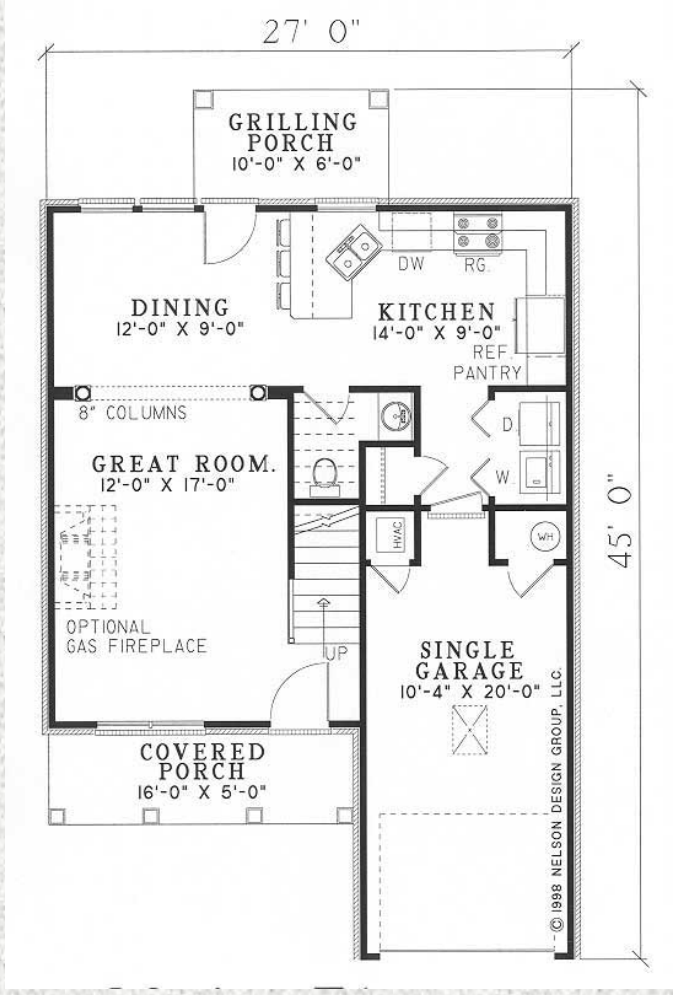
CONTROL ROOM ACCELERATOR PHYSICS

Day 3

The Software Process: Software Engineering and XAL

Outline

1. Software Engineering Introduction
2. Software Process Overview
3. Software Architecture
4. Application frameworks



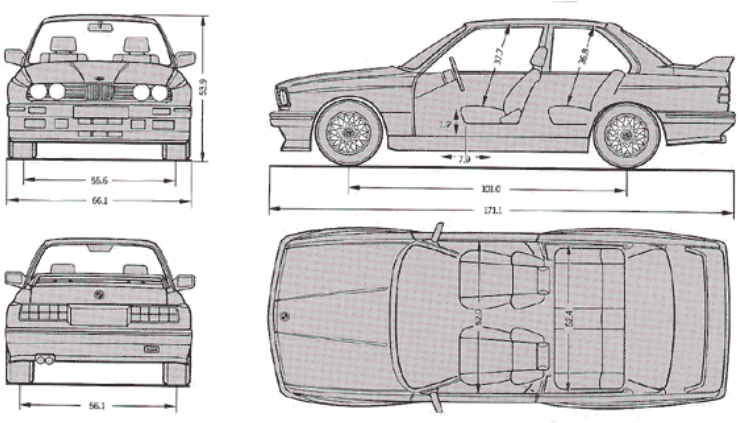
Software Engineering

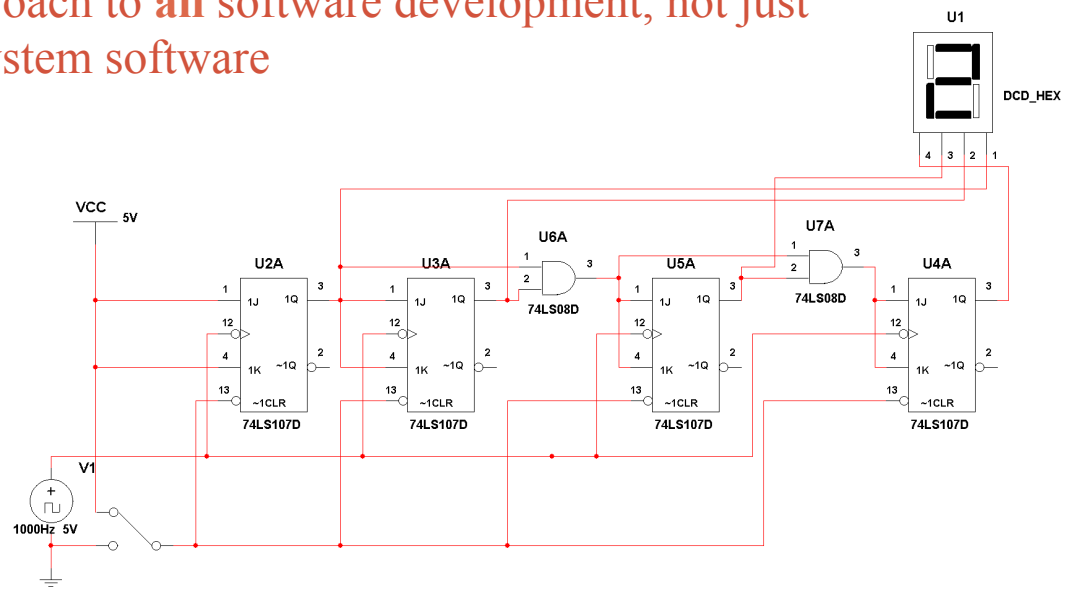
Definition

According to the IEEE…

> "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software: that is, the application of engineering to software."

