Software Product Lines: Past, Present, and Future

Dr. Paul C. Clements

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Sponsored by the U.S. Department of Defense
© 2005 by Carnegie Mellon University
Who We Are: Applied R&D laboratory situated as a college-level unit at Carnegie Mellon University, Pittsburgh, PA, USA

Purpose: Help others make measured improvements in their software engineering practices

First objective: Accelerate the introduction and widespread use of high-payoff software engineering practices and technology identifying, evaluating, and maturing promising or underused technology and practices.

We are a small organization. We have to pick these practices carefully.

One we have chosen is software product lines.
What is a Software Product Line?

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.

- Product lines in manufacturing have been around for centuries.
- Software product lines have not.
Why software product lines?

Because of the high-payoff results they make possible in practice:

Improved productivity by as much as 10x
Decreased time to market by as much as 10x
Decreased cost by as much as 60%
Decreased labor needs by as much as 10X fewer software developers
Increased quality by as much as 10X fewer defects
Real-world examples

Successful software product lines have achieved these results:

- across multiple domains
  - Mobile phones
  - Command and control ship systems
  - Ground-based spacecraft systems
  - Avionics systems
  - Pagers
  - Engine control systems
  - Billing systems
  - Web-based retail systems
  - Printers
  - Consumer electronic products
  - Acquisition management enterprise systems

- in large organizations and small
- in Government and private sectors
Product lines – building a family of products from interchangeable parts – have existed for centuries.

A thousand years ago, Li Chieh, the state architect of the Chinese emperor Hui-tsung, published a set of building codes for official buildings.  
- Standard parts and ways to connect the parts 
- Parameterized variations: lengths, loads 
- Options for components 
- Finishing with brackets, decorations

This book defined a set of reusable designs: a “product line” of buildings.
A large, modern example

These two very different aircraft were designed together and have about 60% of their parts in common.
“A common type-rating enables pilots qualified to fly any 757 or 767 to fly all the others with minimal additional familiarization, saving training time and costs. The 757/767 allows crews to fly more models with a greater variety of fuselage widths, capacities, and range capabilities than any other common-rated family.”

-- www.boeing.com
Another Example:
The Airbus A-320 Family

- 100-200 seats
- 103-146 feet in length
- Same wing
- Same flight deck
Questions

What does a company gain from producing a product line of sandwiches? Of commercial passenger aircraft?

• Simplified production of components
• Simplified management of inventory
• Simplified training
• Streamlined production facilities and process
• Market recognition and mind share
• Flexibility – ability to add new products quickly
Examples are everywhere!

Manufacturing technology – particularly the principle of interchangeable parts – makes the product line concept appear almost everywhere.

Look around this room. How many product lines do you see?

In 1964, IBM introduced the System/360 family of computers. Each model differed widely in performance and features, but they all ran the same programs. The operating system was also a family.
What about Software?

Software engineers knew about interchangeable parts.

We knew it was better to avoid designing and building software from scratch.

In other words, it seemed that reuse was the way to achieve greater productivity.

We began with small-scale reuse.
- Subroutines were the first example.
- We recognized that different systems shared common parts.
Moving to Large-Scale Reuse

Most organizations produce families of similar systems, differentiated by features.

Reuse should pay off.
But reuse has a disappointing history

Focus was small-grained and opportunistic. Results fell short of expectations.
“If you build it...

...they will come.”

It turns out they don’t.

The opportunistic model of reuse has been a thorough failure.
Some companies were trying a new approach

CelsiusTech's ShipSystem 2000: A family of 55 shipboard Command-and-control systems
- Began in 1985 with two systems
- Built from single core asset base
- Hardware-to-software cost ratio changed from 35:65 to 80:20
- Software staff went from 210 to 30
- Cycle time went from 7-10 years to 2-3.
- Reuse ~ 90%
- Integration test of 1-1.5 million SLOC requires 1-2 people
- Re-hosting to a new platform/OS takes 3 months
- Cost and schedule targets are predictably met
- Customer satisfaction is high
Software Product Lines: Past, Present, and Future (Page 19)

The Key Concepts

Use of a common asset base

in production

of a related set of products

Architecture, other core assets

Production Plan

defined by Scope Definition; justified by Business Case

Exit Slide Show
What happened next?

Other companies were trying this approach
• On varying scales
• With varying success

Topics of interest at the time
• Domain analysis
• Software architecture
• Process improvement

• In 1996, the SEI published the CelsiusTech case study
• In 1997, we held our first product line workshop.
• In 2000, the first International Software Product Line Conference was held.
Other Forces Fell Into Place

Rapidly maturing, increasingly sophisticated software development technologies including object technology, component technology, standardization of commercial middleware.

