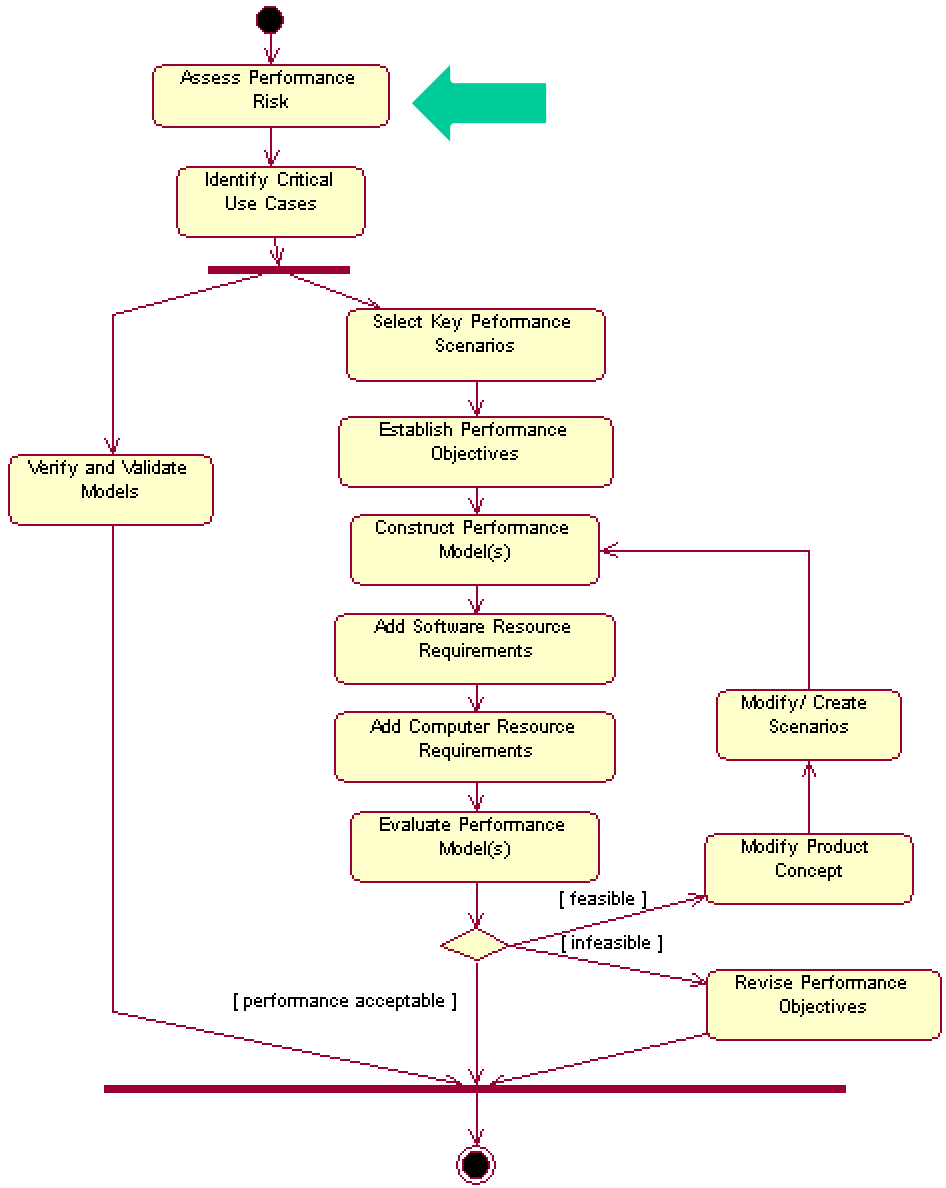


Fig. 1.2 - Schematic illustrating the Waterfall Model

SPE Process for Object-Oriented Systems



1. Assess Performance Risk

- Assessing the performance risk at the outset of the project tells you how much effort to put into SPE activities
  - The SPE effort can be minimal, if the project
    - Is similar to other projects that you have built before
    - Has minimal computer and network usage
    - Is not mission critical or economic survival
- Example:
  - The performance risk in constructing the ATM is small
  - The host software must deal with a number of concurrent ATM users, and response time

