Software Process
2/12/01          Lecture #3           16.070

• Overview of the Software Development Process (SWDP)
• Details on the first phase -- Conceiving
• Example of Conceiving
• Designing, Implementing, Operation
• Summary of SWDP

• Learning Objectives:
  ➢ Describe the SW Development Process as a specialization of the generic Product Development Process (PDP)
  ➢ Use the First Phase of SWDP
# Software Engineering Process

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>D</th>
<th>I</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Generic PDP</strong></td>
<td>Needs, Requirements</td>
<td>Analysis, Architecture</td>
<td>Preliminary Design</td>
<td>Detail Design</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Uckan Text</strong></td>
<td>Requirements Specification</td>
<td>Analysis</td>
<td>Design</td>
<td>Implementation, Testing and Verification</td>
</tr>
<tr>
<td>&quot;C&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Laplante Text</strong></td>
<td>Concept</td>
<td>&quot;Partitioning&quot;</td>
<td>Design</td>
<td>Programming, Test</td>
</tr>
<tr>
<td>&quot;R&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Good SWDP is just a specialization of an underlying Generic PDP**
Conceiving: Needs to Requirements

Interpret the Needs (which exist outside your organization)

• Who is the customer? Who benefits?
• What are their needs? What is the opportunity?
• How will they benefit? What value will they derive?

Define the Requirements (which are set within your organization)

• What exactly is the "problem" to be solved?
• What features should be provided?
• What functions should be performed?
• Must include consideration of technical, financial, regulatory viability

Output: A clear, correct, complete and consistent statement of the goals and requirements -- What the system will do.
Conceiving: Analyze the Requirements

High level goal (Mission objective, problem statement)

Functional Requirements

- **Function** - the input-output processes or transformations that must be performed
  - May include performance (e.g., accuracy)
  - Should be in solution neutral-form (e.g., avoid reference to structure; code)
  - Should specifically identify inputs, outputs, internal processes, constraints

- Timing - throughput, latency

- Human Operator - how the user is informed or operates

*Subtly different use than "C function"*
Conceiving: Analyze the Requirements - cont.

Non-Functional Requirements

• Interfaces - with hardware and software

• Resources and constraints - lines of code, budget, schedule, language to be used, processor to be used

• Test and Verification necessary

• Other - overall desires of the designer, often with respect to style, enterprise strategy, competition
Conceiving: Conceptual Design and Architecture

• Develop the high level concept of the Design Solution

• **Partition** the problem into **modules** so that only the function of the module is visible to the other modules - i.e., the modules hide the internal features

• Partitioning should be based on
  - Intensity of information exchange among internal steps
  - Likelihood that internal steps will change
  - Hiding interfaces to hardware and humans

• Interfaces between modules must be "controlled"

**Output:** Functions, concept and high level partition which collectively are known as the architecture
Example - Controlled Flight into Terrain

Cali-Columbia - On a routine descent, a 757 flies into a mountain, killing all aboard

Accident analysis: Pilots became disoriented in unfamiliar terrain, followed flight controller instructions, crashed.

Need: ?
Requirements

Devise a system to inform the pilot of when the path of the aircraft will impact terrain. The system must use conventional "glass cockpit" flight displays, and must be easily understandable. It must integrate into existing on-board processors, and use C. It must inform the pilots of "clear ahead," "getting close" or "impact expected."
Requirements Analysis

High level goal:

Function:

Timing:

Human operator:

Interfaces:

Resources and constraints:

Test and Verification:

Other:
Definition of Functions

• Determine the relative location of terrain to aircraft
  ➢ input = aircraft location, terrain profile

• Determine the expected path of aircraft
  ➢ input = velocity vector, navigation guidance

• Determine the likelihood of Flight into terrain
  ➢ No, maybe, yes

• Display likelihood to pilot
  ➢ Output = warning to pilot
Architecture

Map Functions onto Modules
Designing

- Convert requirements + architecture + interface control into detailed design specification
  - What each module should do
  - How/what it inputs and outputs
  - What algorithms might be used
- Design includes aspects of
  - Logic design (e.g. pseudo-code, flow charts)
  - Data design (data representation)
Designing - cont.

