Software Engineering
CS5704: Class 8 - 3/15/01

Instructor: Shawn A. Bohner
Voice: (703) 538-8374
Email: bohner@nvc.cs.vt.edu

Teaching Assistant: Sapna George
Voice: (703) 538-8381
Email: sgeorge@vt.edu

Agenda

▲ Mid-Term Overview
  ● Turn in Homework and Project Team Results
  ● Discussion
▲ Chapter 12 – Analysis Modeling
  ● Break (may be mid-way through)
▲ Chapter 13 – Design Concepts and Principles
▲ Homework Assignments
Spring Semester Timeline

Class Begins & Process
Jan — Feb — Mar — Apr — May

Analysis, Design, & Architecture
PM Metrics & Estimation
Midterm Exam
Testing, Techniques
Cross-Life-Cycle Process

Maintenance & Evolution
Final Exam

Advanced SWE Topics
Object-Oriented Development

SW Metrics & Testing Strategies

Project Management

7 weeks, 7 sessions to go… we’ve come a long way…
So much to do and so little time…

Midterm Grading Statistics

<table>
<thead>
<tr>
<th>Adjusted Range</th>
<th>No.</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>92 - 100</td>
<td>7</td>
<td>A</td>
</tr>
<tr>
<td>88 - 91</td>
<td>6</td>
<td>A-</td>
</tr>
<tr>
<td>84 - 87</td>
<td>6</td>
<td>B+</td>
</tr>
<tr>
<td>80 - 83</td>
<td>8</td>
<td>B</td>
</tr>
<tr>
<td>70 - 79</td>
<td>7</td>
<td>B-</td>
</tr>
<tr>
<td>60 - 69</td>
<td>5</td>
<td>C</td>
</tr>
<tr>
<td>50 - 59</td>
<td>2</td>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Range</th>
<th>No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>90 - 100</td>
<td>9</td>
</tr>
<tr>
<td>80 - 89</td>
<td>18</td>
</tr>
<tr>
<td>70 - 79</td>
<td>7</td>
</tr>
<tr>
<td>60 - 69</td>
<td>5</td>
</tr>
<tr>
<td>50 - 59</td>
<td>2</td>
</tr>
<tr>
<td>40 - 50</td>
<td>0</td>
</tr>
<tr>
<td>&lt; 40</td>
<td>0</td>
</tr>
</tbody>
</table>

Average 81.66%
Median 83.00%
Std. Deviation 0.100862
Minimum 57.00%
Maximum 98.00%
Midterm Problem 5

▲ Software deteriorates rather than wears out because:
A. Software must respond to a changing environment
B. Defects are more likely to arise after software has been used often
C. Multiple change introduce errors in component interactions
D. Software spare parts become harder to order

Midterm Problem 18

▲ The ISO quality assurance standard that applies to software engineering is:
A. ISO 9000
B. ISO 9001
C. ISO 9002
D. ISO 9003

▲ Didn’t cover it in reading assignments so, all answers are right.
**Midterm Problem 36**

▲ Risk projection attempts to rate each risk in two ways:

A. Likelihood and cost  
B. Likelihood and impact  
C. Likelihood and consequences  
D. Likelihood and exposure

---

**Midterm Problem 56**

▲ The Rapid Application Development model is:

A. Another name for Component-based Development  
B. A useful approach when a customer can’t define requirements clearly  
C. A high-speed adaptation of the linear sequential model  
D. All of the above
Midterm 61. What is Software and why is software engineering needed?

▲ Software is part of a computer system that is intended to change
▲ Software is a product that is engineered, not manufactured
▲ Software is intangible
▲ Software is complex
▲ Software engineering is needed to systematically address the complexity and size of software products as they are developed and evolved


Defined: The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.

Defined
▲ Process metrics focus
▲ High predictability
▲ Process => Success
▲ Medium Risk
Midterm 63. Briefly describe the Software Planning Steps.

▲ Scoping—understand the problem and the work that must be done

▲ Estimation
  ● How much effort, cost, and time?

▲ Risk
  ● What can go wrong? How can we avoid it? What can we do about it?