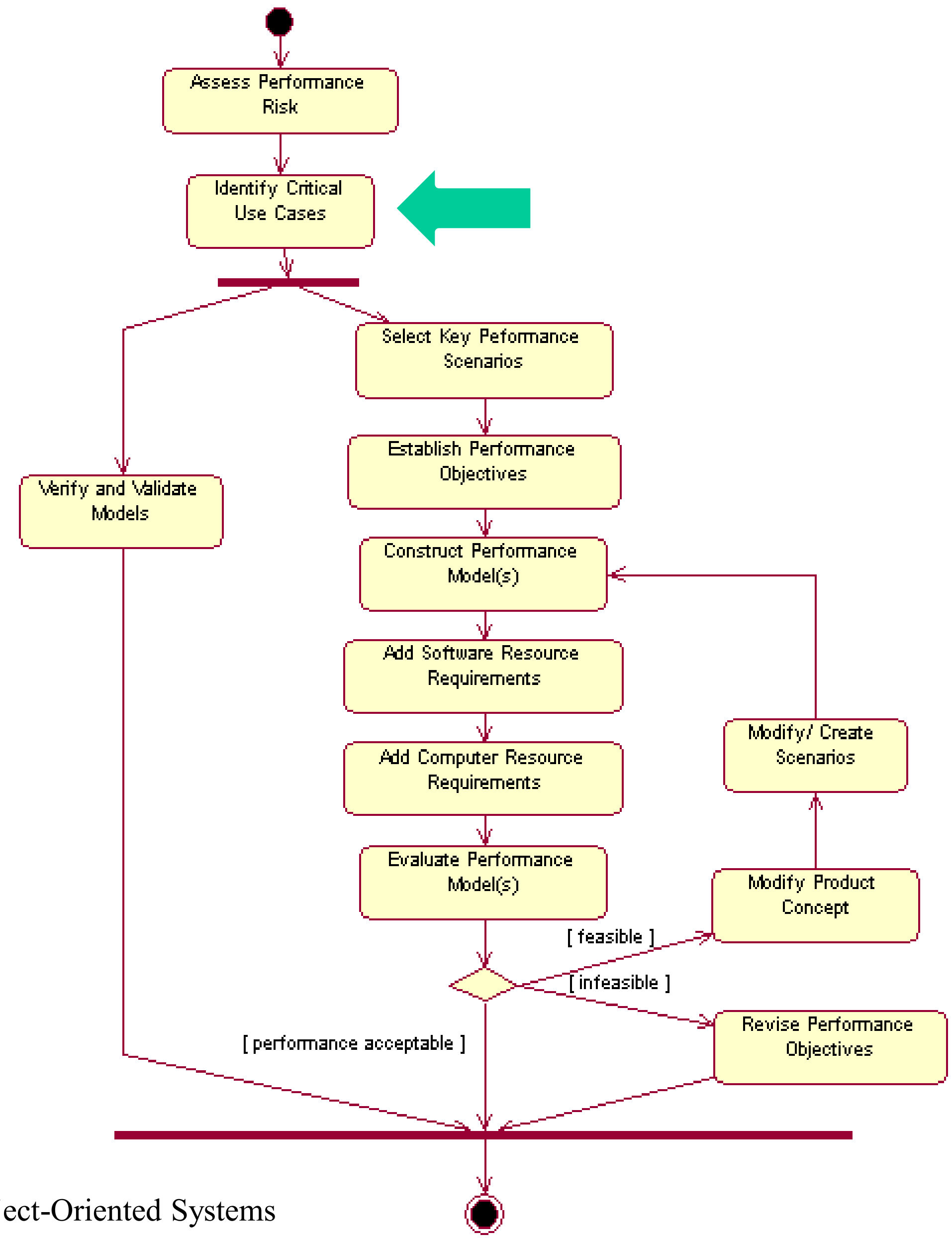
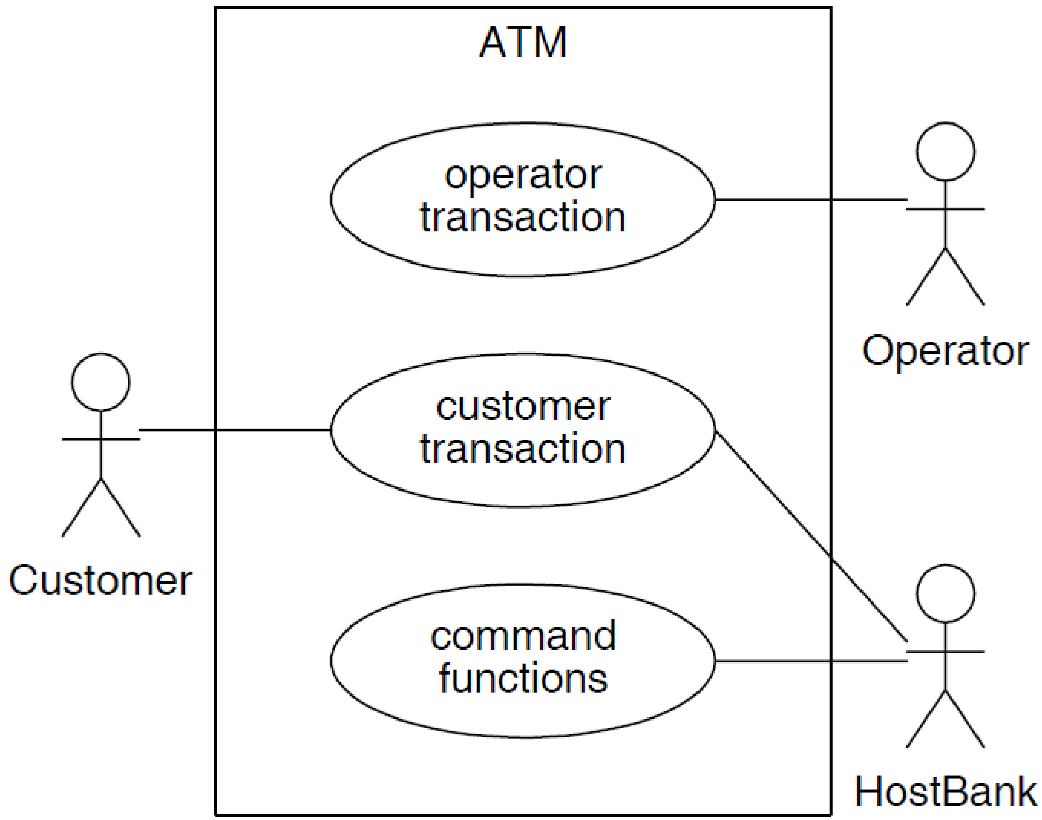


Figure 2-1: The SPE Process for Object-Oriented Systems

2. Identify Critical Use Cases

- The critical use cases are those that are important to the ***operation*** of the system, or are important to ***responsiveness*** as seen by the user
- The selection of use cases is risk driven
  - a risk (e.g. if performance goals are not met, the system will fail or be less than successful)
- Example, ATM use cases include:
  - reloading a currency cassette 
  - customer transaction (e.g., withdraw, deposit) 
  - going off-line 


ATM
operator transaction
customer transaction
command functions
Operator
Customer
HostBank

