2 Reuse-driven Software Engineering Business (RSEB)

- To accomplish key business goals and improve business performance:
  - Industrial-strength *OO Software Engineering (OOSE)* and software reuse must become key parts of the organization’s software engineering strategy.

- 70% of software organizations of over 435 software organizations assessed up to Dec. 94 fell in Capability Maturity Level 1 under the underlines established by SEI of Carnegie Mellon University:
  - operates without formalized procedures, cost estimates and project plans.
  - tools not well integrated with the process.
  - change control is weak.

- Organizations move from single-project development into development on a reuse scale:
  - new organization has to be driven by the necessities of software reuse and has to be operated as a business.

- Improving beyond Level 1 is crucial to long-term survival of most business dependent on software.

- Simultaneous pressures faced by software organizations:
  - Reduce time to market
  - Reduce cost of the product
  - Improve organization’s productivity
  - Improve process’s predictivity
  - Increase product’s reliability
  - Increase product’s quality

- Goals of software quality:
  - Able to operate in heterogeneous environments:
    * DCE (Distributed Computing Environment)
* WWW

* Java

* CORBA

- Able to customize software easily for a particular environment, adapt it to fit a new application, or deliver it many local languages and styles.

- Openness: permits other vendors to provide add-on capability.

2.1 Make reuse cost-effective

• RSEB is a concept of 4 interlocking dimensions:

  - Business orientation

  - Engineering orientation

  - Technical sequence

  - Business Process Reengineering (BPR)

• Business orientation:

  - has to be cost-effective

  - has to be effective in terms of time to market

  - needs a clear business reason for reuse and should be producing a reasonable family of related applications.

  - processes have to be measured and controlled.

• Engineering orientation:

  - Architect the family of applications.

  - Identify a domain of interest within the company.

  - Structure the domain to break out components and component systems using an architecture and model-driven process.

  - creators (component engineers) build components using methods based on, e.g., the OOSE approach of Ivar Jacobson.
- Components incorporated in many application systems by developers (reusers) using, e.g., OOSE-based methods.

- **technical orientation:**

  - the RSEB methods involve a series of OOSE models: architecture, analysis, design, test and code.

  - a structure is traceable thro’ these models both forward and backward.

  - reuse not just code, but also high-level elements.

- **BPR (Hammer and Champy, 1993):**

  - Apply OO techniques to the modeling and transformation of business processes

  - not only key business processes may be reengineered, the software development process that creates business ISs may itself be reengineered.

- Requires OO Business Engineering techniques to provide systematic, incremental approach to improve an organization and the ISs on which it depends:

  - helps manage the risks and costs of change in a dynamic environment.

- use OO methods to describe components, architecture, and the processes that create and reuse components.

### 2.2 The Reuse Business

- run as a *software engineering business*.

  - it has business characteristics.

- Reuse-driven software organization engaged in producing multiple, related applications centered and optimized on the production and reuse of components:

  - These applications form a product line (family).

- It must provide a return on the investment put into it.

  - benefits must offset costs.

  - early years: costs expended greater than benefits received.

  - Later years: costs will be offset by the benefits.
• RSEB carefully evaluates and acquires available products and tools such as:
  - ORB (Object Request Brokers)
  - ActiveX components

• Business objects, architectures, and components are found systematically using OO Business Engineering and OOSE-methods.

• RSEB establishes a *reuse-oriented organization*:
  - separates application and component engineering.

• Integrates all cross-life-cycle activities such as version control, metrics, reviews and inspections, and quality assurance.

• Based on Jacobson’s *use case driven architecture (OOSE)* and *process modeling frameworks (OO Business Engineering)*.
  - Modeling concepts and notations of *UML (Unified Modeling Language)*.

### 2.3 Architect components and applications

• Architecture 1 - Engineer application systems from component systems (a set of customizable and configurable software elements):
  - Assembling reusable components drawn from component systems.
  - Early stage: work with reusable IS use cases.
  - Later stage: work with reusable design components or actual code.

• Architecture 2 – as a set of layers:
A reuse business systematically engineers a layer of related application systems, at the top.

- generates related versions of the applications.

- application-specific software is engineered by extensively reusing component systems existing on 3 lower layers.

- components reusable for specific business or application domain area:
  - insurance applications
  - banking systems, human resources and personnel ISs
  - telecoms switches
  - microwave instruments

- Middeware components provide interfaces to other established entities such as GUIs:

- System software components provide interfaces to hardware, such as OS interfacing to the computer.

  - methods for designing a flexible, layered architecture immature.

- New constructs support layered, modular architectures such as large-grain components like OCXs.