▲ Schedule
  ● How do we allocate resources along the timeline?
  ● What are the milestones?

▲ Control strategy
  ● How do we control quality?
  ● How do we control change?

Midterm 64: 5 Examples of Umbrella Activities (AKA Cross-Life-Cycle Activities)

▲ Software project management
▲ Formal technical reviews
▲ Software quality assurance
▲ Software configuration management
▲ Document preparation and production
▲ Reusability management
▲ Measurement
▲ Risk management
▲ ...
Midterm 65: Draw Incremental Model, what is an Increment?

- System/Information engineering
  - analysis → design → code → test
  - delivery of 1st increment

- Increment 1

- Increment 2
  - analysis → design → code → test
  - delivery of 2nd increment

- Increment 3
  - analysis → design → code → test
  - delivery of 3rd increment

- Increment 4
  - analysis → design → code → test
  - delivery of 4th increment

Calendar time

Midterm 66. Draw RAD... when is it useful?

▲ Tight schedule
▲ Highly compartmentalized functionality
▲ Available team resources
▲ Opportunities for concurrent efforts

- Business modeling
- Data modeling
- Process modeling
- Application generation
- Testing & turnover

60 - 90 days
Midterm 67. Describe Parametric Estimate Approach – Pros and Cons

- A parametric estimation relates cost of a system to one or more parameters such as physical or performance characteristics
  - Moderate to high level of detail
  - Based on actual costs of many systems
  - Uses system parameters to estimate costs

**Advantages**
- Sensitive to significant design changes
- Quantifies effects of cost drivers
- Based on “real world” experience of many systems
- Gives quick, reproducible results

**Disadvantages**
- Inputs are subjective
- Results not as precise as bottom-up
- Requires skilled analyst to develop

---

Midterm 68: What are Metrics Used for? Project and Process examples?

- **Solving problems** — Which choice or improvement should be made?
- **Getting attention** — What situations need to be addressed?
- **Keeping score** — How well is it (or IT) doing?

**Project Management**
- Estimates vs. Actuals
  - Size (change and build)
  - Cost/Budget
  - Effort/Schedule
- Risks (impact & exposure)
  - Resource Availability ...

**Process Management**
- Throughput - rate of changes delivered
- Cost of operations
- Inventory
- # of concurrent changes
- Testing Efficiency
Midterm 69. How to develop a task set.
▲ Task set - tasks, milestones, and deliverables
   ● Functional decomposition provides input
   ● Process provides guidance in doing the task set
▲ Determine type of project
▲ Assess degree of rigor required
▲ Select appropriate software engineering tasks

Midterm 70. Outline Requirements
Engineering parts in sequence
▲ Elicitation — determining what the customer requires
▲ Analysis & negotiation — understanding requirements and shaping relationships to achieve a successful result
▲ Requirements specification — building a tangible model of requirements
▲ System Modeling — building a representation of requirements that can be assessed
▲ Validation — reviewing / assessing the model for correctness, completeness, and consistency
▲ Management — identify, control and track requirements and the changes that will be made to them
Chapter 12
Analysis Modeling

Purpose

▲ The purpose of this chapter is to examine analysis modeling techniques
  ● Entity relationship diagrams to for information models
  ● Data flow diagrams to represent functional models
  ● State transition diagrams for behavioral models

▲ Objectives
  ● Outline key analysis principles through representational models
  ● Illustrate modeling representations
  ● Examine key aspects of analysis modeling
Structured Analysis and Design
Starts with Statement of Scope

▲ A relatively brief description of the system to be built indicating:
  ● Data that are input and output and basic functionality
  ● Conditional processing (at a high level)
  ● Certain constraints and limitations

▲ From these we derive data, functional, and behavioral models

Identifying Objects and Operations

▲ Define “objects” by underlining all nouns in the written statement of scope
  ● Producers/consumers of data
  ● Places where data are stored
  ● “Composite” data items

▲ Define “operations” by circling all active verbs
  ● Processes relevant to the application
  ● Data transformations

▲ Consider other “services” that will be required by the objects
Why Model Data?