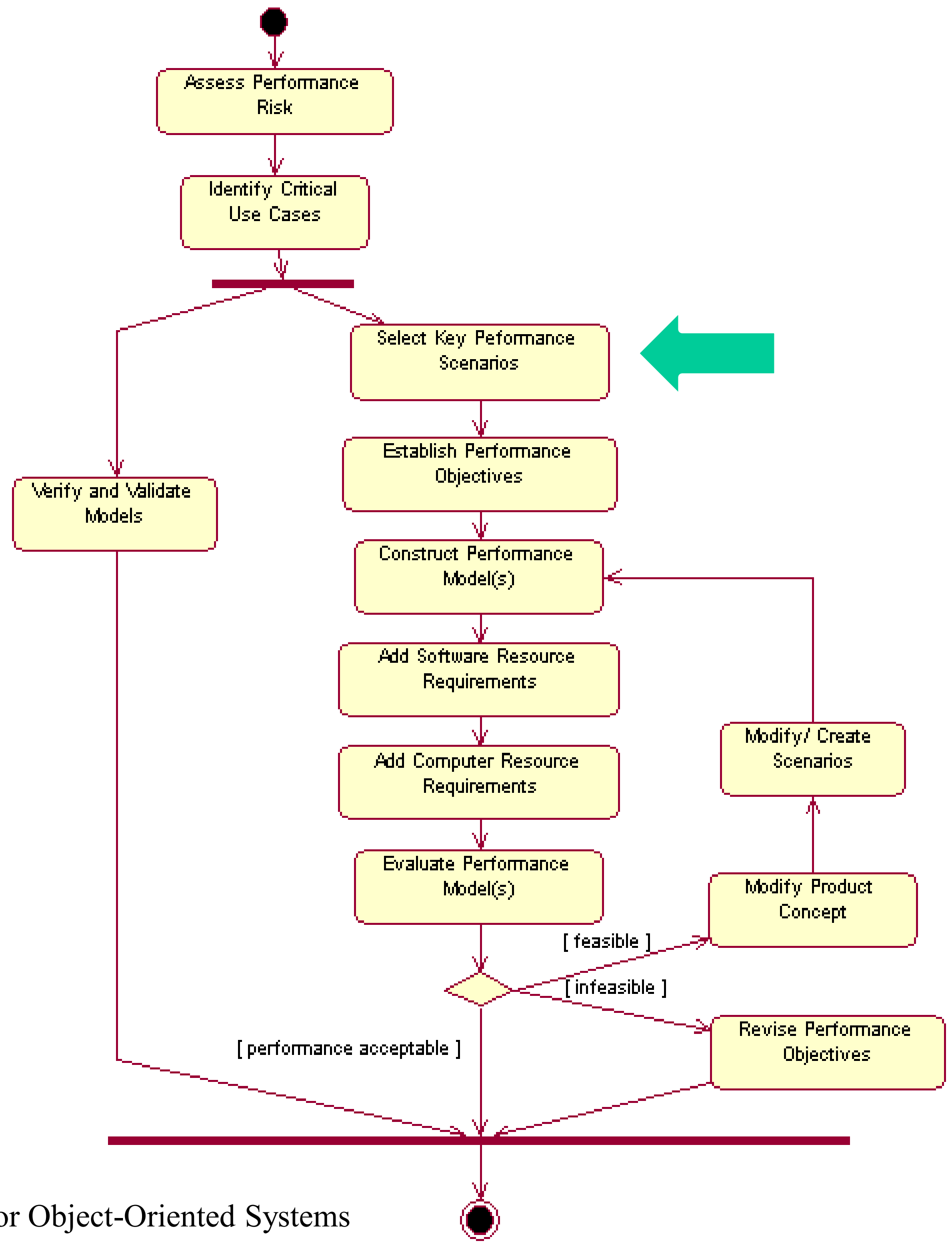
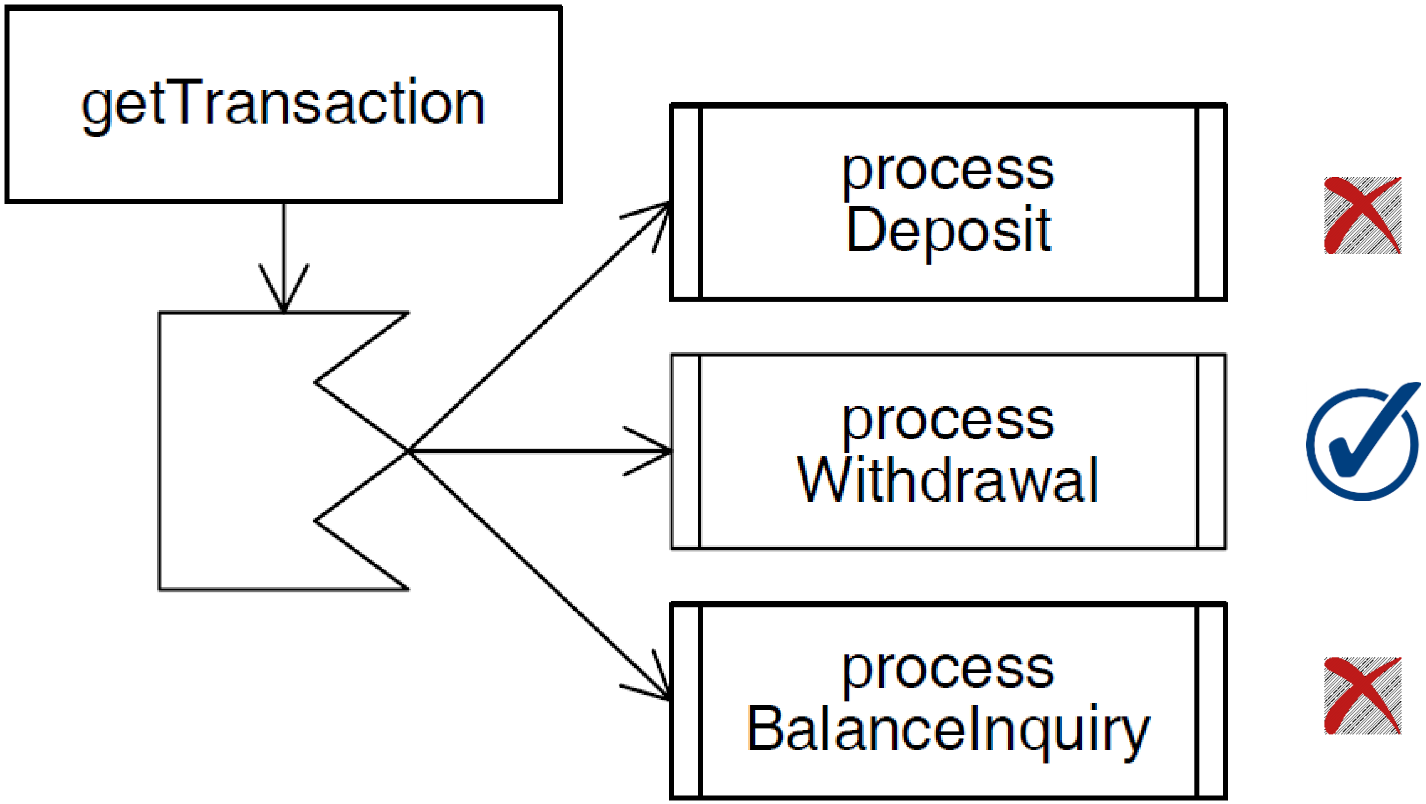


Figure 2-1: The SPE Process for Object-Oriented Systems

3. Select Key Performance Scenarios

- It is unlikely that all of the scenarios for each critical use case will be important from a performance perspective
- The key performance scenarios are
  - Executed frequently
  - Critical to the perceived performance of a system
- Each performance scenario corresponds to a workload
- ***Workload intensity*** specifies the level of usage for the scenario (***arrival rate***)
- Example:
  - Specify ***the workload intensity*** of a customer transaction, that is ***the number of customer transactions*** or ***their arrival rate during the peak period***

