1

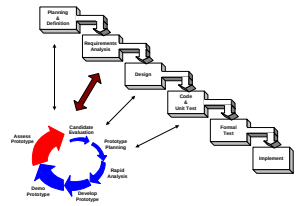
Software Processes

Prototyping Life Cycle Models

In this lecture we'll discuss some examples of prototyping life cycles.

2

Rapid Prototyping Model



SOURCE: J.M. Demasco, "Incorporating Rapid Prototyping into the Software Project Life Cycle," National Conference on Software Engineering, Washington, DC, February 1990.

Another type of life cycle model is a rapid prototyping model…and there are several variations. In the model illustrated here, which I developed for one of my clients, rapid prototyping is really just incorporated as a tool into a pure waterfall or modified waterfall model.

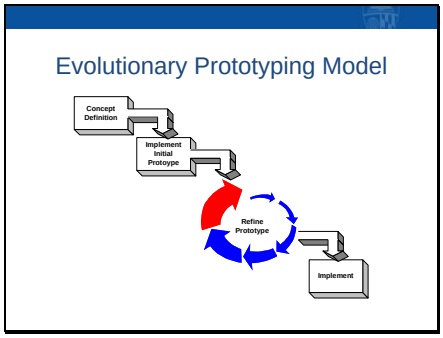
To begin with, what's a rapid prototype? A rapid prototype is a model of a piece of a software product, or the entire software product, that has limited functionality…and that can be produced rapidly…in hours or days. The prototype is working software and can have some user interactivity built into it. A prototype may address the user interface, provide simulated navigation, report mockups, and so forth.

In the model I developed for my client, prototyping could be used in the concept definition phase, the requirements phase, the design phase, or the coding phase. In practice, it was used mostly in the concept definition and requirements phases, at least for this particular client. The whole idea behind a rapid prototype is to provide working software that simulates or approximates some aspect of the product to be built. For example, user menus and navigations, in executable form can give product users a very clear vision of what the user input will look like and can also help to define needed functionality. These software components would then be developed with tools that were different from the tool used to build the prototype…and the prototype would serve as the specification of what needed to be

built. The prototype software is basically thrown away. Rapid prototyping is often useful when users can't visualize product capabilities from static models like traditional requirements or report mock-ups.

Prototyping almost always involves iterating. What distinguishes it from other iterative style processes is that we have rapidly constructed prototypes that are executable software and that only isolated pieces of a product are iterated.

3

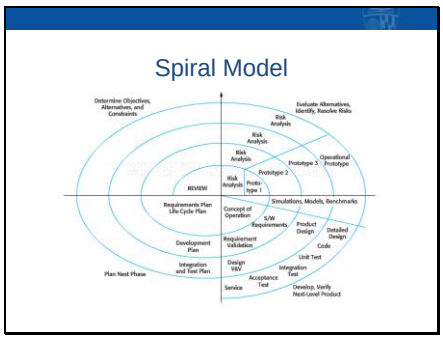


A variation on the earlier rapid prototyping model is the evolutionary prototyping model. In this process, you typically start with some general notion of what the product is supposed to do. You then rapidly build portions of the product, usually starting with the user interfaces and product outputs, and then iteratively evolve the product to have more functionality until it completely meets the customer's needs. In this model, the prototypes are not thrown away, but are evolved into the final product.

This type of model can be useful when requirements are changing rapidly, the customer is reluctant to commit to a set of traditional requirements, or the application area is not well understood either by the customer or the developers.

The challenge of this model is that it's difficult to predict how many prototype iterations will be required, and it requires disciplined management so that the process doesn't degrade into a code and fix model.

4



Another example of both an iterative life cycle model and a prototype life cycle model is the spiral model. I could have also put this model in my lecture on iterative/incremental models, but I decided to put it here. It's a hybrid and could easily fit into either category.

The spiral model is a risk-driven model…and it uses prototypes and iteration to mitigate project risks. Each iteration is often managed as a separate mini-project. Each iteration identifies one or more major project risks…and the iterations continue until all the major risks have been addressed, at which point the final iteration may employ a waterfall-style life cycle.

For each iteration, six basic steps are executed: Determine objectives, alternatives, and constraints; Identify and resolve risks; evaluate alternatives; develop the project deliverables for that iteration and verify that they are correct; plan the next iteration; and commit to an approach for the next iteration.

Here's the basic idea…mapped to the diagram. You start on a small scale in the middle of the diagram, and explore the risks. You then make a plan to handle the risks, and follow the afore-mentioned steps. Any deliverable software is typically considered to be a partial prototype…not necessarily a rapid prototype. Each iteration moves the project to a larger scale. Early iterations are normally cheaper than later iterations. For example, developing a concept of operations is cheaper than developing requirements, and so forth.

Now, this diagram is meant to illustrate the concept and shouldn't be taken literally. For example, it's not necessary that you have exactly four loops in the spiral, and it's not necessary that you perform the six steps in exactly the order I presented them.

The primary disadvantage of the spiral model is that it's complicated and requires careful management attention.

5

Life Cycle Model Scorecard

Life Cycle Model Capability	Rapid Prototype	Spiral Model
Works with poorly understood requirements	E	E
Produces highly reliable system	F	E
Produces expandable system	E	E
Manages risks	F	E
Can be constrained to pre-defined schedule	P	F
Has low overhead	F	F
Allows for mid-course corrections	E	F
Provides customer with progress visibility	E	E
Provides management with progress visibility	F	E
Requires little manager/developer sophistication	P	P

Adapted from S. McConnell, Rapid Development, Microsoft Press, 1997

Here's a scorecard for the prototyping and spiral life cycle models. The two prototyping models I discussed are lumped into one, since they have pretty much the same scores.

As you can see, both the prototyping and spiral models are useful when the requirements are not well-understood, and give the customer good visibility into project progress…but they do not work well for projects that have pre-defined schedule constraints, and they require quite a bit of management and developer sophistication.