▲ Answer questions about the data to be processed
▲ Examines data objects independently of processing
▲ Focuses attention on the data domain
▲ Creates a model at the customer’s level of abstraction
▲ Indicates how data objects relate to one another

What is a Data Object?

Object — something that is described by a set of attributes (data items) and that will be manipulated within the software (system)

Each instance of an object (e.g., a book) can be identified uniquely (e.g., ISBN #)

Each plays a necessary role in the system (i.e., the system could not function without access to instances of the object)

Each is described by attributes that are themselves data items
Typical Objects

▲ External entities (printer, user, sensor)
▲ Things (e.g., reports, displays, signals)
▲ Occurrences or events (e.g., interrupt, alarm)
▲ Roles (e.g., manager, engineer, salesperson)
▲ Organizational units (e.g., division, team)
▲ Places (e.g., manufacturing floor)
▲ Structures (e.g., employee record)

Data Attributes and Relationships

A data object contains a set of attributes that act as an aspect, quality, characteristic, or descriptor of the object.

object: Car
attributes:
- make
- model
- body type
- price
- options code

object: Client
attributes:
- Name
- Address
- Credit

<Orders>

Relationship — indicates “connectedness”; a "fact" that must be "remembered" by the system and cannot or is not computed or derived mechanically.
Entity Relationship Diagrams (ERD) Notation

One common form:

![Diagram](image1)

Another common form:

![Diagram](image2)

Building an ERD

- Level 1—model all data objects (entities) and their “connections” to one another
- Level 2—model all entities and relationships
- Level 3—model all entities, relationships, and the attributes that provide further depth
An ERD Example

Why Model Function?

▲ Answer questions about the processing of data
▲ Focuses attention on the data transforms
▲ Creates a model at multiple levels of abstraction
▲ Indicates how data flows through the computer system
Data Flow Modeling Notation

External Entity: A producer or consumer of data
Examples: a person, a device, a sensor, computer-based system

*Data must always originate somewhere and must always be sent to something*

Process: A data transformer
Examples: compute taxes, determine area, format report, display graph

*Data must always be processed in some way to achieve system function*

(Continued)

Data flows through a system, beginning as input and be transformed into output

Data is often stored for later use
Data Flow Diagramming: Guidelines

▲ All icons must be labeled with meaningful names
▲ The DFD evolves through a number of levels of detail
▲ Always begin with a context level diagram (also called level 0)
▲ Always show external entities at level 0
▲ Always label data flow arrows
▲ Do not represent procedural logic

Context: Level 0 DFD Example

▲ Review ERD to isolate data objects and grammatical parse to determine "operations"
▲ Determine external entities (producers and consumers of data)
▲ Create a level 0 DFD
Constructing a DFD—II

▲ Write a narrative describing the transform
▲ Parse to determine next level transforms
▲ “Balance” the flow to maintain data flow continuity
▲ Develop a level 1 DFD
▲ Use a 1:5 (approx.) expansion ratio

Data Flow Modeling Observations

▲ Each bubble is refined until it does just one thing
▲ The expansion ratio decreases as the number of levels increase
▲ Most systems require between 3 and 7 levels for an adequate flow model
▲ A single data flow item (arrow) may be expanded as levels increase (data dictionary provides information)
Why Model Behavior of System?

▲ Answer questions about the processing events, states, timing, etc.
▲ Focuses attention on the sequence of events and application behaviors
▲ Indicates what events trigger behaviors in the computer system

Behavioral Modeling

▲ Make a list of the different states of a system
  ● How does the system behave?
▲ Indicate how the system makes a transition from one state to another
  ● How does the system change state?
  ● indicate event
  ● indicate action
▲ Draw a state transition diagram (STD)
State Transition Diagram Notation

▲ State—a set of observable circumstances that characterizes the behavior of a system at a given time
▲ State transition—the movement from one state to another
▲ Event—an occurrence that causes the system to exhibit some predictable form of behavior
▲ Action—process that occurs as a consequence of making a transition