Example of Key Performance Scenarios



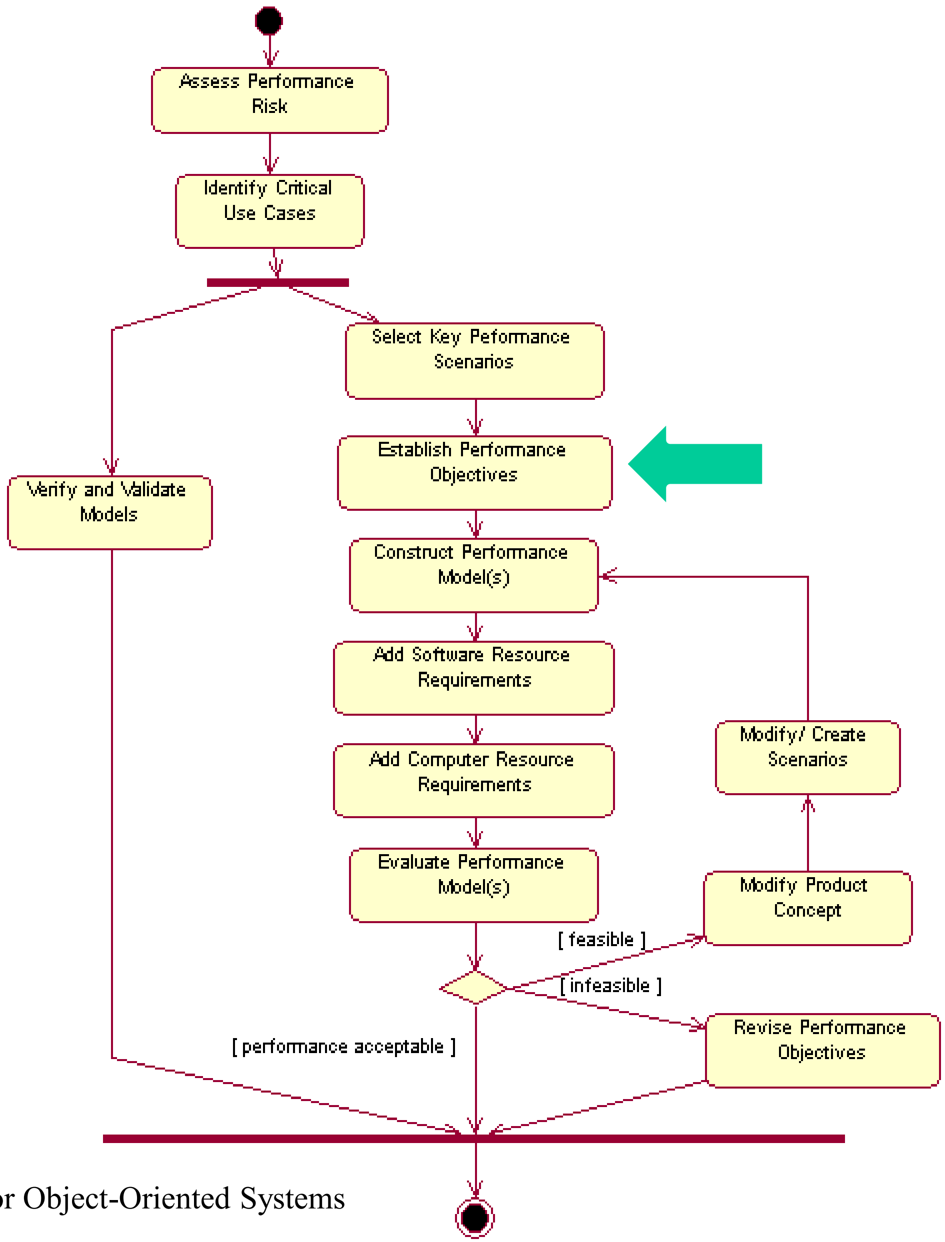


Figure 2-1: The SPE Process for Object-Oriented Systems

4. Establish Performance Objectives

- Performance objectives specify quantitative criteria for evaluating performance characteristics of a system under development
- They are expressed by
  – response time, throughput, or constraints on resource usage
- Example:
  – performance objectives: 30 seconds or less to complete an end-to-end ATM transaction

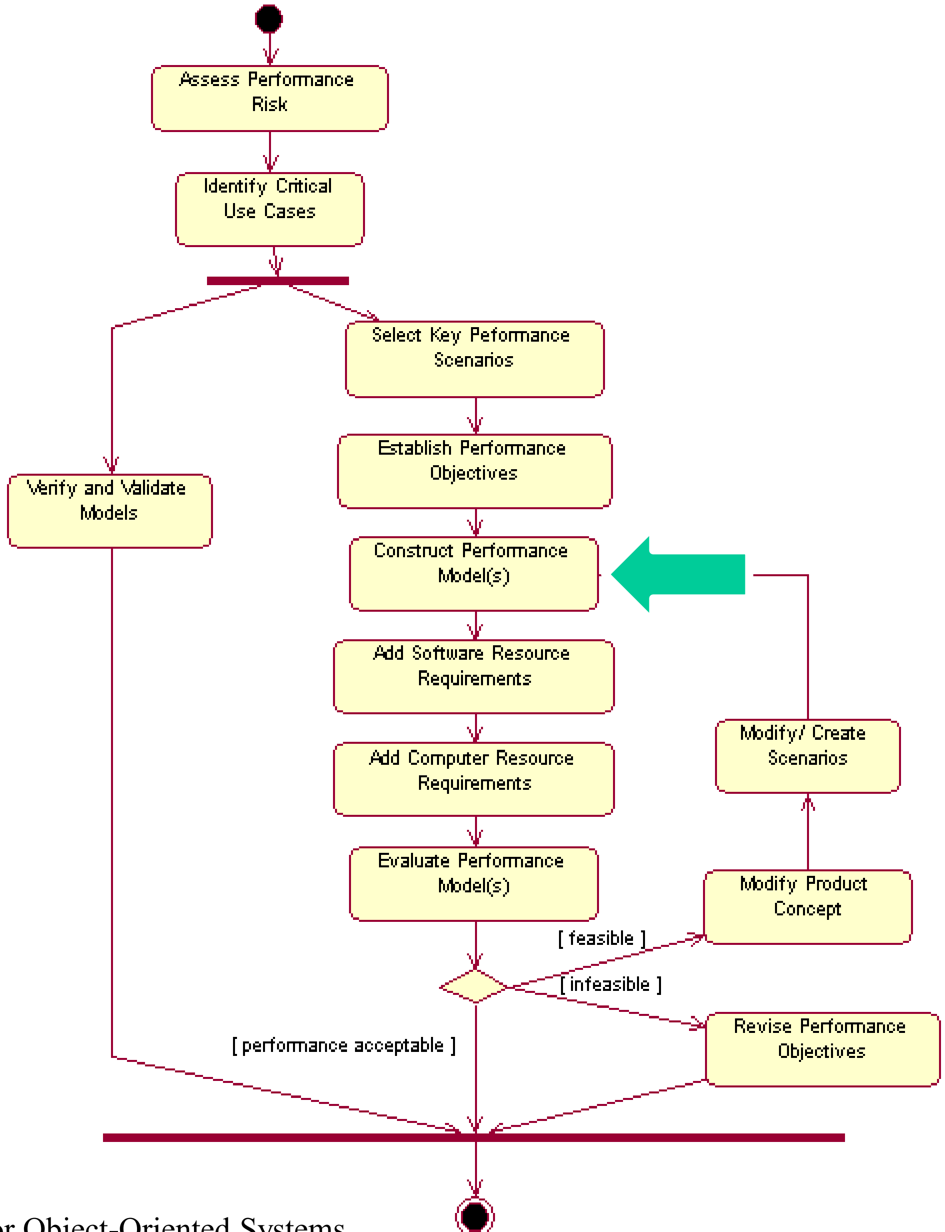
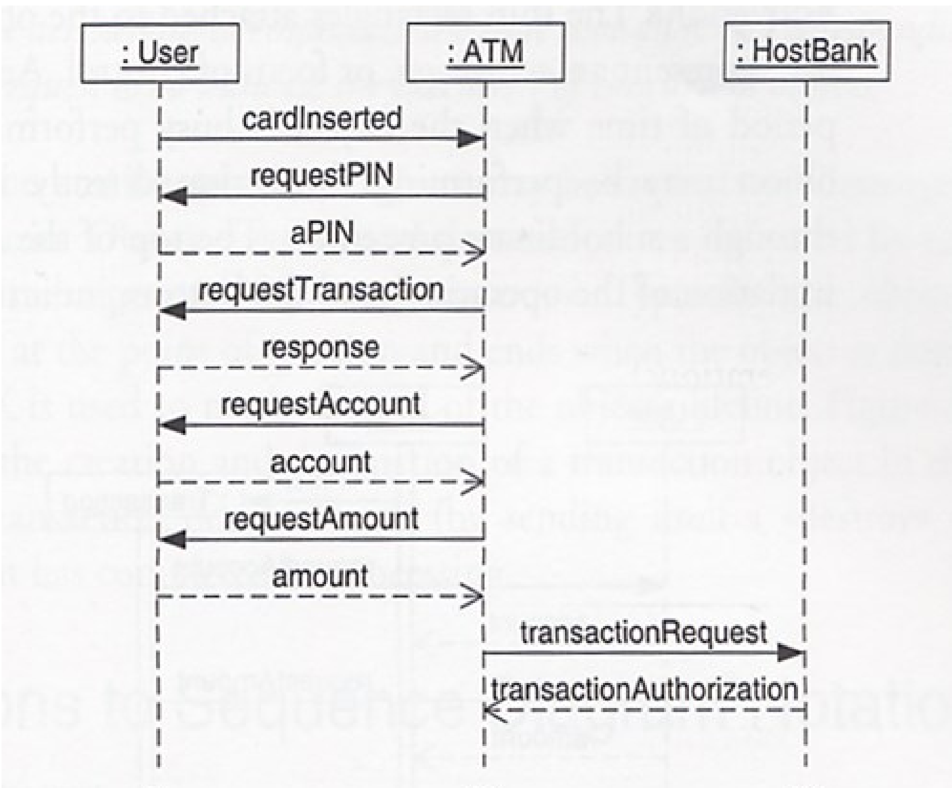


