What is Software Engineering?

- **Software Systems**
  - "An integrated whole composed of diverse, interacting, specialized structures and subfunctions." [IEEE]
- **Software Engineering**
  - Disciplined systematic technological activity for producing and maintaining software products by means of a controlled efficient process.
- **Software Engineering is NOT programming!**

Development Goals

- **Programming**
  - Goal: Write Code

Why Design?

<table>
<thead>
<tr>
<th>Number of Modules</th>
<th>Cost per Component</th>
<th>Interfacing Cost</th>
<th>Integration Cost</th>
<th>Total Cost</th>
<th>System Development Cost Relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What does the code do?

- **Developer**
  - Code
  - ??
Ah, now I know!

**Design Goals**
- Select components
- State Functions
- State Interfaces

Why Should We Document?

*Maintenance*
- Makes up 70%-90% of total system cost
- Majority of maintenance time (50%) spent on understanding -> system documentation
- Maintenance Tasks
  - collection, analysis and prioritization of user trouble reports
  - new system release installations
  - documentation (user’s manuals) changes
  - configuration control issues

Why Test So Much?

Cost of Errors

<table>
<thead>
<tr>
<th>Relative Cost</th>
<th>Design</th>
<th>Code</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Percentage of Time to Code?

- Planning?   1/3
- Component Testing?  1/4
- System Testing?  1/4
- Coding?  1/6

Software Process: Waterfall Model
Spiral Model cont’d

- Development phases reiterates through four cycles:
  - Set goals and determine constraints for the phase
  - Evaluate and resolve risks for the phase
  - Develop the prototype for the phase
  - Plan the next stage activities
- Prototype: a limited, semi-functional, task restricted, partially operational system
  † Analogous to a model or mockup that allows evaluation of development alternatives before commitment

What is the point?

- Computer Science is more than just about programming
  - Requirements Engineering
  - Design
  - Coding
  - Integration
  - Testing
    * component
    * integration
  - Maintenance
- Software Engineering

Module 1 Outline

- What is Software Engineering?
- What is Requirements Engineering?
- What is Design?
  - Structure Charts
- What is Testing?
  - …

Requirements

- Defining what the system should do
  - What the clients needs (as opposed to wants)
  - Not bow the solution should be designed or implemented
- We recognize three iterative activities:
  - Elicitation: capturing information from sources
  - Documentation: “putting it on paper”
  - Validation: confirming it meets users’ needs

Gathering Requirements

- Many sources of requirements
  - Interviews (structured vs. non-structured)
  - Stakeholder documents
  - Questionnaires
  - Existing or similar systems
  - Standards in that domain
  - Rapid prototypes
- Often information is documented in text
Example: Automated Library System

“Vision” Statement
- You have been contracted to develop a computer system for a university library. The library currently uses a 1960s program, written in an obsolete language, for some simple bookkeeping tasks, and a card index, for user browsing. You are asked to build an interactive system which handles both of these aspects online.

Example: More Details on Needs

Books and Journals
- Library contains books and journals
- Books may have several copies
- Some books are short-term loans, all others 3 weeks
- Only staff can borrow journals
- Members can borrow up to 6 items at a time
- Staff can borrow up to 12 items
- New items arrive, old items are disposed of
- Current year’s journals are bound at year-end

Example: More Detail

Borrowing
- System keeps track of when items are borrowed and returned
- Produce reminders when an item is over-due
- (Future) Extend the loan of an item (if not reserved)

Browsing
- Users can search for a book by topic, by author, etc.
- Users can check if item is available
- Can reserve a book
- Anyone can browse the library

User-centered Requirements

From such textual descriptions, could we build a system? How likely would it be to meet users’ needs?

Goal: Center system definition and development on user needs.
- Identify various users of the system
- Define what tasks they undertake with the system, and task outcomes
- Document this in a way that can be used for requirements specification and later in the lifecycle

Use Case Modeling

Use Case:
- “A sequence of actions a system performs to yield an observable result of value to a particular actor.”
- A task which an actor needs to perform with the help of the system

Actor:
- Someone or something outside the system that interacts with the system
- A user of the system in a particular role

Example: Actors and Use Cases

Actors
- BookBorrower
- JournalBorrower
- Browser (person who browses, not SW)
- Librarian

Use Cases
- Borrow copy of a book
- Reserve a book
- Return copy of book
- Borrow journal
- Browse
- Update Catalog
Example Text Description

- **Borrow copy of a book:**
  
  A Bookborrower presents a copy of a book. The system checks that the s/he is a library member, and that s/he has not checked out too many books. If both checks succeed, then the system records that the member now as this copy of the book. Otherwise it refuses the loan.

Module 1 Outline

- What is Software Engineering?
- What is Requirements Engineering?
- What is Design?
  - Structure Charts
- What is Testing?
  - ...

What is Design?

