Weaving the Software Development Process Between Requirements and Architectures

Bashar Nuseibeh
Computing Department
The Open University
Walton Hall
Milton Keynes MK7 6AA, U.K.
Email:B.A.Nuseibeh@open.ac.uk

ABSTRACT

This position paper argues for the concurrent, iterative development of requirements and architectures during the development of software systems. It presents the "Twin Peaks" model – a partial and simplified version of the spiral model – that illustrates the distinct, yet intertwined activities of requirements engineering and architectural design. The paper suggests that the use of various kinds of patterns – of requirements, architectures, and designs – may provide a way to increase software development productivity and stakeholder satisfaction in this setting.

1 INTRODUCTION

There are compelling economic arguments why an early understanding of stakeholders' requirements leads to systems that more closely meet these stakeholders' expectations. There are equally compelling arguments why an early understanding and construction of a software system architecture provides a basis for discovering further system requirements and constraints, for evaluating a system's technical feasibility, and for evaluating alternative design solutions.

Many software development organisations often make a choice between alternatives starting points – requirements or architectures – and this invariably results in a subsequent waterfall-like development process. Such a process inevitably leads to artificially frozen requirements documents – frozen in order to proceed with the next step in the development life cycle; or leads to systems artificially constrained by their architecture that, in turn, constrain their users and handicap their developers by resisting inevitable and desirable changes in requirements.

There is some consensus that a "spiral" (Boehm 1988) life cycle model addresses many of the drawbacks of a waterfall model, and consequently addresses the need for an incremental development process in which project requirements and funding may be unstable, and in which project risks change and thus need to be evaluated repeatedly. This article suggests that a finer-grain spiral development life cycle is needed, to reflect both the realities and necessities of modern software development – a life cycle that acknowledges the need to develop software architectures that are stable, yet adaptable, in the presence of changing requirements. The cornerstone of such a process is that a system's requirements and its architecture are developed concurrently, that they are "inevitably intertwined" (Swartout & Balzer 1982), and that their development is interleaved.

2 TWIN PEAKS: A CONCURRENT, SPIRAL DEVELOPMENT PROCESS

Figure-1 suggests a partial development model that highlights the concurrent, iterative process of producing progressively more detailed requirements and design specifications. We informally call this model *Twin Peaks* to emphasize the equal status we give the specification of requirements and architectures[1]. We could have also used the more general term *design* rather than *architecture*, as we consider our model to be as applicable to the development of detailed design specifications as it is to architectural design specifications. However, from a project management perspective, the abstraction provided by architectures is sufficient for our purposes.

---

[1] The model is an adaptation of one first published in P. Ward and S. Mellor's *Structured development for real-time systems* (Volume 1: Introduction and tools, Prentice Hall, 1985), and subsequently adapted by Andrew Vickers in his student lecture notes at the University of York (UK).
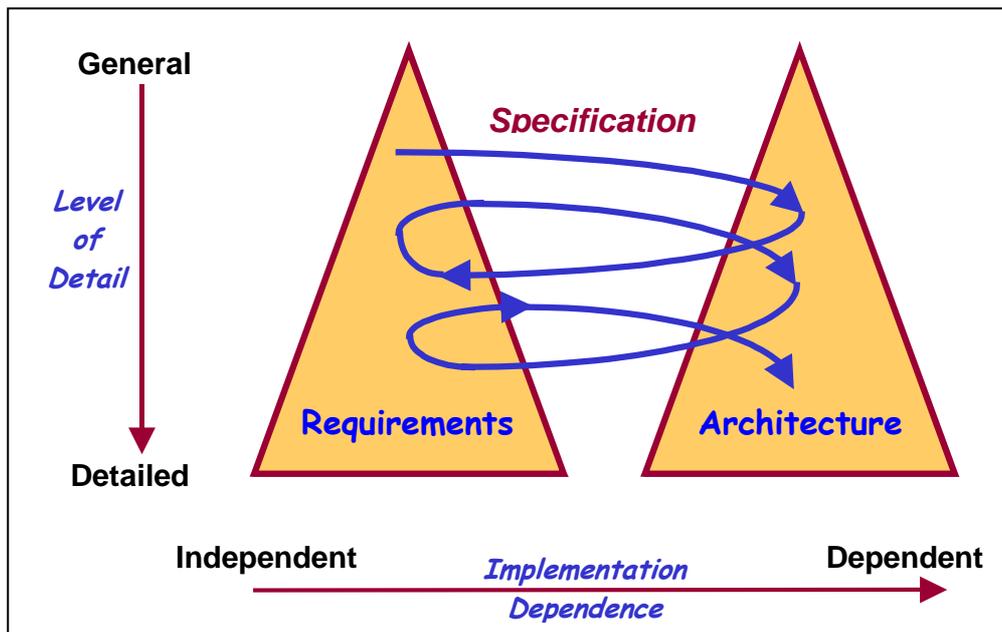
FIGURE-1: TWIN PEAKS – A MODEL OF THE CONCURRENT DEVELOPMENT OF PROGRESSIVELY MORE DETAILED REQUIREMENTS AND ARCHITECTURAL (DESIGN) SPECIFICATIONS.