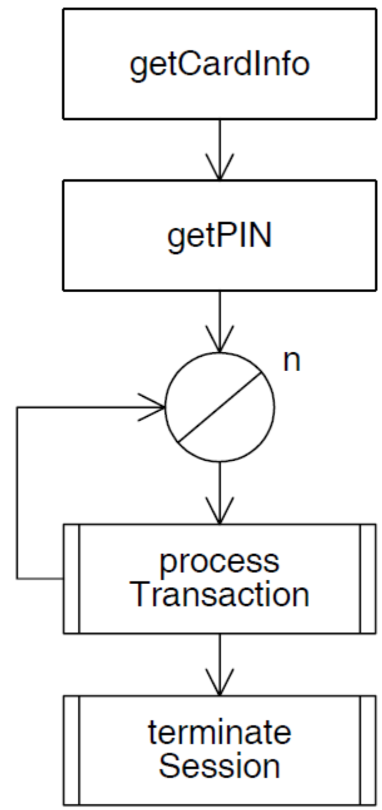
Figure 2-1: The SPE Process for Object-Oriented Systems

5. Construct Performance Models

- We use execution graphs to represent software processing steps in a performance model



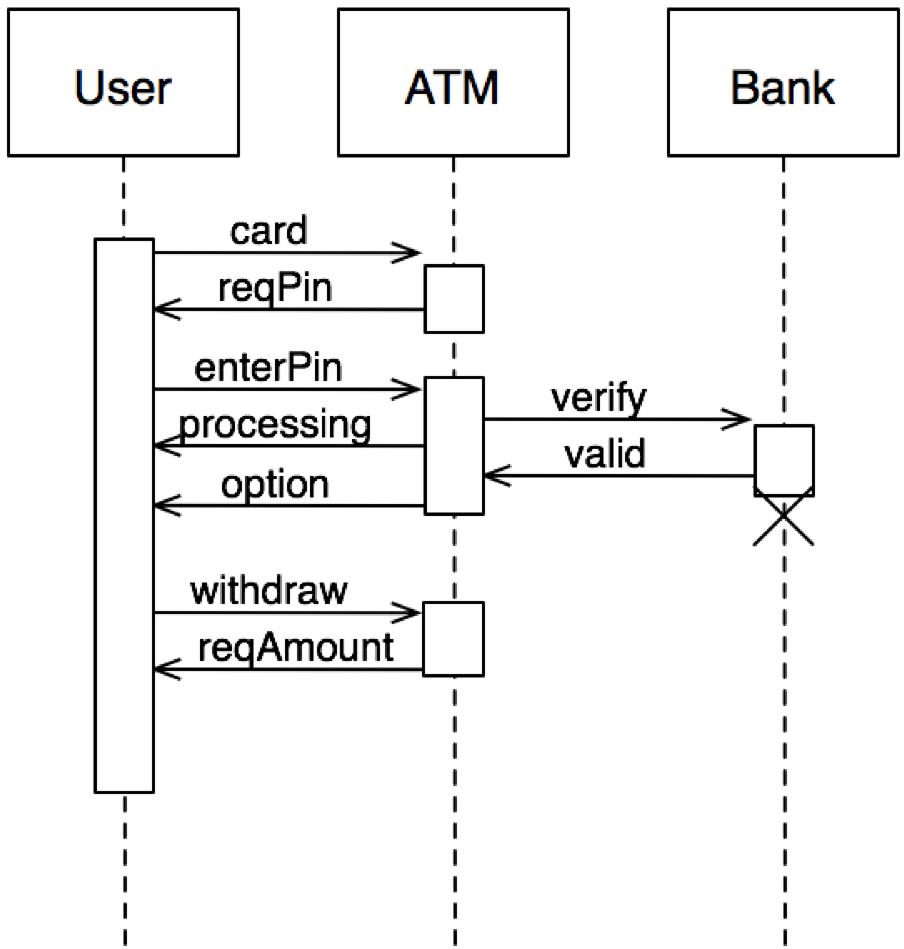
Sequence Diagram



Execution Graph

ATTENTION!

Make sure your sequence diagram is a reflection of the real process!



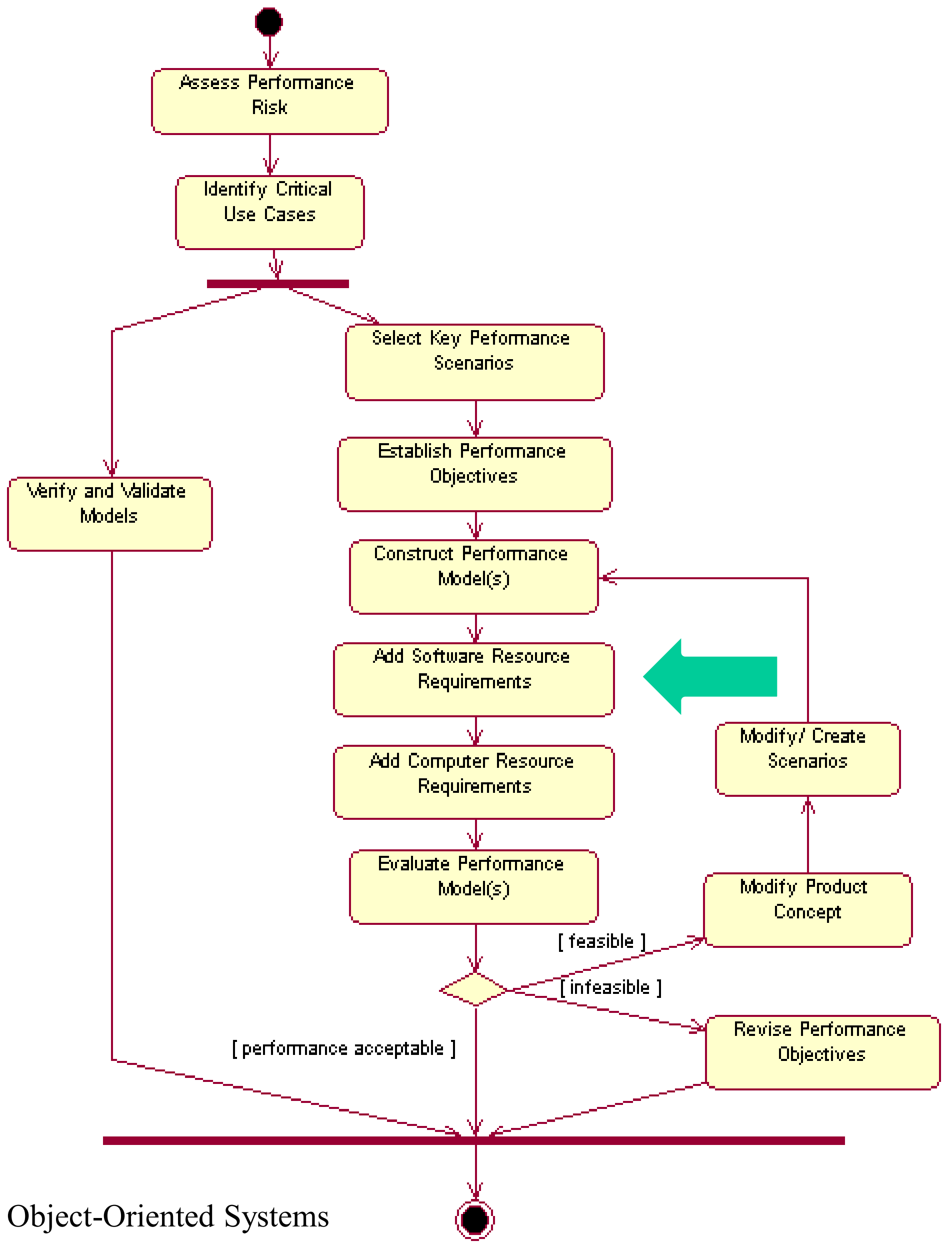
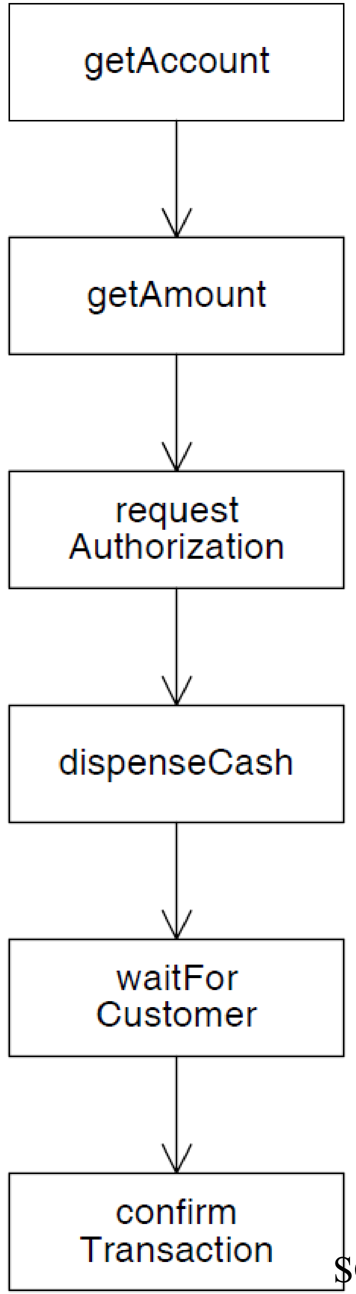


