

I494 Lecture Notes

Class 1 – September 1, 2009

Other than finishing up some of the course mechanics that were left unsaid on Monday, this course focuses on the breadth of development lifecycle strategies. The number of strategies in practice might be very, very large. However, we will look at a manageable number, and then discuss their relative strengths and weaknesses.

The specific models we will discuss are:

- 1) Classic Waterfall
- 2) Spiral
- 3) New Waterfall
- 4) Prototyping
- 5) Build it and Fix it
- 6) Staged Delivery
- 7) Evolutionary Delivery
- 8) Design to Schedule
- 9) Design to Tool
- 10) Buy off the shelf

Let's first review the basic building blocks of the classic waterfall:

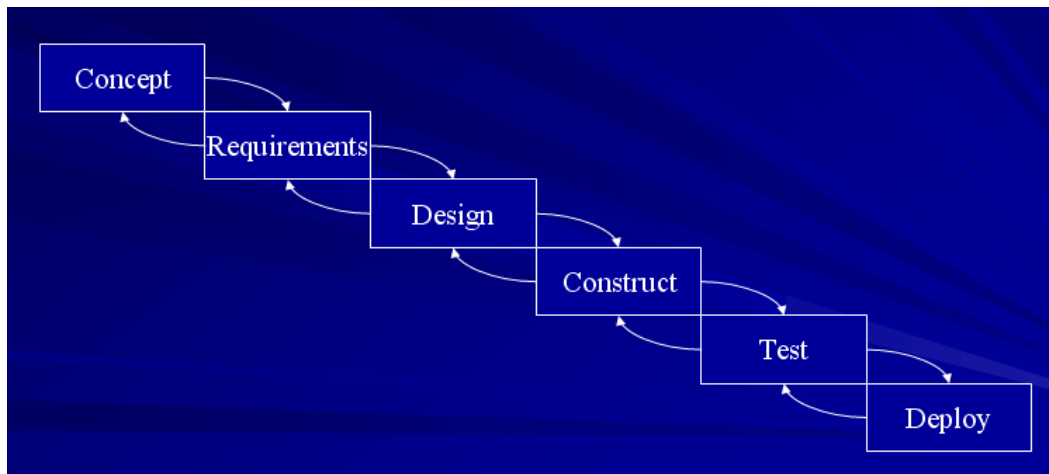


Figure 1: Classic waterfall development model.

Projects start with a concept, which may be initiated by many different constituents (another class). Once a concept is decided to pursue the requirements phase identifies “what” needs to be done. The design phase identifies “how” the solution is to be constructed to satisfy the requirements. Construction is construction. The test phase identifies whether the system operates according to specification. Lastly, the solution is deployed.

These basic building blocks are repeated in many of the other development models, just ordered, emphasized, or excluded in different ways. It is important that you understand what they mean. Sometimes we say these terms and it is assumed that the knowledge is tacit.

The classic waterfall is often criticized for the rigidity which is enforced. In practice the waterfall is usually implemented as the “new” or “modified” waterfall. Probably you can see for yourselves why the newer version is much more reasonable. That being said, the classic waterfall model does have some strengths. First, when you have no risk of changing requirements, or making mistakes, the model is fine. Second, it is a good model for an inexperienced person because it is so highly structured. The weaknesses include the high cost of fixing problems – swimming up the waterfall. The model may also lead to frustration, because there is a long time before what a user may perceive to be concrete deliverables start emerging.

The spiral model of development nibbles away at a problem. There are several decision-making steps, all aimed at eliminating risk, and leading up to a decision about what to do. Then, a waterfall development cycle occurs and the process repeats. The model has been shown to be effective at managing risk well at a reasonable cost. However, it really requires tremendous discipline to be successful.

