Software Project Management 4th Edition

Chapter 3

Project evaluation & estimation

Introduction

- Evolutionary Process model
- Spiral model

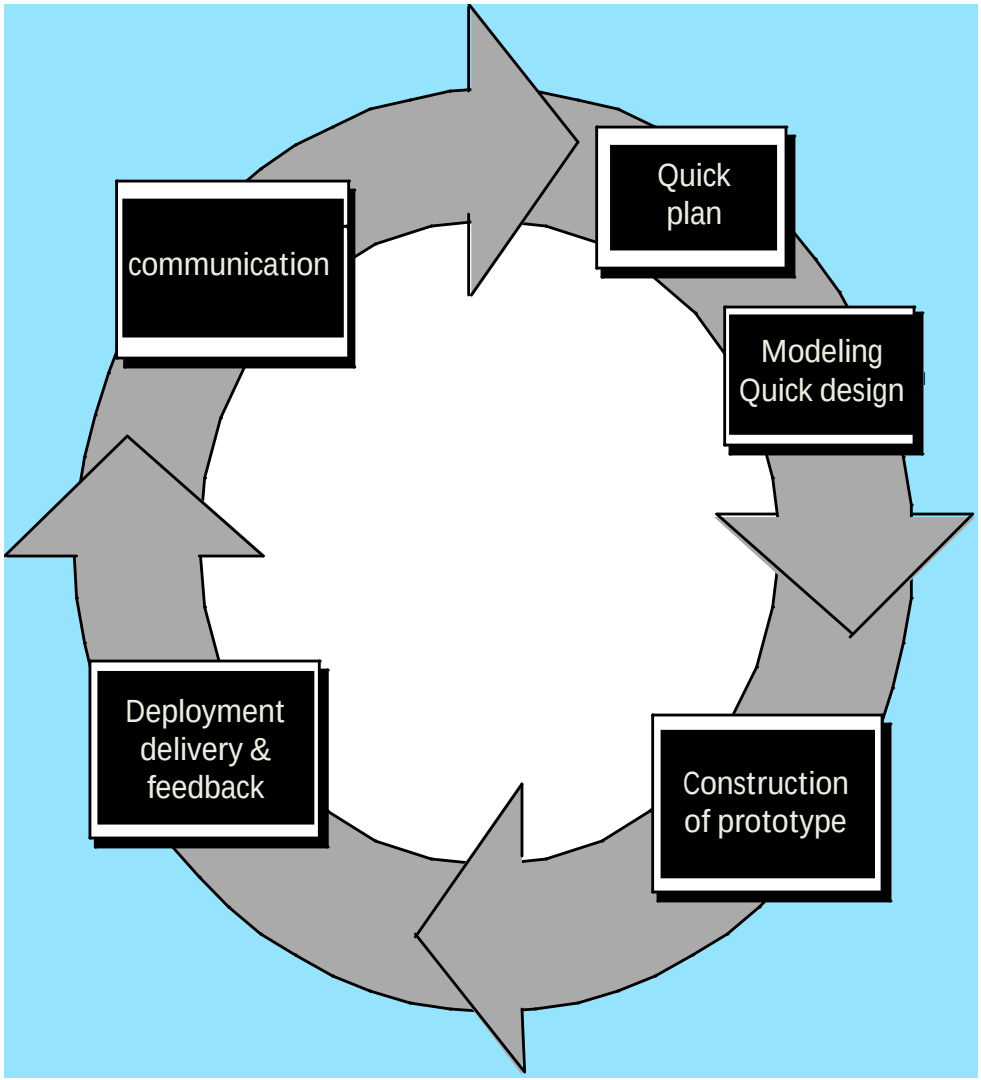
Evolutionary Process Models

Evolutionary Models are characterized in a manner that enables software engineers to develop increasingly more complete versions of the software.

Three Types are there :

- **Prototyping Model**
- **Spiral Model**
- Concurrent

Evolutionary Models: Prototyping



Prototyping Model

- The prototyping paradigm begins with communication.
- The software engineer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory.
- A prototyping iteration is planned quickly and **modeling** (in the form of a "**quick design**") occurs.
- The quick design focuses on a representation of those aspects of the software that will be visible to the customer/end-user (eg. Human interface layouts or output display formats).

