Unit 4 - Software reuse and Reuse landscape

Mrs. Nisha Gautam
School of Computing
IIIT Una.

Software Reuse

- Software reuse means using existing assets in the development of a new system.
- Software reuse involves two main activities: software development with reuse and software development for reuse.
- Reusable assets can be both **reusable artifacts** and **software knowledge.**
- Four types of reusable artifacts as follows :
    - **Data reuse** : Standardization of data formats
    - **Architectural reuse**:
        - Set of generic **design styles** about the logical structure of software; and
        - A set of **functional elements** and reuse those elements in new systems.
    - **Design reuse:** Reuse of abstract design i.e. meet the application requirements.
    - **Program reuse:** Reusing executable code.

Software Reuse (Contd.)

- 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.
    - **Environmental independence:** Reused irrespective of the environment from which they were originally captured.
    - **High cohesion:** Subsystems cooperate with each other to achieve a single objective.
    - **Low coupling:** Less impact on other components makes it more usable.
    - **Adaptability:** To run in any new environment.
    - **Understandability:** Easily comprehended, so the programmers can quickly make decisions about its reuse.
    - **Reliability:** Consistently perform its intended function without degradation or failure.
    - **Portability:** Usable in different environment.

Benefits of Reuse

- Economic benefits
- Increased reliability: Reduces effort in design & implementation.
- Reduced process risks: Reuse fault free components.
- Increased Productivity : Reduced cost and time.
- Compliance with standards
- Accelerated development
- Improved maintainability
- Less maintenance effort and time: modified during reusability.

Reuse Models

- The organization can select one or more reuse models that best meet their business objectives, engineering realities, and management styles.
- Reuse Models are:
  - **Proactive approach**
    - System is designed for all receiving variations.
    - Process to create reusable software assets (RSA) called domain engineering.
    - This approach might be adopted by organizations that can accurately estimate the long-term requirements for their product line.
  - **Reactive approach**
    - Reusable assets are developed if a reuse opportunity arises.
    - This approach works, if
      - It is **difficult to perform long-term predictions** of requirements for product variations; or
      - An organization needs to **maintain an aggressive production schedule with not much resources to develop** reusable assets.
  - **Extractive approach**
    - Falls in between proactive and reactive approach and accumulated both artifacts and experiences in a domain.

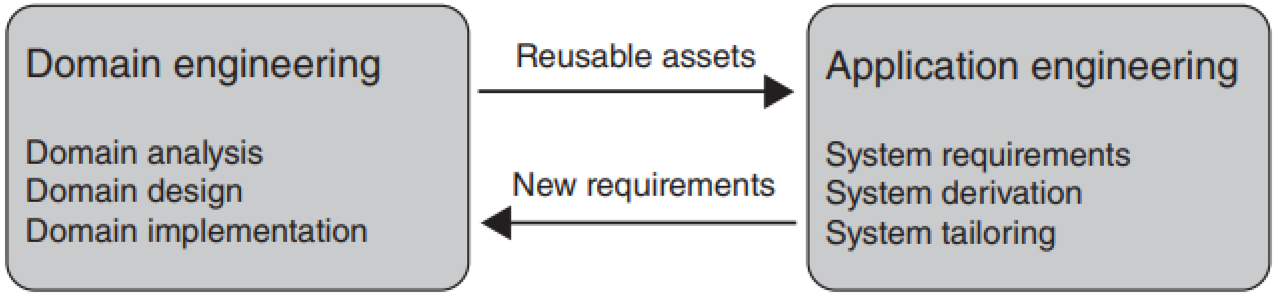
Factors influencing Reuse

- **Managerial:**
  - Less management support:
    - need years of investment before it pays off
    - involves changes in organization funding and management structure
- **Legal:**
  - Proprietary and copyright issues, liabilities and responsibilities of reusable software, and contractual requirements involving reuse.
- **Economic:**
  - Artifacts need to be reused more than 13 times to recoup the extra cost of developing reusable components.
- **Technical:**
  - **Received much attention from the researchers** actively engaged in library development, object-oriented development paradigm and domain engineering.
  - **Collect library assets** in a number of ways: reengineering, design and build new assets, and purchase assets from other sources.
  - **Certification** process through verification and testing.