Our experience on a number of industrial projects support this development life cycle model. Except for very well defined problem domains and strict contractual procedures, most software development projects address issues of requirements specification and design more or less concurrently, and rightly so. At worst, initial requirements gathering is quickly followed by the development or selection of one or more candidate architectures of a system. In many cases, candidate architectures are even provided as part of the requirements. Indeed, candidate architectures may constrain what requirements can be met, and of course the choice of requirements impacts the architecture selected or developed.

As the development process proceeds (and we focus here on specification development in particular), the requirements are elaborated as is the software system architecture. However, although both requirements and architecture are developed concurrently, their distinct content is preserved; that is, the activities of problem structuring and specification (requirements) are separated from (but related to) solution structuring and specification (architecture). This separation, while recognized as important, is often difficult to achieve in alternative life-cycles, since the artificial ordering of requirements and design steps compels developers to focus on either of the two at any one time.

The Twin Peaks model addresses the three management concerns identified by Barry Boehm in an earlier article (Boehm 2000), namely IKIWISI (I'll Know It When I See It), COTS (Commercial off-the-shelf) software, and rapid change.

IKIWISI. Requirements often "emerge" only after significant analysis of models or prototypes of a system has taken place, and users have been given an opportunity to view and provide feedback on the models or prototypes. The Twin Peaks model explicitly allows early exploration of the solution space, thereby allowing incremental development and the consequent management of risk.

COTS. Increasingly, software development is actually a process of identifying and selecting existing software packages (Finkelstein *et al.* 1996, Maiden & Ncube 1998). Package selection requires the identification of desirable requirements (often expressed as features or services), and a matching of these to what is commercially available. Adopting the Twin Peaks model allows rapid and incremental requirements identification and architectural matching. This can be invaluable in quickly narrowing down the choices available, or perhaps making key architectural choices to accommodate existing COTS solutions.

Rapid Change. Managing change continues to be a fundamental problem in software development and project management. *Rapid* change exacebates this problem. We believe that the Twin Peaks model focuses on finer-grain development and is therefore more receptive to changes as they occur. However, analysing the impact of change is still

a difficult task. Key to addressing this task is identifying the *core* requirements of a software system. Such requirements express those stakeholders' goals that are likely to persist for the longest period of time, and that are likely to lead to a software architecture that is stable in the face of changes in other requirements. Core requirements may also be those whose impact on a software architecture is so substantial that they have to be frozen in order to avoid the excessive (unacceptable) cost of changing them.

3 BUILDING MODULAR SOFTWARE INCREMENTALLY

Developing software systems in the context of IKIWISI, COTS, and rapid change suggests that we need to consider different processes of development. IKIWISI means that we need to start design and implementation much earlier than usual; COTS means that we need to consider reuse at the much earlier stage of requirements specification; and rapid change means that we have to be able to do all this much more quickly in order to be competitive.

There is increasing recognition that building systems in terms of components with well defined interfaces offers opportunities for more effective reuse and maintenance. It is not always clear, however, how component-based development approaches fit into the development process. One way is to consider, concurrently, "patterns" of requirements, architectures and designs. The software design community has already identified a number of "design patterns" (Gamma *et al.* 1996) that can be used to express a range of implementations. The software architectures community has identified "architectural styles" (Shaw & Garlan 1996) that are suited to meeting various global requirements. And, the requirements engineering community has promoted the use of "problem frames" (Jackson 2001) or "analysis patterns" (Fowler 1996) to identify classes of problems for which there are existing, known solutions. This begs the question: what are the relationships between these different kinds of patterns?