A global realization of the importance of architecture

A universal recognition of the need for process discipline.

Role models and case studies emerging in the literature and trade journals.

Conferences, workshops, and education programs including product lines in the agenda.

Company and inter-company product line initiatives.

Rising recognition of the amazing cost/performance savings that are possible.

© 2006 by Carnegie Mellon University
What Do We Have Now?

Software product lines have emerged as an important, viable paradigm for software development.

- Convincing evidence that software product line practice can bring about significant improvements in software development
- A body of knowledge and a set of standard models for software product lines
- A growing and energetic community of software product line practitioners
National Reconnaissance Office / Raytheon: Control Channel Toolkit

Ground-based spacecraft command and control systems

Increased quality by 10X
Incremental build time reduced from months to weeks
Software productivity increased by 7X
Development time and costs decreased by 50%
Decreased product risk

© 2006 by Carnegie Mellon University
Cummins, Inc.

World's largest manufacturer of large diesel engines.

- 20 product groups launched, yielding over 1000 separate engine applications
- 75% reuse, 360% productivity gain
- Product cycle time has plummeted. Time to first engine start went from 250 person-months to a few person-months.
- Software quality is at an all-time high.
- Product line approach let them quickly enter and then dominate the industrial diesel engine market.
Nokia Mobile Phones

Product lines with >30 new products per year. Before product line, only 5-10 products per year.

Software product line is so successful, Nokia is selling their core asset base as a product.

Across products there are
- varying keys, displays, features, protocols
- 58 languages and 130 countries
- need for backwards compatibility
- need for product behavior change after release

© 2006 by Carnegie Mellon University
Hewlett Packard

Printer systems
- 2-7x cycle time improvement (some 10x)
- 400% productivity improvement
- Sample Project
  - shipped 5x number of products
  - that were 4x as complex
  - and had 3x the number of features
  - with 4x products shipped/person
Market Maker GmbH: MERGER

Internet-based stock market software

Each product "uniquely" configured

Three days to put up a customized system
Software product lines have emerged as an important, viable paradigm for software development.

- Convincing evidence that software product line practice can bring about significant improvements in software development

- A body of knowledge and a set of standard models for software product lines

- A growing and energetic community of software product line practitioners
The Nature of the Essential Activities

All three activities are interrelated and highly iterative.

There is no "first" activity.
- In some contexts, existing products are mined for core assets.
- In others, core assets may be developed or procured for future use.

There is a strong feedback loop between the core assets and the products.

Strong management at multiple levels is needed throughout. Management orchestrates the processes to make the three essential activities work together.
Software Product Lines are Not About

Just
• Libraries of objects, components, or algorithms
• Reuse when the software engineer is so inclined
• Reuse with no repeatable production process
• A configurable architecture
Different Approaches - 1

**Proactive:** Develop the core assets first
- Define the product line's scope first and use it as a "mission" statement for the architecture and other assets
- Products come to market quickly with minimum code-writing.
- Requires up-front investment and predictive knowledge.

**Reactive:** Start with one or more products
- From these produce the product line core assets and then future products; the scope evolves more dramatically.
- Much lower cost of entry
- Architecture and other core assets must be robust, extensible, and appropriate to future product line needs
Different Approaches - 2

*Incremental:* Develop in stages with the plan from the beginning to develop a product line.
- Develop part of the core asset base, including the architecture and some of the components.
- Develop one or more products.
- Develop part of the rest of the core asset base.
- Develop more products.
- Evolve more of the core asset base.
- ...
Core assets include:

- Requirements and requirements analysis
- Domain model
- Software architecture
- Performance engineering
- Documentation
- Test plans, test cases, and data
- People knowledge and skills
- Processes, methods, and tools
- Budgets, schedules, workplans
- ...and Software

Core asset base

...with attached processes

Building the core asset base
Building the production plan

Core asset base
...with attached processes

Production plan
Product line experience has yielded important lessons

Lessons in software engineering
- Architectures for product lines
- Testing variable architectures and components
- Importance of having and capturing domain knowledge
- Managing variations
- Important of large, pre-integrated chunks

Lessons in technical/project management
- Importance of configuration management; why it's harder for product lines
- Product line scoping: What's in? What's out?
- Tool support for product lines

Lessons in organizational management.
- People issues: how to bring about change, how to launch the effort
- Organizational structure: Who builds the core assets?
- Funding: How are the core assets paid for?
- Interacting with the customer has whole new dimension
SEI Framework for Product Line Practice

We've distilled these lessons into practice areas, areas of expertise necessary for engineering any software system, but which take on a different flavor in a product line context. These form a framework for product line practice.

