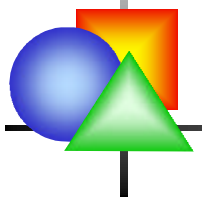
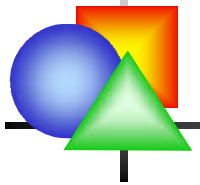
Reuse in the Software Development Process



Roland J. Weiss

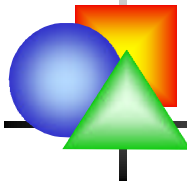
06/06/2003



What is Software Reuse?

- **Definition:** Software Reuse is the process of creating software systems from predefined software components.

- **Systematic reuse** requires a repeatable development process tailored towards reuse.
- **Ad Hoc reuse** exploits arbitrary reusable software artifacts during development.

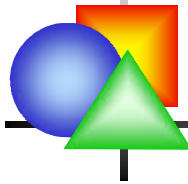


Why Reuse?

1. **Quality:** Reusing tested, revised components avoids repeating errors and harvests existing knowledge and experience.

2. **Speed:** Minimize time-to-market for software, components speed up development.

3. **Costs:** Updating components effects whole application families (maintenance), reduced creation time lowers development costs.



What to Reuse?

Documentation
(styles, templates, …)

Components

Modules

Designs/Design Patterns

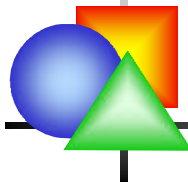
Architectures

Domain models

Sets of requirements

Test sets and frameworks

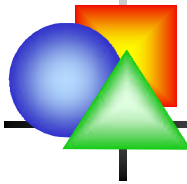
Standard Software



Software Components & Assets

- **Component:** Source or object code that provides a special service through a public interface.
  - The interface consists of provided operations and requirements for proper functioning.
  - Examples: ActiveX, JavaBeans, CORBA, STL.

- **Asset:** Software artifact that is deliberately designed for reuse.

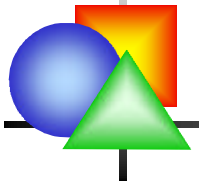


The Two Dimensions of Reuse

1. **Software Engineering Dimension**
   - Evaluate and select relevant technologies (component model, tools, languages/language constructs etc.)
   - Integrate reuse into software development processes
   - Identify and develop reusable components and create systems with them

2. **Business Engineering Dimension**
   - Provide funding and long-term commitment
   - Establish necessary infrastructure
   - Coordination of reuse effort across business units and projects
   - Measure reuse success