- Application systems are built from a combination of concrete and abstract components.

- Component systems can be frameworks or collections of components such as use cases or Java classes.

- Commonality and variability mechanisms: allow effective engineering of components for families of systems.

- OO Business Engineering techniques are applied to identify business objects and application families.
• Reusers may specialize or augment the component systems to adapt them to the new application.

2.3.1 Applications and application systems

• *Families of applications:*
  
  - A suite of applications that work together:
    * e.g., Microsoft Office: a word processor, spreadsheet, database management, etc.

  - A suite of applications that supports customers and workers in a bank:
    * deals with accounts, loans, audits, fund transfers.

• *variants of applications:*

  - different versions of the same application used by different people, countries, or situations.
    * e.g., Ericsson AXE switch: delivered in different forms for a large installation, a small installation or different countries.
    * e.g., different models of HP Laser printer require different versions of firmware.

• *Independent applications* which can be treated as a family:

  - they are built from the same set of components.
    * e.g., Microsoft Foundation Classes (MFC) or a user interface library.
    * e.g., DBMS: implemented from a set of components based on the specialized database architecture that has evolved from building many systems of this type.

2.3.2 Components and component systems

• *component:* an element of a development model that meets 2 criteria:

  - it is loosely coupled to other elements.
  
  - It promises to be reusable.

• Software life-cycle workproducts: potentially components or sources of components.

  - E.g., models used, interfaces, tests, manuals, source code itself.
- E.g., OCXs, Ada packages, C++ components and use cases.

- Some components are fairly formal and structured, some are operational code or tools, others are process descriptions or even less formal documents.

- Component system: a set of related components that accomplishes some function larger than that accomplished by a single component.

- To manage the interfaces between component systems:
  - Export only a subset of the information for use by reusers:
    * publicizing subsets of components thro’ one or more facades.
    * each façade exports only those aspects of the entire component system to be reused.

- More abstract components or component systems contain at least one of a variety of specialization mechanisms.
  - Reuser exploits the mechanism to adapt the component to the particular application.

2.3.3 Layered architecture

- An application explicitly imports components from the component system façade.

- Exports, imports, and facades work together to support both layering and modular pluggability.

- Implement traditional OO frameworks as component systems:
  - have “slots” into which a reuser may plug components to augment their abilities.
  - “slots” may be specialized by using inheritance or several other mechanisms.

- Each application is constructed from components organized as component systems.

- Each component system may be constructed from other, lower-level component systems.

- Reuser constructs each application from a set of application domain, business-specific components or component systems in the top layer.
- can directly reuse some components from lower layers, such as utility classes defined in the middleware or system software layers.

- Middleware layer: provides platform-independent interfaces to distribution and interoperability mechanisms.
  - well-defined, stable interlayer interfaces provide openness and flexibility.

- The architecture allows component systems to evolve independently as new technologies and opportunities arise.

2.4 Software Engineering (SE) processes

- 3 categories of SE processes:
  - component system engineering
  - Application system engineering
  - Application Family engineering

- Overall system architecture is created by the application family engineering process.
  - Identify the architecture and a set of component systems that will support the suite of applications to be built.

- Overall management of the reuse business is a process known as Managing the Reuse Business.

- OO Business Engineering is used to model these software development processes.
  1) Business actors and use cases: model interaction of persons with the software organization.
  2) Models are refined to define the responsibilities of reuse workers, and customized to produce a reuse-oriented software organization structure.

- A business reengineering transition framework and change management techniques: systematically restructure a software development organization into a reuse business.
• People involved in 2 ways in SE processes:

1) Workers e.g., managers, software engineers, librarians, etc.

2) People outside organization: end-users or customers.

- a stick figure: a business actor, i.e., a class of people or roles interacting with a process.

  * **customer**: order, specify and pay for an application system.

  * **end-user**: use an application system when it’s installed.

  * **manufacturer**: receives a new version of an application system, then produces and delivers complete applications to customers and users.

• Application family engineering:

- determines how to decompose overall set of applications into a suite of application systems and supporting component systems.

  * architect the layers, facades and interfaces of the subsystems and component systems that support complete family of related applications.

- an architect: the designer of a family of application systems.

- In some systems, effective component system engineering relies on domain engineering to determine an architecture and set of component systems.

• Application system engineering:

- select, specialize and assemble components from one or more component systems into complete application systems.

- use tools, methods, processes and instructions provided.

- developers express the customers/end-users requirements in terms of available architecture and component models.

  * find appropriate component to reuse if available.

  * seek a new component or develop a model and software to meet the requirements.