Figure 2-1: The SPE Process for Object-Oriented Systems

6. Determine Software Resource Requirements

- Software resource requirements capture computational needs that are meaningful from a software perspective
- Example software resources that are important for an ATM:
  - Screens – the number of screens displayed the ATM Customer
  - Host– the number of interactions with the host bank
  - Log – the number of log entries on the ATM machine
  - Delay – the relative delay in time for other ATM device processing, such as the cash dispenser or receipt printer



Screen	1
Host	0
Log	0
Delay	0

Screen	1
Host	0
Log	0
Delay	0

Screen	0
Host	1
Log	1
Delay	0

Screen	1
Host	0
Log	1
Delay	5

Screen	0
Host	0
Log	0
Delay	10

Screen	0
Host	1
Log	1
Delay	0

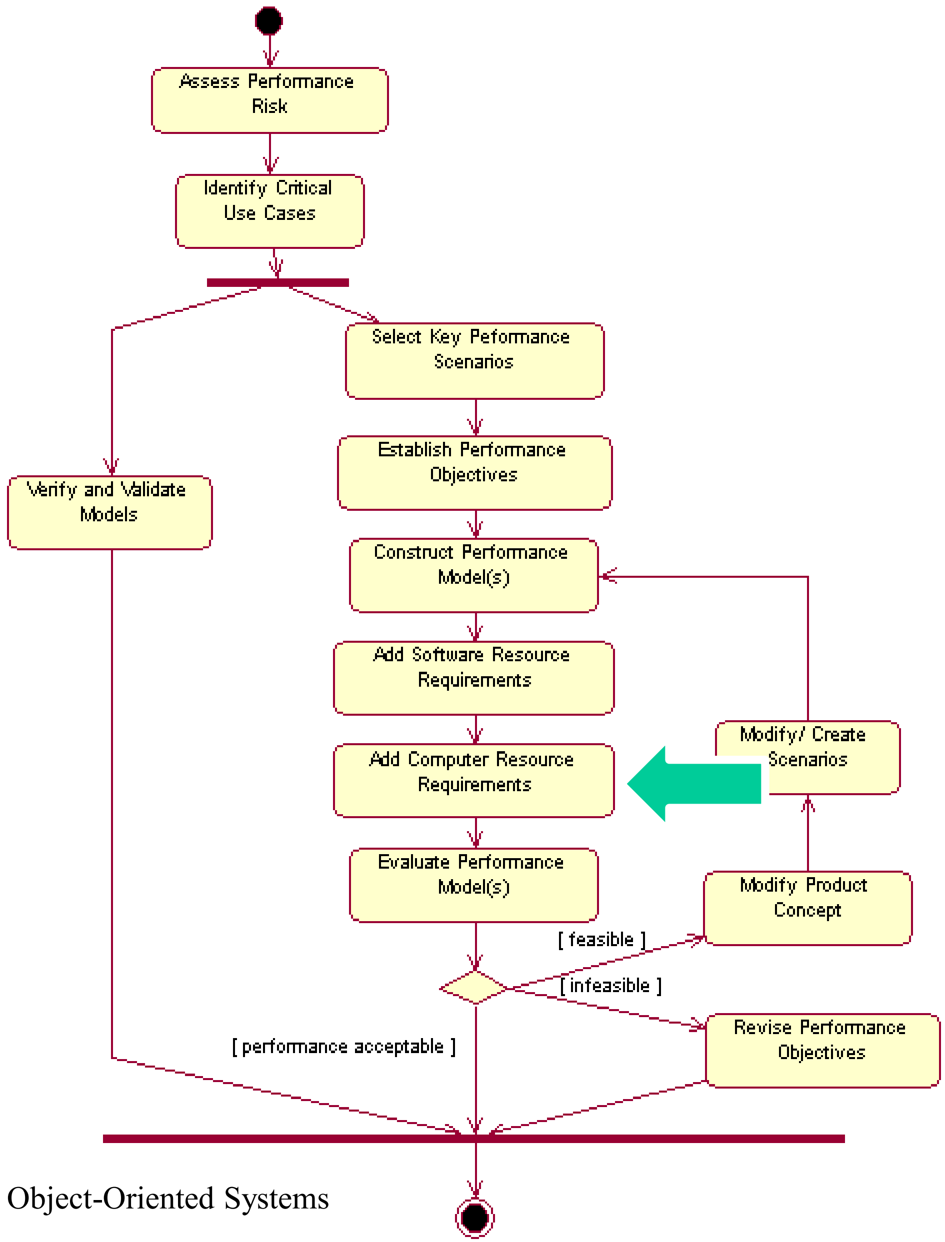


Figure 2-1: The SPE Process for Object-Oriented Systems

7. Add Computer Resource Requirements

- Computer resource requirements map the software resource requirements onto the amount of service key devices in the execution environment
- Example computer resources at an ATM:
  - The types of processor/devices (CPU, Disk, Display, Delay), quantity, speed

Evaluation parameters

Table 2-1: Example Overhead Matrix

Devices	CPU	Disk	Display	Delay	Net
Quantity	1	1	1	1	1
Service Units	Sec.	Phys. I/O	Screens	Units	Msgs.
Screen	0.001		1		
Host	0.005			3	2
Log	0.001	1			
Delay				1	
Service Time	1	0.02	1	1	0.05