- **Specification** is about *What*, and **Design** is the start of the *How*
- **Inputs** to the design process
  - Specification document, including models etc.
- **Outputs** of the design process
  - A design document that describes how the code will be written. Includes design models!
    - What subsystems, modules or components are used
    - How these integrate (i.e. work together)
  - Information allowing testing of the system

Design Goals

- **Qualities Of A Good Design:**
  - Correct, Complete, Changeable, Efficient, Simple
- **Correctness:**
  - It Should Lead To A Correct Implementation
- **Completeness:**
  - It Should Do Everything. Everything? It should follow the specifications.
- **Changeable:**
  - It Should Facilitate Change—Change Is Inevitable

Design Goals cont’d

- **Efficiency**
  - It Should Not Waste Resources. But:
    - Better A Working Slow Design Than A Fast Design That Does Not Work
- **Simplicity**
  - It Should Be As Understandable As Possible
- Important: A design should fully describe how coders will implement the system in the next phase
  - Designs are blue-prints for code construction

Levels of Design

- **Three possible levels:**
  - System Design, if appropriate
    - Part of Systems Engineering (see below)
  - High-level Software Design
    - Architecture, architectural design
  - Low-level Software Design
    - Detailed Design, Module Design
Describing System Architecture

- Model components and how they interact
- Emphasis on component purpose and interaction
  - Not on internals of how components work (e.g. a black box approach)
- Modern techniques/ideas about architecture
  - Styles of architectures
    - Examples: pipe and filter; layers; etc.
  - Frameworks
    - A partially completed design to which you add new components

Logical Architectural View

- We need a high-level logical view of system architecture and its components
  - Many think nothing in UML is particularly good for this. This is at a higher-level of abstraction than the level of classes.
  - Often we draw a simple “box and line” diagram and explain it.
- Architectural styles may be useful. Examples:
  - pipe and filter; layered; client-server

High-level Design

- Goal: create a software system architecture, defining a system in terms of:
  - components;
  - interactions.
- Examples of components:
  - modules, classes
  - clients, servers
  - files, databases
  - layers

High-level Design cont’d

- Interactions:
  - Structural, behavioral
- Examples of interactions:
  - procedure calls
  - composition of objects
  - sharing variable access

Low Level Design

- Internal workings of algorithms
- Implementation details
- Etc.

Module 1 Outline

- What is Software Engineering?
- What is Requirements Engineering?
- What is Design?
  - Structure Charts
- What is Testing?
- …
Structure Charts

- Structure Chart
  - A graphic tool used to hierarchically model the design solution of a problem.

- Contains:
  - Individual modules
  - Data passed to/from modules
  - Control interfaces between modules

- Does NOT contain a complete representation of the internal structure of individual modules.

Structure Charts cont’d

- Basic Elements
  - Calling Module
  - Called Module
  - Control Relationship

Structure Charts cont’d

- Parameter Direction Flow
  - 1. Input:
    - Value Parameters & Const Reference Parameters
  - 2. Output:
    - Return & Reference Parameters (Function changes parameter values independently of parameter’s original [passed] value.)
  - 3. I/O:
    - Reference Parameters (Function changes parameter values dependent upon of parameter’s original [passed] value.)

Structure Charts cont’d

- Types of Parameters
  - 1. Data Parameter:
    - Any parameter upon which NO decision in the called module’s or calling module’s code is based.
  - 2. Control Parameter:
    - Any parameter upon which a decision in the called module’s or calling module’s code is based.

Structure Charts cont’d

- Structure Charts Example
  - void GetNextChar ( . . . ) {
  -   GetChar (ch, uplow, terminator);
  -   if (terminator) . . .
  - }
  - void GetChar(char& chact, uplowtype uplowCase, bool& terminal)

Label with names of actual parameters. Calls to multiple functions are usually listed left-right in order of execution.
Structure Charts Example cont’d

```c
void Move(char ch, . . .) {
    if (ch == plus)
        DrawBox (row, col);
    ...
}
```

Label decision with name(s) of variables used in decision.

Conditional Call

Move

ch

row

col

_draw Box

Draw

Box

Structure Charts Example cont’d

- if else statements

Move

Conditional Calls

row

col

len

Select statements are diagrammed in a similar manner with multiple calls emanating from the decision diamond.

Structure Charts Example cont’d

- for and while loops

Draw

Box

ch

dir

Draw

Line

Conditional Loop

Structure Charts Example cont’d

- Operating System Calls:
- Recursive Routines:
  - Routines that call themselves
- Predefined Module:
  - (library routines)

Calendar

Date

Factoring

Get File

fexists

Structure Charts Example cont’d

- More than one page:

Find Coord

Convert Coord

Interface Diagram #3

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Dir</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>data</td>
<td>in</td>
<td>Cartesian coords</td>
</tr>
<tr>
<td>y</td>
<td>data</td>
<td>in</td>
<td>of point</td>
</tr>
<tr>
<td>scale</td>
<td>data</td>
<td>in</td>
<td>scale of axis</td>
</tr>
<tr>
<td>anggrad</td>
<td>cont</td>
<td>in</td>
<td>angle degrees (rad)</td>
</tr>
<tr>
<td>angle</td>
<td>data</td>
<td>out</td>
<td>polar coords</td>
</tr>
<tr>
<td>dist</td>
<td>data</td>
<td>out</td>
<td>of point</td>
</tr>
</tbody>
</table>
Structure Charts Finished!

- Global Data
  - Must be highly justifiable
- Operation
  - Assign
  - Update
  - Reference

Module 1 Outline

- What is Software Engineering?
- What is Requirements Engineering?
- What is Design?
  - Structure Charts
- What is Testing? (next set of slides)
- …