• Includes the design of test cases for verification [assurance that requirements are met]
  ➢ Nominal, special, limiting cases
  ➢ Coverage to insure all code is executed
  ➢ Most likely cases to find errors

• Other design phase decisions
  ➢ Processor to be used
  ➢ Language, compiler
  ➢ Estimates of processor, memory resources

Output: The set of information/knowledge artifacts which completely describe the design, its interfaces and environment/context
Implementing: Program Building Phase

• Build the system by writing the code. Implement, as a program, the algorithm(s) specified in the design document
  ➢ Convert each algorithm step into a programming language statement. Write readable code!
  ➢ Perform manual simulation
  ➢ Examine efficiency of execution time and space

• Compile and Run Program (or module)

• Perform incremental testing (white box testing)
  ➢ Design Errors occur during analysis, design, implementation
  ➢ Syntax errors occur during implementation, detected by compiler
  ➢ Run-time Errors, e.g., divide by 0
Implementing: Integration and Test

- Incremental Integration
- Formal Testing
  - Specified test cases
  - Acceptance criteria set *a priori*
  - No program changes allowed!
- Subsequent Regression Testing
  - Changes to code
  - Changes to Environment

Output - code (or code in embedded system) plus test report verifying performance
Implementing: Documentation

• Document - leave an understandable record

• Facilitates
  ➢ Use of the program
  ➢ Program debugging
  ➢ Future maintenance/upgrades

• Include
  ➢ Requirements specification
  ➢ Description of inputs, outputs, constraints, formulae
  ➢ Pseudocode/flowchart/state transition diagram for algorithm
  ➢ Source code listing
  ➢ Hardcopy of sample test run
  ➢ User’s guide (optional)
Implementing: Verification and Validation

• Verification
  ➢ Are we building the product right?
  ➢ Have we adhered to the requirements?
  ➢ Can be accomplished by program testing to demonstrate correctness

• Validation
  ➢ Are we building the right product?
  ➢ Are the needs captured in the requirements?
  ➢ Are requirements correct? Properly defined and stated?
  ➢ Can often only be done through simulation or with all-up system
Operating

• Once product is deployed, it must be operated throughout its lifetime/use. Training may be necessary

• Maintenance/Upgrades required due to
  ➢ Errors found in software
  ➢ Areas for improvement indentified
  ➢ Fixes required for hardware problems

• Change Process is a (slightly) shortened version of the SWDP
# Generic Process vs Aerospace and 2167A Processes

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>D</th>
<th>I</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Generic PDP</strong></td>
<td>Needs, Requirements, Analysis, Architecture</td>
<td>Preliminary Design, Detail Design</td>
<td>Chunk Building, Testing, Integration, Testing</td>
<td>Operating, Maintaining, Upgrading, Retiring</td>
</tr>
<tr>
<td><strong>Aerospace Reviews</strong></td>
<td>SRR, CoDR</td>
<td>PDR, CDR</td>
<td>&quot;IRR&quot;, FRR or SAR, ORR</td>
<td>Change Reviews</td>
</tr>
<tr>
<td><strong>2167A Reviews</strong></td>
<td>SRR, SDR, SSR</td>
<td>PDR, CDR</td>
<td>TRR, FCA, PCA, FQR</td>
<td>----, ----</td>
</tr>
</tbody>
</table>
Software Development Process Models

• Waterfall
  ➢ Consists of one execution through the development process steps
  ➢ No feedback
  ➢ Works for routine, standard projects

• Spiral
  ➢ Iterate over the process steps multiple times, providing feedback
  ➢ Continue to evolve the same system
  ➢ Recommended

• Rapid Prototyping
  ➢ Start building the system early, based on vague requirements
  ➢ Get a “feel” for product. Provide feedback
  ➢ Build a new version, incorporating feedback and additional functions
Software Development Process Models

- Requirements
- Architecture
- Preliminary Design
- Detailed Design
- Build
- Test
- Operate
- Upgrade

(time)
Recommended Spiral Model

START

Read approach, functions

Homework

Sketch approach, functions

Write simple version of program

Review, test with some data

Write final version of program

Review, test with valid, invalid, unusual data

Do detailed design on functions, algorithms, arguments, error checks, etc.

Re-read homework for all details

END: Turn in homework

Test

Requirements

Design

Build

Created by G. Kocur, MIT 1.00 Lectures
Summary

• Good software results from Good SWDP -- use one

• Conceiving - and in particular requirements and architecture, are where most of the gains and mistakes are made - be very attentive and careful

• Use a spiral model for your homework
## Generic PDP expanded

<table>
<thead>
<tr>
<th>Conceive</th>
<th>Design</th>
<th>Implement</th>
<th>Operate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mission</td>
<td>Conceptual Design</td>
<td>Preliminary Design</td>
<td>Detailed Design</td>
</tr>
</tbody>
</table>