Success Factors of Reuse

- 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 nontechnical 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



- Refers to a “development-for-reuse” process to create RSA.
- Domain engineering, is the entire process of reusing domain knowledge in the production of new software systems.
- Domain engineering consists of:
  - Domain Analysis:
    - Domain Analysis consists of:
      - 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.
    - Eg : Feature-oriented domain analysis (FODA) method : The method describes a process for domain analysis to discover, analyze, and document commonality and differences within a domain.

Domain Engineering

- Domain Design :
  - Develop a generic software architecture for the family of **products under consideration**;
  - develop a **plan to create individual systems** based on reusable assets.
  - Consider both **functional and non-functional requirements**.
  - **Limitation for this phase is context use** in terms of its applicability or its insufficiency in reuse.
  - **Objective of domain** design is to satisfy as many domain requirements as possible while retaining the flexibility offered by the developed feature model.
- Domain Implementation
  - **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**.

Domain Engineering Approaches

- Draco:
  - First prototype in domain engineering.
  - Based on **transformation technology.**
  - **Domain specific language, optimized transformations**
  - Very complex to apply in production environment.
- DARE:
  - DARE is both a Domain Analysis method and a tool suite supporting the method.
  - DARE Includes lexical analysis tools for extracting domain vocabulary: code, documents, and expert knowledge.
  - The generic architectures, feature tables, and facet tables, and all models and information are stored in a domain book [main work products of the DARE].
- FAST
  - FAST to develop telecommunication infrastructure.
  - Three sub-processes constitute FAST:
    - domain qualification (DQ-worthy for investment is identified),
    - domain engineering (DE-enables the development of product line environments and assets), and
    - application engineering (AE- for developing products rapidly).

Domain Engineering Approaches (Contd.)

- FORM
    - FORM finds commonalities and differences in a product line in terms of features, and uses those findings to develop architectures and components for product lines.
    - Two processes are key to FORM: asset development and product development.
- KobrA
    - Method for component-based application development
    - Two main activities in KobrA are: framework engineering and application engineering.
        - Framework engineering, one makes a common framework that manifests all variations in products making up the family.
        - Application engineering is applied on the framework to build specific applications.

Domain Engineering Approaches (Contd.)

- PLUS
  - In PLUS we do:
    - **For requirements analysis activities, use-case modeling and feature modeling**;
    - Mechanisms to model the static aspects, dynamic interactions, state machines, and class dependency for product lines; and
- PuLSE (product line software engineering)
  - Developed to **enable the conceptualization and deployment of software product lines for large enterprises.**
  - PuLSE methodology comprises three key elements:
    - **The deployment phases**: describe activities for initialization, construction of infrastructure, usage of infrastructure, and management and evolution
    - **The technical components**: how to operationalize the development.
    - **The support components:** guidelines enabling better evolution, adaptation, and deployment.

Domain Engineering Approaches (Contd.)

