Architecture of Product Lines

David Weiss

weiss@cs.iastate.edu.com
Topics

• Software Product Lines
  – What are product lines?
  – Why have product lines?
    • Economics, Benefits
    • Product lines and maintenance
  – Examples of successful product lines

• Architecture
  – Architecture as structures
  – Product line architectures
  – Architecture and maintenance

• How should developers be organized?
  – Open Market Software Development
“Those who ignore history are condemned to repeat it.”

George Santayana
Families

“We consider a set of programs to constitute a family whenever it is worthwhile to study programs from the set by first studying the common properties of the set and then determining the special properties of the individual family members.”

David L. Parnas
Product Lines

“We call a family of products designed to take advantage of their common aspects and predicted variabilities a product line.”

Lai & Weiss

“A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.”

Clements & Northrop
A Product Line Engineering Process

Investment

Domain Engineering

Feedback

Product Line Engineering Environment

Product Engineering

Payback

Products
A Product Line Engineering Process

Investment → Domain Engineering → Product Line Engineering Environment → Product Engineering → Products → Feedback → Payback

Maintenance?
PLE “Nirvana”

Product Manager

Creates using Commonality Analysis

Solution Specification

Specifications Needed Modules

Production Line Module

Core Asset Repository

Solutions

Commonality Analysis

Module Guide

"Uses" Hierarchy

Architecture
Economics Of Families (Simplified)

Cumulative Cost

Number of Family Members

Current Practice

Product Line Engineering
Economics Of Families (More Realistic)

Current Practice

Klein Horizon

Product Line Engineering

Cumulative Cost

Number of Family Members
S = Cumulative savings

$C_T = \text{Cost per family member with current practice}$

$C_F = \text{Cost per family member with domain engineering}$

$N = \text{Number of family members}$

$I = \text{Investment in domain engineering for the family}$
Industry Experience
Lucent
Nokia
Philips
Avaya

(See Software Product Line Hall of Fame
http://splc.net/fame.html)
Lucent
Some 5ESS™ Domains

5ESS Change Time Reduction
Nokia Group Project

- Define a product line with ~25-30 new products a year
- Across products, you must support:
  - Varying number of keys
  - Varying Display Size
  - Varying sets of features
  - A number of languages, and input methods
  - Backwards Compatibility to accessories
  - Different protocols and API’s
- And of course make them segmented (low end, high-end, …).
- Each Feature must be:
  - Configurable (On/Off, various settings)
  - Able to change behavior after product release
  - Plug’n’playable

Source: Anders Heie, SPLC2 Keynote Presentation
Nokia Facts

• This year, we have released 12 new products (lots more on the way)
• We sell phones in more than 130 countries
• We support 58 Languages, amongst which are:
  – Latin Languages
  – Arabic
  – Chinese (With variants)
  – Thai
  – Hebrew
• We support multiple Protocols:
  – CDMA, TDMA, AMPS, GSM, GPRS, and more…
• The HW is constantly changing, and the SW is constantly expanding to provide features
• We have several different UI Series to support

Source: Anders Heie, SPLC2 Keynote Presentation
Philips

Source: Rob van Ommering
Lucent 5ESS™ Switch Configuration: A More Detailed Example
Packet Service Unit Relationships
Packet Service Unit Relationships With Attributes
Relationship Architecture Designer (RAD)
“The art of progress is to preserve order amid change, and to preserve change amid order.”

Alfred North Whitehead
A Product Line Engineering Process

Investment

Domain Engineering

Feedback

Product Line Engineering Environment

Product Engineering

Payback

Products
Domain Engineering

Domain Analysis → Domain Model → Domain Implementation → Product Engineering Environment

Analysis Document, Application Modeling Language, Tools, Process
The Domain Model

• Conceptual Framework
  – Family Definition
    • Commonalities and Variabilities Among Family Members
    • Common Terminology for the Family
    • Abstractions for the Family
  – Economic Analysis
  – Application Modeling Language (AML): Language for stating requirements

• Mechanism for translating from AML to software
  – Alternative 1: Compiler
  – Alternative 2: Composer
Building The Conceptual Framework

• Qualify The Domain
  – Is it economically viable?

• Define The Family
  – What do members of the family have in common and how do they vary?

• Define The Decision Model
  – What decisions must be made to identify a family member?

• Design The Application Modeling Language
  – What is a good way to model a family member?