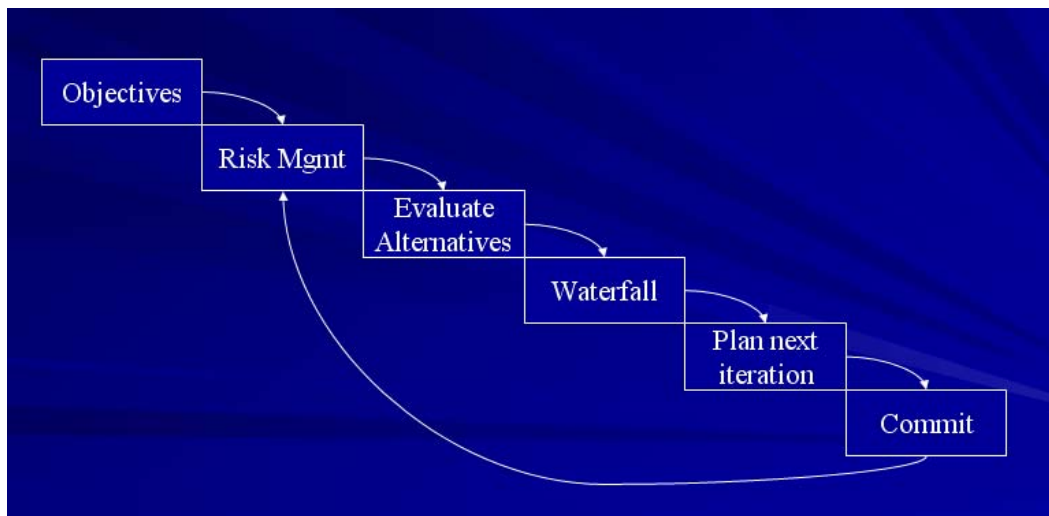


Figure 2: Spiral model of development.

The new, or modified, waterfall model breaks the discrete steps of the classic waterfall model, and allows the steps to overlap. By doing so, projects can start development of some portions of the required functionality before the entire project is designed. While this is certainly a benefit, the model does introduce risk that milestone deadlines become ambiguous. Also, there is a large risk of rework if the tasks are not ordered properly.

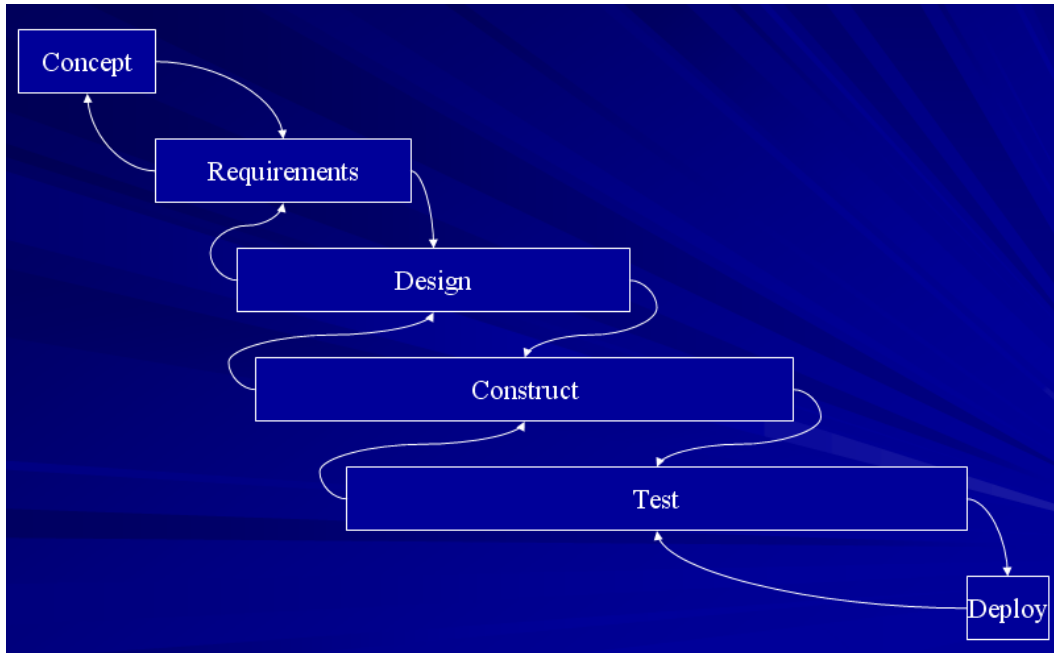


Figure 3: New waterfall model of development.

Prototypes play an important part of any development project. They reduce risk by providing semi-concrete examples of deliverables at early stages of the cycle. Ideally, users/customers will be able to more adequately articulate their needs by critiquing the prototypes. The evolutionary prototyping model is useful for refining designs. The result may be better projects, but the cost may be high because you don't know how much iteration there may be until the project is finished. Also, the focus may be over-emphasized on the user interface, or external aspects of a system, which may really represent very little of the overall effort. Remember that the closer a prototype appears to be a completed system, the more the user's expectation is that the system is really finished.

