Software Evolution and Maintenance

A Practitioner's Approach

Chapter 9

Reuse and Domain Engineering

Outline of the Chapter

9.1 General Idea

- Formal software reuse (off-the-self components) was first introduced by Dough McIlroy.
- Early development of reuse include the concept of program families introduced by David Parnas.
- A set of programs with several common attributes and features is known as a program family.
- At the same time the concepts of domain and domain analysis introduced by Jim Neighbors.
- Domain analysis means finding objects and operations of a set of similar software systems in a specific problem domain.
- In this context, software reuse involves two main activities: software development ***with*** reuse and software development ***for*** reuse.

9.1 General Idea

There are four types of reusable artifacts as follows:

- **Data reuse:** It involves a standardization of data formats. A standard data interchange format is necessary to design reusable functions.
- **Architectural reuse:** This means developing: (i) a set of generic design styles about the logical structure of software; and (ii) a set of functional elements and reuse those elements in new systems.
- **Design reuse:** This deals with the reuse of abstract design. A selected abstract design is custom implemented to meet the application requirements.
- **Program reuse:** This means reusing executable code. For example, one may reuse a pattern-matching system, that was developed as part of a text processing tool, in a database management system.

9.1 General Idea

- The reusability property of a software asset indicates the degree to which the asset can be reused in another project.
- For a software component to be reusable, it needs to exhibit the following properties that directly encourage its use in similar situations
    1. Environmental independence.
    2. High cohesion.
    3. Low coupling.
    4. Adaptability.
    5. Understandability.
    6. Reliability.
    7. Portability.

9.1.1 Benefits of Reuse

- One benefits in several ways from software reuse.
- The most obvious advantage of reuse is economic benefit.
- Other tangible benefits of reuse are as follows:
    1. Increased reliability.
    2. Reduced process risk.
    3. Increase productivity.
    4. Compliance with standards.
    5. Accelerated development.
    6. Improved maintainability.
    7. Less maintenance effort and time.

9.1.2 Reuse Models

- Development of assets with the potential to be reused requires additional capital investment.
- The organization can select one or more reuse models that best meet their business objectives, engineering realities, and management styles.
- Reuse models are classified as:
  - **Proactive.**
  - **Reactive**.
  - **Extractive.**

9.1.2 Reuse Models

Proactive approaches

- In proactive approaches to developing reusable components, the system is designed and implemented for all conceivable variations; this includes design of reusable assets.
- A proactive approach is to product lines what the Waterfall model is to conventional software.
- The term product line development, which is also known as domain engineering, refers to a "***development-for-reuse***" process to create reusable software assets (RSA)
- This approach might be adopted by organizations that can accurately estimate the long-term requirements for their product line.
- This approach faces an investment risk if the future product requirements are not aligned with the projected requirements.

9.1.2 Reuse Models

Reactive approaches

- In this approach, while developing products, reusable assets are developed if a reuse opportunity arises.
- This approach works, if

  (i) it is difficult to perform long-term predictions of requirements for product variations.

  (ii) an organization needs to maintain an aggressive production schedule with not much resources to develop reusable assets. The cost to develop assets can be amortized over several products.
- However, in the absence of a common, solid product architecture in a domain, continuous reengineering of products can render this approach more expensive.

9.1.2 Reuse Models

Extractive approaches

- The extractive approaches fall in between the proactive approaches and the reactive ones.
- To make a domain engineering's initial baseline, an extractive approach reuses some operational software products.
- Therefore, this approach applies to organizations that have accumulated both artifacts and experiences in a domain, and want to rapidly move from traditional to domain engineering.

9.1.3 Factors Influencing Reuse

- Reuse factors are the practices that can be applied to increase the reuse of artifacts.
- Frakes and Gandel identified four major factors for systematic software reuse: ***managerial, legal, economic and technical***.

Managerial

Systematic reuse requires upper management support, because:

(i) it may need years of investment before it pays off.

(ii) it involves changes in organization funding and management structure that can only be implemented with executive management support.

Legal

- This factor is linked with cultural, social, and political factors, and it presents very difficult problems.
- Potential problems include proprietary and copyright issues, liabilities and responsibilities of reusable software, and contractual requirements involving reuse.

9.1.3 Factors Influencing Reuse

Economic

- Software reuse will succeed only if it provides economic benefits.
- A study by Favaro found that some artifacts need to be reused more than 13 times to recoup the extra cost of developing reusable components.

Technical

- This factor has received much attention from the researchers actively engaged in library development, object-oriented development paradigm, and domain engineering.
- A reuse library stores reusable assets and provides an interface to search the repository.
- One can collect library assets in a number of ways: (i) reengineer the existing system components; (ii) design and build new assets; and (iii) purchase assets from other sources.

