Architecture and Design

Guest Lecture for
COMP 180: Software Engineering
Tufts University
Fall 2006

John Klein
Architecture – Definitions

- [http://www.sei.cmu.edu/architecture/definitions.html](http://www.sei.cmu.edu/architecture/definitions.html)

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.


- The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time.
Practitioner Definitions

- “Software architecture is the set of decisions which, if made incorrectly, may cause your project to be cancelled.”
  – Eoin Woods

- “Decomposition of the problem in a way that allows your development organization to efficiently solve it, considering constraints like organizational structure, team locations, individual skills, and existing assets.”
  – John Klein
Architecture/Design Decisions

Frequently Used Tools in the Software Designer’s Toolbox

• Abstraction
• Separation of Concerns

• Patterns – Architecture, Design, Language & Technology
• Organizational Patterns (see Coplien and Harrison, *Organizational Patterns of Agile Software Development*. Prentice Hall, 2004.)
• Notations – UML, SDL, Traces, Formal Specification Languages (Z, CSP, … ), Predicate Logic
Abstraction
All Software is an Abstraction –
Underneath it all is electrons flowing through semiconductors

- Logic gates...

<table>
<thead>
<tr>
<th></th>
<th>MIN</th>
<th>TYP</th>
<th>MAX</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{IH}$</td>
<td>High-level input voltage</td>
<td>2</td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>$I_{IH}$</td>
<td>High-level input current (@ $V_i = 2.4V$)</td>
<td>40</td>
<td></td>
<td>$\mu A$</td>
</tr>
<tr>
<td>$V_{IL}$</td>
<td>Low-level input voltage</td>
<td>0.8</td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>$I_{IL}$</td>
<td>Low-level input current (@ $V_i = 0.4V$)</td>
<td>-1.6</td>
<td></td>
<td>mA</td>
</tr>
</tbody>
</table>

- Early 4-bit counter (DEC PDP-6, circa 1965)

![Image of an early 4-bit counter](image-url)
Abstractions

• Assembly Language

ALIGN 4 ; 0
PUBLIC _main
_main PROC NEAR @B1@8: ; preds: B1.3

    mov edx, DWORD PTR 12[ebp] ; 2
    mov eax, DWORD PTR 8[ebp] ; 2
    cmp eax, 2 ; 10
    mov edx, DWORD PTR [edx] ; 8
    movsx ecx, BYTE PTR [edx] ; 8
    je @B1@1 ; PROB 5% ; 10

• High-Level Languages

public TerminalLogger(Provider theProvider, ILog log, String extension) {
    logDest = log;
    try {
        try {
            myAddress = theProvider.getAddress(extension);
            myTerminal = theProvider.getTerminal(extension);
        } catch (Exception e) {
            tracer.error("Look like a bad extension");
            throw (e);
        }
        myTerminal.addCallObserver(this);
    } catch (Exception e) {
        tracer.error("TerminalLogger constructor caught " + e);
        return;
    }
}
And even higher abstractions…
An Abstraction is a “many-to-one” mapping

- An abstraction is a “virtual machine” that removes some of the unneeded details and complexity from the base machine
- Reduce the “impedance mismatch” between the base machine and the problem to be solved
- The challenge for the designer is to know when to stop…
  - *Everything should be as simple as possible, but no simpler* - Albert Einstein
- Different problems have different sets of “unneeded details”
Separation of Concerns
Separation of Concerns – Examples

• Performance
  – Inter-process communication
  – End-to-end latency
• Security
  – Hardened external interfaces
  – Flow and persistence of cleartext data
• Maintainability
  – Components can be changed independently
• Testability
  – Data can be injected and recorded from individual subsystems
  – Subsystems can run independently
Concerns lead us to choose “Structures” to describe the architecture

• “Software architecture...is the structure or structures of the system...”

• A structure is a binary relation*
  – Define the elements
  – Define the rule

• Examples:
  – Inheritance hierarchy: Element = Class, Rule = “is a subclass of”
  – Pipe and Filter Structure: Elements = Filters and Pipes, Rule = “attached to”
  – Implementation Technology: Elements = Modules and Programming Languages, Rule = “is implemented in”

• “What do the boxes and lines in that diagram mean?

*This description of structures as relations is based on a presentation by David Weiss.
Putting it all together
Using Abstraction and Separation of Concerns to create an Information-hiding Module Structure

• For tonight…Module = Work Assignment

• Concerns –
  – How do you divide the system so that each module can be built by an individual or team?
  – How do you partition the system so that parts can be changed independently?
  – How do you minimize the risk of “unknowns” at the start of a project? How do you deal with “TBDs” in the requirements?

• David Parnas (1972) proposed that using an “information hiding” criteria to decompose the system into modules will satisfy all of these concerns

• Later work by Baldwin & Clark showed the economic value of modularity, and Sullivan, et al showed that Parnas’s criteria was optimal in an economic sense.
Information-hiding criteria

• The information here is “design decisions”, especially those that are likely to change

• Each module hides (ideally) a single decision
  – The “secret” of the module is the design decision that can change without affecting any other module
  – Hide the secret behind an interface

• The interface defines an abstraction

• Modules are “write-time” entities, not “run-time” entities
  – Hide design information, don’t minimize run-time data exchange

• Typical decisions – data representation, persistence mechanism, algorithm implementation, hardware platform, COTS packages (different rates of change)
Information-Hiding Structure

