The process of software architecting

Peter Eeles April 15, 2006

from The Rational Edge: Software architecting is a recognized, emerging discipline in the field of software development. As the third in a series on software architecture, this article describes the various ongoing activities of the software architect during the software project lifecycle.

My first article in this series described what a software architecture is, and the second article defined the characteristics of the role of the software architect. This third installment builds on the previous discussion and considers the themes, or characteristics, that underly the process of software architecting.

Software architecting: Definition and scope

According to the IEEE, software architecting represents

> *the activities of defining, documenting, maintaining, improving, and certifying proper implementation of an architecture.*[1]

The scope of architecting is fairly broad. Figure 1 shows a metamodel that defines various aspects of the process of software architecting. This metamodel is derived from that given in the IEEE 1471 standard and can be considered to be a roadmap through the various aspects of architecting that an architect is concerned with.

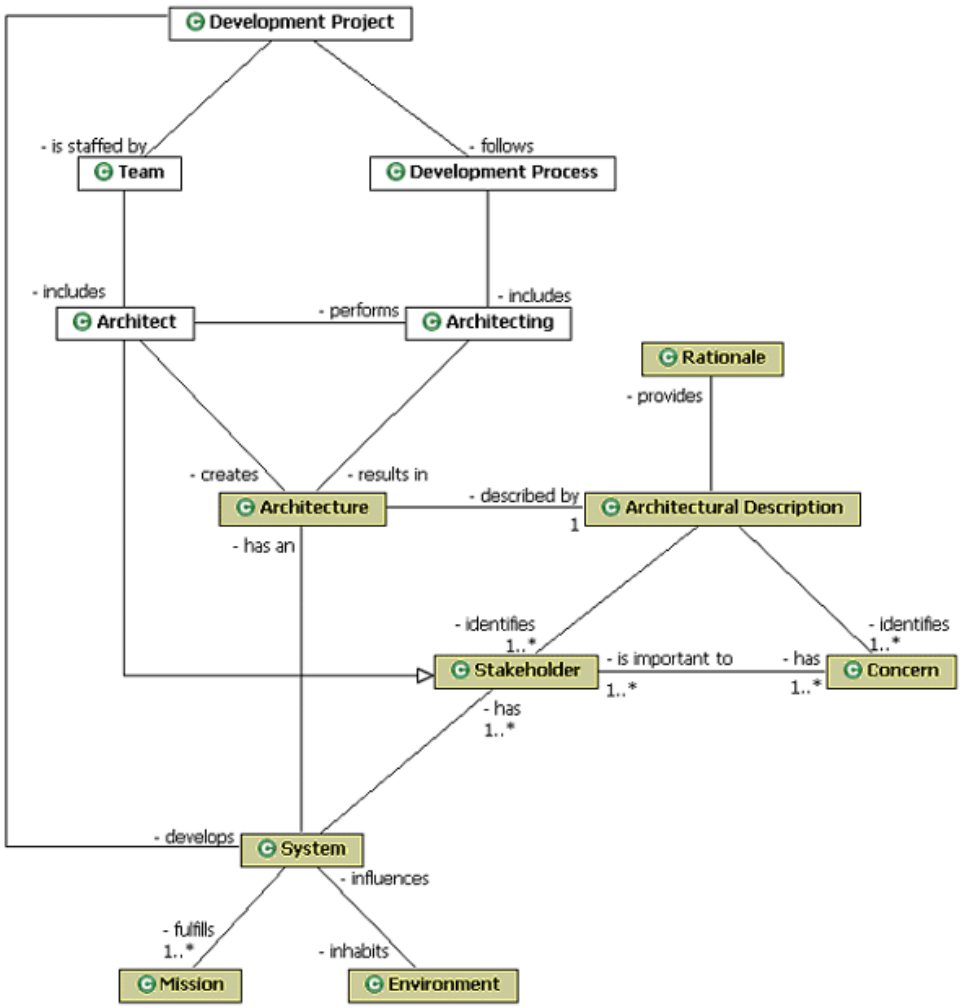

Figure 1: A metamodel of architecting related terms

The elements in shaded boxes in Figure 1 are taken directly from the IEEE 1471 standard, and their various relationships illustrate a number of characteristics regarding a system and its architecture:

- A system has an architecture.
- A system fulfills a mission.
- A system inhabits an environment and is influenced by that environment.
- A system has one or more stakeholders.
- An architecture is described by an architectural description.
- An architectural description identifies one or more stakeholders.
- An architectural description identifies one or more concerns.
- An architectural description provides rationale.
- A stakeholder has one or more concerns, and a concern is important to one or more stakeholders.

Additional elements and relationships in Figure 1 that are not part of the IEEE 1471 standard, shown in the unshaded boxes, can be described as follows:

- A development project is staffed by a team.
- A development project follows a development process.
- A development project delivers a system.
- The development process includes architecting.
- The team includes an architect.
- The architect performs architecting.
- The architect is a kind of stakeholder.
- Architecting results in an architecture.
- The architect creates the architecture.