9.1.4 Success Factors of Reuse

The following steps aid organizations in running a successful reuse program:

- Develop software with the product line approach.
- Develop software architectures to standardize data formats and product interfaces.
- Develop generic software architectures for product lines.
- Incorporate off-the-shelf components.
- Perform domain modeling of reusable components.
- Follow a software reuse methodology and measurement process.
- Ensure that management understands reuse issues at technical and non-technical levels.
- Support reuse by means of tools and methods.
- Support reuse by placing reuse advocates in senior management.
- Practice reusing requirements and design in addition to reusing code.

9.2 Domain Engineering

- The term domain engineering refers to a **development-for-reuse** process to create reusable software assets (RSA).
- It is also referred to as **product line development**.
- Domain engineering is the set of activities that are executed to create **reusable software** assets to be used in specific software projects.
- For a software **product family**, the requirements of the family are identified and a reusable, generic software structure is designed to develop members of the family.
- In the following slides, we explain analysis, design, and implementation activities of domain engineering.

9.2 Domain Engineering

Domain Analysis

- Domain analysis comprises three main steps:
  - identify the family of products to be constructed;
  - determine the variable and common features in the family of products;
  - develop the specifications of the product family.
- The **Feature Oriented Domain Analysis (FODA)** method developed at the Software Engineering Institute is a well-known method for domain analysis.
- The FODA method describes a process for domain analysis to discover, analyze, and document commonality and differences within a domain.

9.2 Domain Engineering

Domain Design

- Domain design comprises two main steps:
  - develop a generic software architecture for the family of products under consideration; and
  - develop a plan to create individual systems based on reusable assets.
- The design activity emphasizes a common architecture of related systems.
- The common architecture becomes the basis for system construction and incremental growth.
- The design activities are supported by architecture description languages (ADLs), namely, Acme, and interface definition languages (IDLs), such as Facebook's Thrift.

Domain Implementation

- Domain implementation involves the following broad activities:
  - identify reusable components based on the outcome of domain analysis;
  - acquire and create reusable assets by applying the domain knowledge acquired in the process of domain analysis and the generic software architecture constructed in the domain design phase;
  - catalogue the reusable assets into a component library.
- Development, management, and maintenance of a repository of reusable assets make up the core of domain implementation.

9.2 Domain Engineering

Application Engineering

- Application engineering (a.k.a. product development) is complementary to domain engineering.
- It refers to a **development-with-reuse** process to create specific systems by using the fabricated assets defined in domain engineering.
- Application engineering composes specific application systems by:

  (i) reusing existing assets;

  (ii) developing any new components that are needed;

  (iii) reengineering some existent software; and

  (iv) testing the overall system.
- Similar to the standard practices in software engineering [27], it begins by eliciting requirements, analyzing the requirements, and writing a specification.

9.2 Domain Engineering

Relationship Between Application & Domain Engineering

- Both domain and application engineering processes feed on each other, as illustrated in Figure 9.1.
- Application engineering is fed with reusable assets from domain engineering, whereas domain engineering is fed with new requirements from application engineering.

