Why and how to make a product-line software-intensive system effective in industrial settings ?

Jean Jourdan

Architectures & System Technologies Lab.
Corporate Research Center
Thomson-CSF

November 6th 1998

Outline

- The realities we face
- The architecture centric product line approach
- Product-line process
- Software architecture
- Reference architectures
- Ongoing experiments and projects
- Conclusion

The realities we face

Current situation

Most organizations developing software-intensive systems have :

- competitive markets :

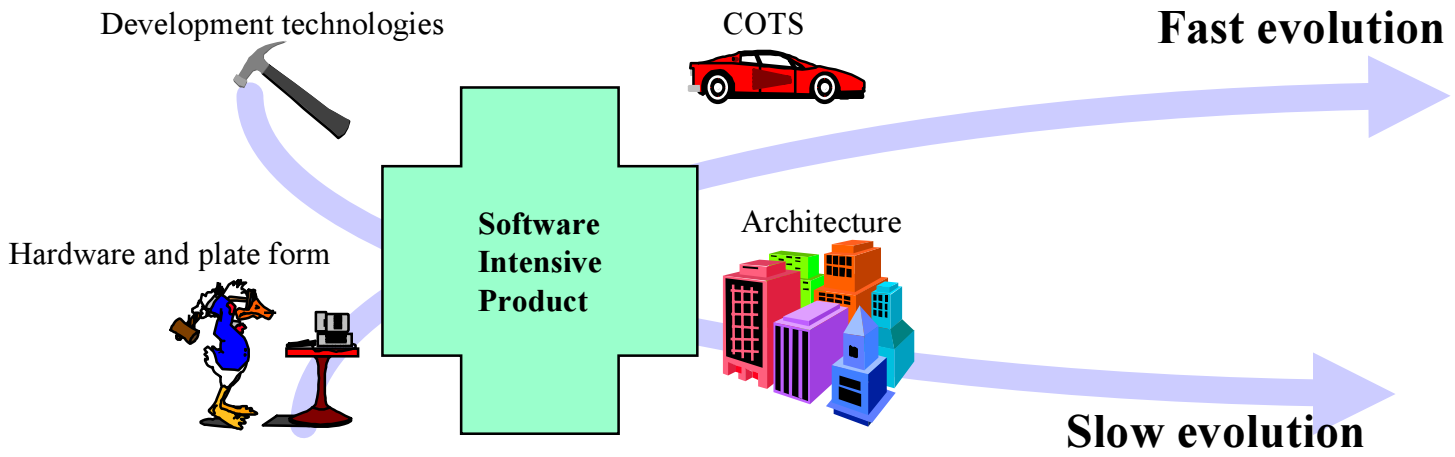
  ⇒ decrease development costs
  ⇒ increase quality
  ⇒ reduce time-to-market
  ⇒ allow predictable market delivery
  ⇒ increase products diversity

- long product cycles :

  ⇒ improve maintenance
  ⇒ allow new technology integration

- more and more complex systems

Life-cycle mismatch



- Ensure an explicit separation between product and technology life-cycles :
  - long life cycles (5 - 15 years): architectures and development technologies
  - short life cycles (1 - 3 years): Hardware and COTS (Components Of The Shelves)
- Make product-line effective in industrial settings:
  - Build applications by assembling reusable components
  - Preserve the possibility to take into account their evolutions and replacement within the complete (specification, design and maintenance) product life-cycle

From stovepipe applications to evolvable application families

Past	Future
• Requirements are fixed • Applications are isolated • No shared substrate	• Requirements change • Applications in families • Evolving substrate
• Architecture and code separation • Implicit architecture • Design info and trade-off discarded • Design is followed by maintenance	• Architecture and code integrated and evolve together • Explicit architecture • Design info and trade-off preserved to guide evolution • Design and maintenance are a single activity
• Premature and irreversible optimization • Static implementation compiles in design decisions to save resources • Tools emphasize "front-end system's analysis"	• Late binding • Implementation and environment use resources to support evolution • Tools support whole system lifetime

The Architecture Centric Product-line Approach

Our goal

Change traditional development cycle in order to build systems by **assembling reusable components**, while :