• Design The Product Engineering Environment
  – What are good mechanisms for using the decision model and the Application Modeling Language?
Defining A Product Line: Commonality Analysis

• A process to understand the extent of commonality and variability
  – Identify commonalities: attributes whose values are the same for all family members
  – Identify variabilities: attributes that may vary across the family
  – Quantify variabilities: define the range of values allowed for each variability
  – Define common terms

• A product line definition document that predicts how products will evolve

• The basis for a product-line architecture
Defining The Family: The Commonality Analysis

• Dictionary of terms
  – Technical terms that define a vocabulary for the domain
    • Primary Condition: The availability of a unit: working: ready, unready, or unusable
  • Commonalities: Assumptions that hold for every member of the family
    – Every unit must be in one of the four primary conditions.
  • Variabilities: Assumptions that define the range of variation for the family
    – Some unit names have inhibit states.
  • Parameters of Variation: Quantification of the variabilities
    – Whether or not a unit name can have an inhibit state: Boolean
Product Line Engineering Environment
For Configuration Control

- Language for specifying relationships among units
- Relationship Architecture Designer (RAD)
- Translator for RAD
  - Generates C from RAD diagrams
Product Line Engineering Environment For Configuration Control

Rules And Operations
- Translator
  - Control Code
  - Application Environment Functions
  - Data Structures
  - Configuration Controller

Variabilities
- RAD Diagram
- Translator
Summary

• The technology to improve the software production process exists
• Reorganizing software production to take advantage of the family viewpoint is the key to improvement
• Similar reorganizations are used in other engineering fields
  – customer involvement
  – shorter time to market
  – more variation across product line
  – maintain consistency across product line
• See The Software Product Line Hall of Fame
  http://splc.net/fame.html
ARCHITECTURE
“If I have seen farther than others, it is because I have stood on the shoulders of giants.”

Isaac Newton (?)
Hagia Sophia, Istanbul
Built 532-537
Designers: Anthemius of Tralles and Isidorus of Miletus (Mathematicians, Engineers, Architects)
Pendentives
Hagia Sophia Interior. Four arches swing across the piers, linked by four pendentives. The apices of the arches and the pendentives support the circular base of the huge central dome.

http://www.patriarchate.org/ecumenical_patriarchate/chapter_4/html/hagia_sophia__page_2.html
St. Paul’s Cathedral, London
Architect: Sir Christopher Wren
Built 1675-1708
View From The Dome (1)
View From The Dome (2)

Millenium Bridge
How Did They Know It Would Stay Up?

Snow-laden Dome Roof Collapses
At 3,500-Seat Auditorium on L.I.
Buckling Load

\[ B \sim ET/R \]

\( T = \) Thickness  
\( R = \) Radius  
\( E = \) Elastic Modulus

Assumptions
1. Spherical  
2. Isotropic  
   - identical properties at each point

How Did They Know It Would Stay Up?
• Prototypes  
• Theoretical Models
What Is Architecture?

“The art or science of building; esp. the art or practice of designing and building edifices for human use, taking both aesthetic and practical factors into account.”


Merriam Webster Online Dictionary

“In wider use, the term ‘architecture’ always means ‘unchanging deep structure.’ ”

Stewart Brand, How Buildings Learn
Attributes Of An Admired Architecture

• Distinctiveness (Istanbul and London landmarks)
• Beauty
• Utility (used every day)
• Persistence (1500 years and more!!)
• Features that delight (whispering gallery, dome view)
• Maintainable
• Safe
• Buildable (safe intermediate states, affordable)
• Different structures for different purposes (load bearing, interior layout, building services, …)
Attributes Of An Admired Architecture

- Distinctiveness (Istanbul and London landmarks)
- Beauty
- Utility (used every day)
- Persistence (1500 years and more!!)
- Features that delight (whispering gallery, dome view)
- Maintainable
- Safe
- Buildable (safe intermediate states, affordable)
- Different structures for different purposes (load bearing, interior layout, building services, …)
What is Software Architecture?

• Literally Hundreds of definitions
  http://www.sei.cmu.edu/architecture/definitions.html

• Architecture is focused on
  – Partitioning the whole into parts
  – Specifying the relations among the parts
  – Satisfying Requirements
    • Functional Requirements
      – End User Features …
    • Other Engineering Requirements
      – Performance & Scalability, Reliability & Availability …
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.

"Externally visible” properties refers to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.