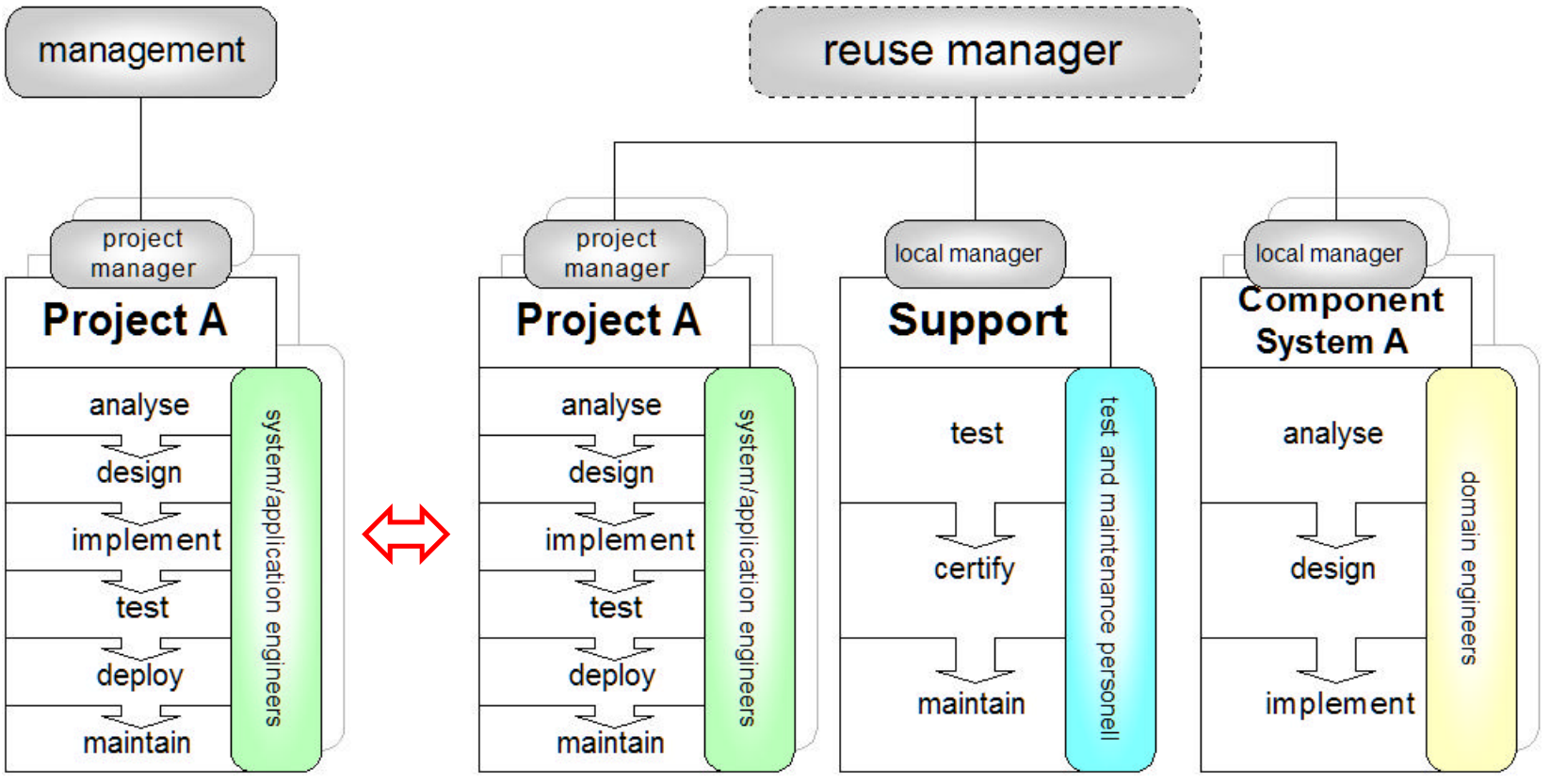
Business Engineering

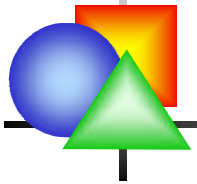


Reuse is Special

- Changes to organizational structures required
  - Special component development and support group(s) required
  - Coordination of reuse efforts essential
- Changes to established development processes required
  - Domain engineering
  - Evaluation of component library, selection of appropriate components
- Reuse works best when applied across projects, business units, or even organizations
  - Greater benefit due to increased reuse opportunities
  - Larger basis for domain engineering

Traditional vs. Reuse Business Organization





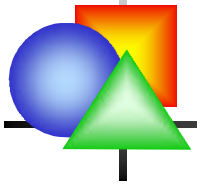
Establishing Reuse

- Initial investments necessary
  - Establish organizational and process changes
  - Develop component repository/library
- Investments begin to recover after 2-5 years
  - Faster application development, …

➔ **Consensus:** Reuse can only succeed with dedicated management support.

- Favored approach: Incremental transition
  - Initiate reuse with pilot project with great reuse potential
  - Extend reuse program to other business units
  - Classical Paper: R. Joos, Software Reuse at Motorola (1994)