- Allowing the improvement/replacement of components over time
- Ensuring independence from COTS (software, middleware, OS, …)
- Gaining early insight into system qualities

Our strategy

Enable widespread **product-line** practices :

⇒ domain specific (family of applications)

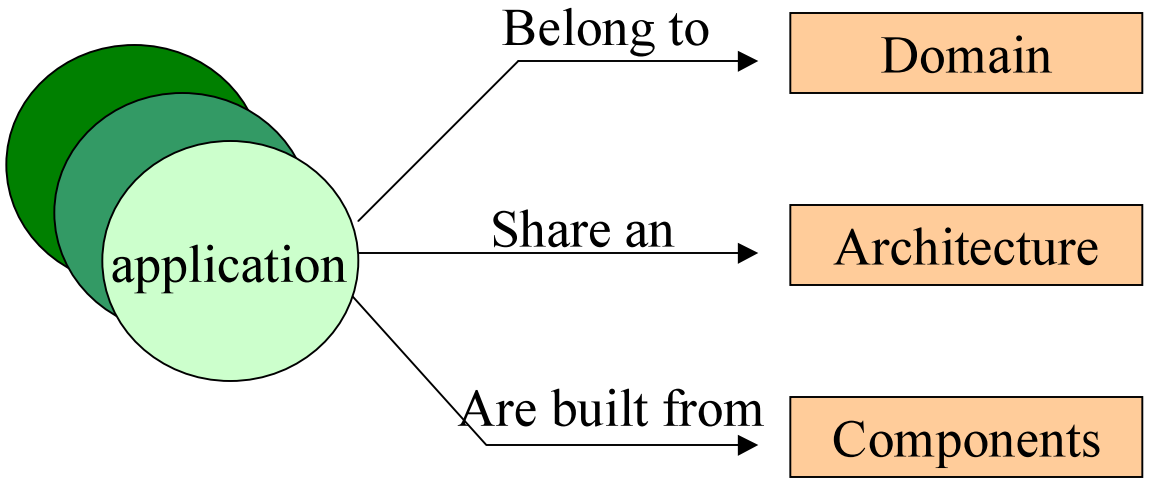
⇒ process driven

⇒ architecture centric

The product-line approach allows organizations to reuse numerous software assets (requirements, designs, source codes, test cases) when building new systems.

Domain specific

A **family of applications** is a group of products sharing a common set of features that satisfy specific needs of a selected market.



The purpose of the product-line approach is to provide a set of packaged reference elements (domain model, architecture, components) and to use them to build new applications.

Contributions of product-lines

Product-lines amortize the investment in :

- Requirements analysis and modeling
- Domain modeling
- Software architecture design and validation
- Documentation
- Test cases
- Implementation