- The quick design leads to the **construction** of a prototype. The prototype is **deployed** and then evaluated by the customer/end-user. ***Feedback*** is used to refine requirements for the software.
- **Iteration** occurs as the prototype is tuned to satisfy the needs of the customer. While at the same time enabling the developer to better understand what needs to be done.
- Ideally, the prototype serves as a mechanism for identifying software requirements. If a working prototype is built, the developer attempts to make use of existing program fragments or applies tools. (e.g. report generators, window managers, etc), that enable working programs to be generated quickly.

- It is true that both customers and developers like the prototyping paradigm. User get a feel for the actual system, and developers get to build something immediately.

Yet, prototyping can be problematic for the following reasons:

1. The customer sees what appears to be working version of the software, unaware that in the rush to get it working we haven't considered overall software quality or long-term maintainability. When informed that the product must be rebuilt so that high levels of quality can be maintained, the cries foul and demands that "a few fixes" be applied to make the prototype a working product. Too often, software development management give up.

2. The developer often makes implementation compromises in order to get a prototype working quickly. An inappropriate Operating system or programming language maybe used simply because it is available and known; an inefficient algorithm may be implemented simply to demonstrate capability. After a time, the developer may become comfortable with these choices and forget all the reasons why they were inappropriate. The less-then ideal choice has now become an integral part of the system.

- Although problems can occur, prototyping can be an effective paradigm for software engineering.
- The key is to define the rules of the game at the beginning; i.e. the customer and developer must both agree that the prototype is built to serve as a mechanism for defining requirements.
- It is then discarded (at least in part) and the actual software is engineered with an eye toward quality.

Doubts of Previous Session

- Drawbacks of RAD Model

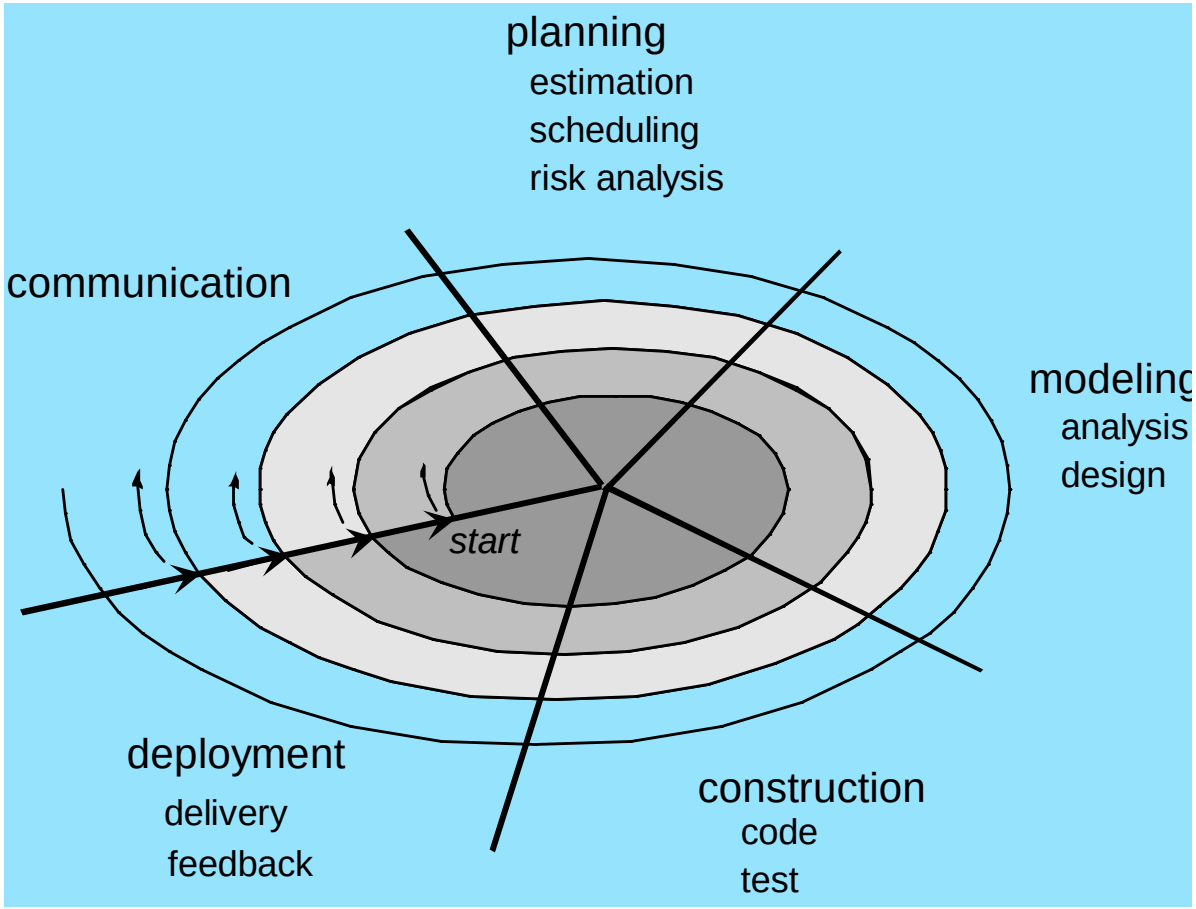
the RAD approach has drawbacks:

(1) For large, but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.

(2) If developers and customers are not committed to rapid fire activities necessary to complete the system in a much abbreviated (reduced) time frame, RAD projects will fail.

(3) RAD may not be appropriate when technical risks are high (eg. When a new application makes heavy use of a new technology)

Evolutionary Models: The Spiral



The Spiral Model

- The spiral model is an evolutionary process model that couples the iterative nature of prototyping with the systematic aspects of the waterfall model.
- It provides the potential for rapid development of increasingly more complete versions of the software. Using a spiral model, software is developed in a series of evolutionary releases.
- During early iterations, the release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered systems are produced.

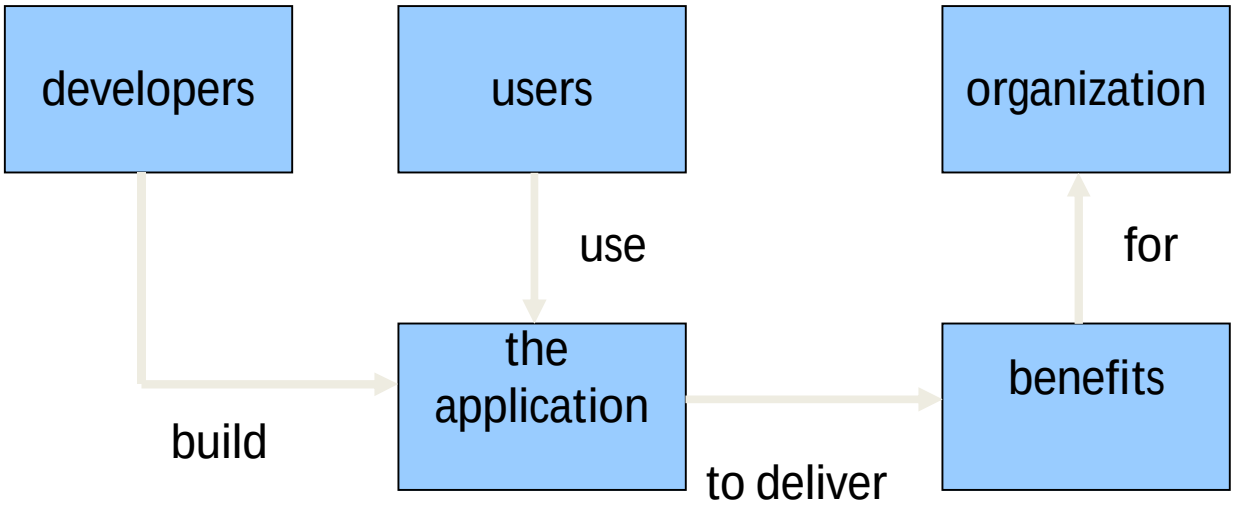
- A spiral model is divided into a set of framework activities defined by the software engineering team. Each of the framework activities represent one segment of the spiral path illustrated in the diagram.
- As this evolutionary process begins, the software team performs activities that are implied by circuit around the spiral in a clockwise direction, beginning at the center.