Architecting is a science

Architecting is a recognized discipline, albeit one that is still emerging. However, with this recognition comes a focus on techniques, processes, and assets that are improving the maturity of the process of architecting. One way of advancing this maturity is to draw upon an existing body of knowledge. In general terms, architects look for proven solutions when developing an architecture, rather than reinventing the wheel. Codified experience in terms of reference architectures, architectural and design patterns, and other reusable elements all have a part to play.

However, there is still some way to go before the process of software architecting is anywhere near as mature as the processes found in civil engineering. This maturity can be considered in many dimensions, including the use of standards, and an understanding of best practices, techniques, and processes. As a result, at this point in time, the experience of an architect has a large bearing on the success of a project.

Architecting is an art

Although architecting can be seen as a science, there are times when it is necessary to provide some level of creativity. This is particularly true when dealing with novel and unprecedented systems. In such cases, there may be no codified experience to draw upon. Just as a painter looks for inspiration when faced with a blank canvas, architects may also, on occasion, see their work more like an art than a science. For the most part, however, the artistic side of architecting is minimal. Even in the most novel of systems, it is normally possible to copy solutions from elsewhere and then adapt them to the system under consideration.

As the process of software architecting becomes more mainstream, it is likely that it will no longer be seen as some mysterious set of practices that only the "chosen few" are able to comprehend, but rather as a broadly accessible set of well-defined and proven practices that have some scientific basis.

Architecting spans many disciplines

The architect is involved in many disciplines within the software development process. The discipline that the architect is most associated with is design. However, the architect is involved in many other disciplines, too. For example, in the requirements discipline, the architect ensures

that those requirements of interest to the architect, in particular, are captured. They are also involved in prioritizing requirements. The architect participates in the implementation discipline, where they define the implementation structures that will be used to organize the source code and also executable work products. The architect participates in the test discipline, ensuring that the architecture is both testable and tested. The architect is responsible for certain elements of the development environment, specifically setting up certain guidelines. The architect also assists in defining the configuration management strategy, since the configuration management structures (which support version control) often reflect the architecture that has been defined. The architect and project manager work closely together, and, as mentioned in Part 2 of this article series, the architect has input to the project planning activities.

While on the subject of the scope of architecting, I should mention the relationship between architecting and design. Although the architect is heavily involved in the design discipline, not all design can be considered architecting. This is because an architecture is only concerned with significant elements, and not all design elements can be considered to be architecturally significant. For example, the detailed design of a screen in the user interface is not normally considered to be architecturally significant and is best left to a user interface designer. The same thinking can be applied to other system elements, such as elements supporting business logic or data logic. The architect establishes constraints only where necessary, and many design decisions are left for the designers to make.

Architecting is an ongoing activity

Experience shows that architecting is not something performed once, early in a project. Rather, architecting is applied over the life of the project where the architecture is "grown" through the delivery of a series of incremental and iterative deliveries of executable software. At each delivery, the architecture becomes more complete and stable. This raises the obvious question of what the focus of the architect is through the life of the project.

Successful software architecting efforts are results-driven. Thus, the focus of the architect changes over time, as the desired results change over time. This is indicated in Figure 2, a figure attributed to Bran Selic.[2]

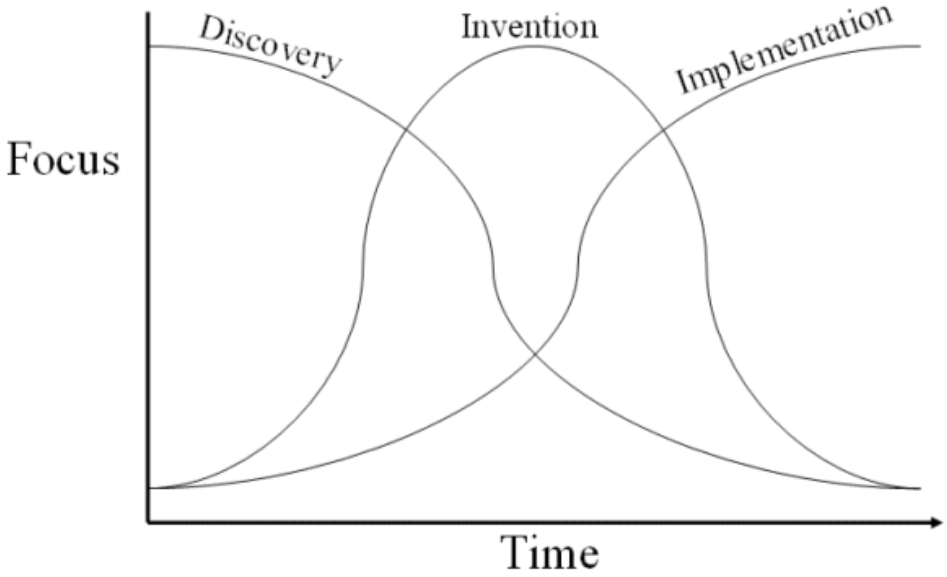

Figure 2: Project emphasis over time

Figure 2 shows that, early on in the project, the architect is focused very much on discovery. The emphasis is on understanding the scope of the system and identifying the critical features and any associated risks -- all elements that clearly have an impact on the architecture. The emphasis then changes to one of invention, where the primary concern is to develop a stable architecture that can provide the foundation for the full-scale implementation effort. Finally, the emphasis changes to implementation, when the majority of discovery and invention has taken place.