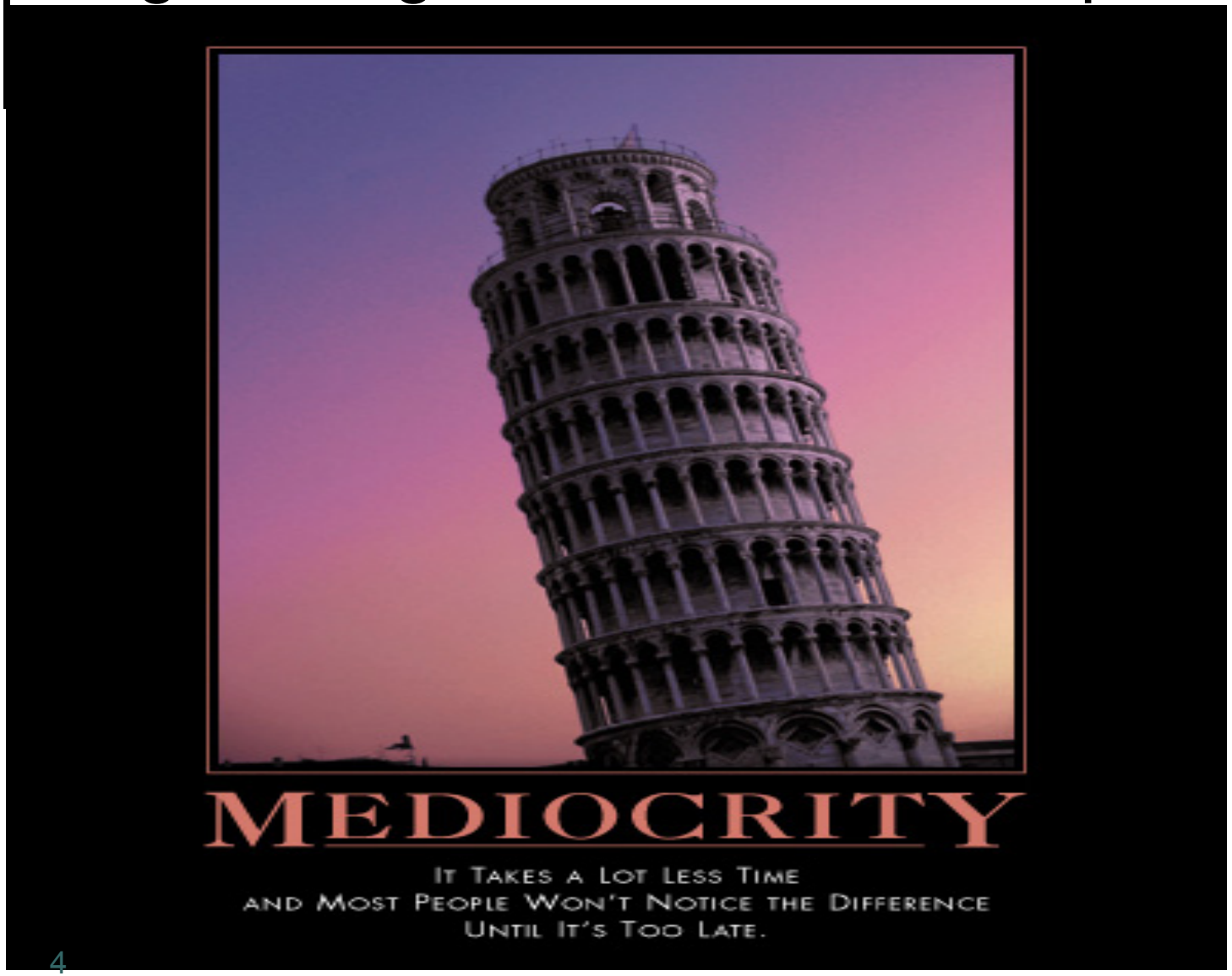
This is a general approach to **all** software development, not just accelerator control system software





Software Engineering Implies **Design**!

Engineering: The Counter Example



Software Engineering: The Counter-Example

It works: But do you want to deal with this?

(Polynomial Graphing Program)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define _                              ;double
#define void                                   x,x
#define case(break,default)            break[O]:default[O]:
#define switch(bool)                   ;for(;x<bool;
#define do(if,else)                    inIine(else)>int##if?
#define true                                   (--void++)
#define false                                  (++void--)