- The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software.
- Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback derived from the customer after delivery.
- In addition, the project manager adjusts the planned number of iterations required to complete the software.

- Unlike other process models that end when software is delivered, the spiral model can be adapted to apply throughout the life of the computer software. Therefore, the first circuit around the spiral might represent a “conceptual development project” which starts at the core of the spiral and continues for multiple iterations until concept development is complete.
- If the concept is to be developed into an actual product, the process proceeds outward on the spiral and a “new product development project” commences.
- The new product will evolve through a number of iterations around the spiral. Later a circuit around the spiral might be used to represent a “product enhancement project.” in essence the spiral, when characterized in this way, remains operative until the software is retired. There are times when the process is dormant (inactive), but whenever a change is initiated, the process starts at the appropriate entry point. (e.g. product enhancement).

- The spiral model is a realistic approach to the development of large scale systems and software. Because software evolves as the process progresses, the developer and customer better understand and react to risks at each evolutionary level.
- The spiral model uses prototyping as a risk reduction mechanism, but more importantly, enables the developer to apply the prototyping approach at any stage in the evolution of the product. It maintains the systematic stepwise approach suggested by the classic life cycle but incorporates it into an iterative framework that more realistically reflects the real world.
- The spiral model demands a direct consideration of technical risks at all stages of the project and, if properly applied, should reduce risks before they become problematic.

Benefits management



- Providing an organization with a capability does not guarantee that this will provide benefits envisaged – need for *benefits management*
- This has to be outside the project – project will have been completed
- Therefore done at *programme level*

Benefits management

To carry this out, you must:

- Define expected benefits
- Analyse balance between costs and benefits
- Plan how benefits will be achieved
- Allocate responsibilities for their achievement
- Monitor achievement of benefits

Benefits

These might include:

- Mandatory requirement
- Improved quality of service
- Increased productivity
- More motivated workforce
- Internal management benefits

Benefits - continued

- Risk reduction
- Economies
- Revenue enhancement/acceleration
- Strategic fit

Quantifying benefits

Benefits can be:

- Quantified and valued e.g. a reduction of *x* staff saving *£y*
- Quantified but not valued e.g. a decrease in customer complaints by *x%*
- Identified but not easily quantified – e.g. public approval for a organization in the locality where it is based

Application

- Risk reduction
- Economies
- Revenue enhancement/acceleration
- Strategic fit

Research

- **The Theater-Level Campaign Model**

A Research Prototype for a New Generation of Combat Analysis Model

Many analysts and decisionmakers argue that an order-of-magnitude leap forward in military modeling for the post-Cold War era — particularly campaign modeling — is essential to improve the quality of analyses, training, acquisition, test and evaluation, and innovative thinking. This research has been a step to ensure that the next-generation campaign models will not be mere rewrites of tools currently in use. The authors investigated alternatives to four aspects of modeling they think are essential to improving theater-level campaign analysis: (1) how to create more flexible structures to simulate the wide range of future scenarios and their associated uncertainties; (2) how to link to more detailed models in an analytically valid way; (3) how to represent ground forces maneuvering at the theater campaign level; and (4) how to represent adaptive behavior and aspects of command and control better in this type of model. This research provides insights into some of the alternatives and suggested some promising directions. The authors built the prototype Theater-Level Campaign (TLC) model and used it as a test bed for the different approaches. In many cases, methods were tried and then, finding they were not promising, that code was removed, and research started over in the true spirit of prototyping. The authors believe this type of prototyping and experimentation is critical to the advancement of the state of the art of campaign modeling and analysis. The various sections of the report describe the results associated with each aspect of the experimentation and conclude with more general observations and recommendations for the future.

Reference Link : http://www.rand.org/pubs/monograph_reports/MR388.html