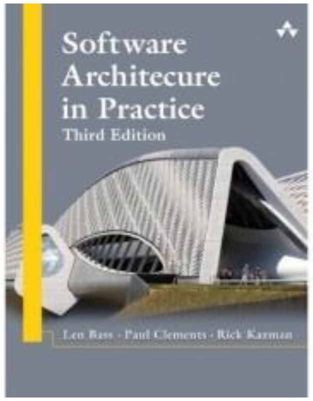
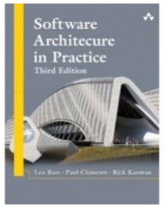
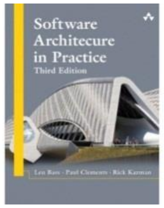


Chapter 25: Architecture and Product Lines



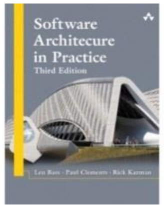
Chapter Outline



Product Lines of Software

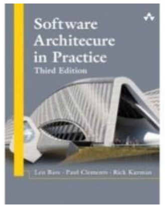
- A product line of software is:

  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



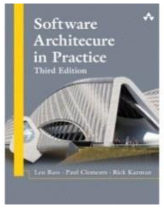
Reuse

- A product line represents strategic (planned) reuse.
- A common set of assets (core assets) that includes
  - Architecture
  - Requirements
  - Test cases
  - Build scripts
  - Documentation
  - …
- These assets are constructed specifically to support reuse.



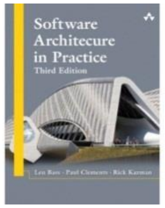
An Example

- You are constructing software that supports a bank loan office.
- There are 21 products in your product line.
- An existing module calculates customer interest payment.
  - Perfectly adequate for 18 of the products
  - Needs 240 lines modification for Delaware.



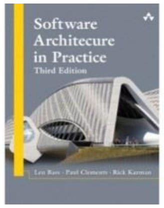
How to Manage the Modifications?

- One strategy is to make another copy of the affected module and insert the necessary changes
  - Called “clone and own”
  - Fast and easy
  - Does not scale!
    - Suppose each of the 21 products has 1000 modules.
    - Potentially huge number of distinct versions of the product to maintain.
- A better strategy is to introduce a “variation point” in the module and manage the variation point with, e.g., a configuration parameter.
  - Setting configuration parameter to “normal” will generate the 18 products as before.
  - Setting the configuration parameter to “Delaware” will generate the new version specifically for Delaware.



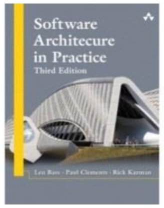
Variation Point (briefly)

- Example mechanisms are
  - Compiler flags
  - Module parameters
  - Resource file settings
- Configuration parameters can be managed and deployed from a data base.
  - Ensures that a record of configuration files is maintained.
  - Allows *tools* to generate various versions.


Software Architecure in Practice
Third Edition
Len Bass · Paul Clements · Rick Kazman

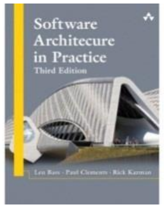
What Makes Product Lines Work?

- Planned reuse of
  - *Requirements*. Most of the requirements are common with those of earlier systems and so can be reused.
  - *Architectural design*. For a new product, if quality goals are the same then the architecture is largely the same.
  - *Software elements*. Software elements are applicable across individual products because the architecture is the same across products.
  - *Modeling and analysis*. Performance models, schedulability analysis, distributed system issues (such as proving the absence of deadlock), allocation of processes to processors, fault tolerance schemes, and network load policies all carry over from product to product.



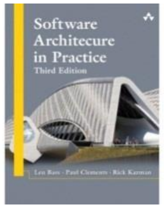
What Makes Product Lines Work? - 2

- Planned reuse of
  - *Testing*. Test plans, test processes, test cases, test data, test harnesses, and the communication paths required to report and fix problems are already in place.
  - *Project planning artifacts*. Budgeting and scheduling are more predictable because experience is a high-fidelity indicator of future performance.
  - and more…