char*O=" <60>!?\\\n"_ doubIe[010]_ int0,int1 _ Iong=0 _ inIine(int eIse){int
O1O=!O _ l=!O;for(;O1O<010;++O1O)l+=(O1O[doubIe]*pow(eIse,O1O));return l;}int
main(int booI,char*eIse[]){int I=1,x=-*O;if(eIse){for(;I<010+1;I++)I[doubIe-1]
=booI>I?atof(I[eIse]):!O switch(*O)x++)abs(inIine(x))>Iong&&(Iong=abs(inIine(x
)));int1=Iong;main(-*O>>1,0);}else{if(booI<*O>>1){int0=int1;int1=int0-2*Iong/0
[O]switch(5[O]))putchar(x-*O?(int0>=inIine(x)&&do(1,x)do(0,true)do(0,false)
case(2,1)do(1,true)do(0,false)6[O]case(-3,6)do(0,false)6[O]-3[O]:do(1,false)
case(5,4)x?booI?0:6[O]:7[O])+*O:8[O]),x++;main(++booI,0);}}}
```

Software Engineering

Motivation

Some of the worst-ever software failures

- 1985: Six people died due to radiation overdose in the Therac-25 X-ray therapy apparatus caused by a software bug (logic error).
- 1996: An Ariane 5 spacecraft exploded 36 seconds after take-off. The problem was due to software originally used in its predecessor, the Ariane 4. A directional correction was applied that exceeded the new aerodynamic tolerances.
- 1999: The Mars Climate Orbiter incinerated in the Martian atmosphere because data that was in expressed in English units was entered into software designed for metric units.
- A software error in the investment model used to manage client assets resulted in AXA Rosenburg Group loosing $217 million. (The SEC fined them an additional $25 million when they told investors market volatility rather than software failure was to blame.)
- The original software in the F-16 fighter jet would have inverted the plane when crossing the equator. (Fortunately, this bug was detected via simulation.)

Software Engineering

Engineering has Scope

- Note that we are implicitly referring to *complex software*
  - Large software systems
  - "Bullet proof" software (medical, military, etc.)
  - Financial
- Simple software does not require significant engineering
  - "Hello World"
  - Readback display, calculator, etc.
- Are you building the Whitehouse or a dog house?

The Software Process

Definition

The development *process* is the structure imposed on the development of software products

- You will chose an process strategy (even if you do not chose)
- This strategy dictates the **entire** software life-cycle
  - Inception
  - Elaboration and Design
  - Documentation
  - Implementation
  - Deployment
  - Maintenance
  - Upgrades

Software Process

Strategies

- Back-loaded strategies (reactive)
  - Example: "Quick and Dirty" implementation (no engineering)
    - Software up and running quickly but brittle.