<table>
<thead>
<tr>
<th>Software Engineering</th>
<th>Technical Management</th>
<th>Organizational Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture Definition</td>
<td>Configuration Management</td>
<td>Building a Business Case</td>
</tr>
<tr>
<td>Architecture Evaluation</td>
<td>Data Collection, Metrics, and Tracking</td>
<td>Customer Interface Management</td>
</tr>
<tr>
<td>Component Development</td>
<td>Make/Buy/Mine/Commission Analysis</td>
<td>Implementing an Acquisition Strategy</td>
</tr>
<tr>
<td>COTS Utilization</td>
<td>Analysis</td>
<td>Funding</td>
</tr>
<tr>
<td>Mining Existing Assets</td>
<td>Planning</td>
<td>Launching and Institutionalizing</td>
</tr>
<tr>
<td>Requirements Engineering</td>
<td>Technical Risk Management</td>
<td>Market Analysis</td>
</tr>
<tr>
<td>Software System Integration</td>
<td>Tool Support</td>
<td>Operations</td>
</tr>
<tr>
<td>Testing</td>
<td></td>
<td>Organizational Planning</td>
</tr>
<tr>
<td>Understanding Relevant Domains</td>
<td></td>
<td>Organizational Risk Management</td>
</tr>
</tbody>
</table>

Sooping

© 2006 by Carnegie Mellon University

Scoping

Scoping bounds a system or set of systems by defining those behaviors or aspects that are in and those that are out.

A scope definition lets you determine if a proposed new product can be reasonably developed as part of the existing (or planned) product line.

We want to draw the boundary so the product line is profitable.
- If the scope is too limited, there will be too few products to justify the investment in the core assets.
- If the scope is too large, the core assets will need to be impossibly general.
- If the scope encompasses the wrong products, the product line will not succeed.
Growing a Scope Definition

The scope starts out broad and very general.

In a product line of Web software
  • Browsers are definitely in.
  • Aircraft flight simulators are definitely out.
  • Email handlers are... well, we aren't sure yet.

The scope grows more detailed as our knowledge increases and the product line matures.

Initially, many possible systems will be "on the cusp," meaning their "in/out" decision must made on a case-by-case basis. That's healthy.

© 2006 by Carnegie Mellon University
Scope precision increases as we learn more...up to a point.

a: space of all possible products
b: early, coarse-grained "in/out" decisions
c: product line scope with a healthy area of indecision
d: product line scope = product line requirements

If many products appear on the cusp over time, you may need to reactively adjust the scope.
Proactively Adjusting the Scope

Companies highly skilled at product line engineering purposely adjust their scope to take advantage of "nearby" market opportunities. Examples:

**CelsiusTech**
- Ship system product line → Air defense system product line
- Forty percent of new system was complete on day one.

**Cummins, Inc.**
- Automotive diesel engine product line → industrial diesel engine product line
- Quickly entered and dominated industrial engine market

**Motorola**
- One-way pager product line → Two-way pager product line
- Same product line architecture used for both
A Snapshot of the Body of Knowledge

Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures by Hassan Gomaa

Domain Architectures: Models and Architectures for UML Applications by Daniel J. Duffy

Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools by Jack Greenfield

Domain-Driven Design: Tackling Complexity in the Heart of Software by Eric Evans

Component-Based Product Line Engineering with UML by Colin Atkinson, et al.

Software Product Lines: Experience and Research Directions, by Patrick Donohoe (Editor)

Design and Use of Software Architectures, By Jan Bosch


Managing Software Re-Use, by Wayne C. Lim

Reuse-Based Software Engineering: Techniques, Organizations, and Controls, by Hafedh Mili, et al.

Practical Software Reuse, by Donald J. Reifer

Software Product Lines: Practices and Patterns, by Paul Clements and Linda Northrop

Source: amazon.com
What Do We Have Now?

Software product lines have emerged as an important, viable paradigm for software development.

