From Spiral to Anchored Processes:  
A Wild Ride in Lifecycle Architecting  

Dr. Peter Hantos  
Xerox Corporation  
701 South Aviation Blvd., MS: ESAE-376  
El Segundo, CA 90245 USA  
+1 310 333 9038  
peter.hantos@usa.xerox.com  

Presentation Abstract  

KEYWORDS  
Lifecycle Architecting, Spiral Lifecycle Model, Incremental/Iterative Lifecycle Model, Anchor Points, Technology Readiness  

INTRODUCTION  
In responding to the Y2000 USC/IEEE Executive Workshop's challenge, the author presents a case study reflecting on the Xerox experiences with the Spiral Model, and provides an overview of the approaches Xerox has chosen to define the optimal lifecycle model.  

LIFECYCLE ANXIETY WAVES AT XEROX  
This historical overview briefly characterizes the 1987-2000 period, focusing on the maturity and stability of the used lifecycle models. The first notable unstable period was 1987 to 1992, when the Waterfall Model's shortcomings were already felt, but no dominant solutions were available. The period from 1996 to 1997 was also a sensitive time, when the Spiral Model's deficiencies were identified, but solid mitigation strategies were not provided. Recent approaches include the adoption of the Anchor Point concept, and the move to an incremental/iterative framework. The Y2000 and beyond lifecycle architecting challenge is the definition of a proper product line lifecycle process.  

ANCHORED PROCESSES  
The Anchor Point concept provides a breakthrough solution to the lifecycle architecting problem. The Xerox approach is characterized as an Anchored Lifecycle Macro Process, providing synchronization for multiple software processes, hardware processes, and business processes as well. While the Lifecycle Macro Process is primarily embracing incremental/iterative processes, it is flexible enough to accommodate all kinds of lifecycle model elements.  

TECHNOLOGY READINESS  
Technology Readiness implies that the assessment of the problems has been completed, and there is no indication that a special solution must be invented beyond normal design engineering practices to satisfy the requirements. Technology Readiness provides the right risk management framework for market-driven, commercial product development.  

SUGGESTED READING  

BIOGRAPHY  
Peter Hantos is Manager of Software Engineering at Xerox, directing all aspects of software quality assurance, software process improvement, product quality assurance and reliability. Previously he was principal scientist of the Corporate Software Engineering Center, where he authored the corporate software development standard and the Software Technology Readiness processes. Earlier at Xerox he held various management positions in manufacturing automation and software development. He also managed a software engineering competency center in Corporate Research and Technology, in charge of software technology transfer from the Palo Alto Research Center and providing methodology, tools and infrastructure support. Prior to joining Xerox, he worked in product development at Spacelabs, Inc. in California. He also has an extensive academic career as Assistant Professor, first at the Technical University of Budapest, and later at the University of California, Santa Barbara. He holds MS and Doctorate degrees in Electrical Engineering from the Technical University of Budapest, Hungary. He is a senior member of the IEEE, and a member of ACM.
Importance of Software Prototyping and the Spiral Model

Modern software development demands the use of Rapid Application Prototyping. Professor Luqi at the Naval Postgraduate School coined the term Computer Aided Prototyping to describe this work. Her leadership showed its effectiveness in gaining understanding of the requirements, reducing the complexity of the problem and providing an early validation of the system design. For every dollar invested in prototyping one can expect a $1.40 return within the life cycle of the system development.

Barry Boehm’s experiments showed that prototyping reduces program size and programmer effort by 40%. It is the technology that is the foundation for his Spiral development method. Prototyping is being used successfully to gain an early understanding of system requirements, to simplify software designs, to evaluate user interfaces and to test complex algorithms. It is a best-in-class software approach.

Fully 30-40% of system requirements will change without prototyping. Rapid Application Prototyping offers the hope of looking at the dynamic states of the system before we build it, whereas most other software engineering focuses on the source code. The special problems of reliability, throughput and response time as well as system features are addressed in the best prototypes.