Example State Transition Diagram

- State—full
- New State—reading operator commands
- Event causing transition—full
- Action that occurs—reading operator commands
- Event—copies done
- Action that occurs—copies done
- Event—empty
- Action that occurs—empty
- Event—problem state
- Action that occurs—problem state
- Event—jammed
- Action that occurs—jammed
- Event—not jammed
- Action that occurs—not jammed
The Control Model

- The control flow diagram is "superimposed" on the DFD and shows events that control the processes noted in the DFD
- Control flows—events and control items—are noted by dashed arrows
- A vertical bar implies an input to or output from a control spec (CSPEC) — a separate specification that describes how control is handled
- A dashed arrow entering a vertical bar is an input to the CSPEC
- A dashed arrow leaving a process implies a data condition
- A dashed arrow entering a process implies a control input read directly by the process
- Control flows do not physically activate/deactivate the processes—this is done via the CSPEC

Control Specification (CSPEC)

*The CSPEC can be:*

- State Transition Diagram (sequential spec)
- State Transition Table
- Decision Tables
- Activation Tables

Combinatorial Spec
Guidelines for Building a CSPEC

- List all sensors that are "read" by the software
- List all interrupt conditions
- List all "switches" that are actuated by the operator
- List all data conditions
- Recalling the noun-verb parse that was applied to the software statement of scope, review all "control items" as possible CSPEC inputs/outputs
- Describe the behavior of a system by identifying its states; identify how each state is reached and defines the transitions between states
- Focus on possible omissions ... a very common error in specifying control, e.g., ask: "Is there any other way I can get to this state or exit from it?"

---

Process Specification (PSPEC)

- Process
- PSPEC
  - Narrative
  - Pseudocode (PDL)
  - Equations
  - Tables
  - Diagrams and/or charts
Real-Time Analysis & Design Methods

▲ While each method introduces its own notation, all:
● Propose an approach for representing control and separating it from data
● Provide a mechanism for specifying events, states, and distributed processing
● Begin at the analysis level and work to derive processor assignments, tasks and program architectures


The Data Dictionary

☐ a quasi-formal grammar for describing the content of data that the software will process and create

☐ a notation for describing control data and the values that control data can take, e.g., "on," or "off"

☐ a repository that also contains "where-used" / "how used" information

☐ a notation that can be represented manually, but is best developed using CASE tools
**Data Dictionary Example**

<table>
<thead>
<tr>
<th>Name:</th>
<th>telephone number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aliases:</td>
<td>phone number, number</td>
</tr>
<tr>
<td>Where/How used:</td>
<td>read-phone-number (input) display-phone-number (output) analyze-long-distance-calls (input)</td>
</tr>
<tr>
<td>Description:</td>
<td>telephone no. = [ local extension</td>
</tr>
<tr>
<td>Format:</td>
<td>alphanumeric data</td>
</tr>
</tbody>
</table>

**Specification Guidelines**

- Use a layered format that provides increasing detail as the "layers" deepen.
- Use consistent graphical notation and apply textual terms consistently (stay away from aliases).
- Be sure to define all acronyms.
- Be sure to include a table of contents; ideally, include an index and/or a glossary.
- Write in a simple, unambiguous style (see "editing suggestions" on the following pages).
- Always put yourself in the reader's position, "Would I be able to understand this if I wasn't intimately familiar with the system?"
Specification Guidelines (continued)

Be on the lookout for persuasive connectors, ask why?
   keys: certainly, therefore, clearly, obviously, it follows that ...

Watch out for vague terms
   keys: some, sometimes, often, usually, ordinarily, most, mostly ...

When lists are given, but not completed, be sure all items are understood
   keys: etc., and so forth, and so on, such as

Be sure stated ranges don't contain unstated assumptions
   e.g., Valid codes range from 10 to 100. Integer? Real? Hex?

Beware of vague verbs such as handled, rejected, processed, ...

Beware "passive voice" statements
   e.g., The parameters are initialized. By what?

Beware "dangling" pronouns
   e.g., The I/O module communicated with the data validation module and
   its control flag is set. Whose control flag?

