

## Keywords

Before providing a theme for software reuse, let's discuss the following definitions: **Design for Reuse** The method, technique or series of steps behind reusing building reusable blocks of information or software where building reusable blocks are components, patterns, procedures, part of a document, test case, a diagram, etc. **Design Reuse** The ability to retarget previously designed building non-reusable blocks or "cores" of information or software for a new design as a means of increasing time to market. **Reuse**

The engineering activities that focus on the recognition of commonalities of systems within and across domains; it consists of the creation of models with different abstractions (ranging from domain models to code) and their use during the engineering of one or more applications. **Code reuse or software reuse** is the idea that a partial or complete software application written at one time can be, should be, or is being used in another program written at a later time. The re-use of application code is a common technique which attempts to save time and energy by reducing redundant work. A very common example of code reuse is the technique of using a software or class library. Many common operations, such as converting information among different well-known formats, accessing external storage, interfacing with external programs, or manipulating information (numbers, words, names, locations, dates, etc.) in common ways, are needed by many different programs. Authors of new programs can use the code in the software library to accomplish these tasks, instead of "re-inventing the wheel" (by actually writing new code directly in the program to perform the operation). Library implementations often have the benefit of being well-tested and covering unusual or arcane cases. Disadvantages include the inability to tweak details which may affect performance or the desired output, and the time and cost of acquiring, learning, and configuring the library. The software library is a good example of abstraction. Programmers may decide to create internal abstractions so that certain parts of their program can be re-used, or may create custom libraries for their own use. **Software Reuse Classes / Components** **Software Stability Model** **Stable Analysis Patterns** **Goals** **Enduring Business Themes** **Stable Design Patterns** **Business Objects** **Capabilities to Achieve the Goals** **Knowledge Map/Topology** **Ontologies** **Stable Architectural Patterns** **Aspects** **Application Frameworks** **Enterprise Frameworks** **Components** **Classes** **Objects** **Procedures** **Process Models** **Systems of Patterns** **Knowledge Patterns** **Pattern Languages** **Pattern Documentation Template** **Scenario Templates** **Use Case Templates** **CRC Cards** **Layout** **Hooks** **Extension Points** **Software Architectures** **Line-Product Architectures** **Model-Driven Architectures** **Adaptive Architectures** **On-Demand Architectures** **Etc.** **Software Reuse: Criteria and Processes** **Scalability** **Adaptability** **Extensibility** **Hooks or ExtensionPoints** **Hooking** **Maintainability** **Customizability** **Integration** **Modifications or Changing** **Augmenting** **Replacing** **Accessibility** **Tracking** **Reusability** **AnyDomain** **Independence** **Measurability** **Composition** **Stable Patterns Classification** **Etc.** **Traditional Software Reuse** **Analysis Patterns** **Design Patterns** **Architectural Patterns** **Software Patterns Classifications** **Gang of Four Patterns** **Seimens Group Patterns** **Process Patterns** **Organizational Patterns** **Anti-Patterns** **Business Patterns** **Enterprise Software Patterns** **Others** **Reuse keywords: Ontology** **Topology** **Quality Factors** **Etc.** **Safety and Reuse: Where is safety and reuse currently addressed in international standards? What processes could ensure certifiable components? How can techniques such as "wrappers" and "safety layers" be used to improve safety in component based systems? How can safety-related aspects of components be specified? Introduction and analysis of further case studies in which software reuse affected safety in critical systems. What are the legal aspects of reuse and safety, in particular concerning the issue of reliability versus safety? Modeling and Reuse** **Models defining the assets to be reused** **Models describing requirements, models of the architecture of composed systems.** **Determining and modeling features on the requirements level.** **Composition and interaction problems in research and industry.** **Artificial intelligence supporting the developer with composition and interaction problems.** **Theoretical foundation of modeling and asset inconsistency detection and specification.** **Methods to assure certain qualities of service after composition.** **Modelling of assets and their desired relationships.** **Detection of undesired negative asset interaction.** **Conformance between model and implementation / reality and corresponding consequences for reasoning about the implementation / reality** **Other Topics in Reuse** **Software variability management** **Aspect-oriented software reuse** **Automated software engineering** **Software generators and domain-specific languages** **Software product lines, software product families, and domain engineering** **Component-based software engineering using Java Beans, DCOM, and others** **Evolution of component-based software systems** **Lightweight approaches to software reuse** **Managing the transition towards a reuse organization** **Legal, managerial, and economic issues of software development with reuse** **Benefit and risk analysis of reuse investments** **Reuse in the e-commerce context: how to address fast-evolving markets** **Generation of non-code artefacts** **Testing of components and generators** **Quality aspects of reuse, e.g. security and reliability** **Success and failure stories of reuse approaches from industrial context** **Different Aspects of Reuse** **Aspect-Oriented Development and its relation with Software Reuse**

- Architecture Description Languages (ADL)
- Cost Models for Software Reuse
- Domain-Specific Software Architectures (DSSA)
- Domain-Specific Languages (DSL)
- Educational Aspects for Software Reuse
- Empirical Studies in Software Reuse
- Generative Programming
- Incremental Models for Reuse Introduction in Software Factories
- Methods, Processes, Tools and Experiences in Component-Based Development (CBD)
- Methods, Processes, Tools and Experiences in Domain Engineering

- Methods, Processes, Tools and Experiences in Model-Driven Development
- Methods, Processes, Tools and Experiences in Software Product Lines
- Methods, Processes, Tools and Experiences in Reusable Web Services
- Non-Technical Aspects in Software Reuse
- Quality Attributes for Reusable Architectures
- Reference Architectures
- Software Reuse Environments
- Software Reuse Process Adaptation
- Software Reuse Metrics
- Software Reuse for Embedded Systems
- Software Reengineering
- Variability in Software Reuse