Building a Commons: Product Line Architectures

Stacy Prowell

sprowell@cs.utk.edu

April 2005

Software Quality Research Laboratory
DEPARTMENT OF COMPUTER SCIENCE @ THE UNIVERSITY OF TENNESSEE

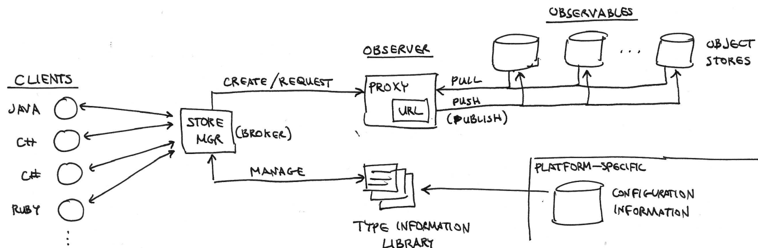
Outline

1. Architecture
2. Product Lines
3. Product Line Architectures
4. Conclusions

Architecture

System **architecture** constrains:

- The **elements** from which the system is constructed
- The **responsibility** assignments of these elements
- Their **relationships** with one another
- **Extensibility** and **contractibility** points of the system



Software Architecture

- The **elements** may be classes, modules, components, libraries, programs, or other complete systems.
- Each element provides a **useful abstraction.** Elements must encapsulate some **data** and/or **processing** from the rest of the system. This is the element's **secret**; information known to the designer of the module, but hidden from (and thereby **simplifying**) the rest of the system.
- Elements have a variety of relationships with one another: **derivation**, **composition**, **uses**, **...**
- Proper separation of interface and implementation–and maintenance of a proper **use hierarchy**–supports extension and contraction.

Software Architecture

- The **elements** may be classes, modules, components, libraries, programs, or other complete systems.
- Each element provides a **useful abstraction.** Elements must encapsulate some **data** and/or **processing** from the rest of the system. This is the element's **secret**; information known to the designer of the module, but hidden from (and thereby **simplifying**) the rest of the system.
- Elements have a variety of relationships with one another: **derivation**, **composition**, **uses**, ...
- Proper separation of interface and implementation–and maintenance of a proper **use hierarchy**–supports extension and contraction.

Software Architecture

- The **elements** may be classes, modules, components, libraries, programs, or other complete systems.
- Each element provides a **useful abstraction.** Elements must encapsulate some **data** and/or **processing** from the rest of the system. This is the element's **secret**; information known to the designer of the module, but hidden from (and thereby **simplifying**) the rest of the system.
- Elements have a variety of relationships with one another: **derivation**, **composition**, **uses**, ...
- Proper separation of interface and implementation–and maintenance of a proper **use hierarchy**–supports extension and contraction.

Software Architecture

- The **elements** may be classes, modules, components, libraries, programs, or other complete systems.
- Each element provides a **useful abstraction.** Elements must encapsulate some **data** and/or **processing** from the rest of the system. This is the element's **secret**; information known to the designer of the module, but hidden from (and thereby **simplifying**) the rest of the system.
- Elements have a variety of relationships with one another: **derivation**, **composition**, **uses**, ...
- Proper separation of interface and implementation–and maintenance of a proper **use hierarchy**–supports extension and contraction.

From Needs to Software

Initially there is some **need**, expressed informally and poorly-understood. Developing an **acceptable product** which satisfies that need is a **learning** experience.

- **Delivering anything changes expectations.** Seeing the product helps users better understand what is possible, and what they might want.
- **Initial assumptions are violated.** What is understood about the problem and solution spaces changes as developers work.
- **Technology changes.** Systems must adapt to the demands of the new technologies.
- **Standards change.** Software standards do not arise from physics, but are arbitrary and, sometimes, even capricious.

From Needs to Software

Initially there is some **need**, expressed informally and poorly-understood. Developing an **acceptable product** which satisfies that need is a **learning** experience.

- **Delivering anything changes expectations**. Seeing the product helps users better understand what is possible, and what they might want.
- **Initial assumptions are violated**. What is understood about the problem and solution spaces changes as developers work.
- **Technology changes**. Systems must adapt to the demands of the new technologies.
- **Standards change**. Software standards do not arise from physics, but are arbitrary and, sometimes, even capricious.

