UML Software Architecture and Design Description

UML is the de facto modeling language for software development. Several features account for its popularity: it's a standardized notation, rich in expressivity.Furthermore, UML supports domain-specific extensions using stereotypes and tagged values.UML models are the first artifacts to systematically represent a software architecture. They're subsequently modified and refined in the development process.

Adherence to UML specification

The UML specification defines the language notation, syntax, and semantics. Based on these self-assessments, adherence to the specification is rather loose. This might be a result of UML's lack of formal semantics and large degree of freedom in its application. On the other hand, it might be that informal use is "good enough" for many practitioners' purposes.Strategic Planning

Stopping criteria for modeling

No objective criteria exist to verify that a model is complete or that it satisfies some tangible notion of quality. When schedule or a deadline becomes the criterion for switching from modeling to implementation, it's usually a poor substitute for a systematic review and a negative indicator for product quality. In correlating respondent demographics with the question of stopping criteria, the criteria varied with different project sizes. Most striking was the comparison between deadline and completeness. This is probably because increased complexity in both the product and the project likewise increases the need for a systematic approach.

Problems with UML descriptions

■ Scattered information. Design choices are scattered over multiple views.
■ Incompleteness. Many projects knowingly complete only a subset of the architectural views.
■ Disproportion. Architects might work out more details for system parts that they consider to be more complex.
■ Inconsistency. Industrial systems are typically developed by teams. Different teams can have different understandings of the system as well as different modeling styles, and this can lead to inconsistent models.

Other problem classes include the following:
■ Diagram quality. UML lets architects represent one design in different ways. Although different ways of organizing diagrams don't change the actual design, they can influence how easy the model is to understand and how it gets interpreted.
■ Informal use. Architects sometimes use UML in a very sketchy manner. These diagrams deviate from official UML syntax, making their meaning ambiguous.
■ Lack of modeling conventions. Our case studies show that engineers use UML according to individual habits. These habits might include layout conventions, commenting, visibility of methods and operations, and consistency between diagrams.

Different uses of UML arise naturally as the design decisions and detail increase in both quantity and complexity throughout the design process.The generality and freedom that enable UML to cater to this wide range of purposes are also the source of its weakness. UML has no formal semantics. This poses a problem when different people use a UML model

Opportunities for improving UML usage

Defect checking and feedback

Current tools provide only limited support for defect checking.The tools should be extended to automatically detect defects. UML profiles have been used to define specific architectural styles and patterns. The pattern defines which building blocks the model developer can use and what types of relations can exist between the blocks. During development, tools can also verify that architecture, design, and implementation conform to this pattern.

UML metrics

Software metrics are a well-established technique for managing source code quality during implementation. Metrics can also help manage the quality of architecture and design. The desired use of metrics is two to four times higher than the actual use. The multiple views of UML models include information not available from source code. Projects could use these early metrics to indicate adherence to or violation of design guidelines and heuristics. The following describe some architecture metrics:

■ **Class dynamicity**. This metric indicates a class's complexity. If a class has many different incoming and outgoing messages and appears in several different sequence diagrams, then the class plays a critical role in the system and deserves more attention during reviews and testing.
■ **Number of classes per use case**. This metric indicates whether related functionality is spread over many parts of a design. A higher value indicates low maintainability and reusability.
■ **Number of use cases per class**. A class that is implemented in many use cases might have low cohesion and so serve a large amount of unrelated functionality. It might also be central to the system

UML practices should improve with increased capabilities in development tools for it. Several areas need improvement:

■ Detection of design flaws, omissions, and inconsistencies.
■ More uniformity in modeling. Modeling standards and development tools for checking them can achieve this purpose.
■ Domain- or project-specific reference architectures and patterns. UML 2.0 provides several facilities for defining such profiles but little support for checking them.
■ More consistency between UML models and system requirements as well as implementations.
■ Defined quality goals for UML models. A quality goal is an incentive to improve a model and identifies spots that must be improved to reach the defined goal.