Chapter XIII

A Reuse Definition, Assessment, and Analysis Framework for UML

Donald Needham, United States Naval Academy, USA
Rodrigo Caballero, United Technologies Research Center, USA
Steven Demurjian, The University of Connecticut, USA
Felix Eickhoff, The University of Connecticut, USA
Yi Zhang, The University of Connecticut, USA

Abstract

This chapter examines a formal framework for reusability assessment of development-time components and classes via metrics, refactoring guidelines, and algorithms. It argues that software engineers seeking to improve design reusability stand to benefit from tools that precisely measure the potential and actual reuse of software artifacts to achieve domain-specific reuse for an organization’s current and future products. The authors consider the reuse definition, assessment, and analysis of a UML design prior to the existence of source code, and include dependency tracking for use case and class diagrams in support of reusability analysis and refactoring for UML. The integration of these extensions into the UML tool Together Control Center to support reusability measurement from design to development is also considered.
Introduction

Software evolution addresses the changes a software system undergoes so as to remain, or become, a viable solution to a problem. Developing reusable software components is a critical aspect of software evolution since a reusable component offers the promise of being beneficial in situations unforeseen at the time of the component’s development. Although reusable software components have been in use for over 30 years (McIlroy, 1968), their benefits, including reduced risk, limited development and maintenance costs, reduced time-to-market, increased quality and reliability, improved interoperability, and the support of rapid prototyping (Software reuse executive premier, 1996; Hall, 1999; Rine & Nada, 2000; Schnietendorf, 2000; Tsagias & Kitchenham, 2000), have proven difficult to demonstrate in practice. Today’s component-based programming languages and their APIs support actual reuse of standard components (e.g., GUIs, communications, databases), but are less successful in attaining the reuse potential of components (Succi, 1995), a target of domain-and-organization-specific reuse (Meekel, 1997; Poulin, 1996) which represents a long-term investment in reuse (Sarshar, 1996). To support reuse potential, we provide an integrated reusability metric, framework, and tool (Price & Demurjian, 1997; Price, Demurjian, & Needham, 1997; Price, Needham, & Demurjian, 2001) with formal underpinnings (Caballero & Demurjian, 2002) for reusability definition, assessment, and analysis of software. We provide the ability to mark components/classes to indicate their reuse level, which can range from general (reuse in multiple contexts) to specific (single use). For example, in a retailing application, a supplier would have a general Item (reusable in many contexts), while an auto-parts supplier would have a less general AutoItem (reusable in that context), and individual retailers would have a specific WalmartItem (not reusable). Given a marking of an application’s classes/components, our reuse metrics can objectively classify and measure dependencies (couplings) within and among classes/components, thereby identifying couplings that promote or hinder future reuse. This process can be automated with an algorithm for reusability assessment and refactoring (Caballero & Demurjian, 2002), and is supported by a Design Reusability Evaluation (DRE) tool (UConn, 2003).

To fully support reuse potential, a reusability definition, assessment, and analysis must be provided for the Unified Modeling Language (UML) prior to the development of software (code), to concentrate on reusing a design model and to monitor reuse as the model evolves when using UML via a tool (e.g., Rational Rose or Together Control Center). Specifically, this chapter details the integration of reuse and refactoring concepts for reusable UML components, allowing these components to be easily incorporated into a product family while tracking reuse assessments during design and following through to the development and maintenance stages. Our approach to reusability assessment and analysis in UML focuses on use cases and class diagrams. In practice, software engineers tend to start the process by defining use cases, followed by classes and the various activity views (sequence and collaboration diagrams), iterating across the various diagrams over time as the design evolves. Our approach supports the marking of use cases and classes with varying levels of generality, and then tracks and enforces this marking as a software engineer modifies the design. For example, as a software engineer starts with UML and defines use cases, the use cases are marked...
Related Content

Modeling of Web Services using Reaction Rules
www.igi-global.com/chapter/modeling-web-services-using-reaction/35869?camid=4v1a

XML Stream Processing: Stack-Based Algorithms
www.igi-global.com/chapter/xml-stream-processing/41505?camid=4v1a

UML- and XML-Based Change Process and Data Model Definition for Product Evolution
Ping Jiang, Quentin Mair, Julian Newman and Josie Huang (2005). *Software Evolution with UML and XML* (pp. 190-221).
www.igi-global.com/chapter/uml-xml-based-change-process/29614?camid=4v1a

Integration of Relational and Native Approaches to XML Query Processing
www.igi-global.com/chapter/integration-relational-native-approaches-xml/41513?camid=4v1a