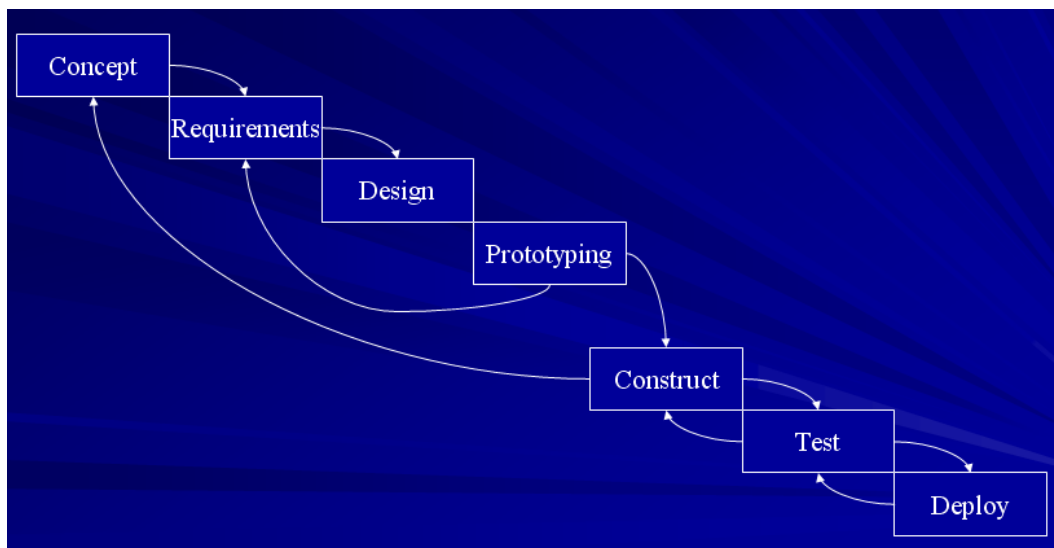


Figure 4: Prototyping model of development.

The other models of development include the build it and fix it model, which is clearly a very sloppy way of doing work. Probably, a lot of us do this with some of our homework assignments for other classes. There is an adage about quality:

“We never have time to build something right the first time, but we always seem to have time to build it again...”

The staged delivery model decomposes the project into functional pieces that can be delivered at different times. Though the decomposition may result in something very much like the spiral model, keep in mind that this is a fully planned project, and all of the pieces are known in advance. This model works well for managing critical delivery stages, and also balances available resources against available time. Care must be taken to ensure that the deliverables are done in the correct order.

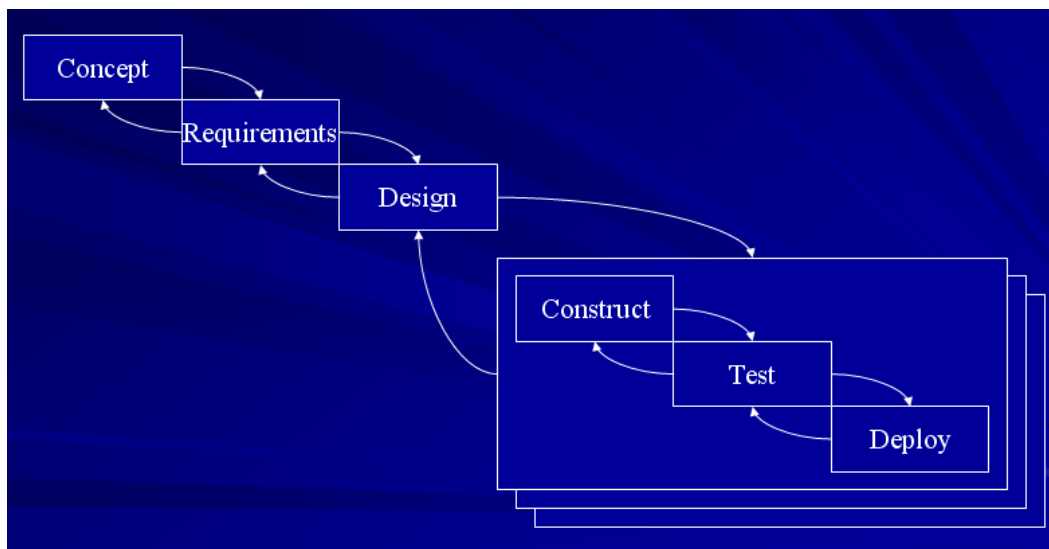


Figure 5: Staged delivery model of development.

The other models that you may encounter include an evolutionary delivery model, which introduces a prototyping loop into the staged delivery model. The “design to schedule” model proceeds according to a predetermined calendar of releases. Each release tries to fit the necessary functionality into the time available. The “design to tool” model places restrictions on the product that functionality that is not natively supported by a development tool is avoided. It is sometimes a stiff constraint, but at least it allows for better estimating of projects.

The final model of development is to buy the solution off the shelf. Sometimes this is referred to as “commercial off the shelf” or COTS. The clear benefit is time to deployment, but what are the risks? Also, how is this model different than the others?