Software Architecture in Practice (2nd edition), (Bass, Clements, Kazman; Addison-Wesley 2003)
Structure (View)

• A structure is a binary relation
  – Set of ordered pairs \{(a,b), (c,d), (b,a), (c,e)\}

• Defining a structure
  – Define the set of elements
    • a,b,c,d,e
  – Define the relation
    • Enumeration
    • Rule

• Example: connected graph
  – Elements: nodes in the graph: a,b,c,d,e,f,g
  – Relation: “is connected to”
    • \{(a,b), (a,c), (b,d), (b,f), (f,e), (c,g)\}
Architectural Structures

- There are many stakeholders in the architecture of a software system – individuals or groups who have an interest in products built using the architecture.

- Product Management
- System Engineering
- Architects
- Software Development
  - Including “maintainers”
    - aka “current engineering”
- System Verification
- Information Development
- Project Management
- R&D Management
- Professional Services
- Services
Architectural Structures

• Different Stakeholders have Different Concerns: Some Examples

• Product Management
  – How can I explain this architecture to customers, in a way that “sells” it to them?
  – What product variations are supported by the architecture?

• System Engineering
  – What functionality does the a product built using this architecture offer to its users?
  – How are the product requirements embodied in the software?

• Architects
  – What changes may be needed in the software in the future, and what changes are likely and need to be especially easy to make in the future?

• Software Development,
  – What implementation constraints must I satisfy when I implement a module?
  – What technologies and standards are used to implement the modules defined by the architecture?
Architectural Structures

• Module Guide
  – Explains the principles used to design the information hiding structure of the architecture, and shows how responsibilities are allocated among the major modules.

• Uses View
  – Describes the allowed “uses” relationships between modules and limits what other modules the implementer of a module may use.

• Process View
  – Defines the distinct processes in the architecture; Specifies the module(s) that make up the process, synchronization between processes ...
Architectural Structures

• Technology View
  – Maps the technology or technologies that will be used to implement each module in the architecture.

• Integration View
  – Highlights what data and modules within the system are externally accessible

• Design View
  – Ties together all of these design perspectives (e.g., workflow, data model, report model ....) to ensure that application customization was consistent across the CRM System.
Architectural Structures

• Module Interface Specs
  – Define Services Provided and Services Needed
  – Define syntax and semantics for accessing services
  – Define data types, program effects, …
  – Define test cases
  – Record design decisions and implementation notes

• Marchitecture View
  – Describes the architecture and the functionality provided at a high-level; Stakeholders include product managers, prospective customers …
Structures and Models

• Every view should have a model associated with it
• Every model should help answer questions about the products
  – Questions are driven by the concern(s) associated with the view
    • What is the buckling load?
  – Typical questions
    • How much does it cost to make a particular type of change?
    • How does performance vary with load on the product?
    • What is the expected availability?
    • How can I find a known bug?
    • What modules do I need to produce a member of my product line?
Architecture & Organization

• Conway’s Law

The structure of a system reflects the structure of the organization that built it.

• Module structure and organizational structure should mirror each other

• Recall: Module is a work assignment
  – Information hiding module hides a design decision
  – Interface is the assumptions that the users of a module can make about it (specification is the contract)
From Architecture To Product: The Decision Model

System Composition Mapping

<table>
<thead>
<tr>
<th>Variability</th>
<th>Decided Param</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>Param1</td>
</tr>
<tr>
<td>V2</td>
<td>Param2</td>
</tr>
</tbody>
</table>

Decisions

Module Mapping

<table>
<thead>
<tr>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
</tr>
<tr>
<td>2.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
</tr>
<tr>
<td>1.3.1</td>
</tr>
</tbody>
</table>

Uses Relation

Implementation Mapping

<table>
<thead>
<tr>
<th>Module</th>
<th>Package</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>com.xx.xx</td>
<td>abc.java</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.4</td>
</tr>
<tr>
<td>2.2.5</td>
</tr>
</tbody>
</table>
## Decision Model Excerpt for Floating Weather Station

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
<th>Value Set</th>
<th>Constraints</th>
<th>Binding Time</th>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>P10: SensorCount</td>
<td>Number of wind speed sensors</td>
<td>(LOW, HIGH)</td>
<td>LOW and HIGH are integers representing the no. of low resolution and high resolution sensors respectively, such that Minlow ≤ LOW ≤ L, Minhigh ≤ HIGH ≤ H, L+H ≤ MaxSensors</td>
<td>Spec</td>
<td>Sensor Monitor, Data Banker, Sensor Device Driver</td>
</tr>
<tr>
<td>P12: SensorRes</td>
<td>Resolution of each sensor</td>
<td>For each sensor, one of {LOWRES, HIGHRES} Default: LOWRES</td>
<td></td>
<td>Spec</td>
<td></td>
</tr>
<tr>
<td>P13: TransmitPeriod</td>
<td>Transmit period</td>
<td>[1..MaxTransmitPeriod] sec. Default: 10</td>
<td></td>
<td>Spec</td>
<td>Message Generation, Transmitter Device Driver, Message Format, Averager, Data Banker</td>
</tr>
</tbody>
</table>
Architecture Summary