From Needs to Software

Initially there is some **need**, expressed informally and poorly-understood. Developing an **acceptable product** which satisfies that need is a **learning** experience.

- **Delivering anything changes expectations**. Seeing the product helps users better understand what is possible, and what they might want.
- **Initial assumptions are violated**. What is understood about the problem and solution spaces changes as developers work.
- **Technology changes**. Systems must adapt to the demands of the new technologies.
- **Standards change**. Software standards do not arise from physics, but are arbitrary and, sometimes, even capricious.

From Needs to Software

Initially there is some **need**, expressed informally and poorly-understood. Developing an **acceptable product** which satisfies that need is a **learning** experience.

- **Delivering anything changes expectations**. Seeing the product helps users better understand what is possible, and what they might want.
- **Initial assumptions are violated**. What is understood about the problem and solution spaces changes as developers work.
- **Technology changes**. Systems must adapt to the demands of the new technologies.
- **Standards change**. Software standards do not arise from physics, but are arbitrary and, sometimes, even capricious.

From Needs to Software

Initially there is some **need**, expressed informally and poorly-understood. Developing an **acceptable product** which satisfies that need is a **learning** experience.

- **Delivering anything changes expectations**. Seeing the product helps users better understand what is possible, and what they might want.
- **Initial assumptions are violated**. What is understood about the problem and solution spaces changes as developers work.
- **Technology changes**. Systems must adapt to the demands of the new technologies.
- **Standards change**. Software standards do not arise from physics, but are arbitrary and, sometimes, even capricious.

From Needs to Software

You are never developing the **right** thing, only the **next** thing, but you should strive to develop it **correctly**.

Remember: For most products, 80% of the total cost is incurred **after** deployment.

(This highlights the value of **documentation**.)

Incremental Architecture

Do not **build** software systems; **grow** them.

This includes the architecture.

Architecture is not Code

Conceptually: software architecture is a collection of **constraints.** It is a set of **standards** and ways of solving common problems, and it must be properly **documented.**

A heavy-handed approach to architecture can **over-constrain** the system, and actually prohibit good solutions.

Documentation

Architecture is Documentation

If you do not document your architecture, you do not have an architecture.

Developers unaware of the architecture will re-implement solutions and violate architectural constraints.

If they re-implement solutions:

- Get little or no benefit from the architecture
- If they violate constraints, they may perceive the architecture as “full of bugs.”

Developers must understand and accept the benefits of the architecture, or they will resist using it.

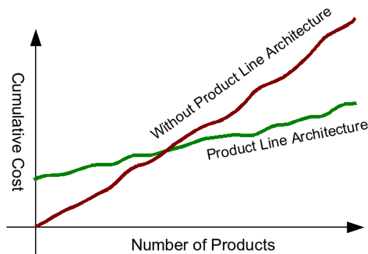
Product Line

Product Line

A **product line** is a family of products with a common set of features satisfying needs specific to a particular market or mission.

It makes sense to think in terms of the individual products as specialized instances of some prototypical product.

Economics of Product Lines



A **product line architecture** trades higher **initial** cost (the cost of developing the architecture) for lower **incremental** costs (the cost of developing a specialized product).

For a successful product line, this can yield significant reduction in **cumulative** costs.

Product Line Architectures

The product line architecture constrains developers, but also relieves them of many considerations and allows them to focus on solving their particular problem.

- **Interchangeable (reusable) parts**
  Recurring problems are solved in a consistent manner. Once one understands the approach, one can more readily understand a given product.
- **Measurement standards**
  Adoption of a standard improves communication, and also improves one's intuition about solutions.

Product Line Architectures

The product line architecture constrains developers, but also relieves them of many considerations and allows them to focus on solving their particular problem.

- **Interchangeable (reusable) parts**
  Recurring problems are solved in a consistent manner. Once one understands the approach, one can more readily understand a given product.
- **Measurement standards**
  Adoption of a standard improves communication, and also improves one's intuition about solutions.

Product Line Architectures

The product line architecture constrains developers, but also relieves them of many considerations and allows them to focus on solving their particular problem.