    - Difficult to upgrade and maintain (inflexible)
  - Thus, most resources are spent in the maintenance and upgrade phases

- Front-loaded strategies (engineering-centric)
  - Most resources are spent in scoping, design, and documenting phase
  - Example: Eclipse IDE
    - Difficult to design, but upgrades and maintenance are easy (flexible)

WARNING: Front-loaded strategies tend to scare (traditional) management since no code is being written in the initial phase of the project
This is likely an artifact of the traditional progress metric "lines of code"
(the metric "lines of code" has little meaning in modern software engineering)

Software Process

Plug for Front-Loaded Development

- Imagine building a house without plans
  - I used 1,000 board-feet of lumber. I'm half finished!
- Without a plan you are essential building and designing simultaneously.
  - "Design flaws" are thus discovered late in the project. Usually corrected with a kluge due to the enormous amount of effort needed for a "re-design."
  - Left with brittle, opaque, undocumented code.
- Implementing the code to a design is fast!
  - It's the thinking part that's hard

Everything in the lifecycle of the software depends upon what you do now!

Why it's bad when home owners change their minds about the bathroom's location late in a building project.

Software Process

Progress Metrics

Old software progress metrics are almost meaningless

Imagine building a large software system without a design

- "I've written 100,000 lines of code – I'm half finished!" (Sound familiar?)

With a software design you have natural metrics, as with a house.

- The foundation is poured
- The house is framed
- The electrical is run

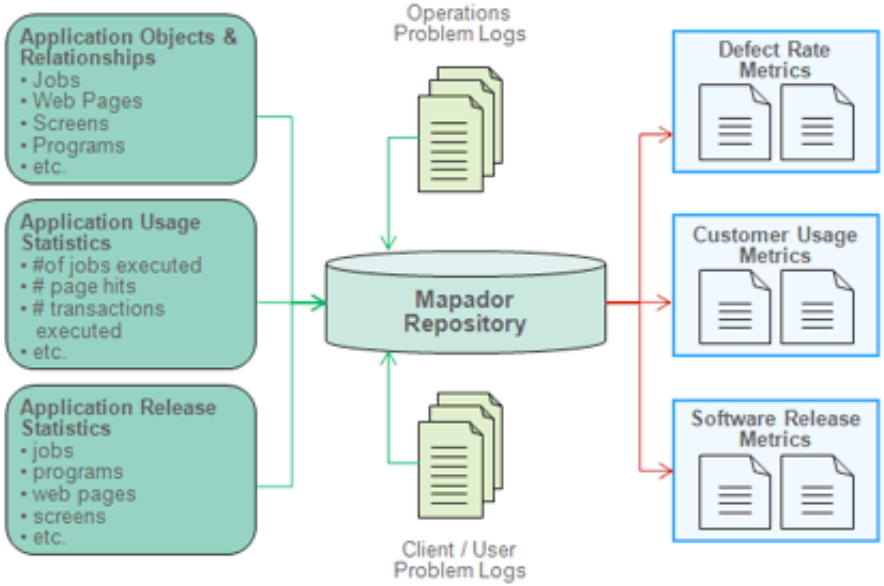
Modern software is component based and assembled with well-defined communication interfaces.

Metrics generally reflect the components of the system

- The motor device interface is designed
- The supervisory control module is implemented
- The user interface is designed

Metrics exist for post release maintenance also!

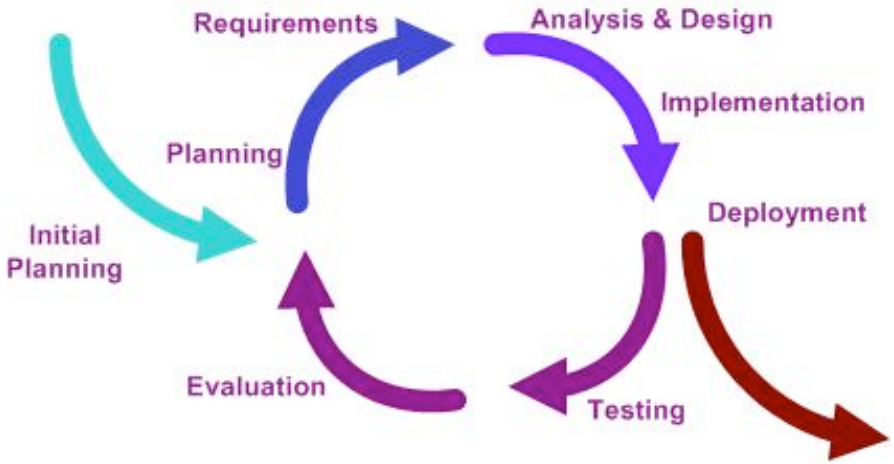
Component-based metric management system for post release



Software Process

Classical Process Phases

- System engineering and analysis
  - What should be done in hardware/software?
- Requirements and analysis
  - What the software should do
- Software design and documentation
  - How to implement it
- Implementation
  - Building it
- Testing and verification
  - Does the software do what it should do?
- Delivery:
  - Out the door, "Over the falls…" (However, requirement change)
  - Iterative delivery (beta, ver. 1, ver. 1.0.1, etc.)



Modern process are typically iterative in their structure.

"Waterfall Model" – obsolete in modern terms due to inflexibility

Software Process

Typical Breakdown for Modern Process

- Time (before deployment):
  - Analysis, Design: 60%
  - Coding: 15%
  - Testing: 25%
- Time: After deployment
  - Maintain/Upgrades: 100%
- Cost
  - Before deployment: 30 - 50%
  - After deployment: 50 – 70%

Software Process

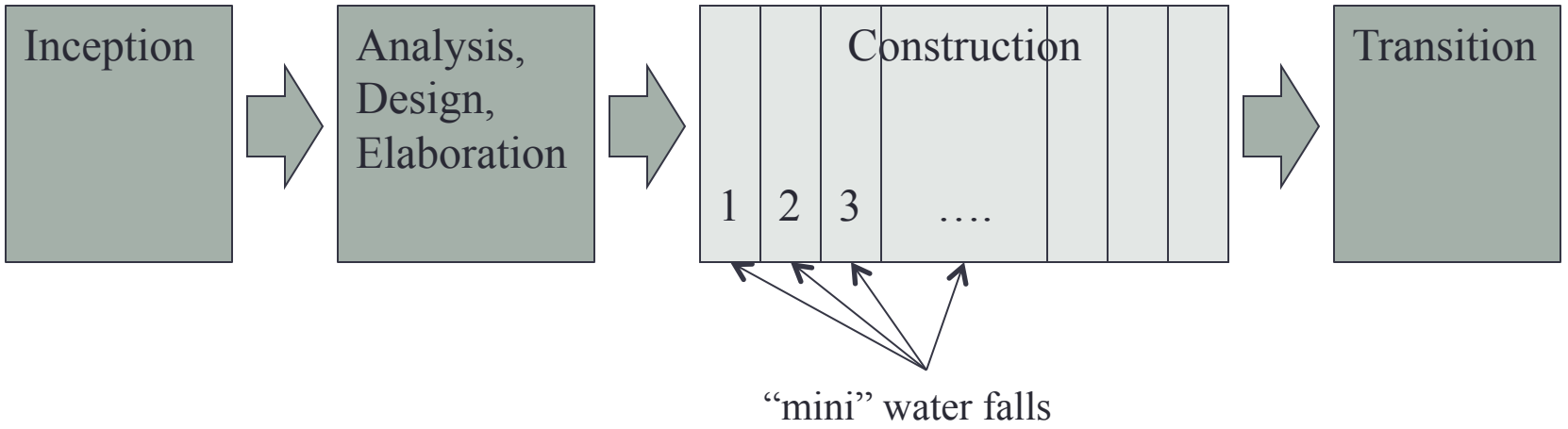
Typical Project Breakdown (cont.)

- Still, for a modern process maintenance is 50-70% of total project cost!
- There will always be upgrades - Why?
  - Fundamental errors in the original system
  - Poor implementation of original system
  - Lack of familiarity with the system
  - Poor documentation of the original system
  - Poor documentation of changes
  - Additional feature requests
- Be careful of jumping into a back-loaded strategy!

Software Process

Modern Process Model

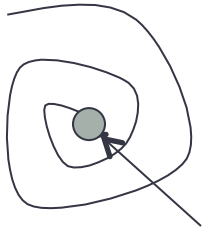
- Most modern (large) software projects employ some type of iterative process model
  - Beta version
  - Release version
  - Correcting original implementation errors
  - Additional feature requests



Software Process

The Spiral" Process Model

- The "Spiral Model" is the most generic process model.
  - Most software process models are special cases
  - Based upon a risk management approach
    - defer elaboration of low risk software elements
    - incorporate prototyping as a risk reduction strategy
    - focus early on reusable software
    - accommodate life-cycle evolution, growth, and requirement changes
    - incorporate software quality objectives into the product
    - focus on early error detection and design flaws
    - set completion criteria for each project activity to answer the question: "How much is enough?"
    - use identical approaches for development and maintenance
    - can be used for hardware-software system development



Now consider engineering in the process

Software Engineering

Engineering Tools for Process Stages

- Inception
- Elaboration
  - **Use cases!**
- Design
  - System requirements
  - Structure diagrams
    - Classes diagrams
    - Component diagrams
  - Behavior diagrams
    - Activity diagrams
    - Interaction diagrams
- Deployment
  - Deployment diagrams

Blue prints

Modeling and Design Tools

Design Patterns

- Standard architecture patterns for common software tasks

UML

- Unified Modeling Language

