Program Families and Software Product Lines

Building Sets of Systems

Looking Ahead

- Increment 2 plans and products
- Metrics: many missing
- Final report
  - Team members with few deliverables should take the lead (teams should agree on this if possible)
  - I will "suggest" this for some where visible output is lacking
Outline

• What is a program family?
• Why develop programs as families?
• The role of architecture
• Designing family architectures

Program Families

*We consider a set of programs a family if they have so much in common that it pays to look at their common aspects before looking at the aspects that differentiate them.*

- David L. Parnas

• What are some examples of program families?
• Why develop programs as families?
Consequence:
Merry-Go-Round of Sequential Development

- Knowledge is not institutionalized (tacit)
- Doomed to repeat lessons of the past

Sequential Development Over Time

... a result of "tactical software engineering"
Inefficiencies of Sequential Development

- **Hypothesis:** much of software development is re-development.
  - Software inevitably exists in many versions
  - Seldom develop truly new applications
- **Implication:** typically much in common among our systems
  ...But very little is reused
  - Difficult to identify commonalities and differences
  - Difficult to reuse code components
  - Difficult to add desired feature to existing design
  - Difficult to adapt other work products (if they exist)
  - Generally easier to re-do than re-use
- **What makes work products difficult to reuse?**

Sequential Development

- System developed through sequence of design decisions leading to product (8)
- Developing new product version requires backing up
  - Some decisions won’t apply to new version
  - Must back up to point where decisions can be re-made. (3, 5 or 6)
- How far we need to back up depends (roughly) on the order of decisions

*From Parnas: On the Design and Development of Program Families*
Objectives of Family Development

- Exploit commonality when building similar systems
  - Deploy systems in multiple versions
  - Quickly build new versions of a system
  - Produce deliverable code and documentation rapidly
  - Reduce cost, improve quality
- Design products to be reused (changed, extended)
  - Easy to add new features/capabilities
  - Easy to produce different versions
- Focus on reuse conceptual structures
  - Code is not the hard part, low ROI
  - Want to reuse requirements, design, etc.
- Requires a strategic view of development
  (encompasses multiple developments over time)

Family Development Model

- Development of new system begins from an intermediate stage
  - Order of decisions is critical
  - Intermediate representation is important
- Branching = different decision
  - All decision above a branch are in common to the family members
- Most similarity achieved by making as many common decisions as possible before creating differences

From Parnas: On the Design and Development of Program Families
Role of Architecture

• Architecture provides the basis for making and representing common design decisions for a family
  – Use architecture to instantiate common design decisions
  – Architectural representation provides the “intermediate representation” for the family
  – Instances of a family share common architecture, differ in design details
• How do we systematically construct one?

Family Architecture Design
Family Design Approach

- Conceptual approach using familiar concepts (there are others)
- Approach based on “design for change”
  - Most-solid-first: apply to common elements
  - Information hiding: apply to variations
- How is the system decomposed into parts?
  - System is decomposed into a hierarchy of information-hiding modules.
  - Structural decisions common to the family members are made first – result in overall structure and interfaces
  - Variabilities are pushed to the leaf modules and hidden when possible

Module Hierarchy

Given a set of likely changes C1, C2, … Cn and following these rules, what happens:
- To each change?
- To things that change together?
- Change separately?
Face Recognition Authentication:
Basic Application

1. Take a picture of John Smith
2. The system searches a pre-established data-base of faces
3. If recognized, authenticates individual

Likely Changes

- Identified requirements that might change, e.g.
  1. Look and feel of user interface
  2. The input devices used
  3. The specific database used
  4. The face recognition service or algorithm
  5. Etc.
Architectural Concept

- Take advantage of existing web services
- Support a family of possible applications
- How should the face recognition module be designed?

Assertion

- It always pays to build a systems as a family
- Hint: what is the relationship between designing for a family and designing for maintainability?
Summary

- When developing similar systems pays to develop as a family
- Applies to variation produced by maintenance
- Approach: develop a family architecture
  - Analyze what’s in common and what varies
  - Apply most solid first
    - Put decisions that won’t change in the architecture
    - Hide decisions that will vary

Software Product Line Engineering
Software Product-Line

“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 and Northrop

Product-Line Development

- A product-line development strategy sets out to maximize reuse over a family of similar software systems.
  - Reuse is planned and systematically executed
  - Common assets are developed as a distinct product
  - The process focuses on developing means of production before any particular product
  - Process develops processes as products
Generic PL-Process View

Customer Needs/ Business Goals

<table>
<thead>
<tr>
<th>Identify commonalities/variabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implement Production Environment</td>
</tr>
<tr>
<td>Model Application</td>
</tr>
<tr>
<td>Produce Family Instance</td>
</tr>
</tbody>
</table>