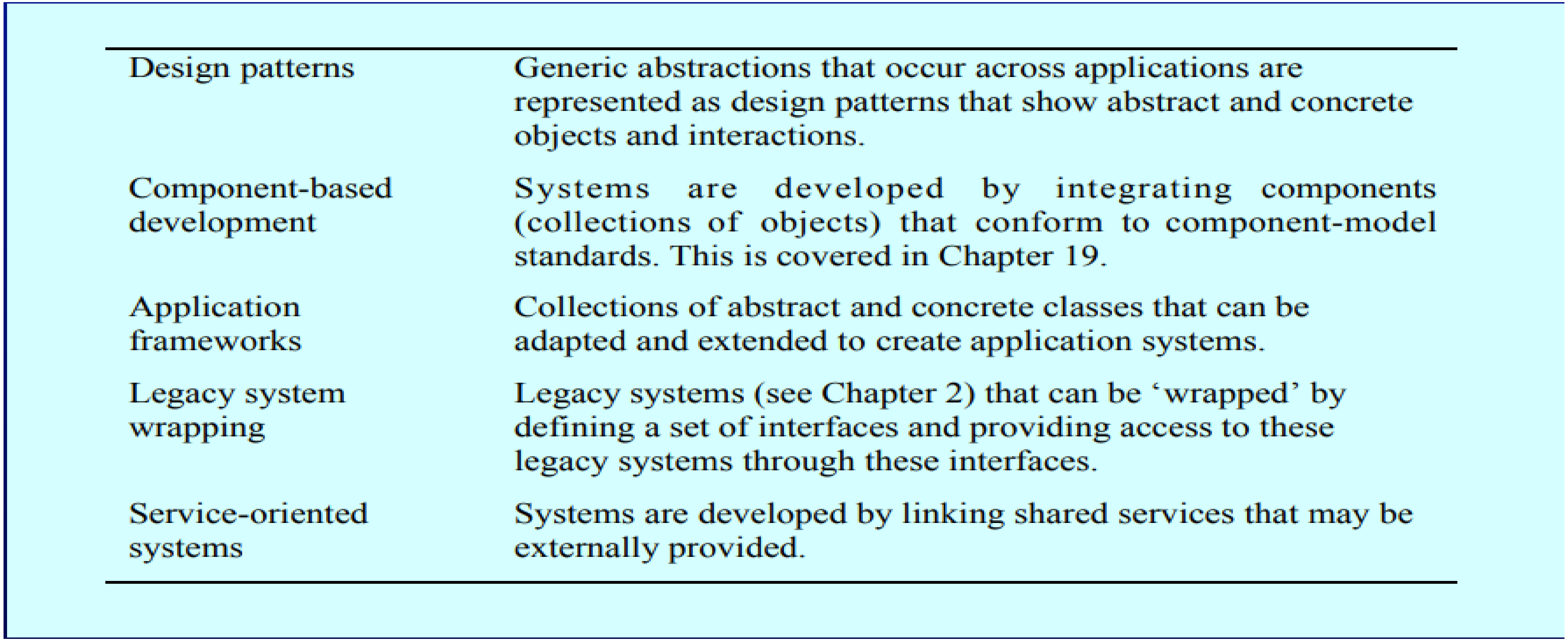
- RSEB
  - use-case-driven reuse method based on UML
  - designed to facilitate both asset reuse and the development of reusable software.
  - RSEB supports both domain engineering and application engineering.
- Coala
  - Koala is a language to **describe architectures for product lines**
  - developed at Philips Corporation
  - To **support product variations, diversity interfaces and switches are provided** in Koala
- Organization Domain modelling (ODM): reuse library frameworks or knowledge based reuse model.
- Capture tool : hypertext based tool using navigation.
- DSSA (Domain- specific software architecture) : central role of software architecture.