SysML

- Systems Modeling Language

EMF

- Eclipse Modeling Framework

中文、Deutche, English, Français, 日本語,헌글, etc.

We use UML to design and model our software systems

Software Engineering

Example: Accelerator Control Project Inception

Accelerator project is proposed, designed, modified, etc.

- Somehow a set of requirements is produced involving the control system.
- These requirements, in turn, contain specifications on high-level control

- Know your scope! Examples
  - Upgrade/modification to a single existing control application
  - Implementing a single application on existing system
  - A control requirements change
  - Implementing new control application suite on existing system
    - Size of the suite is important!
  - Design and implementation of a new control system

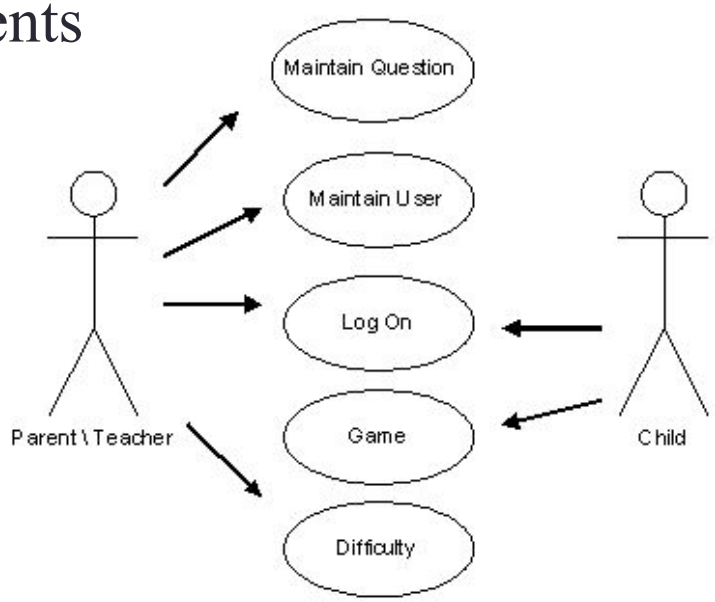
Software Engineering

Requirements ⇔ Elaboration

- Software requirements are essential for design
  - In turn, design strategies are suggested by requirements
  - Elaboration!

- Use Cases help you elaborate requirements
  - Why is the software used?
  - How is the software be used?
  - Is there commonality?

- Use Case Diagrams
  - Visualize software use and requirements
  - Flesh out essential requirements
  - Identify shared and hierarchical functionality



Software Engineering

Use Cases

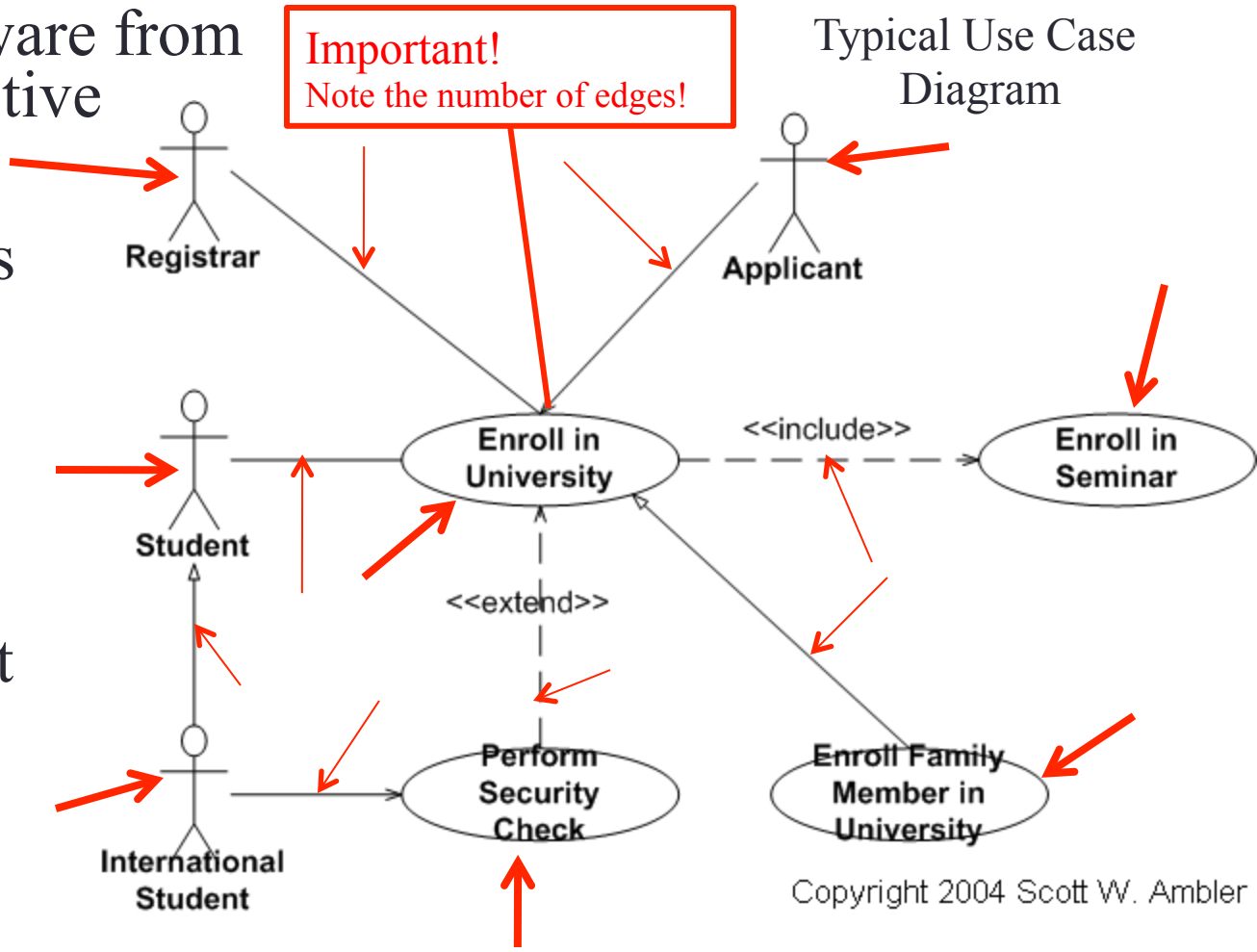
- Viewing the software from the user's perspective

Use Case Diagrams

Components of the diagram

- Actors (roles)
- Scenarios
- Relations

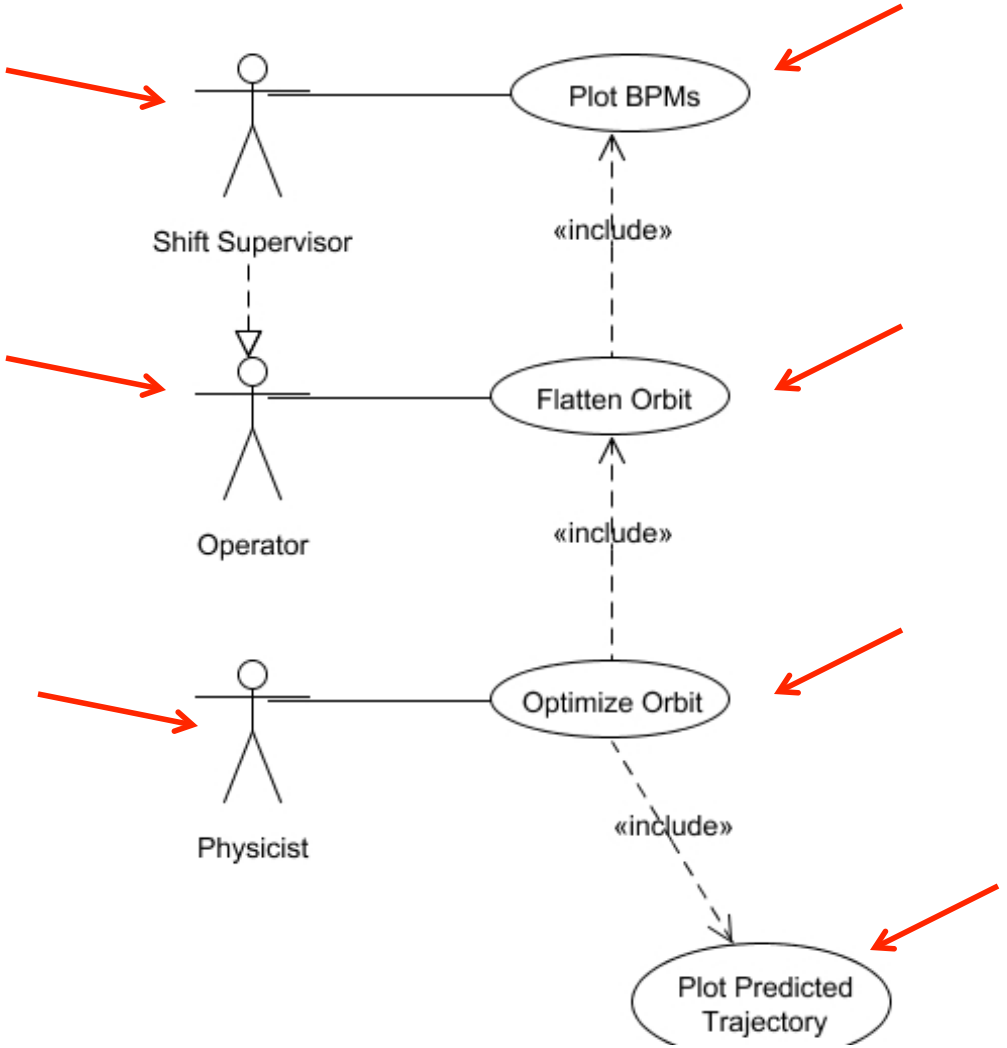
- Identify important activities



Use Cases in Accelerator Control

Example: Orbit Correction Application

- Users
  - Shift supervisor
  - Operator
  - Physicist
- Scenarios
  - Check orbit (Plot BPMs)
  - Flatten orbit
  - Optimize orbit
  - Plot predicted Trajectory

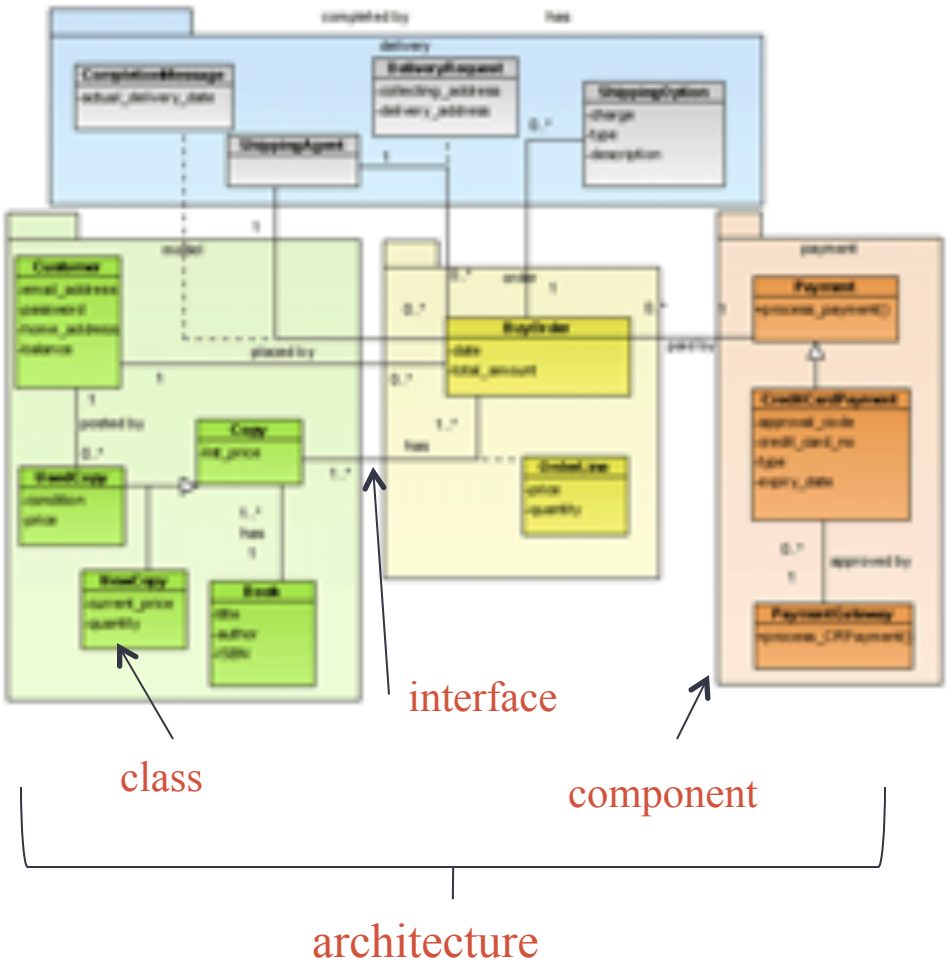


Software Engineering

Design "Blueprints"

Designing systems on paper that supports behavior necessary to perform use cases

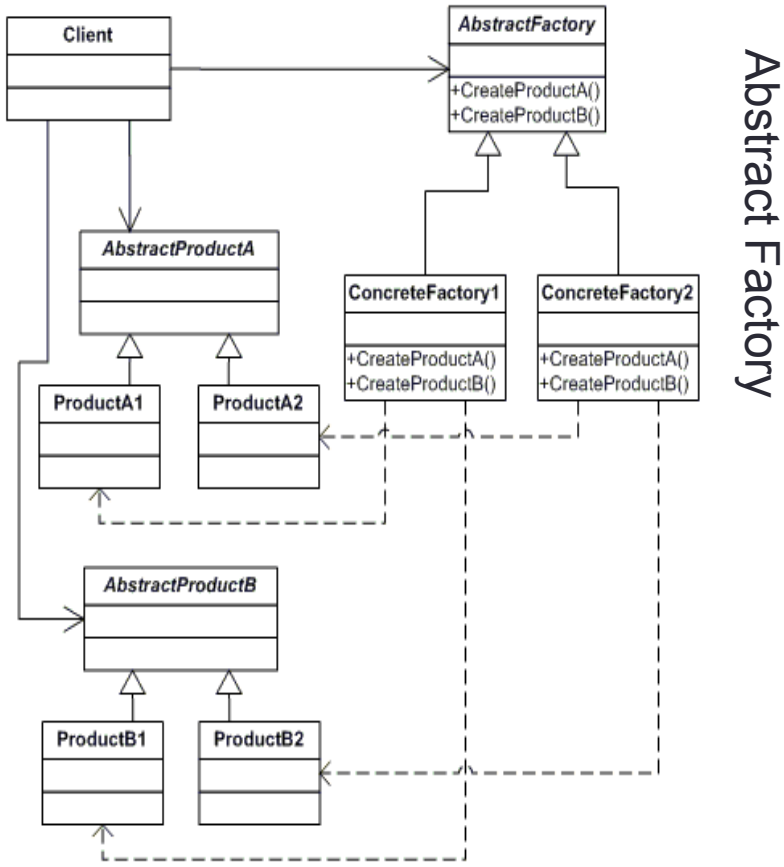
- Architecture
  - Overall structure of software
- Components
  - Autonomous blocks of code performing common task
- Interfaces
  - Communication protocols by which components interact
- Classes
  - Atomic units of code from which components are built
  - Basic elements of the problem domains



Software Engineering

Design Patterns

- Design Patterns
  - Common architectural solutions to common engineering tasks
  - Analogous to amplifiers, DSPs, A/D converters, etc.



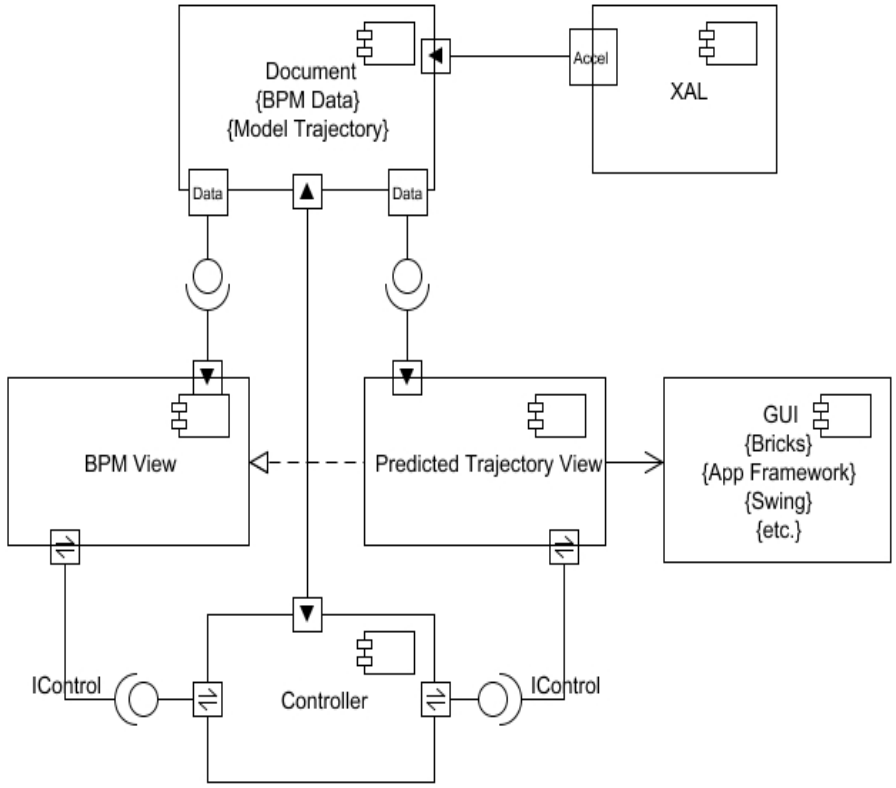