• Structures are at the heart of architecture
• Structures should be designed to satisfy specific concerns
• Each structure should have an associated model to help know whether or not the concern(s) are satisfied
• We know many of the important structures for software architecture
  – Models form a product line as well
  – Documentation forms a product line as well
• Architecture and organization are closely linked
• Architecture drives production of products
Conclusions

• Product line architectures should be based on a product line of architectures
  – Common structures
  – Common models
  – Common documentation
  – Common tests

• Product line organizations should be based on a product line of organizations
  – Consistent PLE goals, business goals, organizational goals
  – Use motivators enabled by PLE

• Product lines are everywhere
How Should Developers Be Organized?
Motivating Developers

• Work with whom you want, when you want

• Hawthorne Effect
  – Compensation tied to value and productivity
    • Individually or by small group
  – Rapid feedback on productivity

Organizing Development: The Module Marketplace

• Issue: How To Assign Work
• Issue: How To Compensate Developers
Basic Assumptions

- The Product Line Architecture is organized into a collection of (information hiding) modules
- Each module needs an interface specification and an implementation
- It is possible to establish a priority ordering among modules
- There is a Module Approval Board (MAB) that can
  - Set module priorities, with attendant rewards
  - Evaluate the quality of module interface specifications
  - Evaluate the quality of module implementations
- Before it can be used in a product, every module must be approved
  - The interface specification must be approved before the implementation is approved
- Interface specification defines module
  - new interface spec means new module
    - New implementation may also mean new module
Assigning Work: A Family Of Possibilities

- Hierarchical, carefully controlled organization
  - Management decides who does what, with whom, and when

- Bidder’s Market
  - Bidding window established by MAB
  - (Teams of) developers bid on developing modules
    - Low bidder wins (track record considered)
  - Winning bidder develops interface spec and implementation
  - Winning bidder has first rights on later revisions

- Open market
  - (Teams of) developers submit interface specs, implementations as they wish
Compensating Developers: A Family Of Possibilities

- Management assigns salaries
- Management assigns salaries, plus rewards for approval of interface spec, implementation
- Management assigns salaries, plus royalties for module use
- Royalties for module use
  - Royalties based on revenues of products in which module is used
  - Royalties based on contribution of module to products
  - Royalties reduced for modules with uncorrected defects
Open Market Software Development
Organizing Development: Open Market Software Development

• Work assignment strategy: Bidder’s market
• Compensation strategy: Royalties for module use
Motivations

• Developers are motivated by chances to work with people they respect and like

• Performance should be rewarded in proportion to contribution and quality of the work

• Design for change should be encouraged
Complications

• Quality Standards
  – For each field defect, reduce royalties by $1/n$

• Determining Royalties
  – For each product in which a module is used, royalties are proportional to
    • Contribution of the module to the product
    • Quality of the module
    • Revenue gained from the product

• Encouraging Improvements
  – Improvers get a share of royalties, up to a maximum
  – Major improvements are rewarded more than minor improvements

• Sales Bias
  – Avoid collusion between development teams and sales

• Bookkeeping
• Training New Developers
Side Effects

- No more performance evaluation
- No more development management
- No more HR
- Developers will form teams on their own
Summary

• Product Line architecture permits bidding system on module basis
  – Use product lines and their architectures to break Conway’s Law

• Family of work assignment, compensation combinations
  – Bidding
  – Royalties

• Motivate developers through feedback on quality, productivity
Backup
References

Product Lines
Software Product Line Hall of Fame: http://www.sei.cmu.edu/productlines/plp_hof.html

Building Architecture
– Petroski, Henry; To Engineer Is Human, St. Martin’s Press, 1985
– Levy, Matthys, Salvadori, Mario; Why Buildings Fall Down, W.W. Norton, 1992

Software Architecture & Views
– Documenting Software Architectures Views and Beyond, Paul Clements et. al.

Hawthorne Effect
Categorizing Product Lines

• Product Line as evolution of existing family members
  – Modify members to produce new products in product line
    • Versions of Linux

• Product Line as named set of products
  – Audi: A4, A6, A8, …
  – Avaya S8600, S8700, …

• Product Line as configurable options
  – Mac OSX