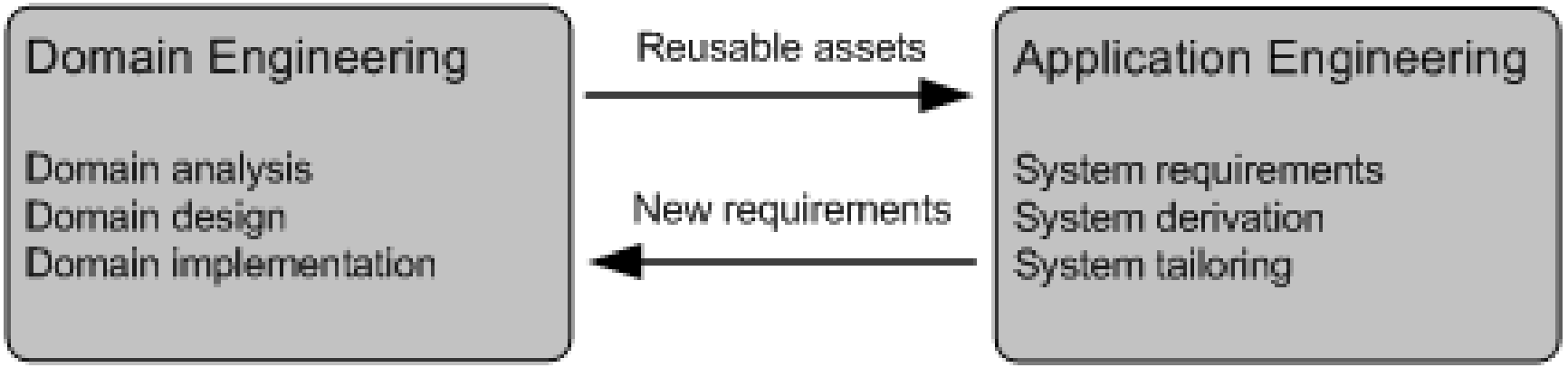


Figure 9.1 Feedback between domain and application Engineering

9.2 Domain Engineering

Domain Engineering Approaches

- The following nine domain engineering approaches reported in literature:
  - Draco
  - Domain Analysis and Reuse Environment (DARE)
  - Familyoriented Abstraction, Specification, and Transportation (FAST)
  - Feature-Oriented ReuseMethod(FORM)
  - ”Komponentbasierte Anwendungsentwicklung” (KobrA)
  - Product line UML-basedsoftware engineering (PLUS)
  - Product Line Software Engineering (PuLSE)
  - Koala
  - Reusedriven Software Engineering Business (RSEB)

9.3 Reuse Capability

- Reuse capability concerns gaining a comprehensive understanding of the development process of an organization with respect to reusing assets and establishing priorities for improving the extent of reuse.
- The concept of reuse opportunities is used as a basis to define reuse **efficiency** and reuse **proficiency**.
- An asset provides a reuse opportunity when the asset – to be developed or existing – satisfies an anticipated or current need.
- There are two broad kinds of reuse opportunities:
  - ***Targeted reuse opportunities*** are those reuse opportunities on which the organization explicitly spends much efforts.
  - ***Potential reuse opportunities*** are those reuse opportunities which will turn into actual reuse, if exploited. Not always a targeted opportunity turns into a potential opportunity.

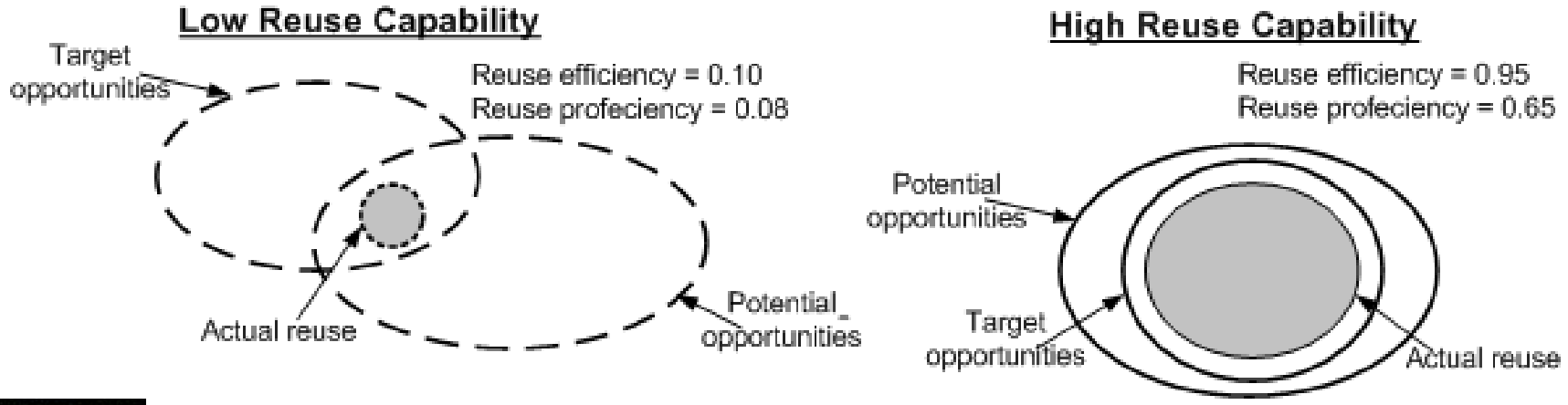
9.3 Reuse Capability

- We define reuse proficiency and reuse efficiency by means of R_A, R_P , and R_T , where
  - R_A counts the actual reuse opportunities exploited.
  - R_P counts the potential opportunities for reuse.
  - R_T counts the targeted opportunities for reuse.
- Reuse **proficiency** is the ratio R_A/R_P .
- Reuse **efficiency** is the ratio R_A/R_T .
- Reuse ***effectiveness*** is represented as: $N\ (C_{NR} - C_R)/C_D$, where:
  - N = number of products, systems, or versions developed with the reusable assets.
  - C_{NR} = cost of developing new assets without using reusable assets.
  - C_R = cost of utilizing, that is, identifying, assessing, and adapting reusable assets.
  - C_D = cost of domain engineering, that is, developing assets for reuse and building a reuse infrastructure.