Example: Architecture

Design Blueprints for an Accelerator Application (Orbit Flattening)

Document/View/Controller Design Pattern

- XAL Application Framework supports this architecture for building applications
  - *Document* centralizes and encapsulates the application's data
  - *Views* provide different perspectives of the data
  - *Controller* consolidates interaction between users, views, and data.

Component Diagram
Orbit Display Application Design

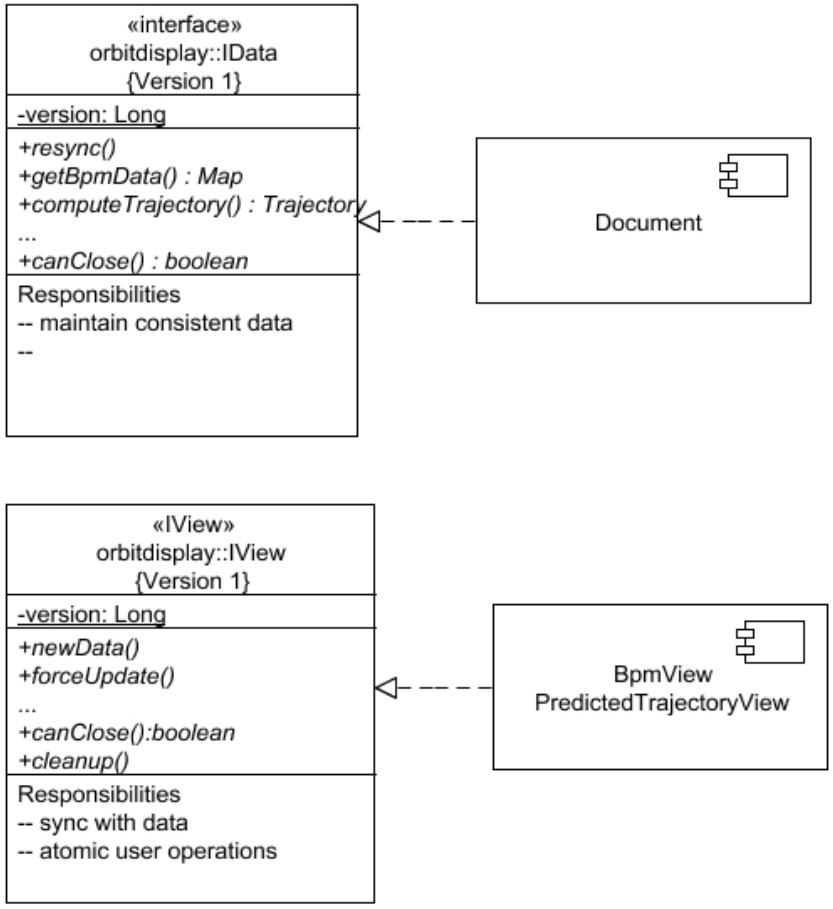


Note this is a high-level model, the components themselves require signification attention. All the more importance of getting this part right.

Software Engineering: Interfaces

Example: The IControl and IData Interfaces

- Software Interfaces are one of the most design intensive software entities
  - Must provide well-defined, accommodating, connections between software components
  - The capabilities of your software component is then *only as good as your interface allows it*
  - You must anticipate and accommodate!



Software Engineering

Modeling **Behavior**

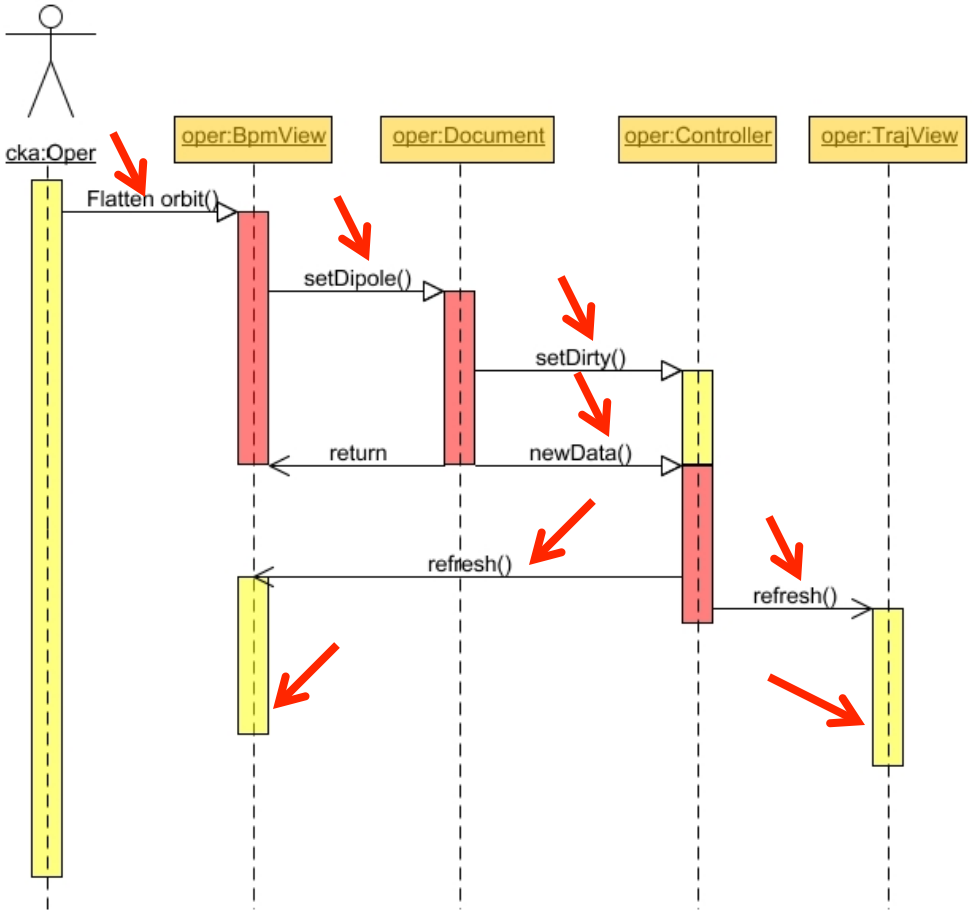
- There are a variety of techniques for abstracting and visualizing software behavior (especially in UML)
  - **Activity Diagram**: Depicts high-level processes, including data flow, to model the complex logic within a system
  - **Sequence Diagram**: Models the sequential logic, in effect the time ordering of messages between classifiers.
  - **State Machine Diagrams**: Describes the state an object or interaction may be in, as well as the transitions between states
  - **Timing Diagram**: Depicts the change in state or condition of a classifier instance or role over time. Typically used to show the change in state of an object over time in response to external events

Software Engineering

Sequence Diagram

Operator Invoked Orbit Flattening

- Operator invokes flattening command
- View computes new corrector values and sends them to document
- Document notifies controller that we are currently inconsistent
- Once correctors are set, document notifies controller there is new configuration
- Controller notifies all views that the data has changed
- Views perform any necessary updates



Software Engineering
Implementation

With software blueprints, implementation takes about 15-30% of the effort.

Before modern software techniques, design and testing phases were skipped (at least formally)

- Jump immediately into "coding" (how many lines of code?)
- However, design has always occupied most of the effort

