**11.1 Typical Developer Motivations**

* *Compared to the general population,* developers **are much more motivated** by possibility for growth, personal life, opportunity for technical supervision, and interpersonal relations with their peers.
* *Compared to their managers,* developers **are somewhat more motivated** by possibility for growth, personal life, and technical supervision opportunity.

**Using the Top Five Motivation Factors**

Achievement

* Ownership
* Goal Setting

Possibility for Growth

Opportunity to focus on the work Itself

Personal Life

Technical-Supervision Opportunity

**Using other Motivation Factors**

Rewards and Incentives

Pilot projects

Performance Reviews

**11.4 Morale Killers**

Hygiene Factors

* Unrestricted access to a computer
* Appropriate lighting, heating and A/C
* Legal copies of all software used

**Other Morale Killers**

Management manipulation

Excessive schedule pressure

Lack of appreciation for development’s efforts

Inappropriate Involvement of technically inept management

Not involving developers in decisions that affect them

**12 Teamwork**

**12.1 Software Uses of Teamwork**

* Developing and reviewing the project’s requirements
* Developing the project’s architecture and the design guidelines that will be used by the whole project.
* Developing coding standards that will be used by the whole project.

**12.2 Teamwork’s Importance to Rapid Development**

* **Variations in Team Productivity**
* **Cohesiveness and Performance**

**12.3 Creating a High-Performance Team**

* High performance, jelled cohesive teams have:
  + A shared elevating vision or goal
  + A sense of team identity
  + A results-driven structure
  + Competent team members
  + A commitment to the team
  + Mutual trust
  + Small team size
  + Autonomy

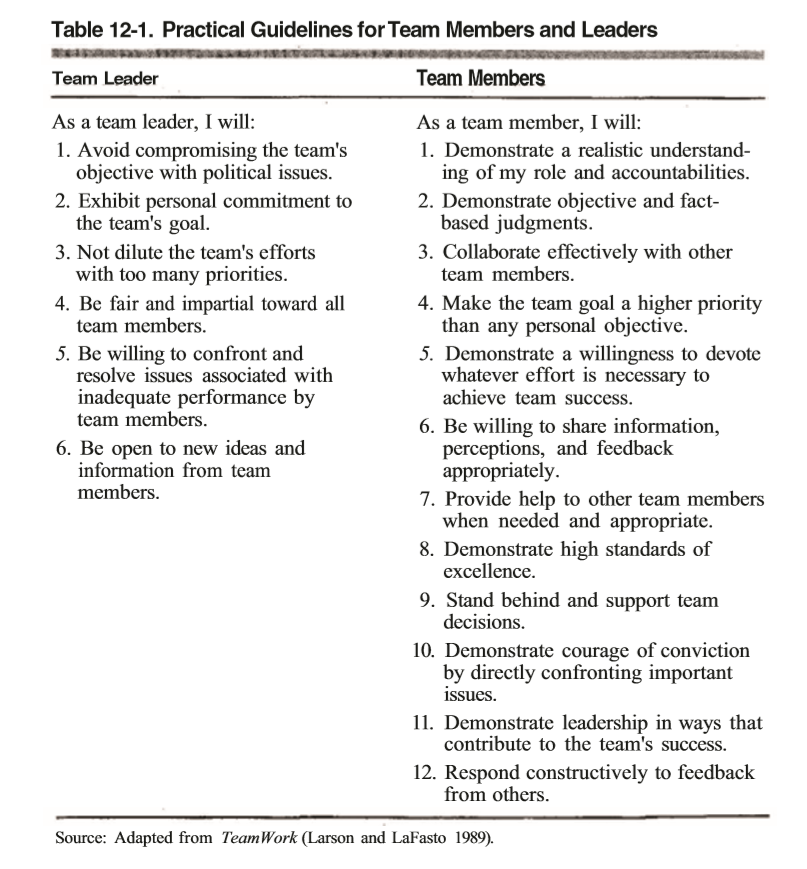
**12.4 Why Teams Fail**

* Lack of common vision
* Lack of identity
* Lack of recognition
* Productivity roadblocks.
* Ineffective communication
* Lack of trust.
* Problem personnel

**12.5 Long-Term Teambuilding**

* Lower startup costs.
* Lower risk of personnel problems.
* Less turnover
* The idleness question