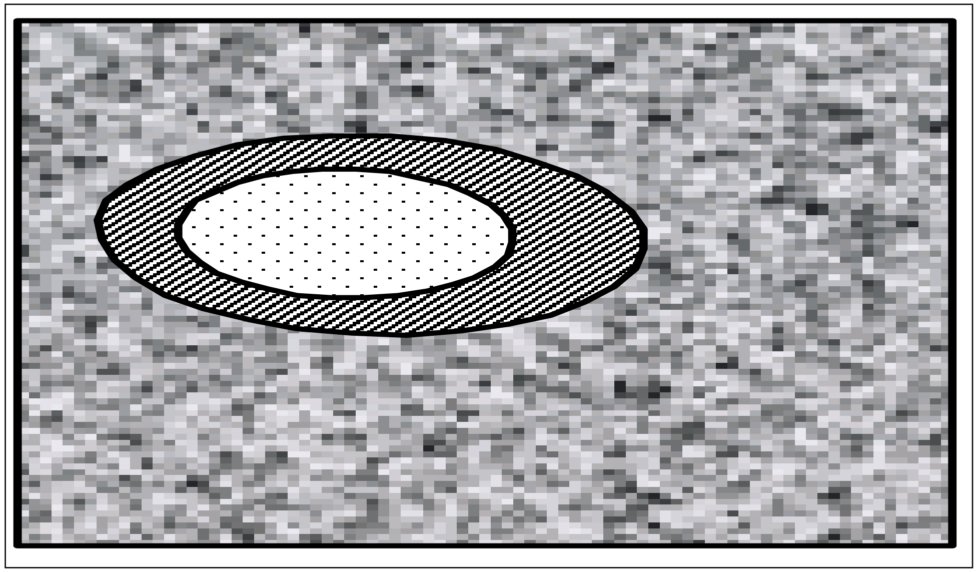
Costs of Product Lines

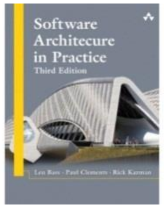
- Up front planning takes time.
- It takes 2-3 systems to start seeing savings begin for product lines.
- Alternatively, the product line can be evolved when new products are defined. But this may require rework of the architecture.



Product Line Scope

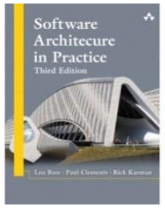
- Some products are in scope (white)
- Some products are out of scope (speckled)
- Some products need to be decided on a case by case basis (lined)





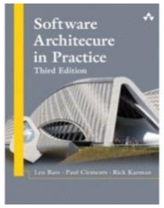
Scope is a Critical Decision

- Too broad a scope and the systems become difficult to design and construct.
- Too narrow a scope and there are too few systems to justify the additional expense and complexity.
- Scoping decisions made by
  - finding commonality and variation points among potential products
  - interactions between marketing and the architect to define as broad a scope as technically feasible without introducing excessive cost when building products.



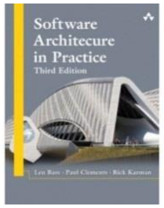
Variability General Scenario

Portion of Scenario	Possible Values
Source	Actor requesting variability
Stimulus	Requests to support variations in: Hardware, Feature sets, Technologies, User interfaces, Quality attributes, etc. for the range of products affected, such as: • All • A specified subset • Those that include feature set *x* • New products
Artifacts	Requirements, architecture, component x, test suite y, project plan z…
Environment	Variants are to be created at: • Runtime • Build time • Development time
Response	The requested variants can be created.
Response Measure	A specified cost and/or time to create the core assets and to create the variants using these core assets.



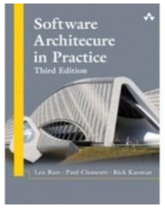
Architectural Mechanisms for Variability

- Inclusion or omission of elements:
  - Through build procedures for different products, you can include a different number of replicated elements.
  - For instance, high capacity variants might be produced by adding more servers.
  - The actual number should be unspecified, as a point of variation. This may be chosen dynamically.


Software Architecure in Practice
Third Edition
Len Bass · Paul Clements · Rick Kazman

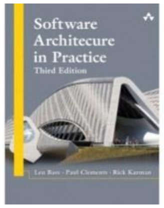
Architectural Mechanisms for Variability - 2

- Selection of different versions of elements that have the same interface but different behavioral or quality attribute characteristics.
  - Selection can occur at compile, build time, or run time.
  - Selection mechanisms include
    - static libraries, which contain external functions linked after compilation time,
    - dynamic link libraries,
    - add-ons (e.g. plug-ins, extensions, themes)



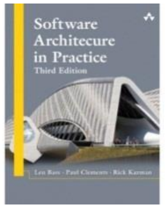
Other Variability Mechanisms

- *Extension points.* These are identified places in the architecture where additional behavior or functionality can be safely added.
- *Reflection.* This is the ability of a program to manipulate data on itself or its execution environment or state. Reflective programs can adjust their behavior based on their context.
- *Overloading.* This is a means of reusing a named functionality to operate on different types. Overloading promotes code reuse, but at the cost of understandability and code complexity.



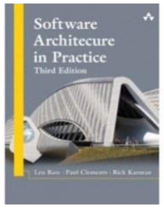
Other Variability Mechanisms

- Inheritance; specializing or generalizing a particular class
- Component substitution
- Add-ons, plug-ins
- Templates
- Parameters (including text preprocessors)
- Generator
- Aspects
- Runtime conditionals
- Configurator


Software Architecure in Practice
Third Edition
Len Bass · Paul Clements · Rick Kazman

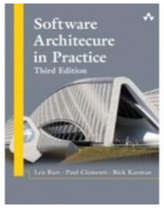
Each Mechanism Has a Cost

- Costs include
  - The skill set required to implement mechanism
  - One time cost for establishing mechanism
  - Recurring cost for each instance
  - Impact on qualities such as
    - Performance
    - Modifiability



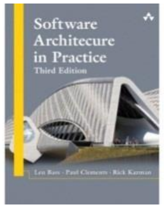
Evaluating a Product Line Architecture

- Largely the same as a normal evaluation.
- Special attention should be paid to variation points.
- Can be applied to
  – core assets, to assess suitability for product line
  – instance architecture, to assess suitability for a particular desired product.



Key Product Line Issues

- Adoption strategies
  - Top down versus Bottom up
  - Proactive adoption versus reactive adoption
  - Incremental versus Big-bang
- Organizational structure
  - Dedicated group for core assets
  - Core asset group composed of various product members
  - Paying for core asset development



Summary

- A product line is strategic, planned reuse.
- The product line approach to developing software can pay huge dividends.
- Requires an understanding of:
  - Architecture
  - Organizational issues
  - Process issues