- Convincing evidence that software product line practice can bring about significant improvements in software development
- A body of knowledge and a set of standard models for software product lines
- A growing and energetic community of software product line practitioners
We now have a community of software product line practitioners

We seek to
• understand the principles and practices behind software product engineering.
• help others successfully adopt the paradigm
• share experience

Major forums, workshops and conferences
• Software Product Line Conferences (SPLC 1-3)
• Product Family Engineering (PFE 1-5)
• Merged in 2004 to become SPLC / SPLC-Europe

Community web sites
• www.sei.cmu.edu/productlines
• www.softwareproductlines.com
SEI Software Product Line Curriculum

Five one- and two-day courses
  • Software Product Lines
  • Adopting Software Product Lines
  • Developing Software Product Lines
  • Product Line Technical Probe Team Training
  • Product Line Technical Probe Leader Training

Three certificate programs
  • Software Product Line Professional Certificate
  • Product Line Technical Probe Team Member Certificate
  • Product Line Technical Probe Leader Certificate
Current research topics

Product lines and aspects

- An aspect is a concern that cuts across units of code in a program.
- Aspects represent a way to achieve multi-dimensional separation of concerns.
- Making a "global" change merely requires changing the aspect and then weaving the aspect through the code.
- Example: Changing a program's overall fault-handling approach

Can aspects be used to represent the variation points of a software product line?
**Call for Participation**

**Aspects and Software Product Lines:**
**An Early Aspects Workshop at SPLC-Europe 2005**

*Rennes, France*

26 September 2005

http://www.early-aspects.net/events/splc2005ws/ / http://www.see.uni-essen.de/SPLC2005

**Background**

Aspect-oriented software development (AOSD) techniques aim to systematically identify, modularise, represent and compose broadly-scoped properties such as security, mobility, availability and real-time constraints – the so called crosscutting concerns – throughout the software life cycle. The initial focus of AOSD techniques has been at the programming level resulting in a range of aspect-oriented programming (AOP) techniques, and a number of application studies validating them in real-world application scenarios. Recently, however, the term “Early Aspects” has emerged to refer to AOSD techniques that focus on treatment of crosscutting concerns that arise before implementation, especially in requirements analysis and architecture design. Five Early Aspects workshops have been held to date to explore this area (see http://early-aspects.net/). One very promising application of early aspects is in
Current research topics

Product lines economic models

Here is our "classic" economic model:

Without Product Line Approach

With Product Line Approach

Cumulative Cost

Number of Products

This comes from a world where economic advantage is anecdotal, intuitive, and based on single data points.
Individuales Who Benefit – Anecdotally

CEO

COO

Technical Manager

Architect

Core Asset Developer

Marketer

End User

Customer

© 2006 by Carnegie Mellon University
Simple

Structured Intuitive Model for Product Line Economics

- \( C_{org}() \): cost to an organization of adopting the product line approach
- \( C_{cab}() \): development cost to develop a core asset base
- \( C_{unique}() \): cost to develop unique software that itself is not based on a product line platform.
- \( C_{reuse}() \): cost to reuse core assets in a core asset base

Example: Cost of building a product line of \( n \) products =

\[
C_{org}() + C_{cab}() + \sum_{i=1}^{n} (C_{unique}(product_i) + C_{reuse}(product_i))
\]
Product line practice patterns

Patterns represent pre-packaged solutions to known problems.

Design patterns are well-known in software engineering.

Organizational or process patterns also provide solutions or solution strategies to attack recurring problems

- How do I determine the scope of my product line?
- How do I establish a production capability?
- How do I launch a product line effort?
- How do I adopt the product line approach across my whole organization?
Example: “What to Build” Pattern

Patterns can provide
- Guidance for managers, blueprint for organizations
- Basis for process enactment, workflow and rules engines
Other research topics

Product line testing

Variability mechanisms and product line architectures

Lightweight, low-cost, low-barrier product line approaches

“Non-traditional” product lines
• Product lines of components for other product lines
• Cross-organizational product lines
In the Future

Software product line practice will be come a first-class part of software engineering professional practice.
• Product line engineering will be a standard part of software engineering curricula
• Software architects will routinely consider building architectures for a family of systems, using variability mechanisms wisely
• Product/project managers will always ask whether a project being launched can be a product line.
• Decision-makers will have the tools, models, and experience base necessary to help them make the right decisions and carry them out.
• Development tools will handle the development of a family of systems built and maintained in parallel.
Key Themes Among Successful Product Lines

Sophistication in the domain

A legacy base from which to build

Architectural excellence

Process maturity

Management commitment

Capacity for introspection
Three Observations

Reusing software is not the same as establishing a software product line.

It is essential to maintain the reused software (and its associated technical and developmental artifacts) and manage its evolution and growth separately from the places where it is used.

Mature product line organizations prioritize the health and welfare of their core asset base – that is, their product production capability – over that of any individual product.