Specification Guidelines (continued)

When a term is explicitly defined in one place, try
   substituting the definition for other occurrences of the term

When a structure is described in words, draw a picture

When a structure is described with a picture, try to redraw
   the picture to emphasize different elements of the structure

When symbolic equations are used, try expressing their
   meaning in words

When a calculation is specified, work at least two
   examples

Look for statements that imply certainty, then ask for proof
   keys; always, every, all, none, never

Search behind certainty statements—be sure restrictions
   or limitations are realistic
Chapter 13
Design Concepts and Principles

Purpose

▲ The purpose of this chapter is to describe fundamental design principles that are essential to understanding software design methods

● Basic concepts are introduced and a fundamental design model is discussed.
● The design model consists of the data design, architectural design, interface design, and component-level design

▲ Objectives

● Outline key software design principles
● Examine key aspects of software design at architecture, interface, and component-level design
Mapping Analysis to Design

Where Do We Begin?
Key Design Principles

▲ The design process should not suffer from ‘tunnel vision.’
▲ The design should be traceable to the analysis model.
▲ The design should not reinvent the wheel.
▲ The design should “minimize the intellectual distance” between the software and the problem as it exists in the real world.
▲ The design should exhibit uniformity and integration.
▲ The design should be structured to accommodate change.
▲ The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered.
▲ Design is not coding, coding is not design.
▲ The design should be assessed for quality as it is being created, not after the fact.
▲ The design should be reviewed to minimize conceptual (semantic) errors.

From Alan Davis [DAV95]

Fundamental Design Concepts

▲ Abstraction—data, procedure, control
▲ Refinement—elaboration of detail for all abstractions
▲ Modularity—compartmentalization of data and function
▲ Architecture—overall structure of the software
  ● Structural properties
  ● Extra-structural properties
  ● Styles and patterns
▲ Procedure—the algorithms that achieve function
▲ Hiding—controlled interfaces
Data Abstraction

Door

- manufacturer
- model number
- type
- swing direction
- inserts
- lights
  - type
  - number
- weight
- opening mechanism

implemented as a data structure

Procedural Abstraction

open

details of enter algorithm

implemented with a "knowledge" of the object that is associated with enter
Stepwise Refinement

open

walk to door; reach for knob;
open door; walk through; close door.

repeat until door opens
turn knob clockwise;
if knob doesn't turn, then
take key out;
find correct key;
insert in lock;
endif
pull/push door
move out of way;
end repeat

Modular Design

easier to build, easier to change, easier to
Modularity: Trade-offs

What is the "right" number of modules for a specific software design?

Sizing Modules: Two Views

What's inside??

How big is it??
Functional Independence

COHESION  -  the degree to which a module performs one and only one function.

COUPLING  -  the degree to which a module is "connected" to other modules in the system.

Architecture

“The overall structure of the software and the ways in which that structure provides conceptual integrity for a system.”

Mary Shaw [SHA95a]

Structural properties. This aspect of the architectural design representation defines the components of a system (e.g., modules, objects, filters) and the manner in which those components are packaged and interact with one another. For example, objects are packaged to encapsulate both data and the processing that manipulates the data and interact via the invocation of methods.

Extra-functional properties. The architectural design description should address how the design architecture achieves requirements for performance, capacity, reliability, security, adaptability, and other system characteristics.

Families of related systems. The architectural design should draw upon repeatable patterns that are commonly encountered in the design of families of similar systems. In essence, the design should have the ability to reuse architectural building blocks.
Why Use Information Hiding?

▲ Reduces the likelihood of “side effects”
▲ Limits the global impact of local design decisions
▲ Emphasizes communication through controlled interfaces
▲ Discourages the use of global data
▲ Leads to encapsulation—an attribute of high quality design
▲ Results in higher quality software

Homework Assignment for 3/23/01

▲ Read Pressman Chapters
  ● Chapter 20
  ● Chapter 21
▲ Continue Refining Requirements
  Specification for “Dog E-Dating System”
  ● Data Model, data flows, state-transition diagrams, use cases
  ● Extra Credit – Comprehensive CSPEC
▲ Have a great week!