CHAPTER 30 COMPONENT BASED SOFTWARE ENGINEERING

Overview

The chapter describes component-based software engineering (CBSE) as a process that emphasizes the design and construction of computer-based systems using reusable software components. This has the potential advantage of delivering highly reliable software products in a very short time. CBSE encourages the use of predictable architectural patterns and standard software infrastructures that improve overall product quality. CBSE encompasses two parallel engineering activities, domain engineering and component-based development (CBD). Domain engineering explores the application domain with the specific intent of finding functional, behavioral, and data components that are candidates for reuse and places them in reuse libraries. CBD elicits requirements from the customer and selects an appropriate architectural style to meet the objectives of the system to be built. The next steps are to select potential components for reuse, qualify the components to be sure they fit the system architecture properly, adapt the components if they must be modified to integrate them, then integrate the components into subsystems within the application. Custom components are engineered only when existing components cannot be reused. Formal technical reviews and testing are applied to ensure the quality of the analysis model and the design model. The resulting code is tested to uncover errors in the newly developed software.

Engineering of Component-Based Systems

- Software team elicits requirements for system to be built
- Architectural design is established
  - Team determines which of the requirements are amenable to composition rather than construction
  - Are commercial off-the-shelf (COTS) components available to implement the requirement?
  - Are internally developed reusable components available to implement the requirement?
  - Are the interfaces for available components compatible within in the proposed system architecture?
- Team attempts to remove or modify requirements that cannot be implemented with COTS or in-house components?
- For those requirements that can be addressed with available components the following activities take place:
  - component qualification - candidate components are identified based on services provided and means by which consumers access them
  - component adaptation - candidate components are modified to meet the needs of the architecture or discarded
  - component composition - architecture dictates the composition of the end product from the nature of the connections and coordination mechanisms
  - component update - updating systems that include COTS is made more complicated by the fact that a COTS developer must be involved
• Detailed design activities commence for remainder of the system

Domain Engineering

• Domain analysis
  1. define application domain to be investigated
  2. categorize items extracted from domain
  3. collect representative applications from the domain
  4. analyze each application from sample
  5. develop an analysis model for objects
• Domain model
• Software architecture development
• Structural model
  ▪ consists of small number of structural elements manifesting clear patterns of interaction
  ▪ architectural style that can be reused across applications in the domain
  ▪ structure points are distinct constructs within the structural model (e.g. interface, control mechanism, response mechanism)
• Reusable component development (new software components must be developed and integrated with existing software components)
• Repository of reusable artifacts or components

Structure Point Characteristics

• Abstractions with limited number of instances within an application and recurs in applications in the domain
• The rules governing the use of a structure point should be easily understood and structure point interface should be simple
• The structure point should implement information hiding by isolating all complexity contained within the structure point itself

Component-Based Development

• Analysis
• Architectural design
  ▪ component qualification
  ▪ component adaptation
  ▪ component decomposition
• Component engineering
• Testing
• Iterative component update

Component Qualification Factors

• Application programming interface (API)
• Development and integration tools required by the component
• Run-time requirements (resource usage or performance)
• Service requirements (component dependencies)
• Security features
• Embedded design assumptions
• Exception handling

Component Adaptation Techniques

• White-box wrapping - integration conflicts removed by making code-level modifications to the code
• Grey-box wrapping - used when component library provides a component extension language or API that allows conflicts to be removed or masked
• Black-box wrapping - requires the introduction of pre- and post-processing at the component interface to remove or mask conflicts

Component Composition Architectural Ingredients

• Data exchange model - similar to drag and drop type mechanisms should be defined for all reusable components, allow human-to-software and component-to-component transfer
• Automation - tools, macros, scripts should be implemented to facilitate interaction between reusable components
• Structured storage - heterogeneous data should be organized and contained in a single data structure rather several separate files
• Underlying object model - ensures that components developed in different languages are interoperable across computing platforms

Representative Component Standards

• Object Management Group (OMG)/CORBA (common object request broker architecture)
• Microsoft COM (component object model)
• Sun JavaBeans Components

Basis for Analysis and Design for Reuse

• Standard data - standard global data structures identified in application domain and components developed to use these data structures
• Standard interface protocols - three interfaces levels should be established (interamodular, external technical, and human/machine)
• Program templates - used in architectural design of new programs

Classifying and Retrieving Components

• Describing reusable components
  • concept - what the component does
• content - how the concept is realized
• context - specifies conceptual, operational, and implementation features of the software component within its domain of application

• Library indexing methods
  ▪ uncontrolled indexing vocabularies - syntax free, no restrictions
  ▪ enumerated classification - hierarchical listing of the domain objects grouped by class relations
  ▪ faceted classification - based on 1 to 8 basic descriptive features from the application domain
  ▪ attribute - value classification similar to faceted classification using unlimited number of fixed terms

• Reuse environment elements
  ▪ component database capable of storing software components and classification information to allow their retrieval
  ▪ library management system to allow access to database
  ▪ software component retrieval system that enables client software to retrieve components and services from library server
  ▪ CBSE tools that support integration of reused components into a new design or implementation

Economics of CBSE

• Quality -- with each reuse additional component defects are identified and removed which improves quality.
• Productivity -- since less time is spent on creating plans, models, documents, code, and data the same level of functionality can be delivered with less effort so productivity improves.
• Cost -- savings projected by estimating the cost of building the system from scratch and subtracting the costs associated with reuse and the actual cost of the software as delivered.
• Cost analysis using structure points -- can be computed based on historical data regarding the costs of maintaining, qualification, adaptation, and integrating each structure point.
• Reuse metrics can be computed for CBSE