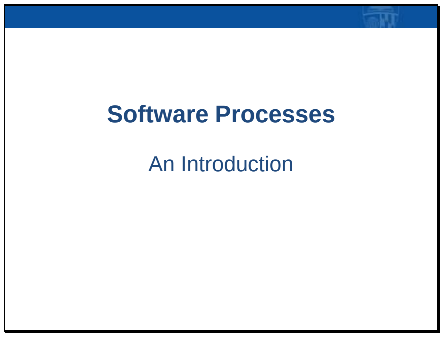
1

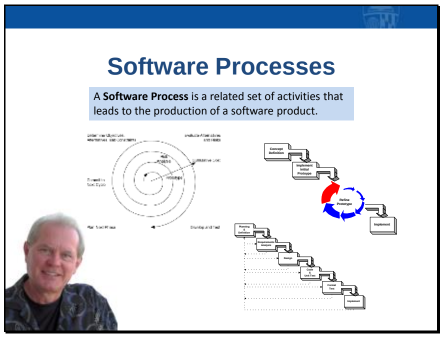


Software Processes

An Introduction

In this lecture we’ll discuss the importance of software process models.

2



Software Processes

A **Software Process** is a related set of activities that leads to the production of a software product.

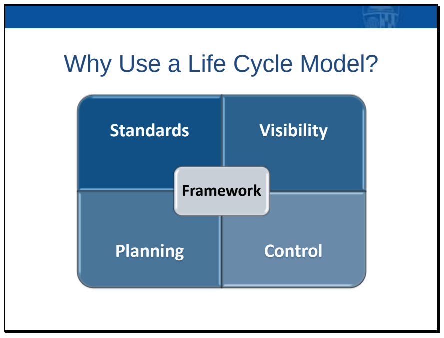
In this course module, we’re going to discuss software processes. So…what is a software process anyway? A software process…commonly called a software life cycle…is a set of related activities that results in the production of a software product.

Every software project goes through some type of process that results in the delivery of a software product. That process may be very well defined…or it may be rather chaotic. But…it will take place nevertheless.

In practice, there are many different kinds of software life cycles. In some life cycles, the progression of tasks may be essentially linear in nature, starting with a planning effort, then a determination of requirements, development of a design, development of the code, and so forth. In other life cycles, the activities may be more iterative in nature, with requirements design, coding, and test activities being performed multiple times. And some life cycles may include combinations of sequential and iterative activities.

In this lecture, I’ll talk about some of the general characteristics of software life cycles, and in subsequent lectures I’ll discuss some examples of specific life cycle models.

3



Why Use a Life Cycle Model?

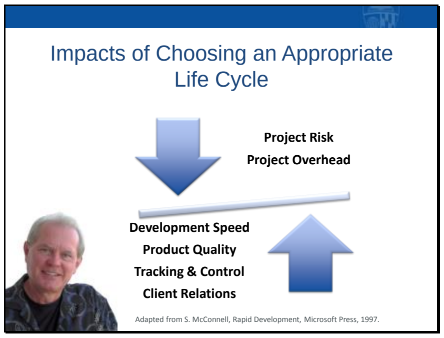
So why use a life cycle model in the first place? A project life cycle can be an extremely important tool for planning and managing a software project.

Used correctly, a life cycle model provides a framework or foundation for conducting all the activities associated with a project. It serves as a framework for establishing a standard set of project terminology, activities, and project deliverables…and specifies the basic order in which the project activities will take place and the project artifacts will be delivered. This is very important from management, project team, and stakeholder standpoints.

A life cycle can also increase the visibility of project progress to all project stakeholders.

The project life cycle should serve as the foundation for project planning, estimating, and scheduling…though, unfortunately, it sometimes isn't used that way. And…it should provide a mechanism for project tracking and control…though, again, it sometimes isn't used that way.

4



Impacts of Choosing an Appropriate Life Cycle

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

One of the decisions that needs to be made early on in a project is what type of life cycle model to use. Some organizations leave it up to the project manager to define the project life cycle that is used.

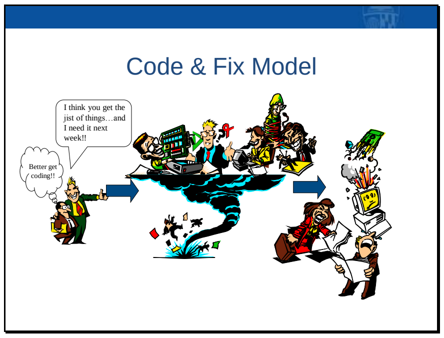
Some organizations have a single life cycle model that's used for all projects. In general, that may not be such a good idea unless the life cycle is appropriate for the majority of the organization's project portfolio.

And, some organizations have multiple life cycle models that are chosen based upon how best they fit the characteristics of a particular project. This has many potential benefits.

Choosing a life cycle that is appropriate for the specific project at hand has a number of potentially positive impacts…such as increased development speed, better product quality, improved project tracking and control, improved client relations, decreased project risk, and decreased project overhead.

On the other hand, choosing a life cycle that is not appropriate for the project at hand can significantly mitigate these benefits.

5


Code & Fix Model
Better get coding!!
I think you get the jist of things…and I need it next week!!

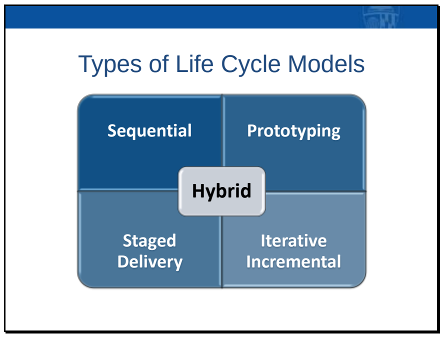
As I mentioned earlier, every software project uses some type of life cycle model.

The simplest life cycle model is often called the code and fix model. It's a model that is very common, but seldom useful.

In this model, you start with a general idea of the product to be built. That general idea can be informal and unwritten, or maybe consist of an informal list of things that stakeholders think might be necessary. Then, the development team progresses through a series of informal activities…usually dominated by coding and fixing…until the team has a product they think can be released. Successes are usually characterized by luck and heroic efforts, and are difficult or impossible to repeat on a regular basis.

Now…to be fair…this model does have some advantages. It's very simple, so there's no overhead involved in planning, documenting, and usually performing quality assurance activities. And…since you typically jump right into coding…there's a perception that you're showing very quick progress…though this perception is often misleading. For very small projects…maybe one-time uses, proofs of concept, or throw-away prototypes…the code and fix model might be okay.

For more typical projects, however, this model can be dangerous. It doesn't yield the benefits that were just presented…and is best avoided.

6 
Types of Life Cycle Models

For our discussion purposes in this course module, I've grouped some commonly used project life cycle models into a couple of different categories…sequential models, in which the project activities are performed in an essentially linear sequence…culminating in delivery of a product at the end of the process; prototyping models, which incorporate the use of one or more product prototypes; staged delivery models, which deliver a product in stages rather than all at once; and iterative/incremental models, which incorporate performing development activities iteratively, building the product in discrete increments.

All of these models can yield the benefits I listed earlier…if they are applied to the appropriate projects and executed correctly.
In the forthcoming lectures, I'll talk about specific examples of life cycle models in each of these categories…as well as some hybrid models.