9.3 Reuse Capability

- The linkage between the two concepts: efficiency, proficiency and reuse capability has been illustrated in Figure 9.2.
- Assume that the areas of the ovals denote the counts of the assets corresponding to those opportunities.
- In terms of the elements of the figure, reuse efficiency is calculated by dividing the area of the actual reuse oval by the area of the target oval.
- Reuse proficiency is calculated by dividing the area of the actual reuse oval by the area of the oval representing potential reuse.

Figure 9.2 Re-use capability ©IEEE, 1993



9.4 Maturity Models

- A reuse maturity model is an aid for performing planning and self-assessment to improve an organization's capability to reuse existing software artifacts.
- Maturity model helps the organization's understanding of their existing and future goals for reuse activities.
- Maturity model can be used in planning systematic reuse. Organizations developing and maintaining.
- In this section we discuss briefly three maturity models:
  - Reuse Maturity Model,
  - Reuse Capability Model.
  - RiSE Maturity Model.

9.4.1 Reuse Maturity Model

- In circa 1991, Koltun and Hudson presented the first reuse maturity model (RMM).
- The model provides a concise form of obtaining information on reuse practices in organizations.
- The model comprises five levels and ten dimensions of reuse maturity as shown in Table 9.1.
- The columns of the table indicate the different levels of reuse maturity.
- Maturity improves on a scale from 1 to 5, where level 1 corresponds to Initial/ Chaotic state and level 5 corresponds to the Ingrained state.
- This model was not applied in real case studies, but are considered as the key insights for the reuse capability model developed by Ted Davis

9.4.1 Reuse Maturity Model

Dimension of Reuse	Reuse Maturity Levels				
	1. Initial/ Chaotic	2. Monitored	3. Coordinated	4 Planned	5. Ingrained
Motivated/ Culture	Reuse discouraged	Reuse encouraged	Reuse incentivized re-enforced rewarded	Reuse indoctrinated	Reuse in the way we do business
Planning for reuse	None	Grassroots activity	Targets of opportunity	Business imperative	Part of strategic plan
Breadth of reuse	individual	Work group	Department	Division	Enterprise wide
Responsible for making reuse happen	Individual initiative	Shared initiative	Dedicated individual	Dedicated group	Corporate group with division liaisons
Process by which reuse is leveraged	Reuse process chaotic; unclear how reuse comes in	Reuse questions raised at design reviews (after the fact)	Design emphasis placed on off the shelf parts	Focus on developing families of products	All software products are genericized for future reuse
Reuse assets	Salvage yard (no apparent structure to collection)	Catalog identifies language and platform specific parts	Catalog organized along application specific lines	Catalog includes generic data processing functions	Planned activity to acquire or develop missing pieces in catalog
Classification activity	Informal, individualized	Multiple independent schemes for classifying parts	Single scheme catalog published periodically	Some domain analyses done to determine categories	Formal, complete consistent timely classification
Technology support	Personal tools, if any	Many tools, but not specialized for reuse	Classification aids and synthesis aids	Electronic library separate from development environment	Automated support integrated with development environment
Metrics	No metrics on reuse level, pay-off, or costs	Number of lines of code used in cost models	Maturity tracking of reuse occurrences of catalog parts	Analyses done to identify expected payoffs from developing reusable parts	All system utilities, software tools and accounting mechanisms instrumented to track reuse
Legal, contractual accounting considerations	Inhibitor to getting started	Internal accounting scheme for sharing costs and allocating benefits	Data rights and compensation issues resolved with customer	Royalty scheme for all suppliers and customers	Software treated as key capital asset

Table 9.1: Reuse Maturity Model [43] (©[1996] ACM).

9.4.2 Reuse Capability Model

- RCM comprises two models, namely
  - an assessment model, and
  - an implementation model.
- An organization can use the assessment model to:
  - understand its current capability to reuse artifacts, and
  - discover opportunities to improve its reuse capability.
- A set of critical success factors are at the core of the assessment model.
- The success factors are described as goals that an organization uses to evaluate the present state of their reuse practice.
- The organization can apply the implementation model in prioritizing the critical factor goals by grouping them into stages.

9.4.2 Reuse Capability Model

Assessment model:

- The success factors in the assessment model are grouped into four categories: **application development, asset development, management, and process and technology.**
- The critical success factors have been listed in Table 9.2.

