Reuse and Domain Engineering

Idaho State University | Computer Science

Isaac Griffith

CS 4423 and CS 5523
Department of Computer Science
Idaho State University

ROAR

Idaho State University
Computer Science

Outcomes

After today's lecture you will be able to:

- Understand and be able to describe the general types and ideas surrounding reuse and reusability
- Understand and be able to describe the general ideas of domain engineering
- Understand and be able to use the basic measures of reuse capability

ROAR

Reuse and Domain Engineering

CS 4423/5523

ROAR

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.

ROAR

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:
  1. a set of generic design styles about the logical structure of software
  2. 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.

ROAR

Idaho State University
Computer Science

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

ROAR

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

ROAR

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**

ROAR

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.

ROAR

Reactive Approaches

- In this approach, while developing products, reusable assets are developed if a reuse opportunity arises.
- This approach works, if
  1. it is difficult to perform long-term predictions of requirements for product variations.
  2. 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.

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.

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**
  - **technical**

ROAR

Managerial

Systematic reuse requires upper management support, because:

1. it may need years of investment before it pays off.
2. 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.

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:
  1. reengineer the existing system components
  2. design and build new assets
  3. purchase assets from other sources.

Reuse Success Factors

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.

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.

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.

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
  - catalog the reusable assets into a component library
- Development, management, and maintenance of a repository of reusable assets make up the core of domain implementation.

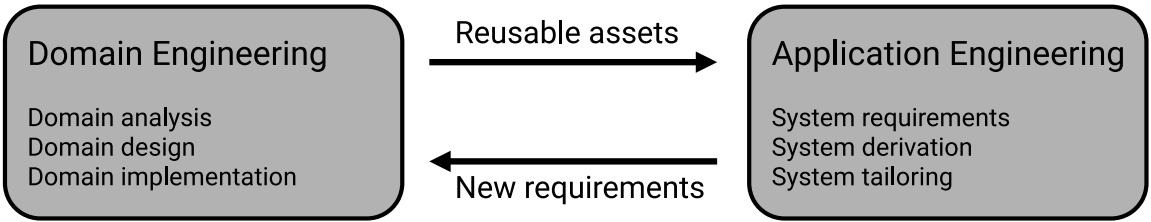
Application Engineering

- 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:
  1. reusing existing assets;
  2. developing any new components that are needed;
  3. reengineering some existent software; and
  4. 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.

Domain Engineering

Relationship Between Application & Domain Engineering

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



ROAR

Domain Engineering Approaches

- The following nine domain engineering approaches reported in literature:
  - Draco
  - Domain Analysis and Reuse Environment (DARE)
  - Family-oriented Abstraction, Specification, and Transportation (FAST)
  - Feature-Oriented Reuse Method(FORM)
  - "Komponentbasierte Anwendungsentwicklung" (KobrA)
  - Product line UML-based software engineering (PLUS)
  - Product Line Software Engineering (PuLSE)
  - Koala
  - Reuse-driven Software Engineering Business (RSEB)

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.

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.

ROAR

Reuse Capability

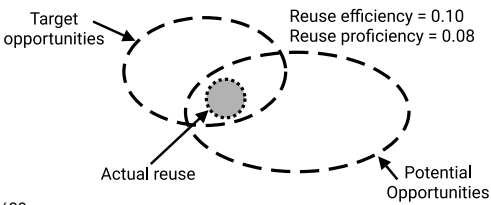
- Reuse **effectiveness** is represented as: $N(C_{NR}\text{-}C_R)/C_D$, where:
  - $N =$ number of products, systems, or versions developed with the reusable assets.
  - $C_{NR} =$ cost of developing new assets without using reusable assets.
  - $C_R =$ cost of utilizing, that is, identifying, assessing, and adapting reusable assets.
  - $C_D =$ cost of domain engineering, that is, developing assets for reuse and building a reuse infrastructure.

ROAR

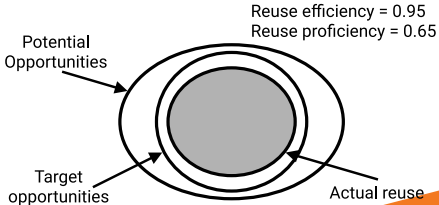
Reuse Capability

- The linkage between the two concepts: efficiency, proficiency and reuse capability are illustrated below
- 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.

Low Reuse Capability

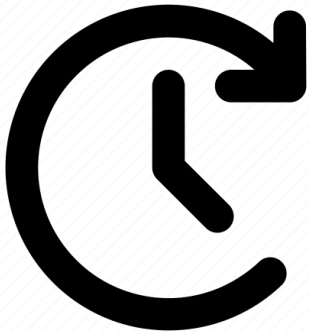


High Reuse Capability



For Next Time

- Review EVO Chapter 9.1 - 9.3
- Read EVO Chapter 9.4 - 9.6
- Watch Lecture 24



ROAR

Are there any questions?