$\Rightarrow$ Product-lines = strategic reuse

Prerequisites to build a product-line

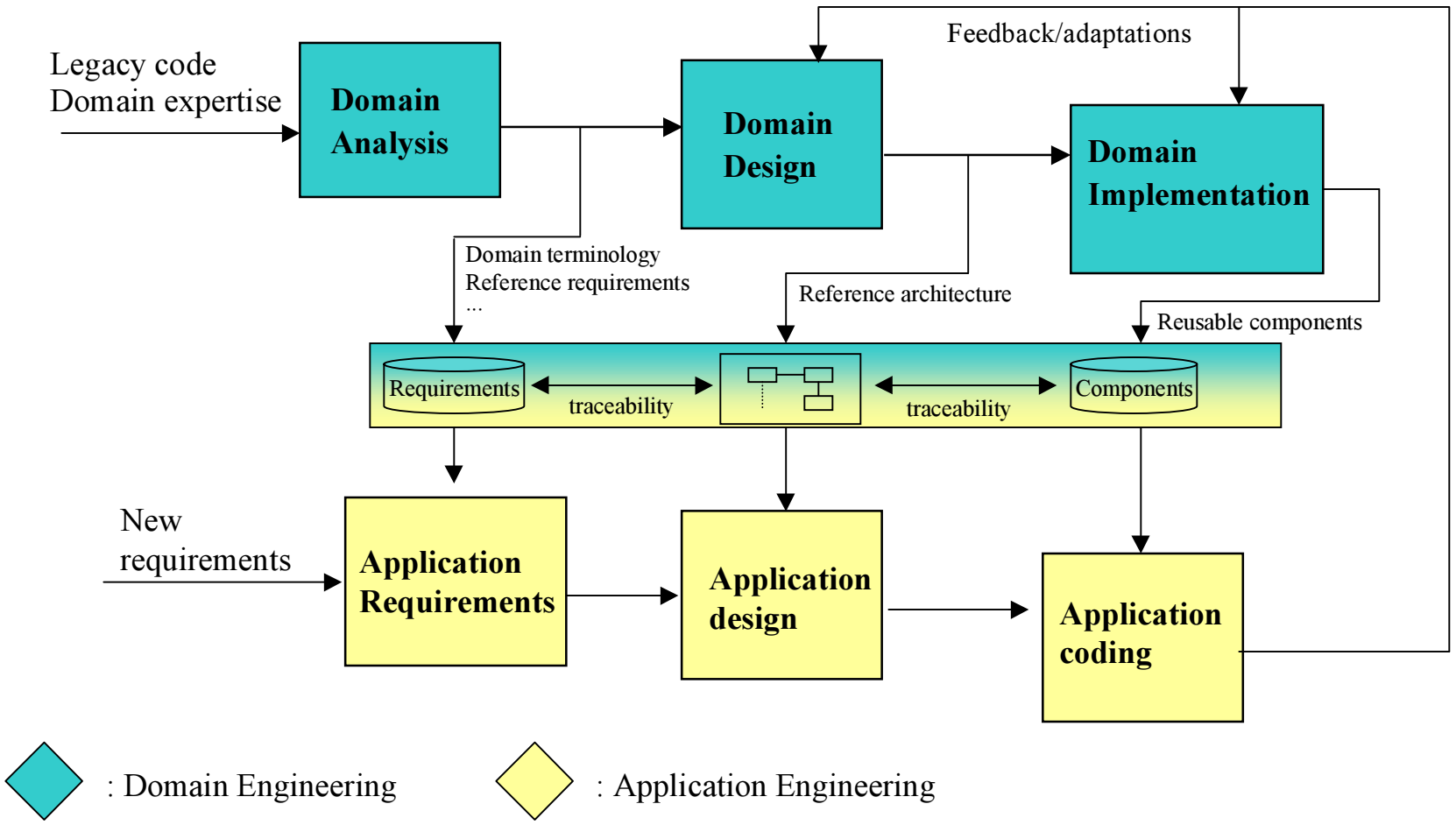
- Extensive domain experience
- Existing legacy systems
- Coming systems
- Domain technology is relatively stable
- Operating environment is relatively stable
- Variability can be expressed
- Components are available
- Corporate necessity to migrate to product-line

Non prohibitive factors

- Software size
- Domains
- Business goals
- Complexity or demanding requirements

Product-line process

Product-line process



Domain engineering

- Domain analysis :
  - requirements description
  - domain definition (entities, attributes, relations, constraints, …)
  - system analysis
  - legacy code description (architecture and traceability with requirements)
- Reference architecture design :
  - reference requirements
  - reference architecture
  - traceability
- Development :
  - library of reusable components
  - Exploitation environment

Application engineering

The goal is to reuse the work-products of domain engineering in order to produce a new application satisfying specific requirements.

It entails the following activities :

- Identify specific requirements
- Identify architectural modifications
- Modify/adapt/generate components
- Build the global application

⇒ Depends strongly on the environment produced by DE

Software Architecture

What is software architecture ?

The software architecture of a computing system is an abstract description of components and their connections, through multiple and complementary views.

- Components & connectors
- Classes & relations
- Structured programming
- Formula translator
- Machine language

Definitions

Hayes-Roth, 1994 : DSSA

An abstract system specification consisting primarily of functional components described in terms of their behaviors and interfaces and component-component interconnections.

Architecture are usually associated with a rational that document and justifies constraints on component and interconnection or explains assumptions about the technologies which will be available for implementing applications consistent with the architecture.