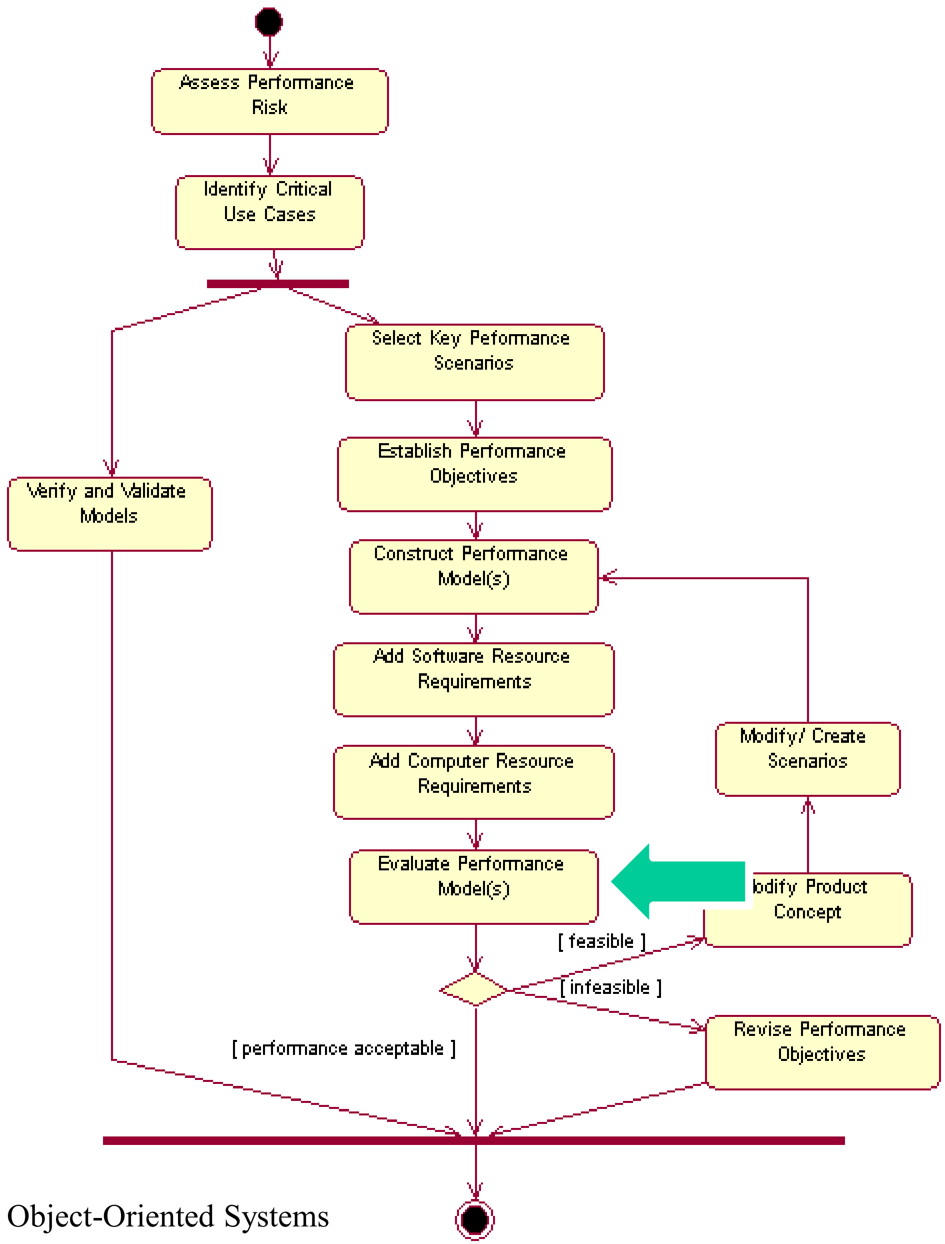
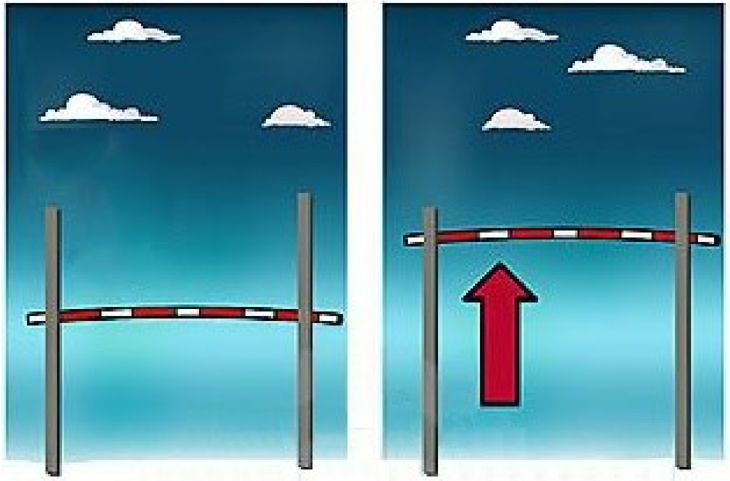


Figure 2-1: The SPE Process for Object-Oriented Systems

8. Evaluate the Models

- If the model indicates that there are problems, there are two alternatives:
  - Modify the product concept: looking for feasible cost-effective alternatives for satisfying the use case instance
  - Revise performance objectives: no feasible alterative exists, we modify performance goals to reflect this reality