Figure-2 suggests that such patterns of requirements, designs, and architectures can be treated as a resource for component-based development. Indeed, the figure suggests that – in line with the Twin Peaks model – the "starting point" of development may be requirements, design, or architecture. If a given architecture is fixed, this impacts on the kinds of problems that can be addressed by that architecture, and the kinds of designs that may be possible. If the requirements are more rigid, then the candidate architectures and design choices are also limited. If a certain design is chosen, then both the architectures in which this design fits and the problems that are addressed by this design are also limited.
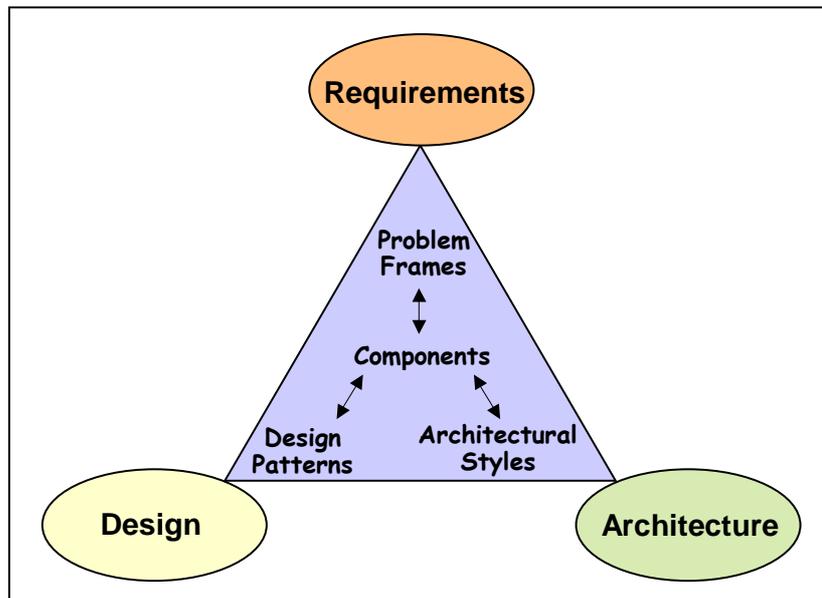
FIGURE-2: PART OF THE SOFTWARE DEVELOPMENT TERRAIN, WITH REQUIREMENTS, ARCHITECTURE, AND DESIGN RECEIVING SIMILAR ATTENTION. PATTERNS OF EACH HAVE AN IMPACT ON THE KIND OF SYSTEM (COMPONENTS) DEVELOPED, AND THE RELATIONSHIP BETWEEN THEM IS A KEY DETERMINANT OF THE KIND OF THE DEVELOPMENT PROCESS ADOPTED.
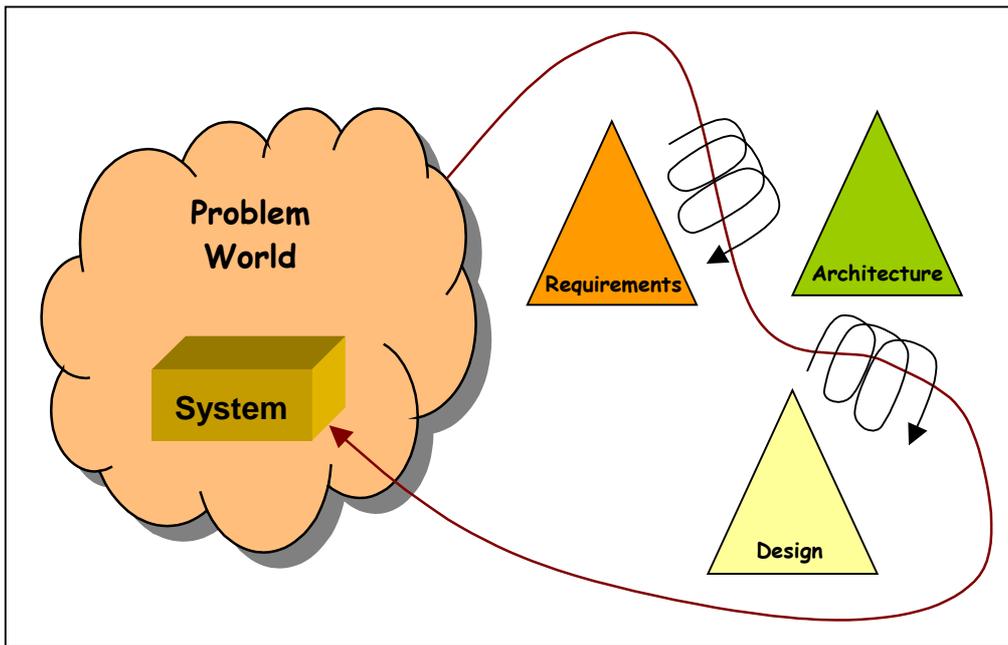
FIGURE-3: WEAVING THE SOFTWARE DEVELOPMENT PROCESS FROM PROBLEM TO SOLUTION. THE TRIANGULAR ICONS INDICATE ARTEFACTS THAT GROW (FROM TOP TO BOTTOM) AS THE PROCESS PROCEEDS. THERE IS AN IMPLICIT GLOBAL FEEDBACK LOOP AS WELL, IN WHICH THE SYSTEM INSTALLED IN THE WORLD CHANGES THE WORLD, AND POTENTIALLY NECESSITATES ANOTHER WEAVING JOURNEY.