They were simply coding and designing simultaneously

This is analogous to building a house without blueprints

- No documentation
- No meaningful metrics for progress, operation, etc.
- Weak integration
- Does not support team development

"Which component are you working on?"

"What's a component?"

Software Engineering

Testing (Quality control)

- Critical software must be tested
  - Verification: Is it doing what is supposed to do?
  - Validation: Does it compute the answer

  For example

  (verification): My new simulation code gives the same results as ORBIT

  (validation): My new simulation code is reproducing experimental results

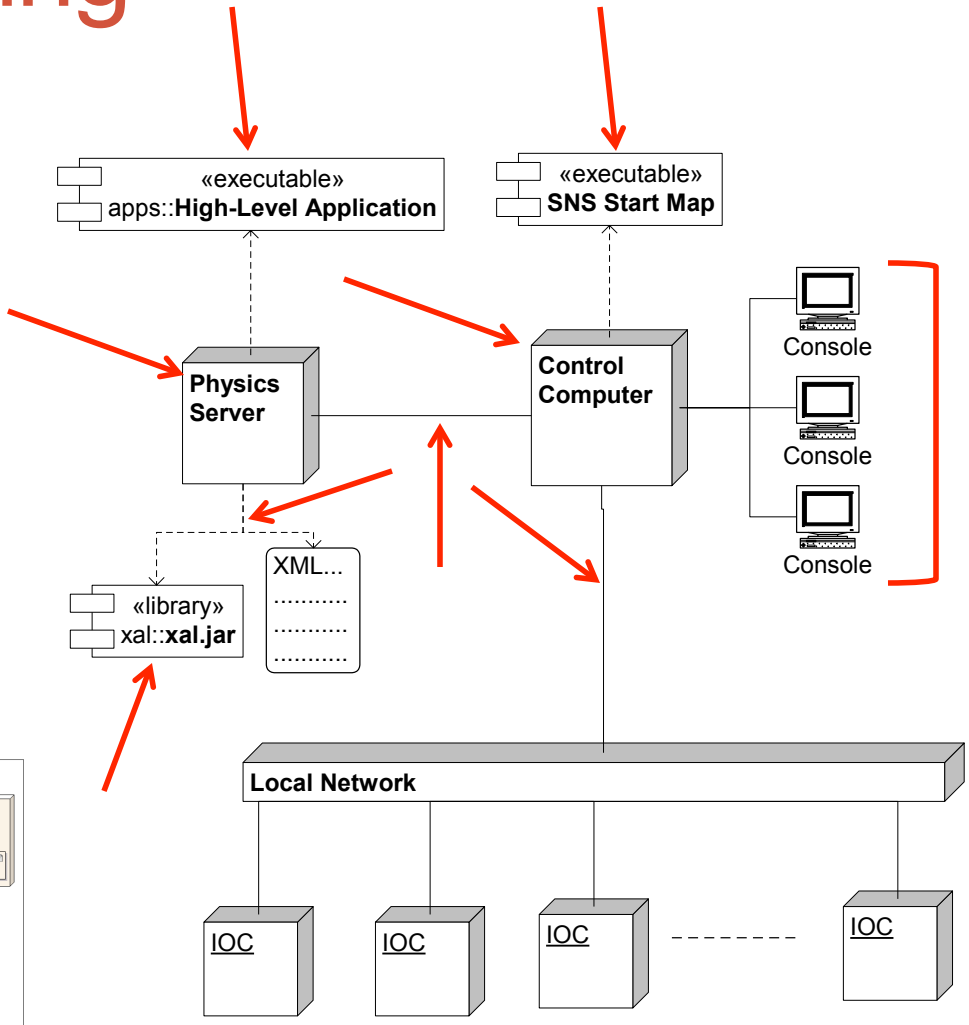
- New software must be tested for compatibility against existing software
  - Test suites expedite this process
  - JUnit is a standardized method for created Java based test suites

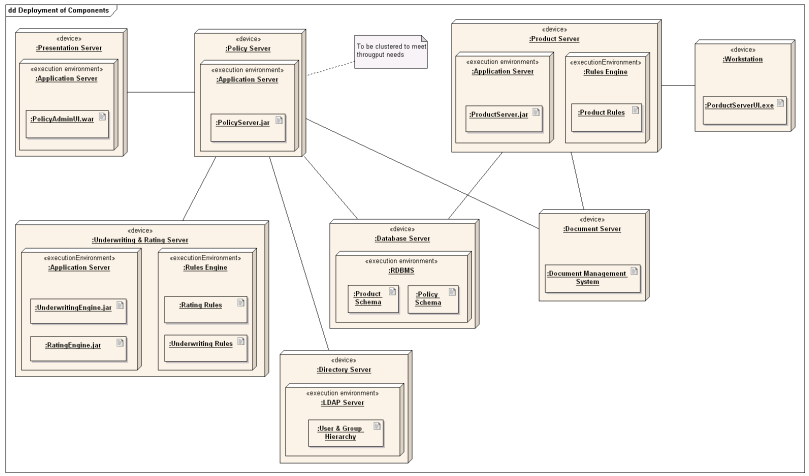
- Continuous Integration Environments
  - Integrated source management, testing, and building (e.g., Maven, Jenkins, Hudson, etc.)
  - New source is automatically verified against a project test suite when committed and/or during builds
  - Projects, code, versioning and builds are kept consistent and reports of its state are continually generated

Software Engineering

Deployment

- Deployment Diagrams
  - Design/Visualize the physical system
  - Diagram shows
    - what hardware components exist
    - which software components run on each node
    - how the pieces are connected (JDBC, RMI, etc.)





Software Engineering

Iteration and Version Control

Last (Important) Word on Software Engineering

- Breaking from the Waterfall – Refactoring, upgrades, and maintenance
  - Requirements change
  - Original implementation errors
  - Features are added
  - Improvements due to unfamiliarity with original system

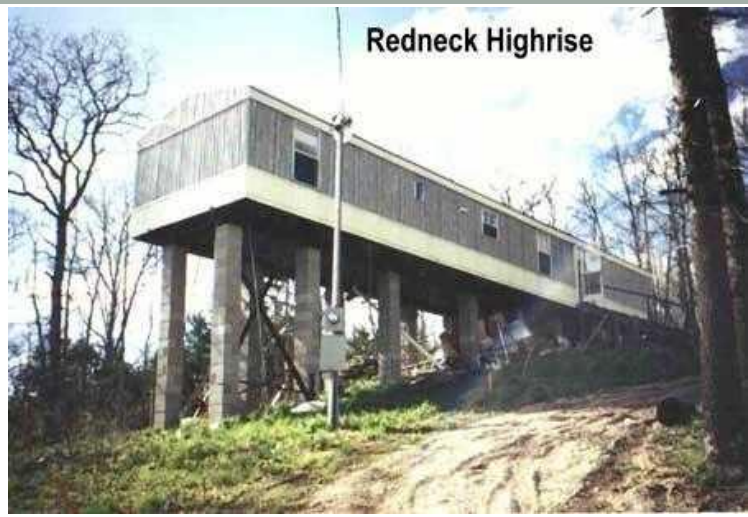
- Thus, version control is critical
  - Tools such as CVS, Subversion, Git, Mercurial, etc. provide an environment for maintaining consistent software versioning
  - These tools also enable team development, collaboration, historical record, site extensions, etc.

- Continuous integration tools can simplify the version control of software projection (once configured)

Summary

Software Engineering "Rules of Thumb"

- Develop software iteratively
- Manage requirements
- Work out use-cases
- Use component-based architectures
- Visually model software
- Verify software quality
- Version control and integration



Design and build a solid foundation for complex software systems