Application Development Factors	Asset Development Factors	Management Factors	Process and Technology Factors
Asset awareness and accessibility	Needs identification	Organizational commitment	Process definition and integration
Asset identification	Asset interface and architecture definition	Planning and direction	Measurement
Asset evaluation and verification	Needs and solution relationships	Cost and pricing	Continuous process improvement
Application integrability	Commonality and variability definition	Legal and contractual constraints	Training
	Asset value determination		Tool support
	Asset reusability		Technology innovation
	Asset quality		

Table 9.2: Critical Success Factors

9.4.2 Reuse Capability Model

Assessment model:

- Each critical success factor is defined in terms of one or more goals.
- The goals describe what is to be achieved – and not how those goals can be realized.
- Therefore, there is much flexibility in achieving those goals.
- As an example, the **needs identification** factor has the following goals:
  - Identify the current needs for solutions of the developer.
  - Identify the anticipated needs for solutions of the developer.
  - Identify the current needs for solutions of the customer.
  - Identify the anticipated needs for solutions of the customer.
  - Use the identified needs as a reference to develop or acquire reusable assets to meet the specified needs.

9.4.2 Reuse Capability Model

Implementation model:

- The goals are divided into four stages,

Opportunistic

- A common reuse strategy does not fit all projects so each project develops its own strategy to reuse artifacts.
- The strategy includes:

  (i) defining reuse activities in the project plan.

  (ii) using tools to support the reuse activities.

  (iii) identifying the needs of the developers and developing or acquiring reusable artifacts.

  (iv) identifying reusable artifacts throughout the lifecycle of the project.

9.4.2 Reuse Capability Model

Implementation model:

Integrated

- The organization defines a reuse process and integrates it with its development process.
- It is important for the organization to support the reuse process by means of policies, procedures, resource allocation, and organizational structure.

Leveraged

- To extract the maximum benefits from reuse in groups of related products, a strategy for reuse in product lines is developed.

Anticipating

- Reusable assets are acquired or developed based on anticipated customer needs.

9.4.3 RiSE Maturity Model

- The RiSE maturity model was developed during the RiSE project through discussions with industry partners.
- The RiSE maturity model includes:

  (i) reuse practices grouped by perspectives and in organized levels representing different degrees of software reuse achieved.

  (ii) reuse elements describing fundamental parts of reuse technology, such as assets, documentation, tools and environments.
- The five maturity levels are as follows:

  Level 1: Ad-hoc reuse.

  Level 2: Basic Reuse.

  Level 3: Initial Reuse.

  Level 4: Integrated Reuse.

  Level 5: Systematic Reuse.

9.4.3 RiSE Maturity Model

- In the RiSE Maturity Model, fifteen factors were considered, and those are divided into four perspectives: **organizational, business, technological, and processes.**

Factors of Influence	Levels				
	1. Ad-hoc	2. Basic	3. Initial	4 Organized	5. Systematic
Planning for reuse	- Nonexistent	- Grassroots activity - Reuse is viewed as single-point opportunities - Individual achievements are rewarded	- Targets of opportunity - Organization responsible for reuse - A key business strategy	- Business imperative - Reuse occurs across all functional areas	- Part of a strategic plan - Discriminator in business success
Software reuse education	- Lack of expertise by staff members - Frequent resistance to reuse	- Basic definitions of reuse are agreed upon	- The staff has the expertise and how to obtain benefits with reuse	- The staff members know the reuse vocabulary and have reuse expertise	- All definitions, guidelines, standards are in place, enterprise-wide
Legal, contractual, accounting considerations	- Inhibitor to getting started	- Internal accounting scheme for sharing costs allocating benefits	- Data rights and compensation issues resolved with customer	- Royalty scheme for all suppliers and customers	- Software treated as key capital asset
Funding, costs, and Financial Features	- Costs of reuse are unknown	- Costs of reuse are "feared"	- Payoff of reuse is "known" and understand for a given domain - Investments made in reuse, payoffs expected - Costs of reuse are "known"	- All costs associated with an asset's development and all savings from its reuse are reported and shared	- All costs associated to a product line or a particular asset and all saving from its reuse are reported and shared
Rewards and incentives	- Reuse is discouraged by management	- Reuse is encouraged	Reuse is motivated reinforced, rewarded	- Reuse is indoctrinated	- Reuse is "the way we do business"
Independent reusable assess development team	- Individual initiative (personal goal as time allows)	- Shared initiative	- Dedicated individual	- Dedicated group	- Corporate group (for visibility not control) with division liaisons

Table 9.3: RiSE maturity model levels: organizational factors [42]