**12.6 Summary of Teamwork Guidelines**



**13 Team Structure**

**13.1 Team-Structure Considerations**

* Problem resolution
* Creativity
* Tactical execution

**Kinds of teams**

* **Problem-resolution team**
* **Creativity team**
* **Tactical-execution team**

**Additional Team-Design Features**

* **Clear roles and accountabilities**
* **Monitoring of individual performance and providing feedback**
* **Effective communication**
* **Fact-based decision making**

**13.2 Team Models**

* Business Team
* Chief-Programmer Team
* Skunkworks Team
* Feature Team
* Search and rescue Team
* SWAT Team
* Professional Athletic Team
* Theater Team
* Large Teams

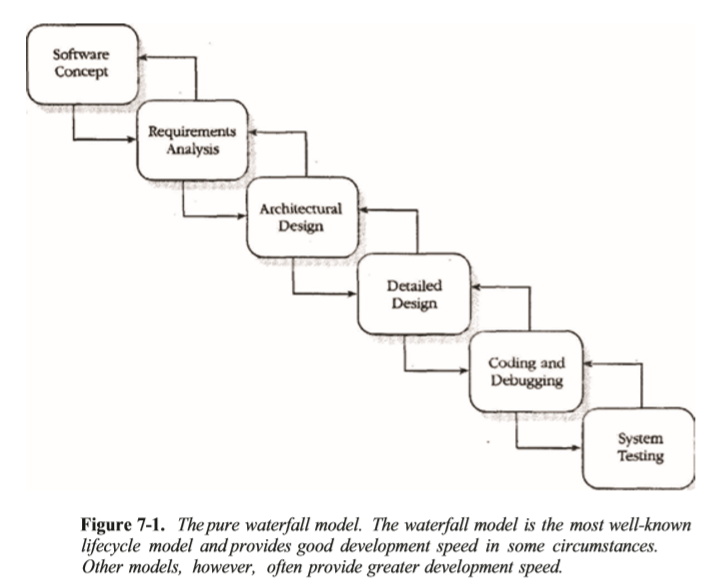
**13.3 Managers and Technical Leads**

Lifecycle Planning

7.1 Pure Waterfall

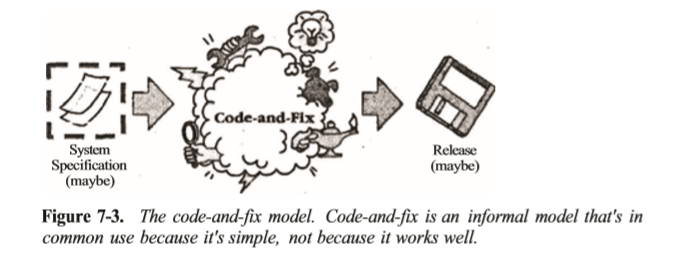
The waterfall model works well for projects that are well understood but complex, because you can benefit from tackling complexity in an orderly way.

The waterfall model works especially well if you have a technically weak staff or an inexperienced staff because it provides the project with a structure that helps to minimize wasted effort.