Software is hard because it has a weak theoretical foundation. Most of the existing software theory focuses on the static behavior of the software - analysis of the source listing. There is little theory on its dynamic behavior - how it performs under load. To avoid serious network problems telecommunications software systems are over-engineered with plenty of capacity for two or three times the expected load. Without analysis of the dynamics, application designer have no idea of the resources they will need once their software is operational. Software prototyping has proven its metal in helping designers avoid these problems in their production systems.

Much has been written about the best way to develop software applications. But there is no "best way." Both prototyping and requirements are necessary. The tried-and-true process of synthesis and analysis is used to solve software-engineering problems. Bottom-up is the synthesis. Top down is the analysis. Bottom up is prototyping. Top down is developing requirements. As early as 1975 Winston Royce wrote that requirements must be validated to be useful. Prototyping is the best way to validate the requirements and encourage synthesis. Prototyping also eases communication with the customer and with the designer. Formal written requirements are needed to establish a clear definition of the job, to control changes and to communicate the system capabilities between the customer and the developer.

So where does this leave us? Start with an English language written statement of a problem and broadly outline its solution. Now build a prototype for the elements where you need insight. Analyze the prototype using computer aided prototyping technology and synthesize a new solution either by refining the prototype or building a new one.
Once you and the customer agree on the workings of the prototype, write requirements that include features, performance goals, product costs, product quality, development costs and schedule estimates.

What do I mean by prototype? Prototyping is the use of approximately 30% of the ultimate staff to build one or two working versions of various aspects of a system. It is not production code but it may eventually become pre-production code or it may be completely discarded. In the prototyping effort, we aren't concerned with the maintainability of the code nor are we concerned with formally documenting it. Code resulting from prototyping is often used to train the programmers. Only after we have written specifications resulting from the experience with the prototype should we start the formal development process. If we are fortunate enough that some of the code that was developed for the prototypes can be carried forward, that's great, if not, there is no loss.

A prototype produces "running" software and the production development produces "working" software.

Experience shows that early prototyping is fundamental to the success of operations support telecommunications software products. The Spiral Model provides a disciplined approach and framework for using prototyping on a large project. The reasons why prototyping is fundamental include:

1) The prototype provides a vehicle for systems engineers to better understand the environment and the requirements problem being addressed.

2) A prototype is a demonstration of what's actually feasible with existing technology, and where the technical weak spots still exist.

3) A prototype is an efficient mechanism for the transfer of design intent from system engineer to the developer.

4) A prototype lets the developer meet earlier schedules for the production version.

5) A prototype allows for early customer interaction.

6) A prototype demonstrates to the customers what is functionally feasible and stretches their imagination, leading to more creative inputs and a more forward-looking system.

7) The prototype provides an analysis test bed and a vehicle to validate and evolve system requirements.

Now for a balanced view we must consider prototyping pitfalls. If the initial prototype is too far off the mark, we can get some disastrous results such as fielding unresponsive systems that really turn the user off, or we could concentrate on short term
needs, or tinker with algorithms, or develop sub-optimal systems. To avoid these pitfalls, we should write requirements to force us to do a careful analysis of the users overall problem before plunging into the code. It is difficult to manage and schedule prototyping and hard to get people off the prototype into the real system. Specifically, getting them to deal with size, performance, and the build constraints and practicalities of a production system can be a management problem.

In one project we tried to use structured system analysis and failed. Even though it is an excellent analysis tool, it is a horrible way to communicate with the customer. The customer just doesn't want to learn the language of structured system analysis. With the tool in hand System Engineers tend to jump from the general to the detailed adding more confusion. Whereas, English text feature memos provide a convenient way of communicating across the customer/engineer boundary; but, they are hard to keep current and leave too much to the imagination. This is where the power of the Spiral model comes in. The customer sees what the engineer thinks he wants and is a good position to correct false impressions. With the corrections in mind, the engineer spins another cycle on the Spiral, getting it right before investing in implementation.

Nevertheless, structured system analysis makes sure that our thinking is clear. It is a valuable tool for the system analyst. Just do not share the analysis with the customer or the developer. They who will want to talk with the analyst after they read the formal written requirements. Demonstrating the system concept with a prototype shows that it is possible to build such a system, irons out confusion in the requirements, makes the discussions fun and effective.