9.4.3 RiSE Maturity Model

Factors of Influence	Levels				
	1. Ad-hoc	2. Basic	3. Initial	4 Organized	5. Systematic
Product family approach	- Isolated products - No family product approach	- Common features and requirements across the products - Commonalities and reuse possibilities were identified	- Product line domain analyses performed	- Focus on developing families of products - Domain engineering performed	- Domain analysis performed across all product lines - Product family approach
Software reuse education	- Chaotic development process unclear where reuse comes in	- Reuse questions raised at design reviews (after the fact) - Development process defined (some reuse activity indications)	- Design emphasis placed on reuse of off-the-shelf parts - Product line domain analyses performed - Shared understanding of all the activities needed to support reuse	- Focus on developing families of products - Reuse-based processes are in place to support and encourage reuse - Domain engineering performed	- All software products generated for future reuse - Domain analyses performed across all product lines - Product family approach

Table 9.4: RiSE maturity model levels: business factors [42]

Factors of Influence	Levels				
	1. Ad-hoc	2. Basic	3. Initial	4 Organized	5. Systematic
Repository systems usage	- Salvage yard (No apparent structure to collection)	- Catalog identifies language and platform specific parts - Simple structure like concurrent versions systems - Considered mainly source code	- Catalog includes generic data processing functions - Considered software components, reports and document models	- Catalog organized along application specifications - Have all data needed decide which assets to build/acquire - Considered screen generators database elements and test cases	- Planned activity to acquire or develop missing pieces in catalog - Considered all artifacts of software development life cycle
Technology support	- Personal tools, if any	- A collection of tools, e.g., CM, but not specialized to reuse - General-purpose analyzers combined to assess reuse levels	- Classification, aids and synthesis aids - Standardization on components and architecture - Tools customized to support reuse	- Digital library separate from development environment	- Automated support integrated with development system - Fully integrated with development and reporting systems

Table 9.5: RiSE maturity model levels: technological factors [42]

9.4.3 RiSE Maturity Model

Factors of Influence	Levels				
	1. Ad-hoc	2. Basic	3. Initial	4 Organized	5. Systematic
Quality models usage	- No quality model adoption	- Some quality activities were incorporated in the software development process	- Software development process guided by a quality model	- High quality model usage in the engineering department	- Quality model completely adopted in the organization activities
Software reuse measurement	- No metrics on level of reuse, payoff or cost of reuse	- Number of lines of reused code factored into cost models	- Manual tracking of reuse occurrences of catalog parts	- Analyses performed to identify expected payoffs from developing reusable parts	- All system utilities, software tools, and accounting mechanisms instrumented to track reuse
Systematic reuse process	- No reused-based process	- Some reuse activities were adopted in the development process - Planning to adapt the software development process of the organization for a reuse-based process	- Development process of the organization is adapted to reuse concepts	- Reuse benefits and concepts are clear for the engineering team - Development process is reused-based	- Systematic reuse process is enterprise-wide
Origin of the reused asses	- No reuse assets	- Build from scratch, some times indirectly	- Build from existent products; adapting existing products	- Build from existing products; extracted through a reengineering process	- Planning the design and building of reusable assets according to product family
Previous development of reusable assets	- No development of reusable assets	- Parallel with development	- Before development	- Before development	Before development

Table 9.6: RiSE maturity model levels: processes factors [42]

9.5 Economic Models of Software Reuse

- Project managers and financial managers can use the general economics model of software reuse in their planning for investments in software reuse.
- The project manager needs to estimate the costs and potential payoffs to justify systematic reuse.
- Increased productivity is an example of payoff of reuse.
- We will discuss **cost model** of Gaffney and Durek, application **system cost model** of Gaffney and Cruickshank, and **business model** of Poulin and Caruso.

9.5.1 Cost Model of Gaffney and Durek

- Two cost and productivity models proposed by Gaffney and Durek for software reuse are:
  - first order reuse cost model.
  - higher order cost model.
- The cost of reusing software components has been modeled in the first order reuse cost model.
- Whereas the higher order cost model considers the cost of developing reusable assets.

9.5.1 Cost Model of Gaffney and Durek

First Order Reuse Cost Model

- In this model, we assume the following conditions:
  - The reused software satisfies the black-box requirements in the sense that it is stable and reliable.
  - Users of the reusable components have adequate expertise in the context of reuse.
  - There is adequate documentation of the components to be reused.
  - The cost of reusing the components is negligible.
- Three broad categories of program code are used in a project:
  - S_n: It represents the new code added to the system.
  - S_o: It represents the original source code from the pre-existing system. S_o includes both lifted code and modified code. Lifted code means unchanged, original code taken from pastreleases of a product. The source code from modified (partial) parts are not considered as reused code.
  - S_r: It represents the reuse source code that are not developed or maintained by the organization. The reuse code is obtained from completely unmodified components normally located in a reuse library.