7.2 Code-and-Fix

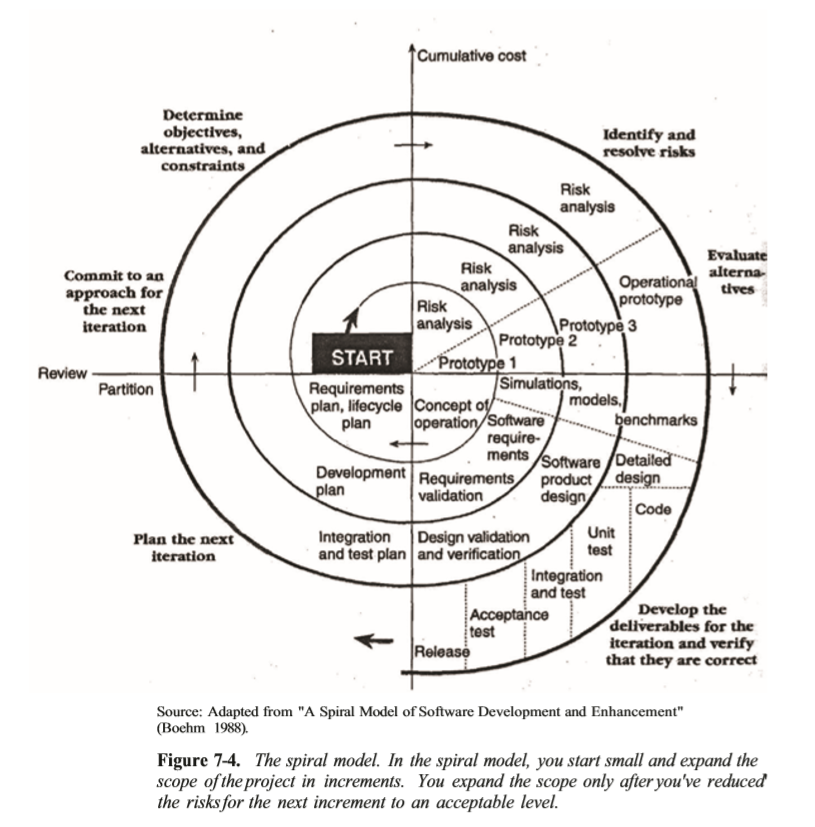
When you use the code-and-fix model, you start with a general idea of what you want to build. You might have a formal specification or you might not. You then use whatever combination of informal design, code, debug and test methodologies suits you until you have a product that’s ready to release.



For any kind of project other than a tiny project, this model is dangerous.

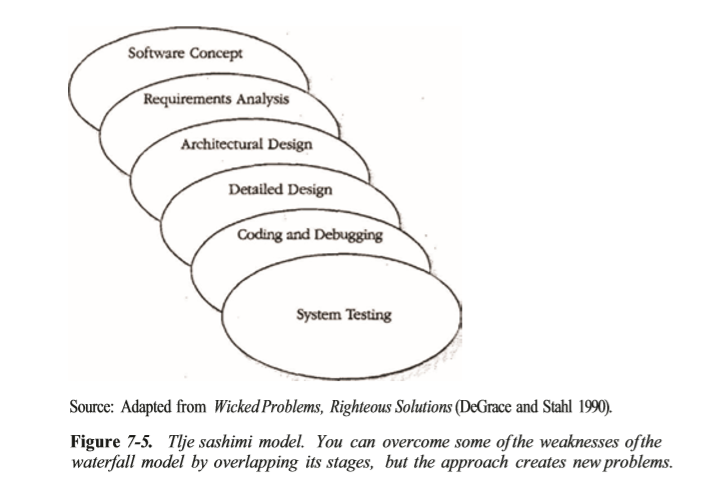
7.3 Spiral

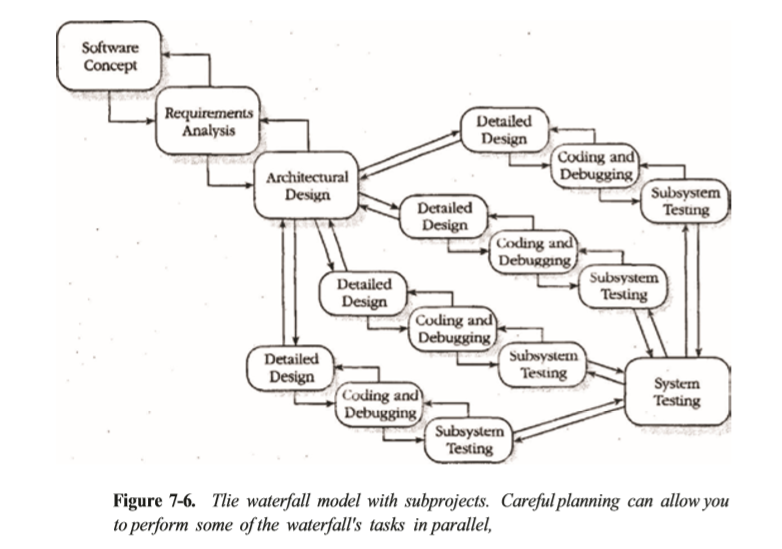
In the spiral mode, the early iterations are the cheapest. You spend less developing the concept of operation than you do developing the requirements, and less developing the requirements that you do developing the design, implementing the product, and testing it.