David Garlan and Perry, 1995 : CMU

The structure of the components of a system, their interrelationships, and principles and guidelines governing their design and evolution over time.

Bass Clements and Rick Kazman, 1998 : SEI

The software architecture is the structure or structures of a system, which comprise software components, the externally visible properties of those components, and the relationships among them.

Architectural views

A software architecture incorporates different views, including :

- Structural view : components and their connections
- Dynamic view : data flow and control flow in the architecture
- Computing view : hardware and software hosting the architecture
- more specific views ...

⇒ Complementary views

Architectural design

Lack of guidance for architectural design :

- Software processes and design notations are fine for defining/ordering activities, but that is not enough
- Language-specific mechanisms (classes, inheritance, …) are not enough either (too low level).

Good designers rely a lot on experience to build elegant, flexible architectures. There is a need of high level design description to capture this experience.

⇒ Study of architectural styles to support the design process.

Architectural style

An architectural style :

- Defines a family of architectures constrained by :
  - Component/connector vocabulary
  - Topology
  - Data and control flows
  - Semantic constraints
- Encapsulates rationale about architectural elements
- Emphasizes constraints on the elements and their relationships

Some architectural styles

Identified styles (Garlan and Shaw) :

- Pipes and Filters
- Layered organizations
- Distributed processes
- Repositories
- Event-based, implicit invocation
- Object-oriented
- Main program/Subroutine, explicit invocation

⇒ Lack of homogeneity
⇒ No quality attributes

Reference Architecture

Reference architecture

A reference architecture is a generic, adaptable software architecture that specifies software components and their relationships through multiple views.

It is an abstract structure that captures common aspects of a product family and encapsulates variable features:

Adaptations can affect :

- the components
- the connections
- the topology
- the constraints

$\Rightarrow$ Control flow is stable

Reference architecture

Requirements

Functional
- common
- variable

Non functional
(global properties, performances)
- common
- variable

Explicit architecture description
- component diagrams
- component dependencies
- component activity diagrams
- deployment diagrams

Design trade-off
- different alternatives
- key attributes
- decision techniques
- implementation constraints

Product-line goal
- cost estimation
- code generation

Implementation

Components

Middleware

Hardware platform

Evolution techniques

Explicit architectural views **enhanced** with specific evolution techniques

- structuration techniques (layer pattern, micro kernel pattern, delegation, …)
- decision / propagation techniques

- Properties
- trade-off

- resources
- performances

- implementation constraints
- components configurations

Structuration techniques

- AOP
- framework OO

Decision techniques

- decision tree
- optimization
- reuse contracts

Propagation techniques

- hypertext/XML

Ongoing Experiments and Projects

Esprit Project PRAISE

http:/www.esi.es/Projects/Reuse/Praise/

- PRAISE : Product-line Realization and Assessment in Industrial SEttings
- Partners :
  - Thomson-CSF / LCR (prime)
  - Bosch
  - Ericsson
  - ESI (European Software Institute)
- Duration : 18 months (Kickoff : September 1998)

Alcatel/Thomson-CSF Common Research Lab

- LCAT : Product-line Realization and Assessment in Industrial SEttings
- Research topics :
  - Technology for the development of architecture-centric software product lines (4 domains)
  - Multi-target execution support for real-time, distributed systems
- Kickoff : September 1998

Conclusions

Conclusion : Advantages of product-lines

Reduction of development and maintenance costs

- analysis/design/code reuse

Acceleration of application generation

Quality improvement

- architecture has been tested and validated on previous applications
- code can be highly optimized (in terms of efficiency, safety,…)

Conclusion : weaknesses of product-lines

Product-lines are costly:

- involve highly qualified people and a large adherence
- several applications are needed to amortize the cost
- steep learning curve

Reference architectures are hard to design:

- lack of guidelines and techniques
- generalize from concrete examples
- think architecture and look for patterns

Lack of tools to represent/validate reference architectures

Lack of integrated tools to develop and exploit product-lines

Research directions (reminder)

- Defining an adapted software process
- Architecture design using styles
- Architecture evaluation using quality attributes
- Extensions of UML for architectural representation
- Design of reference architectures and variability and decision representation
- Integrated tools to support product-line
- Assessment model