9.5.1 Cost Model of Gaffney and Durek

First Order Reuse Cost Model

The effective size, denoted by S_e, is an adjusted combination of the modified source code and the new source code, as given in the following equation:

$$S_e = S_n + S_o(A_d \times F_d + A_i \times F_i + A_t \times F_t)$$

where:

A_d = is a normalized measure of design activity,

A_i = is a normalized measure of integration activity,

A_t = is a normalized measure of testing activity, and

$A_d + A_i + A_t = 1$.

Letting S_r denote the estimated size of reusable components, the relative sizes of reusable components is given by R, where R is expressed as follows:

$$R = S_r / (S_e + S_r)$$

9.5.1 Cost Model of Gaffney and Durek

First Order Reuse Cost Model

- Let C be the cost of software development for a given product relative to that for all new code (for which C = 1).
- Let R be the proportion of reused code in the product as defined earlier ($R \leq 1$).
- Let b be the cost, relative to that for all new code, of incorporating the reused code into the new product. Note that b = 1 for all new code.
- The relative cost for software development is:

  [(relative cost of all new code) * (proportion of new code)]

  +

  [(relative cost of reused software) * (proportion of reused software)].

Therefore: $C = (1)(1 - R) + (b)(R) = (b - 1)R + 1$ and the associated relative productivity is: $P = 1 / C = 1 / (b - 1)R + 1$

b must be < 1 for reuse to be cost effective.

9.5.1 Cost Model of Gaffney and Durek

First Order Reuse Cost Model: Example

Activity	Activity Code	Activity Cost
Requirements	Req	0.08
Design	Des	0.37
Implementation	Imp	0.22
Test	Test	0.33

Table 9.7: Relative costs of development activities

Component Type	Activities to be Completed	Relative Reuse Cost (b)
Requirements	Des, Imp, Test	0.92
Design	Req, Imp, Test	0.63
Code	Req, Test	0.41
Requirements, Design, Code,	Test	0.33

Table 9.8: Relative reuse cost (b)

- If we want to reuse a requirements component, the relative cost to reuse requirements is b = (0.37 + 0.22 + 0.33) = 0.92.
- If code is reused, then the additional tasks will involve requirements and testing, the relative cost to reuse code is b = (0.08 + 0.33) = 0.41.

9.5.1 Cost Model of Gaffney and Durek

Higher Order Reuse Cost Model

- Estimating the cost of developing reusable components is key to formulating a reuse cost model.
- By combining the development cost of the reusable components into the economic model, we have:

$$C = (1 - R) \times 1 + (b + \frac{a}{n})R,$$

where a is the cost of developing reusable components relative to the cost of building new non-reusable components from the scratch, and n is the number of uses over which the cost of reusable components is amortized.

- Now, the model can be rewritten as:

$$C = (b + \frac{a}{n} - 1)R + 1$$

9.5.2 Application System Cost Model of Gaffney and Cruickshan

- An application system cost model based on domain engineering and application engineering was proposed by Gaffney and Cruickshank.
- The cost of an application system is expressed as the sum of two component costs:
  - the investment in domain engineering apportioned over N application systems.
  - the cost of application engineering to develop a specific system.
- Therefore, the cost of an application system, C_s is equal to the prorated cost of domain engineering plus the cost of application engineering.
- Let the cost of application engineering be the cost of the new code plus the cost of the reused code in the new application system, and let R denote the fraction of code that is reused code.

9.5.2 Application System Cost Model of Gaffney and Cruickshan

Now, we have $C_s = C_{dp} + C_a$

$$\rightarrow C_s = C_d / N + C_n + C_r$$

where:

$C_{dp} = C_d/N$ and $C_a = C_n + C_r$

C_s = the total cost of the application system

C_d = the total cost of domain engineering

C_{dp} = the prorated portion of C_d shared by each of the N application systems

C_a = the cost of an application system

C_n = the cost of the new code in the application system

C_r = the cost of the reused code in the application system

9.5.2 Application System Cost Model of Gaffney and Cruickshan

Each of the costs, C_d, C_n, and C_r, is taken to be the product of a unit cost (LM/KSLOC) and an amount of code (KSLOC), where LM/KSLOC stands for labor-months/1000 source lines of code. Hence,

$$C_d = C_{de} * S_t, \; C_n = C_{vn} * S_n,$$

and

$$C_r = C_{vr} * S_r$$

The equation for reuse cost is:

$$C_s = C_{us} S_s = \frac{C_{de} S_t}{N} + C_{vn} S_n + C_{vr} S_r$$

where
C_{us} = unit cost of the application system
C_{de} = unit cost of domain engineering
C_{vn} = unit cost of new code developed for this application system
C_{vr} = unit cost of reusing code in this application system
S_t = expected value of the unduplicated size of the reuse library, measured in source statements
S_n = amount of new code in terms of source statements developed for this application system
S_r = amount of reused code incorporated into this application system in source statement
S_s = total size of the application system in source statement