Reuse Capability

Reuse Landscape

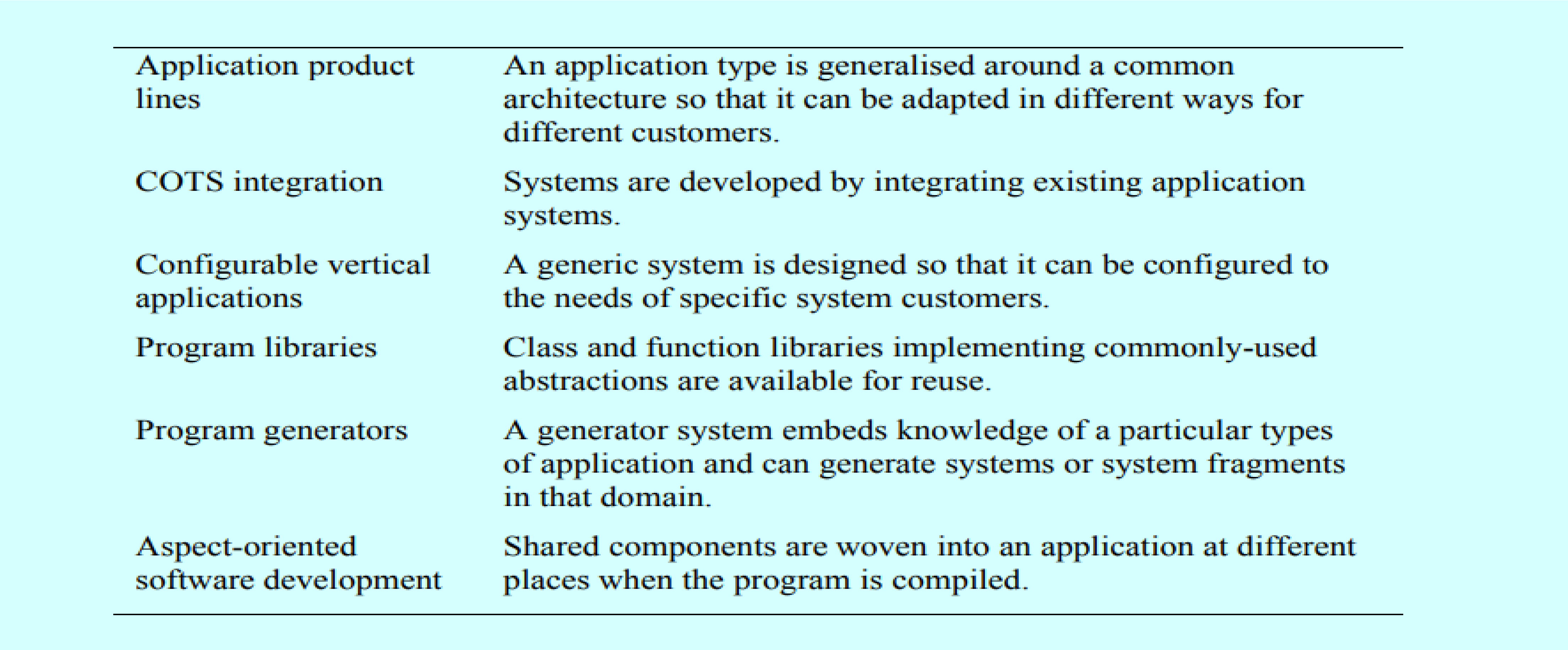
- Although reuse is often simply thought of as the **reuse of system components**, there are many different approaches to reuse that may be used.
- **Possible range of reuse** varies from simple functions to complete application systems
- The reuse landscape covers the range of possible reuse techniques.

Reuse approaches



Design patterns	Generic abstractions that occur across applications are represented as design patterns that show abstract and concrete objects and interactions.
Component-based development	Systems are developed by integrating components (collections of objects) that conform to component-model standards. This is covered in Chapter 19.
Application frameworks	Collections of abstract and concrete classes that can be adapted and extended to create application systems.
Legacy system wrapping	Legacy systems (see Chapter 2) that can be 'wrapped' by defining a set of interfaces and providing access to these legacy systems through these interfaces.
Service-oriented systems	Systems are developed by linking shared services that may be externally provided.

Reuse approaches



Application product lines	An application type is generalised around a common architecture so that it can be adapted in different ways for different customers.
COTS integration	Systems are developed by integrating existing application systems.
Configurable vertical applications	A generic system is designed so that it can be configured to the needs of specific system customers.
Program libraries	Class and function libraries implementing commonly-used abstractions are available for reuse.
Program generators	A generator system embeds knowledge of a particular types of application and can generate systems or system fragments in that domain.
Aspect-oriented software development	Shared components are woven into an application at different places when the program is compiled.

Concept Reuse

- **Limit the reuse opportunities,** when you reuse program or design components, you have to follow the design decisions made by the original developer of the component.
- So, **abstract form of reuse** is concept reuse when a particular approach is described in an implementation independent way and an implementation is then developed.
- The two main approaches to concept reuse are:
  - Design patterns;
  - Generative programming

Design patterns

- A design pattern is a way of **reusing abstract knowledge** about a problem and its solution.
- A pattern is a **description of the problem** and the essence of its solution.
- Patterns **rely on object characteristics** such as inheritance and polymorphism.

Pattern elements

- **Name**
  - A meaningful pattern identifier.
- **Problem description**
- **Solution description**
  - Not a concrete design but a template for a design solution that can be instantiated in different ways.
- **Consequences**
  - The results and trade-offs of applying the pattern

The observer pattern

- **Name**
  - Observer.
- **Description**
  - Separates the display of object state from the object itself.
- **Problem description**
  - Used when multiple displays of state are needed.
- **Solution description**
  - See slide with UML description.
- **Consequences**
  - Optimizations to enhance display performance are impractical