- **Interchangeable (reusable) parts**
  Recurring problems are solved in a consistent manner. Once one understands the approach, one can more readily understand a given product.
- **Measurement standards**
  Adoption of a standard improves communication, and also improves one's intuition about solutions.

Product Line Architectures

- **Standard notations**
  "By relieving the mind of all unnecessary work, a good notation sets it free to concentrate on more advanced problems." – Alfred North Whitehead
- **Well-understood tools**
  Understand the space for which the tool is applicable, its limitations, and the cost of its use.

All these free developers to focus on their **core** problem, for which their individual expertise is most relevant (the "**sweet spot**").

Product Line Architectures

- **Standard notations**
  "By relieving the mind of all unnecessary work, a good notation sets it free to concentrate on more advanced problems." – Alfred North Whitehead
- **Well-understood tools**
  Understand the space for which the tool is applicable, its limitations, and the cost of its use.

All these free developers to focus on their **core** problem, for which their individual expertise is most relevant (the "**sweet spot**").

Product Line Architectures

- **Standard notations**
  "By relieving the mind of all unnecessary work, a good notation sets it free to concentrate on more advanced problems." – Alfred North Whitehead
- **Well-understood tools**
  Understand the space for which the tool is applicable, its limitations, and the cost of its use.

All these free developers to focus on their **core** problem, for which their individual expertise is most relevant (the "**sweet spot**").

Products

If a product is part of a product line, it makes sense to make the product's architecture **consistent** with the product line architecture.

Each product has a distinct architecture, but the architecture can be quickly understood by understanding the product line architecture.

Frameworks

Because the product line architecture is a set of constraints, it is not, itself, a product.

A product line architecture may be supported by a **framework**, but a framework is **not** a product line architecture.

The Commons

A product line architecture motivates a **commons.**

- The product line architecture supports **interoperability** of disparate products.
- The product line architecture encourages many kinds of **consistency**:
  - **Developers** see a consistent set of components and rules for combining them; this makes it easier to understand each other's products and encourages collaboration. (Errors are reduced.)
  - **Users** get a consistent experience; their experience with one product translates to others. (Misuse of products is reduced.)
- As new products are developed and old ones are updated, the product line architecture both grows and matures.

A community process is necessary, but there must be centralized management of the architecture.

The Commons

A product line architecture motivates a **commons.**

- The product line architecture supports **interoperability** of disparate products.
- The product line architecture encourages many kinds of **consistency**:
  - **Developers** see a consistent set of components and rules for combining them; this makes it easier to understand each other's products and encourages collaboration. (Errors are reduced.)
  - **Users** get a consistent experience; their experience with one product translates to others. (Misuse of products is reduced.)
- As new products are developed and old ones are updated, the product line architecture both grows and matures.

A community process is necessary, but there must be centralized management of the architecture.

The Commons

A product line architecture motivates a **commons.**

- The product line architecture supports **interoperability** of disparate products.
- The product line architecture encourages many kinds of **consistency**:
  - **Developers** see a consistent set of components and rules for combining them; this makes it easier to understand each other's products and encourages collaboration. (Errors are reduced.)
  - **Users** get a consistent experience; their experience with one product translates to others. (Misuse of products is reduced.)
- As new products are developed and old ones are updated, the product line architecture both grows and matures.

A community process is necessary, but there must be centralized management of the architecture.

The Commons

A product line architecture motivates a **commons.**

- The product line architecture supports **interoperability** of disparate products.
- The product line architecture encourages many kinds of **consistency**:
  - **Developers** see a consistent set of components and rules for combining them; this makes it easier to understand each other's products and encourages collaboration. (Errors are reduced.)
  - **Users** get a consistent experience; their experience with one product translates to others. (Misuse of products is reduced.)
- As new products are developed and old ones are updated, the product line architecture both grows and matures.

A community process is necessary, but there must be centralized management of the architecture.

