The Role of Software Architecture in Requirements Engineering

David Garlan
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 USA

Problem Space versus Solution Space

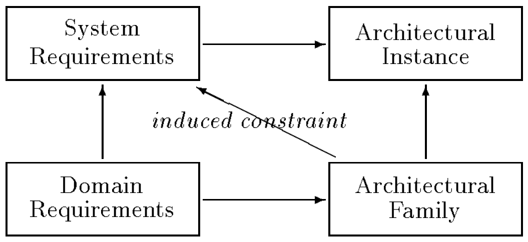
Requirements engineering is concerned fundamentally with the shape of the problem space. Its primary goal is to determine the dimensions of the problem that the software system is to solve.

In contrast, software architecture is concerned with the shape of the solution space. Its primary goal is to determine the structure of a solution to a problem posed by a set of requirements. Thus, understanding the role of software architecture in requirements engineering is largely a matter of understanding the dynamic relationship between identification of a problem to be solved and a description of its solution.

Architectural Instance versus Architectural Family

An architectural instance describes the high-level structure of a *particular* system. While there is as yet no established definition of what constitutes an architectural description, most would agree that a system's architecture describes at least how the system is decomposed into major components and how those components interact. A "good architecture" is one that exposes the important properties – those that are needed to reason about satisfaction of the system's requirements.

In contrast, an architectural family defines architectural constraints on a collection of related systems. Architectural families can range from generic, idiomatic patterns and styles (such as "pipe and filter system", "client-server organization', "table driven interpreter") to reference architectures (such as the OSI layered communication standard, user interface tool kits, or the organization of a compiler). A "good architectural style" is one that guarantees certain integrity constraints while allowing a range of variability that subsumes the class of systems to be built over the lifetime of the product family.



Significance

Traditionally software development has been concerned with the relationship between requirements and architectural instances. In that context, problems drive solutions: requirements for a specific system are used by the software architect to develop an architectural instance that satisfies the requirements. This tends to promote the creation of an innovative design and implementation for each new system.

More recently software developers have begun to focus more on the relationship between requirements and architectural families. Here, oddly, solutions drive problems: the architecture of a family of systems determines the range of variability allowable in a product line. Thus any new system must conform to the shape of the solution if it is to be built cost-effectively. This encourages a reuse-oriented view of software development, since new systems are built by reusing an established architectural framework.

If the idea of solutions driving problems seems strange, it is. The resolution is to realize that there is another important role beyond requirements engineering and software architecting: namely that played by the domain specialist. This person playing this role is responsible for determining the likely range of product variability as input to the software (meta) architect who then develops or selects a common architecture for that family of products.