Weaving together requirements and architecture

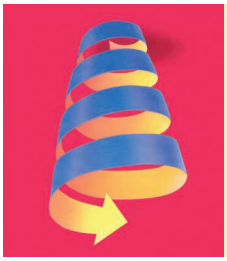
Weaving Together Requirements and Architectures

Bashar Nuseibeh, The Open University

Compelling economic arguments justify why an early understanding of stakeholders' requirements leads to systems that satisfy their expectations. Equally compelling arguments justify an early understanding and construction of a software-system architecture to provide a basis for discovering further requirements and constraints, evaluating a system's technical feasibility, and determining alternative design solutions.

Software-development organizations often choose between alternative starting points—requirements or architectures. This invariably results in a waterfall development process that produces artificially frozen requirements documents for use in the next step in the development life cycle. Alternatively, this process creates systems with constrained architectures that restrict users and handicap developers by resisting inevitable and desirable changes in requirements.

The spiral life-cycle model addresses many drawbacks of a waterfall model by providing an incremental development process, in which developers repeatedly evaluate changing project risks to manage unstable requirements and funding. An even finer-grain spiral life cycle reflects both the realities and necessities of modern software development. Such a life cycle acknowledges the need to develop software architectures that are stable, yet adaptable, in the presence of changing requirements. The cornerstone of this process is that developers craft a system's requirements and its architecture concurrently, and interleave their development (W. Swartout and R. Balzer, "On the Inevitable Intertwining of Specification and Implementation," *Comm. ACM*, vol. 25, no. 7, 1982, pp. 438-440).



Twin Peaks intertwines software requirements and architectures to achieve incremental development and speedy delivery.

THE TWIN PEAKS MODEL

Except for well-defined problem domains and strict contractual procedures, most software-development projects address requirements specification and design issues simultaneously—and justifiably so. Achieving a separation of requirements and design steps is often difficult because their artificial ordering compels developers to focus on either aspect at any given time. In reality, candidate architectures can constrain designers from meeting particular requirements, and the choice of requirements can influence the architecture that designers select or develop.

Based on our experience in industrial software-development projects, my colleagues and I use an adaptation of the spiral life-cycle model. We informally call this model Twin Peaks to emphasize the equal status we give to requirements and architectures. Although this model develops requirements and architectural specifications concurrently, it continues to separate problem structure and specification from solution structure and specification, in an iterative process that produces progressively more detailed requirements and design specifications, as Figure 1 suggests.

The Twin Peaks model addresses the three management concerns identified by Barry Boehm ("Requirements that Handle IKIWISI, COTS, and Rapid Change," *Computer*, July 2000, pp. 99-102):

- *I'll Know It When I See It (IKIWISI).* Requirements often emerge only after users have had an opportunity to view and provide feedback on models or prototypes. Twin Peaks explicitly allows the user to explore the solution space early, permitting incremental development and consequent risk management.
- *Commercial off-the-shelf software (COTS).* Increasingly, software development is actually a process of identifying and selecting desirable requirements from existing commercially available software packages. With Twin Peaks, developers can identify requirements and match architectures with commercially available products, rapidly and incrementally. The developer benefits by quickly narrowing the selections or making key architectural decisions to accommodate existing COTS solutions.
- *Rapid change.* Managing change continues to be a fundamental problem in software development and project management. Focusing on finer-grain development, Twin Peaks is receptive to changes as they occur. Analyzing and identifying a soft-

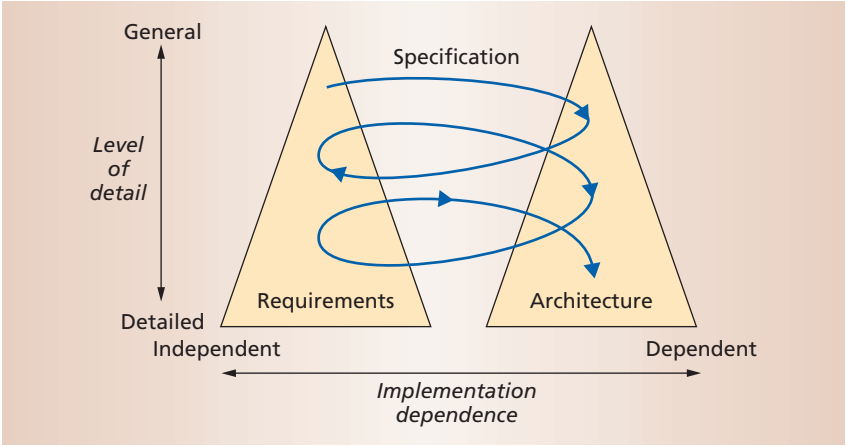


Figure 1. The Twin Peaks model develops progressively more detailed requirements and architectural specifications concurrently. This is an adaptation of the model first published in Paul Ward and Stephen Mellor's* Structured Development for Real-Time Systems: Introduction and Tools*, vol. 1, Prentice Hall, Upper Saddle River, N.J., 1985, and subsequently adapted by Andrew Vickers in his student lecture notes at the University of York, UK.