- **Element:** Modules
- **Rule:** “refines the secret of”
- **This is not O-O...don’t think inheritance**
- **Example:**

```
<table>
<thead>
<tr>
<th>External Interface</th>
<th>Platform Services</th>
<th>Communication Services</th>
<th>CRM Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Source Interface</td>
<td>System Installation Module</td>
<td>Communications Channel</td>
<td>Campaign Management</td>
</tr>
<tr>
<td>System Integration</td>
<td>System Administration</td>
<td>Client Channel Control</td>
<td>CRM Behavior Services</td>
</tr>
<tr>
<td>External Platform Abstraction</td>
<td>Distributed System Module</td>
<td>Media Processing</td>
<td>CRM Integration</td>
</tr>
<tr>
<td>Basic Transport</td>
<td>License Service</td>
<td>System State</td>
<td>CRM Data Integration</td>
</tr>
<tr>
<td></td>
<td>Logging Service</td>
<td>Resource Presence and Availability</td>
<td>Production Line Module</td>
</tr>
<tr>
<td></td>
<td>Alarming</td>
<td>Work Item Services</td>
<td>Context Management</td>
</tr>
<tr>
<td></td>
<td>Internationalization Support</td>
<td>Event Management</td>
<td>Customization Management</td>
</tr>
<tr>
<td></td>
<td>Product Naming and Branding</td>
<td>Resource Assignment Service</td>
<td>Conventional CM</td>
</tr>
<tr>
<td></td>
<td>Scheduled Task</td>
<td>Communications Workflow</td>
<td>Build Management</td>
</tr>
<tr>
<td></td>
<td>Security Services</td>
<td></td>
<td>Package Management</td>
</tr>
<tr>
<td></td>
<td>User Interface Framework</td>
<td></td>
<td>Modification Request Tracking</td>
</tr>
<tr>
<td></td>
<td>Application Design Environment</td>
<td></td>
<td>Product Dependency</td>
</tr>
<tr>
<td></td>
<td>Monitoring and Reporting</td>
<td></td>
<td>Product Component Dependency</td>
</tr>
<tr>
<td></td>
<td>System Data Model</td>
<td></td>
<td>Product Deployment Policy</td>
</tr>
<tr>
<td></td>
<td>Data Management</td>
<td></td>
<td>License Policy Module</td>
</tr>
<tr>
<td></td>
<td>Knowledgebase Management</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Workflow Services</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
Modules exist in many forms

- A set of programs and shared data
- Abstract interface and implementation
- A state machine
- A class
- An abstract data type
- An abstraction
- A collection of macros and preprocessor directives

Decision Binding Time

- When do we make each decision?
- Model for decision times is system and technology specific
  - When? How?
- Examples:
  - Specification-time
  - Architecture-time
  - Design-time
  - Code-time
  - Compile-time
  - Link-time
  - Package-time
  - Install-time
  - Configuration-time
  - Run-time
## Example from a Java System Under Development

<table>
<thead>
<tr>
<th>Binding Time</th>
<th>Examples of decisions</th>
<th>Binding Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture-time</td>
<td>Feature content&lt;br&gt;Platform - programming language, application server, inter-process communication mechanism</td>
<td>Product Specification&lt;br&gt;Architecture Specification</td>
</tr>
<tr>
<td>Development-time</td>
<td>Definition of extension points (events and filters)</td>
<td>Code and Metadata</td>
</tr>
<tr>
<td>Package-time</td>
<td>Selecting components for a module</td>
<td>Installation-builder scripts</td>
</tr>
<tr>
<td>Install &amp; Deploy-time</td>
<td>Selecting which modules to install&lt;br&gt;Setting default parameters&lt;br&gt;Entering initial parameters</td>
<td>Installation options&lt;br&gt;Metadata&lt;br&gt;Setting values in Enterprise Database</td>
</tr>
<tr>
<td>Customization-time</td>
<td>Associating a handler to an extension event</td>
<td>Setting values in Enterprise Database</td>
</tr>
<tr>
<td>Run-time</td>
<td>“Plug-and-play” automatic selection of a component based on environment</td>
<td>Java dynamic class loading</td>
</tr>
</tbody>
</table>
 Reality Check

- Real world is imperfect
  - It is hard to isolate each design decision in a single module
  - It is hard to define interfaces that reflect the desired abstractions
- Why?
  - Reuse of existing assets
  - Mergers and acquisitions
  - Organizational constraints – people, processes, locations
  - Conway’s Law – The structure of the system reflects the structure of the organization that builds it - “...[an] organization had eight people who were to produce a COBOL and an ALGOL compiler...five people were assigned to the COBOL job and three to the ALGOL job. The resulting COBOL compiler ran in five phases, the ALGOL compile ran in three.”
So what’s an architect to do in the “real world”? 

• Do the best you can
  – Focus on modularizing high-change/high-risk areas of the system
  – It’s OK to have a top-level module that is less cohesive and collects the “leftover” unrelated set of decisions
• There is still value in an “information hiding” and “separation of concerns” mindset
  – Basic tools for dealing with complex systems
  – Examples – Quality Attributes, General Scenarios, Architecture Tactics
• Track the deviations from ideal, know the weaknesses of your design
  – What is going to be hard to change?
  – Is the deviation worth the risk?
Review

• Architecture is the structure or structures that describe the system
• Structure = Mathematical Relation = Elements & Rule
• Abstraction – One-to-Many mapping
• Separation of Concerns – Concerns tell us what structures we need to describe
• Information Hiding – Hide each design decision within a module
• Decision Binding Times
• Reality gets in the way
References