From a requirements engineering perspective, it is essential that a satisfactory problem structuring is achieved as early as possible. A vehicle for achieving such a structuring is Jackson's problem frames[2]. However, given that existing architectures may influence how problems are structured, it may be necessary – as Jackson suggests – to "reverse engineer" some problem frames from existing architectural designs.

4 PROJECT MANAGEMENT: WEAVING THE DEVELOPMENT PROCESS

The Twin Peaks model of software development shares much in common with Extreme Programming (XP) (Beck 1999), such as the goal of exploring implementation possibilities early and iteratively. The focus of Twin Peaks, however, is very different from, but perhaps complementary to, the XP model in that it focuses on the front-end activities of the software development life cycle; that is, on requirements and architectures. This potentially addresses some of the issues of scale that are often claimed as weaknesses of XP. Early understanding of requirements and the 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 architecture. Of course, the focus on requirements and architectures in itself is not sufficient to achieve scalability. Modularity and iteration are also crucial. Twin Peaks is inherently iterative, and combined with the use of patterns can facilitate incremental development of large scale systems more quickly, using tried and tested components derived from well-understood patterns.

The resultant overall software development process inevitably takes a much more complex path from problem to solution. Figure-3 is a hugely simplified illustration that tries to convey the winding route in which development proceeds as requirements, architectures, and designs are elaborated iteratively and often concurrently.

5 CONCLUSIONS

With the many advances in software development in recent years, it is perhaps appropriate to re-visit the relationships between requirements and design. Although the conceptual differences between requirements and design are now much better understood and articulated (Jackson 1995), the *process* of moving between the problem world and the solution world is much less so (Goedicke & Nuseibeh 1996). 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.

---

2 Recall that a problem frame defines the shape of a problem for which there is a known solution.

We have suggested that a more robust and realistic development process is one that allows both requirements engineers and system architects to work concurrently and iteratively to describe the artefacts they wish to produce. In this way, problems are better understood through consideration of architectural constraints, and architectures can be developed and adapted based on requirements.

We have exposed only the tip of the iceberg. 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 manage the impact of these changes?

The answers to these questions will have significant impact on the way software is developed and projects are managed. Particular impact will be felt in some key emerging development contexts:

- ❑ *Product lines and product families* – where there is a need for stable architectures that tolerate changing requirements.

- ❑ *COTS systems* – where there is a need to identify and match existing units of architectures to requirements (as opposed to developing system requirements from scratch).

- ❑ *Legacy systems* – where there is a need to incorporate existing system constraints into requirements specifications.

For software systems that need to be developed quickly, with progressively shorter times-to-market as a key requirement, development processes that facilitate fast, incremental delivery are essential. The Twin Peaks model that we have presented in this article represents much of the existing current state-of-the-practice in software development. It is also based on accepted research into evolutionary development as embodied in Boehm's spiral model (Boehm 1988). What is missing, however, is a more explicit recognition by the software development community that such a model represents acceptable practice.

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

6   REFERENCES
[1]   B. Boehm (1988), "A Spiral Model of Software Development and Enhancement", *Computer*, 21(5):61-72, IEEE CS Press.

[2]   B. Boehm (2000), "Requirements that Handle IKIWISI, COTS, and Rapid Change", *Computer*, 33(7):99–102, July 2000, IEEE CS Press.

[3]   K. Beck (1999), *Extreme Programming Explained: Embracing Change*, Addison-Wesley.

[4]   A. Finkelstein, M. Ryan and G. Spanoudakis (1996), "Software Package Requirements and Procurement", In Proceedings of 8th International Workshop on Software Specification & Design (IWSSD-8), 141-146, Schloss Velen, Germany, 22-23 March 1996, IEEE CS Press.

[5]   M. Fowler (1996), *Analysis Patterns,* Addison Wesley.

[6]   E. Gamma, R. Helms, R. Johnson, J. Vlissides (1995), *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, 1995.

[7]   M. Goedicke & B. Nuseibeh (1996), "The Process Road Between Requirements and Design", *In Proceedings of 2nd World Conference on Integrated Design and Process Technology,* 176-177, Austin, Texas, USA, 1-4 December 1996, SDPS.

[8]   M. Jackson (1995), *Software Requirements & Specifications: A Lexicon of Practice, Principles, and Prejudices*, Addison-Wesley.

[9]   M. Jackson (2001), *Problem Frames: Analyzing and Structuring Software Development Problems*, Addison-Wesley.

[10] N. Maiden and C. Ncube (1998), "Acquiring Requirements for Commercial Off-the-Shelf Package Selection", *Software,* 15(2):46-56, IEEE CS Press.

[11] M. Shaw and D. Garlan (1996), *Software Architectures,* Prentice-Hall.

[12] W. Swartout and R. Balzer (1982), "On the Inevitable Intertwining of Specification and Implementation", *Communications of the ACM*, 25(7): 438-440.