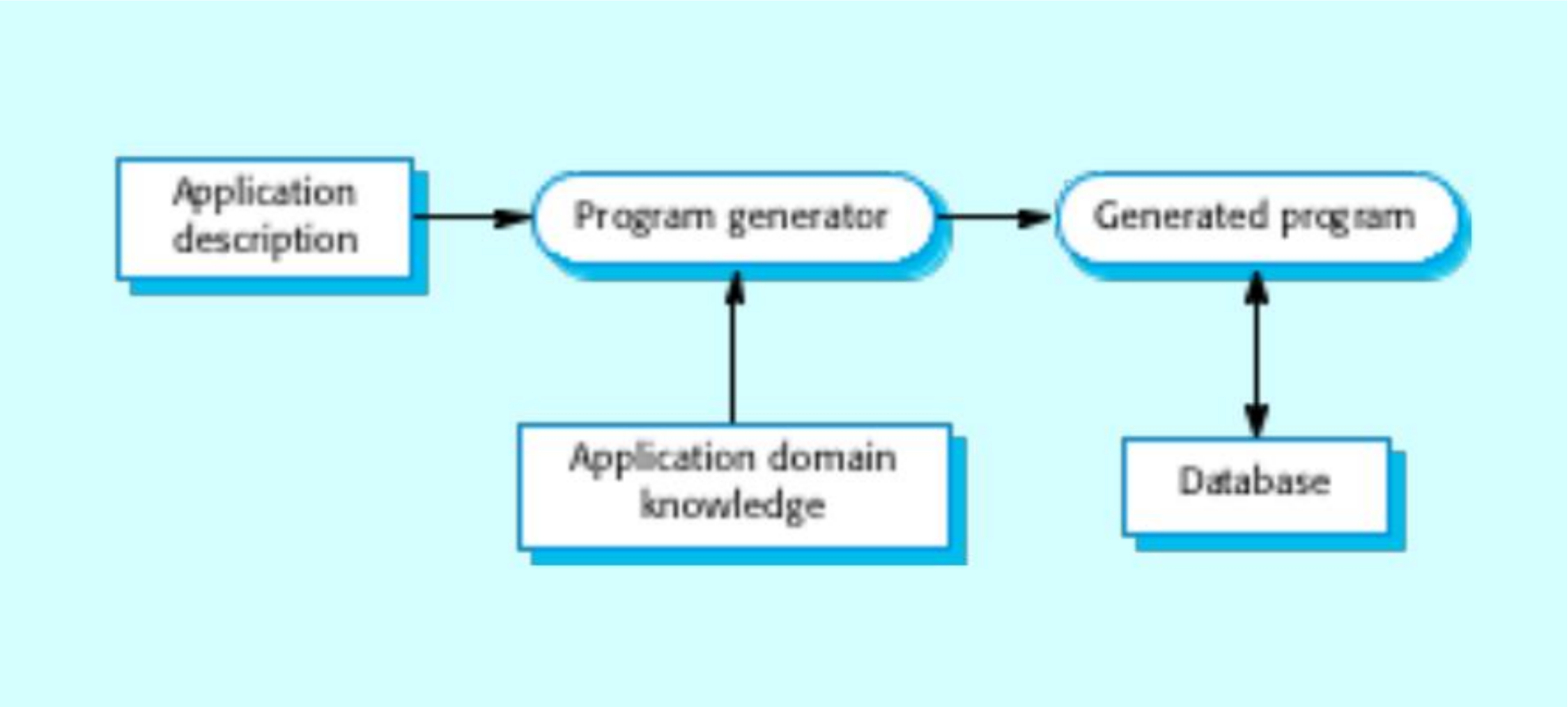
Generator-based Reuse

- Program generators involve the **reuse of standard patterns and algorithms.**
- These are embedded in the generator and parameterized by user commands. A program is then automatically generated.
- Generator-based reuse is possible **when domain abstractions and their mapping to executable code can be identified.**
- A domain specific language is used to compose and control these abstractions.

Types of program generator

- Types of program generator
  - **Application generators** for business data processing;
  - **Parser and lexical analyzer generators** for language processing;
- **Generator-based reuse is very cost-effective** but its applicability is limited to a relatively small number of application domains.
- It is easier for end-users to develop programs using generators compared to other component-based approaches to reuse.