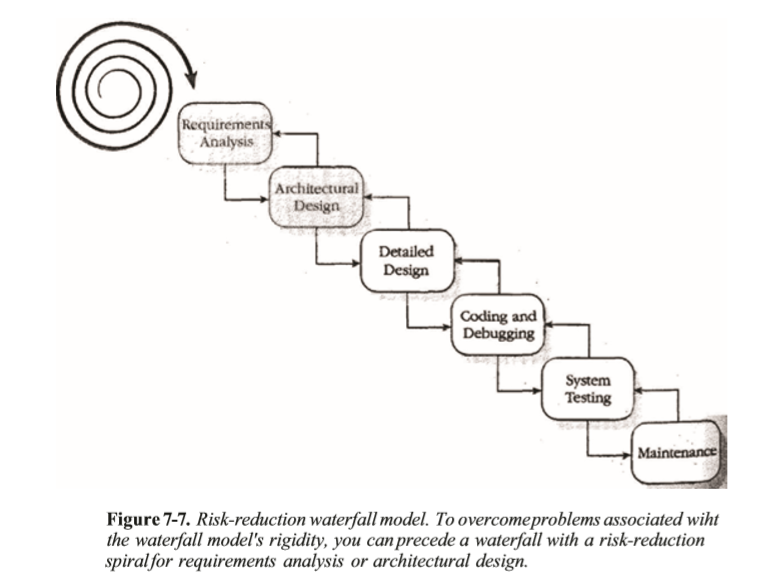


The spiral model provides at least as much management control as the traditional waterfall model.

7.4 Modified Waterfalls



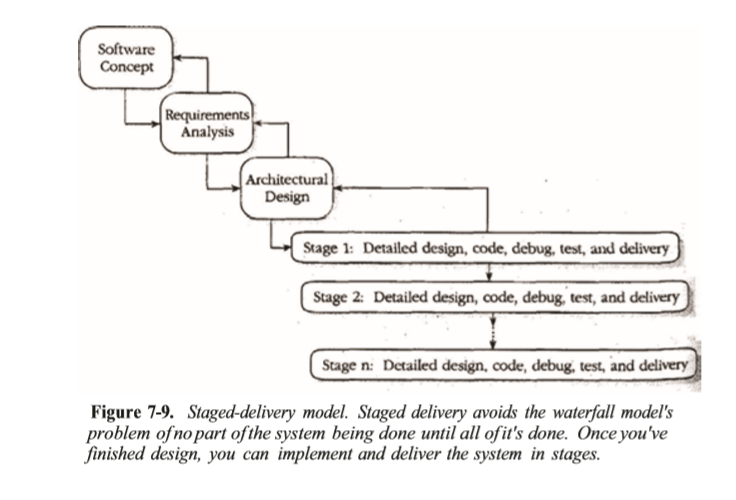




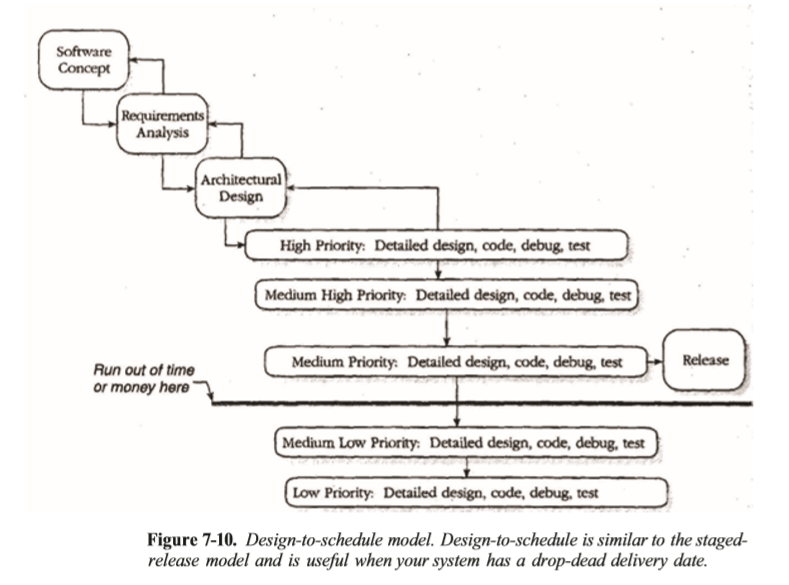
7.5 Evolutionary Prototyping

Evolutionary prototyping is especially useful when requirements are changing rapidly, when your customer is reluctant to commit to a set of requirements, or when neither you nor your customer understands the application area well