Here are several real-world uses of effective of prototypes:

Project A: Order Reading and Analysis Software

Size of Prototyping Effort: 12K lines of C Code (10% of final system module)

Purpose:

Find a method for order reading and analysis, applicable to variable formats.

Experience:

1. Final requirements based on prototype results.
2. Alerted developers' to the possibility of a having a tunable system.
3. Early evaluation of functional decomposition and performance showed bottlenecks in the dispatcher.
4. Eliminated the possibility of reusing code from another project.
5. Prototype was thrown away due to decomposition and performance problems.

Duration and Staff of Prototype:

Center for Software Engineering  Larry Bernstein  02/01/2000
The prototype took four people for eight months. Subsystem Development took 40 people three years.

**Project B: Outside Plant Data Base System**

Size of Effort: 5% of 500K line of source code.

Purpose:
To evaluate database structures for an outside plant data and to experiment with approaches to handling multiple future states of equipment usage.

Experience:

1. A data base structure using hyper graph theory was invented.
2. An algorithm for handling both time-driven and event-driven assignments was invented.
3. The prototype became the basis for the production code.
4. UNIX flat files were used to model loop plant by way of a directed graph.
5. The prototype showed the practicality of porting from Unix lat files to a schema in the Mainframe’s database system.

Duration:

The prototype took 3 people for 15 months. The subsystem development took 50 people two years.

**Project C: Store and Forward Message Switch**

Size of Effort: 2% of total System of 500K lines of code

Purpose:
To evaluate new scheduling algorithm for an existing system.

Experience:

In a store and forward message switching system, we found the buffer overload strategy was unstable. After the system went into overload and returned to normal processing, it would immediately poll for more traffic. Polling had a higher priority than distributing the messages already queued in the mistaken belief that polling must be the highest priority task to meet the response time requirement. This exhausted even more buffers, drove the system into overload again and caused it to stay in overload longer than before.

To convince the customer and ourselves that lowering the priority of polling would solve the problem, a prototype system was put together in the test lab. It
demonstrated stable over-load response with an imperceptible increase in
response time. With the prototype in hand, schedules for a system release with
improved over-load response were adopted.

Demonstration of the prototype avoided an emotionally charged battle with the
users over perceived response time degradation.

The prototype became pre-production code that took one year to field after it was
working in the lab.

Duration of Prototype

The prototype took 1 person four months. The production code required three
people for one year.

**Project D: Order entry**

Size of prototyping effort: 10%

Purpose of Prototype:

1. Evaluate Human Interface by the user.
2. Validate economic assumptions.
3. Train software developers before requirements are available.

Experience:

The Human Interface was changed to put more data on a single screen as the user
preferred to see those transactions that could be completed with a single
transaction and therefore the screens became denser.

The table structures were changed to make them easier to change as needs
changed and to make it possible to keep the data consistent in the short daily
update cycle.

The economics proved in for the system. The great dependency of the viability of
the project to the transaction response time was established.

An earlier version was rushed to production without adequate prototyping. This
version failed. Not only were there response time problems but also the system
was difficult to operate and could not scale. Note that the users were thrilled with
the functional capability and very disappointed that they could not use it. It took
project manager courage to halt system deployment.
Duration and Staff:

The prototype approximately seven people for nine months.

Summary

These prototyping experiences show how important the prototype was in getting to an accurate system definition. This definition must reflect the changing needs of the user, be practical to build and fit within the environment. The prototype is the ideal vehicle to get system engineer, developer and user to make tradeoffs among

a. what the system will do,
b. the software technology used,
c. the time it will take to build it,
d. the development cost, and
e. the deployment cost.

The prototype allows people to deal rationally with any the conflicts between the system requirements and system economics. They provide a framework for dealing with potential solution in concrete terms. The prototype establishes the feasibility of the design.

The prototype lets the developer ‘fail small’ so that he can ‘succeed big.’

Larry Bernstein
Founder, Have Laptop - Will Travel
Email: lbernstein@ieee.org
Phone: 973-258-9213