9.5.2 Application System Cost Model of Gaffney and Cruickshan

Let $S_n/S_s = l - R$ and $S_r/S_s = R$, where R is the proportion of reuse. The reuse cost equation can be rewritten as:

$$C_{us} = \frac{C_{de}}{N}\frac{S_t}{S_s} + C_{vn}(1-R) + C_{vr}R$$

Now let $S_t/S_s = K$, the library relative capacity. Thus, the basic reuse cost equation is:

$$C_{us} = \frac{C_{de}}{N}K + C_{vn} - (C_{vn} - C_{vr})R$$

The basic reuse cost equation assumes a single reuse of S_r units (SLOC, KSLOC) in each of the "N application" systems. Thus, this expression is applicable to systematic reuse of units of code relatively dense in functionality.

9.5.3 Business Model of Poulin and Caruso

- Poulin and Caruso developed a model at IBM to improve measurement and reporting software reuse. Their results are based on a set of data points as follows:
- **Shipped source instruction (SSI):** SSI is the total count of executable code lines in the source files of a product.
- **Changed source instruction (CSI):** CSI is the total count of executable code lines that are new, added, or modified in a new release of a product.
- **Reused source instruction (RSI):** RSI is the total source instructions shipped but not developed or maintained by the reporting organization.
- **Source instruction reused by others (SIRBO):** SIRBO is the total lines of source instructions of an organization reused by others. It is calculated as follows:

SIRBO = (Source instructions per part) × (The number of organizations using the part)

9.5.3 Business Model of Poulin and Caruso

- **Software development cost (Cost per LOC):** This metric concerns the development of new software, and it is calculated in two steps:
  - Let S denote the total cost of the organization, including overhead; and divide S by the total outputs of the organization in number of lines of code (LOC).
- **Software development error rate (Error rate):** It is a historical average number of errors uncovered in the products. To estimate the cost of avoiding maintenance, a historical average value is used.
- **Cost per error:** To quantify the advantage of better quality reusable assets, the historical mean cost of maintaining components with traditional development methods is used as a base line. Now, the cost per error metric is calculated in two steps:
  - let S denote the sum of all costs; and divide S by the number of errors repaired.

9.5.3 Business Model of Poulin and Caruso

- The discussed metrics are combined to form three derived metrics:
  - reuse percent.
  - reuse cost avoidance.
  - reuse value added.

9.5.3 Business Model of Poulin and Caruso

Reuse Percent

$$Reuse\ percent\ of\ a\ product\ =\ \frac{RSI}{RSI + SSI} \times 100\%$$

$$Reuse\ percent\ of\ a\ product\ release\ =\ \frac{RSI}{RSI + CSI} \times 100\%$$

9.5.3 Business Model of Poulin and Caruso

Reuse Cost Avoidance

- The purpose of this metric is to measure reduced total product costs as a result of reuse.
- One must retrieve and evaluate the reusable assets to choose the appropriate ones to be integrated into the system being developed.
- For example, the cost of integrating a reusable software element is 20% of the cost of developing the same element anew.
- The financial benefit due to adopting reuse in the development phase of a project is calculated as follows:

$$Development\ cost\ avoidance\ =\ RSI \times (1 - 0.2) \times (new\ code\ cost)$$

- In addition, saving in maintenance cost attributed to reuse is much more than those during software development, because of the fewer defects in reused components. The saving is:

$$Service\ cost\ avoidance\ =\ RSI \times (error\ rate) \times (cost\ per\ error)$$

The total reuse cost avoidance is calculated as the sum of cost avoidance in the development and maintenance activities, which is:

$$Reuse\ cost\ avoidance\ =\ Development\ cost\ avoidance\ +\ Service\ cost\ avoidance.$$

9.5.3 Business Model of Poulin and Caruso

Reuse Value Added

- The main idea behind RVA is to provide a metric to reward an organization that reuses software components and helps other organizations by developing reusable components.
- Reuse value added is derived from SSI, RSI, and SIRBO:

$$Reuse\ value\ added\ = \frac{(SSI + RSI) + SIRBO}{SSI}$$

- Organizations with no involvement in reuse have an RVA = 1.
- An RVA = 2 indicates the organization is twice as effective as it would be without reuse.

Summary

General Idea

Domain Engineering

Reuse Capability

Maturity Models

Economic Models of Software Reuse