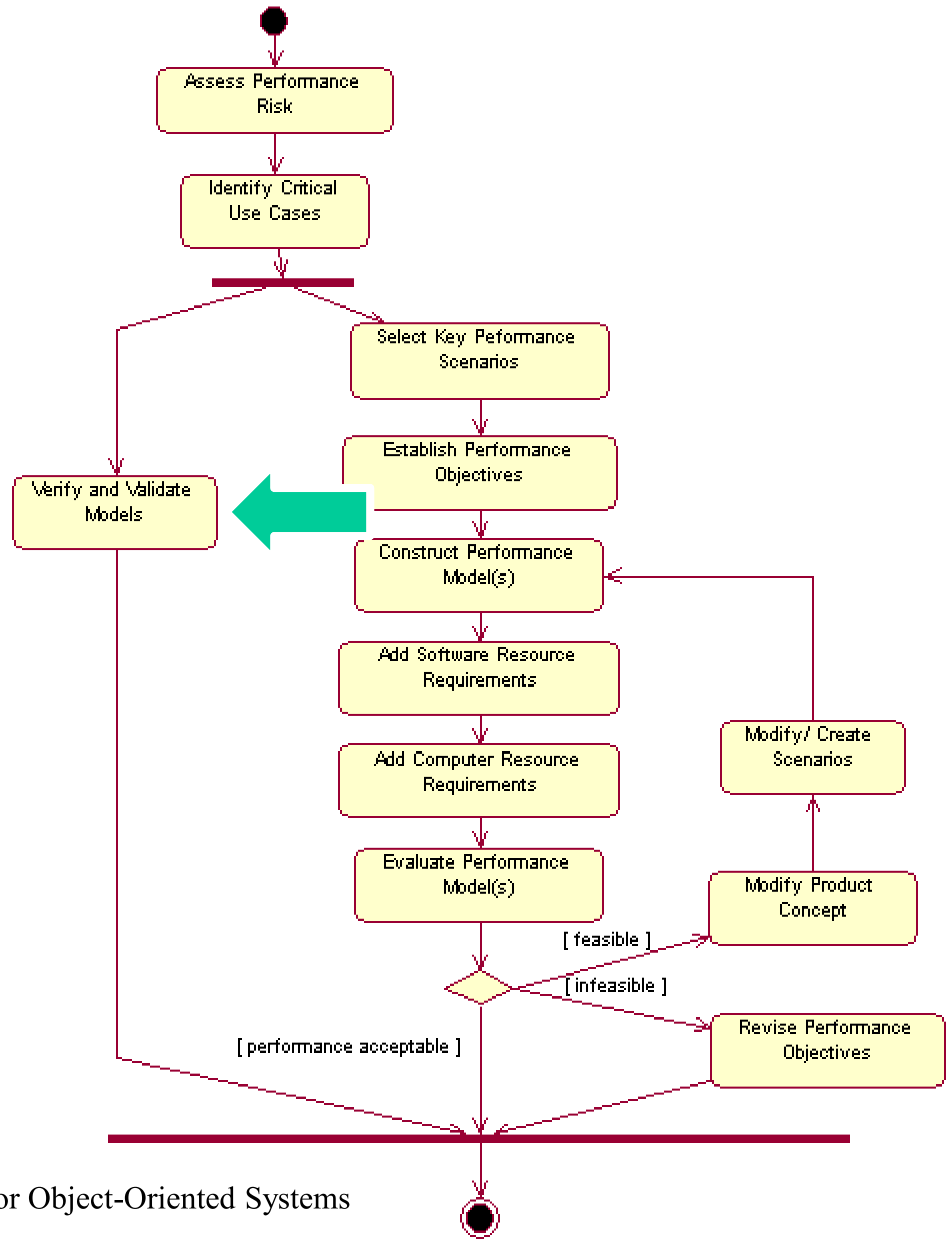


Figure 2-1: The SPE Process for Object-Oriented Systems

9. Verify and Validate the Models

- Model ***verification*** and ***validation*** are ongoing activities that proceed in parallel with the construction and evaluation of the models
- Model ***verification*** is aimed at determining whether the model predictions are an accurate reflection of the software's performance
- Model ***validation*** is concerned with determining whether the model accurately reflects the execution characteristics of the software

SPE Modeling

SPE Modeling Strategies

1. Simple-Model Strategy
2. Best- and Worst Strategy
3. Adapt-to-Precision Strategy

1. Simple-Model Strategy

- Leverages the SPE effort to provide rapid feedback on the performance of the proposed software.

Start with the simplest possible model that identifies problems with the system architecture, design, or implementation plans

2. Best- and Worst-Case Strategy

- The models rely upon estimates of resource requirements for the software execution
- The precision of the models depends on the quality of these estimates
- It is difficult to precisely estimate resource requirements early in the software process

Use best- and worst-case estimates of resource requirements to establish bounds on expected performance and manage uncertainty in estimates

3. Adapt-to-Precision Strategy

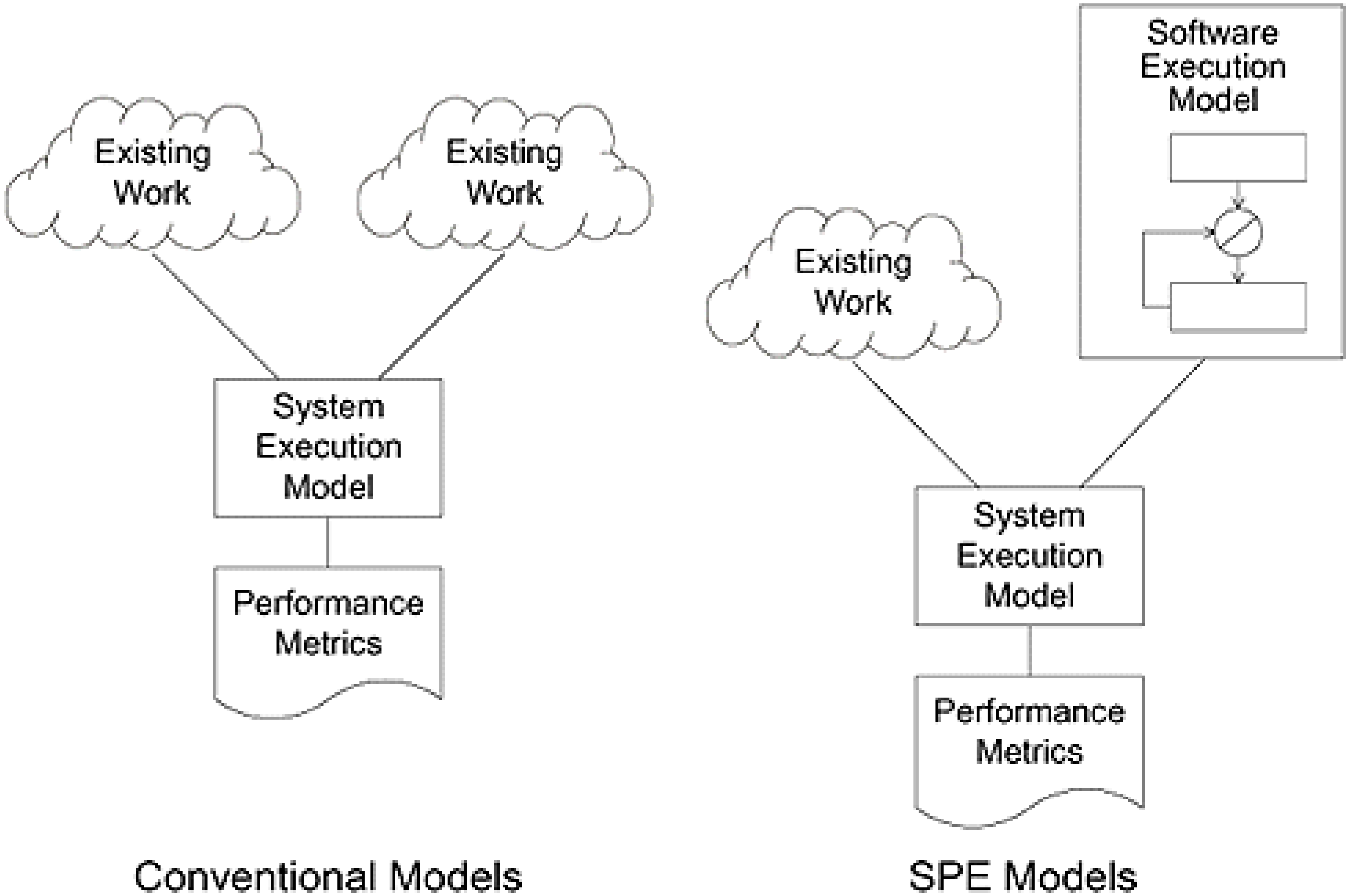
- The simple-model strategy is appropriate for early life cycle studies.
- The adapt-to-precision strategy is used later in the development process

Match the details represented in the models to your knowledge of the software processing details

Conventional Modeling Procedure

- Study the exiting computer system
- Construct a system execution model
- Measure current execution patterns
- Characterize workloads
- Develop input parameters to calculate performance metrics
- Validate the model by solving the performance metrics
- Calibrate the model

Conventional vs. SPE Models



References

- Course Notes for *Performance of Computer Systems* by Håkan Grahn, Department of Software Engineering and Computer Science, Blekinge Institute of Technology Sweden
- Course Notes for CS 672 by Daniel A. Menasce, Department of Computer Science, George Mason University