Software Engineering

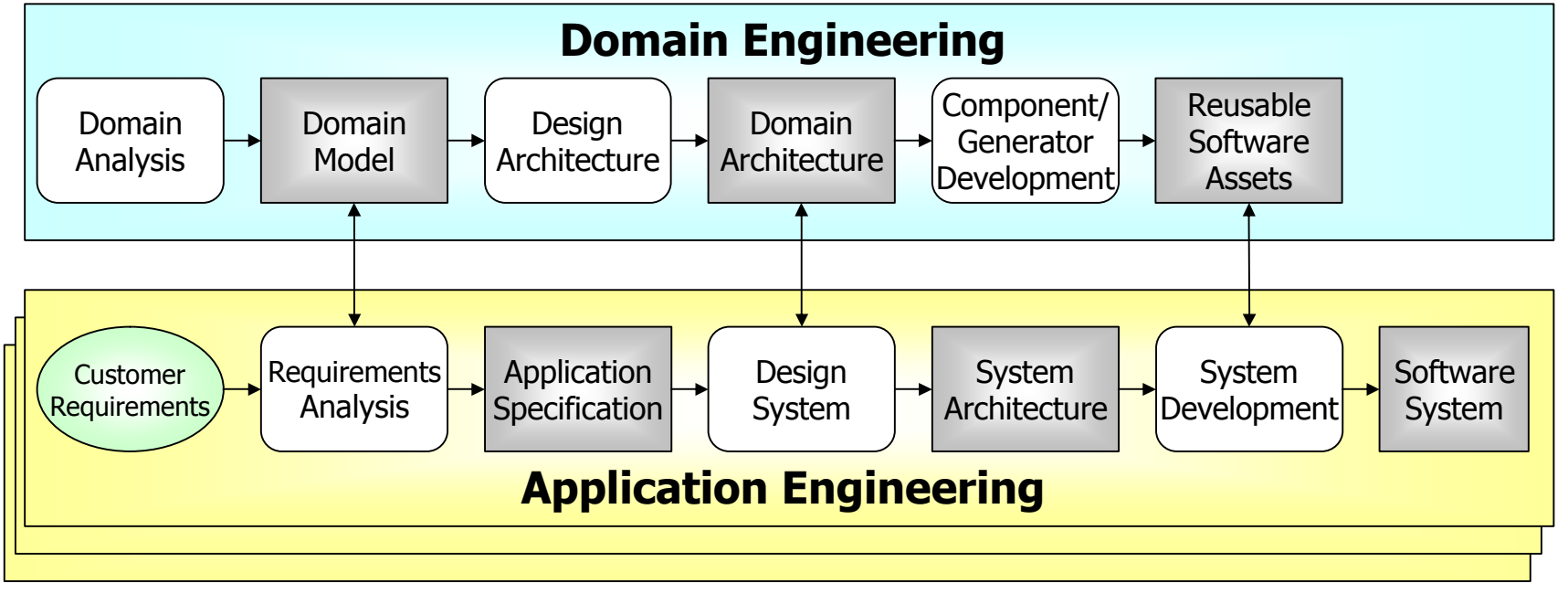


Two Major Activities

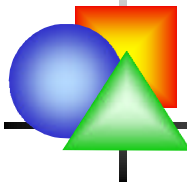
1. Application/System Engineering
   - Create sets of related applications (families) based on reusable components

2. Domain Engineering
   - Detect commonalities and variabilities to create domain model
   - Develop reusable software artifacts (components, generators)

Two Life Cycle Model



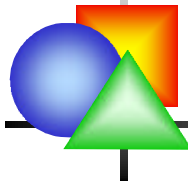
- Correlates life cycles of domain and application engineering
- Domain and application engineers have to communicate



Domain Engineering

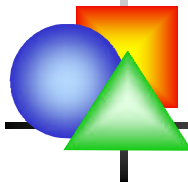
1. Domain analysis
   - Domain scoping (identification, description)
   - Domain modeling (definition, lexicon, feature and concept models)
2. Domain design
   - Develop architecture (decomposition of a system's elements, their relations and constraints)
   - Devise production plan (assembly of concrete system; integrating change requests, measuring the production process)
3. Domain implementation

➔ Czarnecki/Eisenecker: Generative Programming (2000)



Domain Analysis

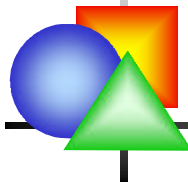
- Bottom-Up Domain Analysis
  - Examine existing applications
  - Identify reusable components according to commonalities and variabilities
  - Example: AT&T reuse effort
- Top-Down Domain Analysis
  - Develop enterprise models
  - Detect overlap and future trends
  - Which parts should be created from reusable components, which should be added to the component repository
- Combine both approaches
- DA goes beyond recording existing domain expertise
- Formalized DA methods: FODA, ODM



OO Technology and Reuse

- OOD/A traditionally focused on creating single applications
- Objects not well suited for reuse without additional efforts

- Advanced techniques
  1. Frameworks: Implementation skeletons for related applications
  2. Design Patterns: Capture expert knowledge for common design problems



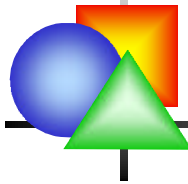
Policy Based Programming

- Design Patterns demanding for users: Understand & Implement

- Approach: Generate custom pattern implementations from pattern templates

1. Pattern templates are implemented in terms of policies
2. Policies capture various design decisions inherent to the pattern, e.g. allocation strategy, threading policy
3. Default policies implement established solutions, user can provide custom policies

➔ A. Alexandrescu: Modern C++ Design (2001)



Summary & Conclusions

- Reuse requires changes to business and software engineering processes
- Reuse relies on cooperation and trust: New enterprise culture necessary
- Reuse offers great potential, but holds serious risks if implemented inappropriately