Investment

Family Production Environment

Payback

Product

Process

Customer Needs/ Business Goals

Infrequent/ Higher Cost

Customer System Requirements

<table>
<thead>
<tr>
<th>Economic Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commonalty Analysis</td>
</tr>
<tr>
<td>Implement Production Environment</td>
</tr>
<tr>
<td>Family Production Environment</td>
</tr>
<tr>
<td>Model Application</td>
</tr>
<tr>
<td>Build Family Instance</td>
</tr>
</tbody>
</table>

Domain Engineering

Domain Analysis

Domain Implementation

Frequent/ Low Cost

Domain Qualification
Product-Line Development Over Time

Adapt Assets
Create Product
Product

Reusuable Assets & Generators

Adapt Assets
Create Product
Product

Adapt Assets
Create Product
Product

PL Requirement
PL Design

Time

... a result of “strategic” software engineering

Development Approach

- Identify collections of programs that can be considered families.
- Design the family for “reproducibility”
  - Create a common design (architecture) for the family (module structure)
  - Create parameterized, reusable modules
  - Create a process for producing family members infrastructure to make it easy to produce family members from design and artifacts (“uses” relation)
- Create a means (e.g., language) for modeling family members
  - To support validation of customer requirements by analyzing the behavior of the model
  - To specify instances of the family from which deliverable code and documentation can be generated
- “Generate” family members from models using infrastructure and assets
Basic Mechanisms

- **MetaProgram Instantiation**: a point in the output text for a metaprogram instance at which the metaprogram's body should be expanded and included
- **Value Substitution**: a point at which the value of a referenced slot or generator should be included
- **Value Selection**: a point at which one of several alternative text fragments should be expanded and included
- **Value Repetition**: a point within the output text at which zero or more concatenated instances of a text fragment should be expanded and included

Fixed Size, Type Stack

```java
public class intStack {

    static final int maxSize = 1024;
    int data[] = new int[maxSize];
    int size = 0;

    public void add (int p1) throws stackFull {
        if (size == maxSize) throw new stackFull ();
        data [size++] = p1;
    }

    public int get () throws stackEmpty {
        if (size == 0) throw new stackEmpty ();
        return data [--size];
    }
}
```

© 1999 PHS – Grady Campbell
Fixed Size, Variable Type

```java
< program stacks (name: text, datatype: text, maxsize: text) >
public class <name>Stack {
    <datatype> data[] = new <datatype>[<maxsize>];
    int size = 0;

    public void add (<datatype> p1) throws stackFull {
        if (size == <maxsize>) throw new stackFull();
        data[size++] = p1;
    }

    public <datatype> get () throws stackEmpty {
        if (size == 0) throw new stackEmpty();
        return data[<--size];
    }
}
```

Risks and Rewards
Sequential Development Inefficiencies

- Key issue: distance between desire and gratification
- What problems result?
- How much of SE seeks to address this problem?
- Benefits from closing loop?

Idealized AE Process: External View

- Very quickly explore requirements space
- Customer works closely with application engineer
  - What are the advantages (compare to sequential development)?
Simplified P-L Economics

Cumulative Cost

4C
3C
2C
C

Number of family members

With reuse
Payback Point
Without reuse

C = average cost to develop family member

Shrinking Schedules

Figure: Change in end-to-end system development schedules over time

From "A Case Study in Successful Product Line Development. SEI Report SEI-96-TR-016, Clements & Brownword"
Increasing Reuse


Reduced Staffing Needs

Summary Rewards

- Exploit commonality between similar systems
  - Leverage existing expertise
  - Capture experience and results in concrete form for reuse
- Rewards
  - Achieve large-scale productivity gains
  - Improve time to market
  - Improve quality
  - Increase customer satisfaction
  - Respond rapidly to requirements changes (within P/L scope)
  - Reduce long term cost
  - Better use of skilled personnel (software engineers)

Summary Risks

- Increased up-front investment
- Requires expertise adapting
  - Products
  - Processes
  - Organization
- May not yield savings if...
  - Incorrect analysis
  - Inadequate design
  - Unpredictable changes
Summary

- Exploit commonality between similar systems
  - Leverage existing expertise
  - Capture experience and results in concrete form for reuse
- Unique in simultaneously improving quality while reducing cost, development time, and effort
- Best fit for product-oriented development organizations with documented successes
  - Cummins Diesel
  - Hewlett-Packard
  - Bosch
  - BMW
  - Daimler-Benz
  - Nokia, Etc.
- But increased up-front investment

Resources

- Books
- Web
  - SEI: http://www.sei.cmu.edu/productlines/
  - Big Lever http://www.softwareproductlines.com/index.html
    - Tools supporting industrial product-line engineering
  - MetaCase http://www.metacase.com/mep/
    - Tool for building application generators
Questions?