The Commons

A product line architecture motivates a **commons.**

- The product line architecture supports **interoperability** of disparate products.
- The product line architecture encourages many kinds of **consistency**:
  - **Developers** see a consistent set of components and rules for combining them; this makes it easier to understand each other's products and encourages collaboration. (Errors are reduced.)
  - **Users** get a consistent experience; their experience with one product translates to others. (Misuse of products is reduced.)
- As new products are developed and old ones are updated, the product line architecture both grows and matures.

A community process is necessary, but there must be centralized management of the architecture.

Designing the Architecture

1. **Define the product line.**
   What is the common domain of the products? Are they all intended to solve some common problem, or provide similar capabilities, or target a common market? Write down the definition of the product line.
2. **Identify a small set of initial common problems.**
   What common capabilities must all the products share? Database access, object storage, data reduction and visualization, hardware abstraction, etc.
3. **For each common problem, propose a solution.**
   The solution may be to adopt an existing system (for object serialization), an existing standard (such as XML), or a rule (each product is responsible for visualization).
4. **Grow the product line architecture.**
   As products are developed, watch for capabilities which may be common to the product line and propose a uniform solution as part of the architecture.

Designing the Architecture

1. **Define the product line.**
   What is the common domain of the products? Are they all intended to solve some common problem, or provide similar capabilities, or target a common market? Write down the definition of the product line.
2. **Identify a small set of initial common problems.**
   What common capabilities must all the products share? Database access, object storage, data reduction and visualization, hardware abstraction, etc.
3. **For each common problem, propose a solution.**
   The solution may be to adopt an existing system (for object serialization), an existing standard (such as XML), or a rule (each product is responsible for visualization).
4. **Grow the product line architecture.**
   As products are developed, watch for capabilities which may be common to the product line and propose a uniform solution as part of the architecture.

Designing the Architecture

1. **Define the product line.**
   What is the common domain of the products? Are they all intended to solve some common problem, or provide similar capabilities, or target a common market? Write down the definition of the product line.
2. **Identify a small set of initial common problems.**
   What common capabilities must all the products share? Database access, object storage, data reduction and visualization, hardware abstraction, etc.
3. **For each common problem, propose a solution.**
   The solution may be to adopt an existing system (for object serialization), an existing standard (such as XML), or a rule (each product is responsible for visualization).
4. **Grow the product line architecture.**
   As products are developed, watch for capabilities which may be common to the product line and propose a uniform solution as part of the architecture.

Designing the Architecture

1. Define the product line.
   What is the common domain of the products? Are they all intended to solve some common problem, or provide similar capabilities, or target a common market? Write down the definition of the product line.
2. Identify a small set of initial common problems.
   What common capabilities must all the products share? Database access, object storage, data reduction and visualization, hardware abstraction, etc.
3. For each common problem, propose a solution.
   The solution may be to adopt an existing system (for object serialization), an existing standard (such as XML), or a rule (each product is responsible for visualization).
4. Grow the product line architecture.
   As products are developed, watch for capabilities which may be common to the product line and propose a uniform solution as part of the architecture.

Summary

- Architecture is essential to a maintainable product.
- A product line architecture supports collaboration and faster product development.
- The product line architecture must be well documented. Note that this reduces the documentation requirements for each product.
- There must be centralized management of the product line architecture.
- Architectures must be developed incrementally.

Online

Software Quality Research Laboratory

http://www.cs.utk.edu/sqrl/

Software Engineering Institute

http://www.sei.cmu.edu/architecture
http://www.sei.cmu.edu/plp/plp_init

Use Hierarchy

Use

One element *A* **uses** another element *B* iff the correct functioning of *A* depends on the availability of a correct implementation of *B*.
(*A* doesn't work until *B* works.)

- **Use** is different from **calls** or **invokes**.
- The task OneSecond must be invoked every second by the Scheduler. OneSecond **uses** the Scheduler, but the Scheduler **invokes** OneSecond.

Use Hierarchy

Use Hierarchy

A well-designed **uses** relation forms a proper **hierarchy**; there should be no loops in the uses relation. For this reason you must **document** the uses relation and maintain it. It cannot be automatically deduced; it must be **designed**.

If A uses B, and B uses C, and finally C uses A, then **nothing** works unless **everything** works. Changes to one component can become impractical, and the system can behave in unexpected ways.

Split functionality, use an **observer**, etc.