- design and implement an increment of the eventual application system.
- system improvement.

- Component system engineering:

  - design, construct and package components into component systems.

  - use appropriate code, templates, models, dictionary, documents and even custom tools.

  1) collect and analyze requirements about both needs and trends from a wide range of sources.

     * business models

     * architects

     * domain experts

     * application users

  - Goal: a consistent model that explicitly expresses commonality and variability across the suite of applications that will reuse these components.

     * use techniques for cost-benefit estimation

  - design concrete components that will be reused as is, specialized.

  - mechanisms used: inheritance and other mechanisms based on problem-oriented languages, parameterized templates, generators and domain-specific kits.

  - components range from simple set of code elements and interface models to complex set of code elements, templates, tests and scripts.

  2) component engineers architect, incrementally design, implement and test the component system.

  3) certification and packaging of the component system for retrieval by reusers.

  4) adapt to changes of infrastructure, etc.

2.5 Establish and manage a reuse business

- organizational structures and management policies and plans are developed as the reuse business is established.
- the reuse business needs to be managed and optimized.

- To deal with complexity and scale of organization: RSEB has to manage risks in a systematic way

- incremental adoption to deal with the scale

- Responsibilities:

  - make sure that plans for the future are made, followed and communicated to upper management.
  
  - decisions on the investment of resources: money, people and time.

- Overall management functions:

  - determine who shall own and maintain a particular component system.
  
  - collect metrics data to manage and optimize the reuse business.
  
  - determine how many workers are needed in future.
  
  - Trade off the management variables such as time to market, cost, functionality, and quality and resolving conflicts over resources and schedule.

- Systematic transition to an RSEB as a process of business reengineering.

  - RSEB approach combines concepts from 3 areas:

    * business process reengineering
  
    * change management
  
    * incremental systematic reuse adoption

- Installing a reuse business - 4 types of organization:

  - those where RSEB improves internal business processes of the organization: customer interaction, MISs or manufacturing processes.
- those in which RSEB is to engineer software into a hardware product: telecommunications switch, defense product, printer, or instrument.

- those in which RSEB develops application systems for outsourcing customers: Anderson Consulting.

- Those in which RSEB develops software products: Microsoft.

2.5.1 Improving internal processes

• In this type of organization, the reuse business exists entirely within one organization: a bank or insurance company.

• Mission:

  1) understand company’s present processes better in order to implement them in software more effectively.

  2) participate in a business reengineering effort.

• Rapid and effective development of a set of integrated and related application systems offers a competitive advantage.

• E.g.: HP’s Corporate ISs:

  - produces a variety of human resource, manufacturing, and financial application software systems for all of the divisions serving the corporation’s 90,000 worldwide employees.

  - systems are customized to local divisional and country needs, but support HP-wide policies and conform to HP-wide standards.

  - engineer component systems that accommodate:

    * corporate ISs identify commonalities that extend across entire corporation.

    * seek out variabilities for different lines of business and different country cultures.

2.5.2 Embedded software
• in this type of organization, the reuse business exists wholly within one business.

- produces products for external users:

  * HP producing laser printers and instruments.

  * Ericsson producing AXE switching system.

  * a defense contractor producing weapons systems for armies, navies and air forces.

- product: based on special purpose hardware, operated and specialized by software.

- design for customization.

  * adapt switching systems to the practices of different telephone companies and the cultures and regulations of different countries.

- design for customization.

• Architectural group: responsible for planning the evolution of the architecture in response to anticipated and actual changes in product features and technology.

• These plans become the input to the software engineers to develop component systems.
2.5.3 Software house

- In this type of organization, the reuse business exists as an organization separate from the user of the software.

- 2 patterns:
  - As an outsourcing vendor, e.g., Anderson Consulting that contracts to develop and operate information services of a client.
    * build business models of the common needs of many clients.
    * assist clients to reengineer their businesses.
    * looks for commonalities and variabilities in these models that provide basis for appropriate component systems.
  - The software house takes full responsibility to build systems or subsystems under contract to a client, but the client will operate the system.
    * often confine their efforts to clients in one industry segment, e.g., power utilities, financial services, or manufacturing.
    * build suitable component systems with variability enabling vendor to adapt them to different circumstances of different clients.

2.5.4 Software product vendor

- In this organization, software products are sold to customers either off the shelf in retail establishments or thro’ direct sales channels.

  - marketed ready-to-use component systems.

  - e.g., vendors of utility software, multimedia software and frameworks fall within this category:

    * Microsoft Office: a suite of interoperable application systems built from common components.

    * banking application frameworks such as CommonPoint by Taligent or Orbix by Iona.