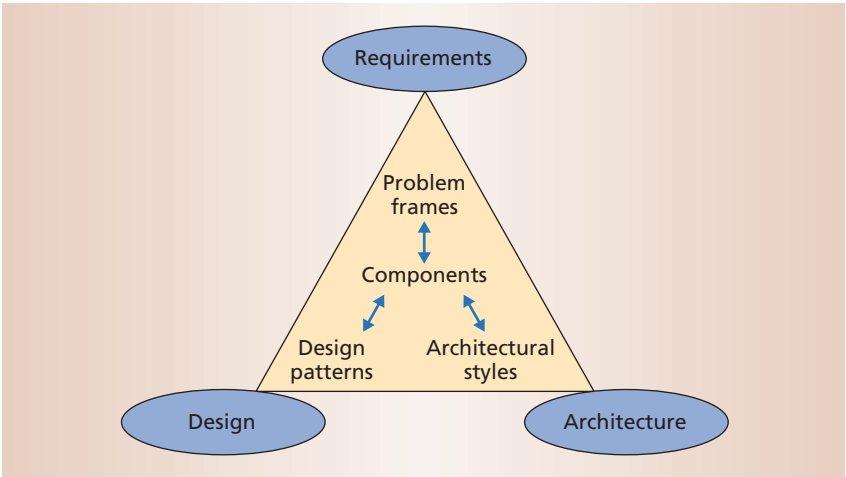


Figure 2. Part of the software-development terrain, with requirements, architecture, and design receiving similar attention. Patterns of each affect the kind of system (components) developed, and the relationship between them is a key determinant of the kind of process developers adopt.

ware system's core requirements are requisite to developing a stable software architecture amid changing requirements.

Developing software systems in these contexts requires considering different development processes. Addressing IKIWISI means starting design and implementation earlier than usual; using COTS requires considering reuse at an earlier stage of requirements specification; remaining competitive while adapting to rapid change requires us to perform all development tasks more quickly.

BUILDING MODULAR SOFTWARE INCREMENTALLY

Building systems with well-defined component interfaces offers opportunities for effective reuse and maintenance. It is unclear, however, how component-based development approaches fit into the development process. One approach is to consider the use of requirements, architecture, and design *patterns*. The software design community has already identified *design patterns* for expressing a range of implementations. The software architectures community has identified suitable *architectural styles* for meeting various global requirements. The requirements engineering community has promoted the use of Michael Jackson's *problem frames* and Martin Fowler's *analysis patterns* to identify problems for which solutions exist.

What relationships connect these different patterns? Figure 2 suggests that we can treat patterns of requirements, designs, and architectures as the starting point for component-based development. For example, a given fixed architecture can limit the kinds of problems that we can address and the possible designs that we can develop, while rigid requirements can limit the candidate architectures and design choices.

From a requirements engineering perspective, achieving a satisfactory problem structuring using problem frames as early as possible is essential. Given that existing architectures can influence how developers structure problems, some problem frames may need to be reverse engineered from existing architectural designs.

WEAVING THE DEVELOPMENT PROCESS

Twin Peaks shares much in common with Kent Beck's Extreme Programming, such as the goal of exploring implementation possibilities early and iteratively. Twin Peaks is complementary to XP in that it focuses on software-development front-end activities—requirements and architectures. This potentially addresses some of the issues of scale that are often claimed to be XP's weaknesses.

Early understanding of requirements and choice of architecture are key to managing large-scale systems and projects. XP focuses on producing code—sometimes at the expense of the wider picture of requirements and architectures.

Of course, focusing on requirements and architectures in itself is not sufficient to achieve scalability. Modularity and iteration are also crucial. Twin Peaks is inherently iterative, and combining it with tried

and tested components derived from well-understood patterns can facilitate incremental development of large-scale systems. The resultant overall software-development process inevitably takes a more complex path from problem to solution.

Although the conceptual differences between requirements and design are now much better understood and articulated, the *process* of moving between the problem world and the solution world is not as well recognized (Michael Goedicke and Bashar Nuseibeh, "The Process Road between Requirements and Design," *Proc. 2nd World Conf. Integrated Design and Process Technology*, SDPS, Austin, Texas, 1996, pp. 176-177). Researchers and practitioners are struggling to develop processes that allow rapid development in a competitive market, combined with the improved analysis and planning that is necessary to produce high-quality systems within tight time and budget constraints.

The Twin Peaks model represents much of the existing, but implicit, state of the practice in software development.

A more robust and realistic development process allows both requirements engineers and system architects to work concurrently and iteratively to describe the artifacts they wish to produce. This process allows developers to better understand problems through consideration of architectural constraints, and they can develop and adapt architectures based on requirements.

Many difficult questions remain unanswered:

- What software architectures (or architectural styles) are stable in the presence of changing requirements, and how do we select them?
- What classes of requirements are more stable than others, and how do we identify them?
- What kinds of changes are systems likely to experience in their lifetime, and how do we manage requirements and architectures (and their development processes) in order to minimize the impact of these changes?

The answers to these questions will influence key emerging software-development contexts including

- *product lines and product families*, which need stable architectures that tolerate changing requirements;
- *COTS systems*, which require identifying and matching existing architectures to requirements (as opposed to developing system requirements from scratch); and
- *legacy systems*, which can incorporate existing system constraints into requirements specifications.

Processes that embody Twin Peaks characteristics are the first steps in tackling the need for architectural stability in the face of inevitable requirements volatility.

Development processes that facilitate fast, incremental delivery are essential for software systems that need to be developed quickly, with progressively shorter times-to-market as a key requirement. The Twin Peaks model represents much of the existing, but implicit, state of the practice in software development. While it is based on accepted research in its evolutionary development, the software-development community has not yet recognized that such a model represents acceptable practice. ✸

***Bashar Nuseibeh** is a professor of computing at The Open University, United Kingdom, and director of the Centre for Systems Requirements Engineering at the Department of Computing, Imperial College, London. Contact him at B.A. Nuseibeh@open.ac.uk or visit http://mcs.open.ac.uk/ban25.*

Editor: Barry Boehm, Computer Science Department, University of Southern California, Los Angeles, CA 90089; boehm@sunset.usc.edu