7.6 Staged Delivery

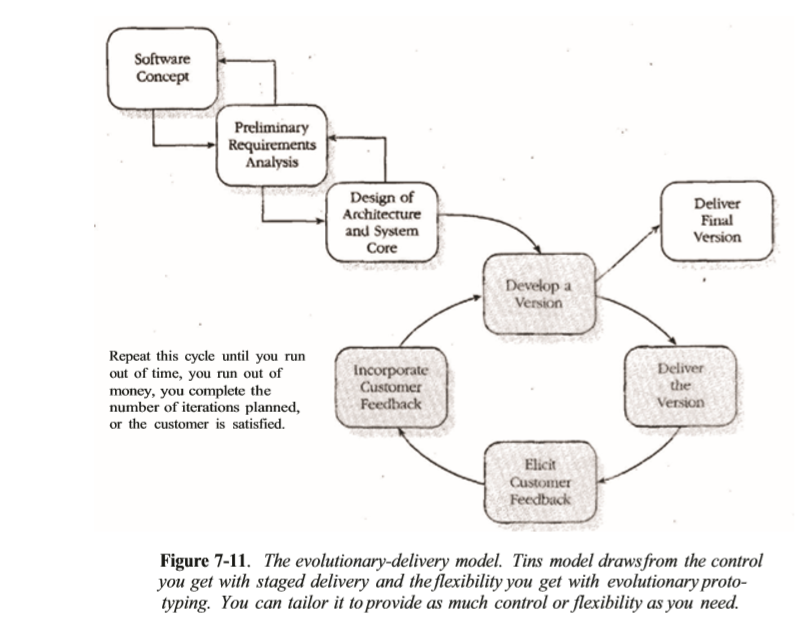


7.7 Design-to-schedule

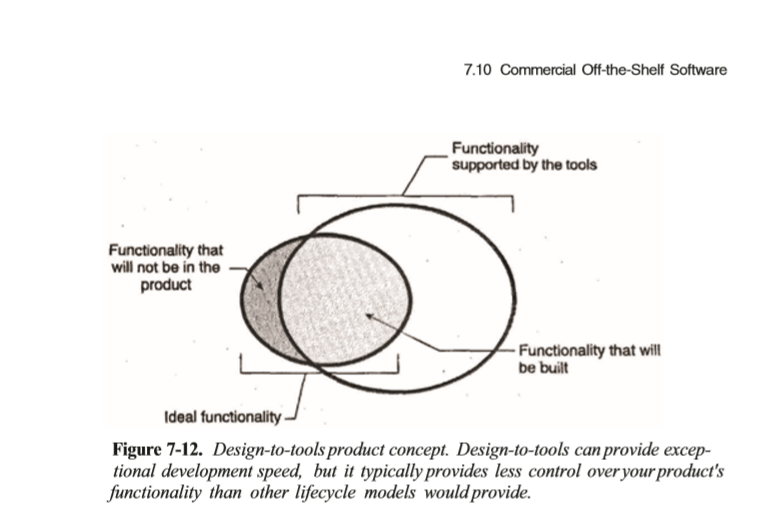


7.8 Evolutionary Delivery

You develop a version of your product, show it to your customer, and refine the product based on customer feedback.



7.9 Design-to-Tools



7.10 Commercial Off-the-Shelf Software

However, when you actually build your own software, you have to make design, cost, and schedule concessions, and the actual custom-built product will fall short of the ideal you envisioned.

7.11 Choosing the Most Rapid Lifecycle for Your Project

To choose the most effective lifecycle model for your project, examine your project and answer several questions:

• How well do my customer and I understand the requirements at the beginning of the project? Is our understanding likely to change significantly as we move through the project?

• How well do I understand the system architecture? Am I likely to need to make major architectural changes midway through the project?

• How much reliability do I need?

• How much do I need to plan ahead and design ahead during this project for future versions?

• How much risk does this project entail? • Am I constrained to a predefined schedule?

• Do I need to be able to make midcourse corrections?

• Do I need to provide my customers with visible progress throughout the project?

• Do I need to provide management with visible progress throughout the project?

• How much sophistication do I need to use this lifecycle model successfully?