It should be noted that discovery, invention, and implementation are not strictly sequential. For example, some implementation will occur early in the project, as architectural prototypes are constructed, and there will be some discovery late in the project as lessons are learned and different strategies for implementing certain elements of the architecture are put in place.

It is worth noting that the process of architecting is not complete until the system is delivered; it is therefore necessary for the architect to be involved until the end of the project. There is often a strong desire to remove the architect from a project once the architecture has stabilized in order to use this precious resource on other projects. However, there may still be architectural decisions to be made until late in the project. In practice, a middle ground is often found where, once the major decisions have been made that affect the architecture, the architect becomes a part-time member of the team. However, the architect should not disengage completely. Of course, a much more flexible situation is where the role of the architect is fulfilled by a team, since some of the members may be used on other projects, whereas those that remain continue to ensure the architectural integrity of the system.

Architecting is driven by many stakeholders

As described earlier in this chapter, an architecture fulfills the needs of a number of stakeholders. The process of architecting must therefore accommodate all of these stakeholders to ensure that their concerns, specifically those that are likely to have an impact on the architecture,

are captured, clarified, reconciled, and managed. It is also necessary to involve the relevant stakeholders in any reviews of the solution to these concerns.

The effort involved in accommodating all of the stakeholders in the process of architecting should not be underestimated. The stakeholders influence many aspects of the process, including the manner in which the requirements are gathered, the form in which the architecture is documented, and the way in which the architecture is assessed.

Architecting often involves making tradeoffs

Given the many factors that influence an architecture, it is clear that the process of architecting involves making tradeoffs. Quite often, the tradeoff is among requirements, and the stakeholders may be consulted to assist in making the correct tradeoff. An example of a tradeoff is between cost and performance: throwing more processing power at the problem will improve performance, but at a cost. This may represent a conflict in requirements and, assuming that the architect has been diligent in his or her work by exploring all options, is a matter that needs to be resolved by the stakeholders whose needs are in conflict.

Other tradeoffs occur in the solution space: for example, the use of one technology over another, or one third-party component over another, or even the use of one set of patterns over another. Making tradeoffs is not something that can or should be avoided. Part of the architect's job is to consider alternatives and make tradeoffs among them.

Architecting acknowledges prior experience

Architects rarely work from a blank sheet of paper. As noted earlier, they actively seek prior experience that may be codified in architectural patterns, design patterns, off-the-shelf components, and so on. In other words, the architect seeks out reusable assets. Only the most ignorant architect does not consider what has gone before.

> *A reusable asset is a solution to a recurring problem. A reusable asset is an asset that has been developed with reuse in mind.*[3]

While it is true that elements of an architecture are reusable in the context of the current system, an architect may also consider the architecture, or elements of it, as reusable assets that can be used outside of the current system.

Architecting is both top-down and bottom-up

Many architectures are considered in a top-down manner, where 1) stakeholder needs are captured and requirements developed, before the architecture is defined; 2) architectural elements designed; and 3) then these elements implemented. However, it is rare for an architecture to be driven *totally* from the top down in this fashion. It is also common for an architecture to be driven from the bottom up, as a result of lessons being learned from any executable software that has been created, such as an architectural proof-of-concept. To some extent, the architecture is also driven bottom-up as a result of design or implementation constraints that have been specified, in

which cases these elements are "givens" that the architecture must accommodate. For example, a constraint might be that the design will use a relational database or interface to an existing system.

Successful architects acknowledge that both approaches to architecting are necessary, and their architectures are created both "top-down" and "bottom-up." This could be referred to as the "meet-in-the-middle" approach to architecting.

Summary

This article has focused on defining the core characteristics of the process of software architecting. In concluding this series on architecture, next month's article will focus on the benefits of treating architecture as a fundamental IT asset.

Acknowledgements

The contents of this article have been derived from a forthcoming book, provisionally entitled "The Process of Software Architecting." As a result, the content has been commented upon by many individuals that I would like to thank, who are Grady Booch, Dave Braines, Alan Brown, Mark Dickson, Luan Doan-Minh, Holger Heuss, Kelli Houston, Philippe Kruchten, Nick Rozanski, Dave Williams, and Eoin Woods.

Notes

[1] IEEE Computer Society, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems: IEEE Std 1471-2000.

[2] Bran tells me he drew this graph on a napkin for Philippe Kruchten.

[3] From the "Reusable Asset Specification," Object Management Group Inc., Document number 04-06-06, June 2004.