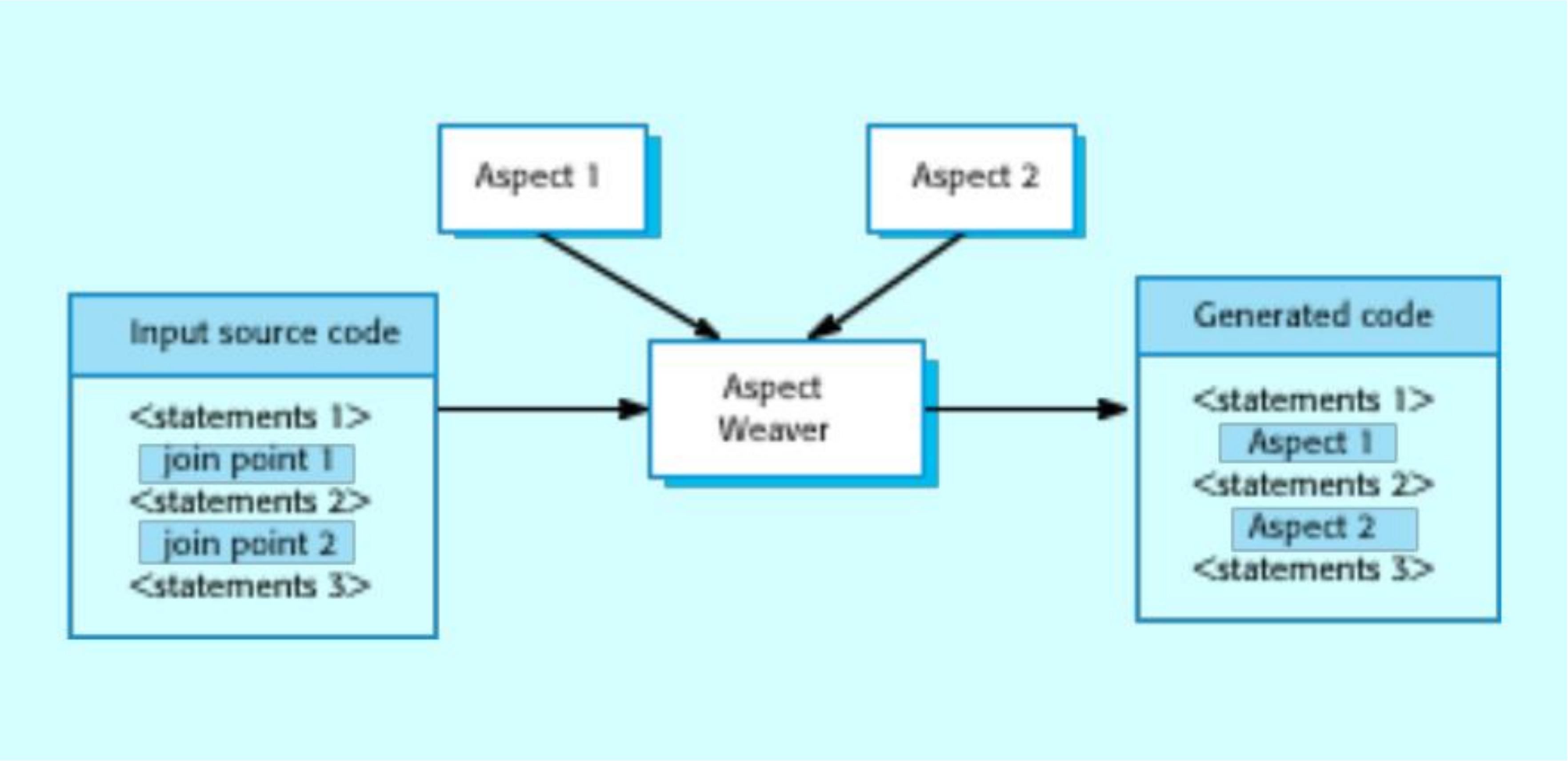
Reuse through program generation



Aspect-oriented development

- Aspect-oriented development addresses a major software engineering problem - **the separation of concerns.**
- Concerns are often not simply associated with application functionality but are cross-cutting - e.g. all components may monitor their own operation, all components may have to maintain security, etc.
- **Cross-cutting concerns are implemented as aspects** and are dynamically woven into a program. The concern code is reuse and the new system is generated by the aspect weaver.

Aspect-oriented development



THANK YOU..