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

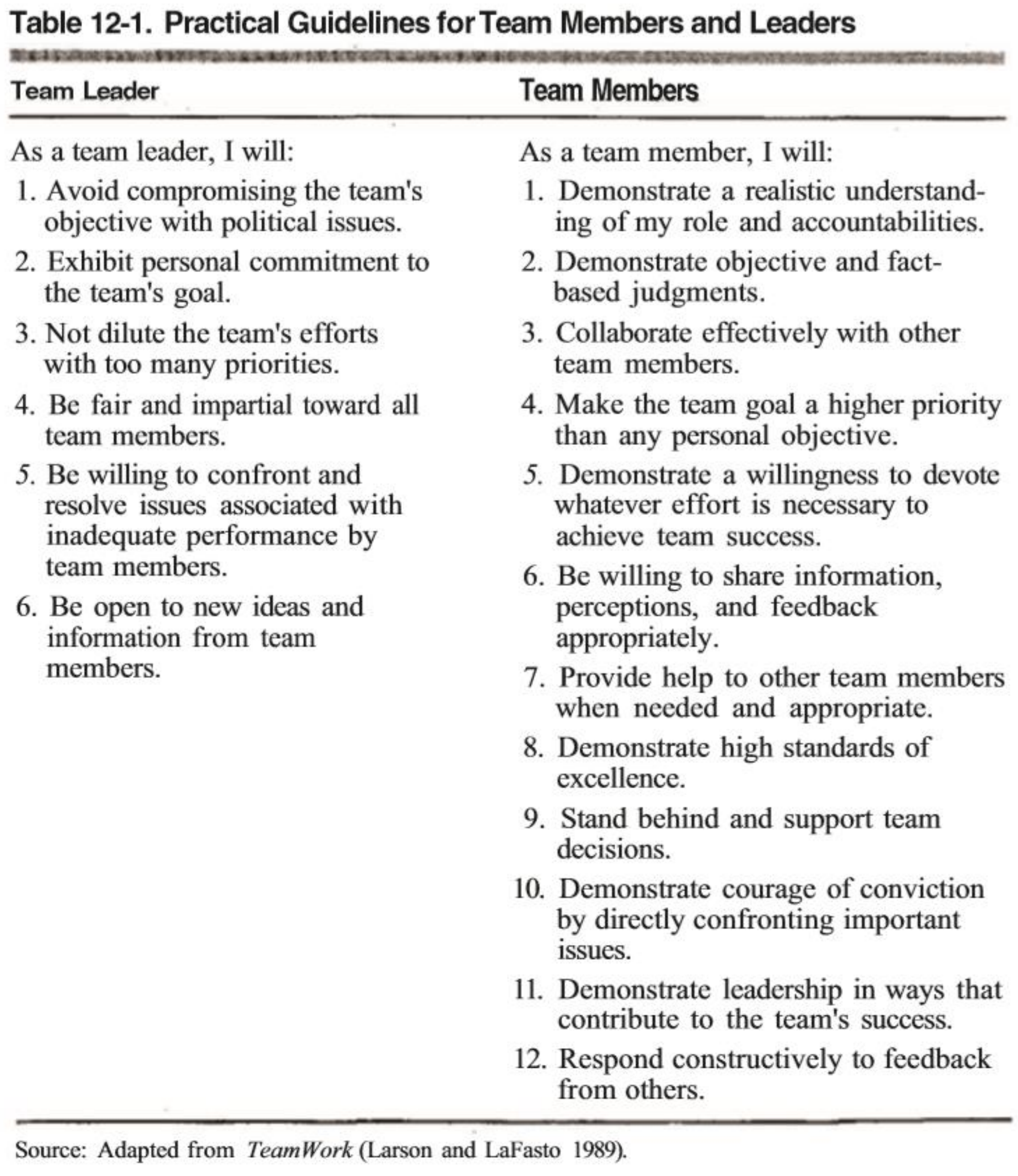


Table 12-1. Practical Guidelines for Team Members and Leaders

Team Leader	Team Members
As a team leader, I will:	As a team member, I will:
1. Avoid compromising the team's objective with political issues.	1. Demonstrate a realistic understanding of my role and accountabilities.
2. Exhibit personal commitment to the team's goal.	2. Demonstrate objective and fact-based judgments.
3. Not dilute the team's efforts with too many priorities.	3. Collaborate effectively with other team members.
4. Be fair and impartial toward all team members.	4. Make the team goal a higher priority than any personal objective.
5. Be willing to confront and resolve issues associated with inadequate performance by team members.	5. Demonstrate a willingness to devote whatever effort is necessary to achieve team success.
6. Be open to new ideas and information from team members.	6. Be willing to share information, perceptions, and feedback appropriately.
	7. Provide help to other team members when needed and appropriate.
	8. Demonstrate high standards of excellence.
	9. Stand behind and support team decisions.
	10. Demonstrate courage of conviction by directly confronting important issues.
	11. Demonstrate leadership in ways that contribute to the team's success.
	12. Respond constructively to feedback from others.

Source: Adapted from *TeamWork* (Larson and LaFasto 1989).

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.

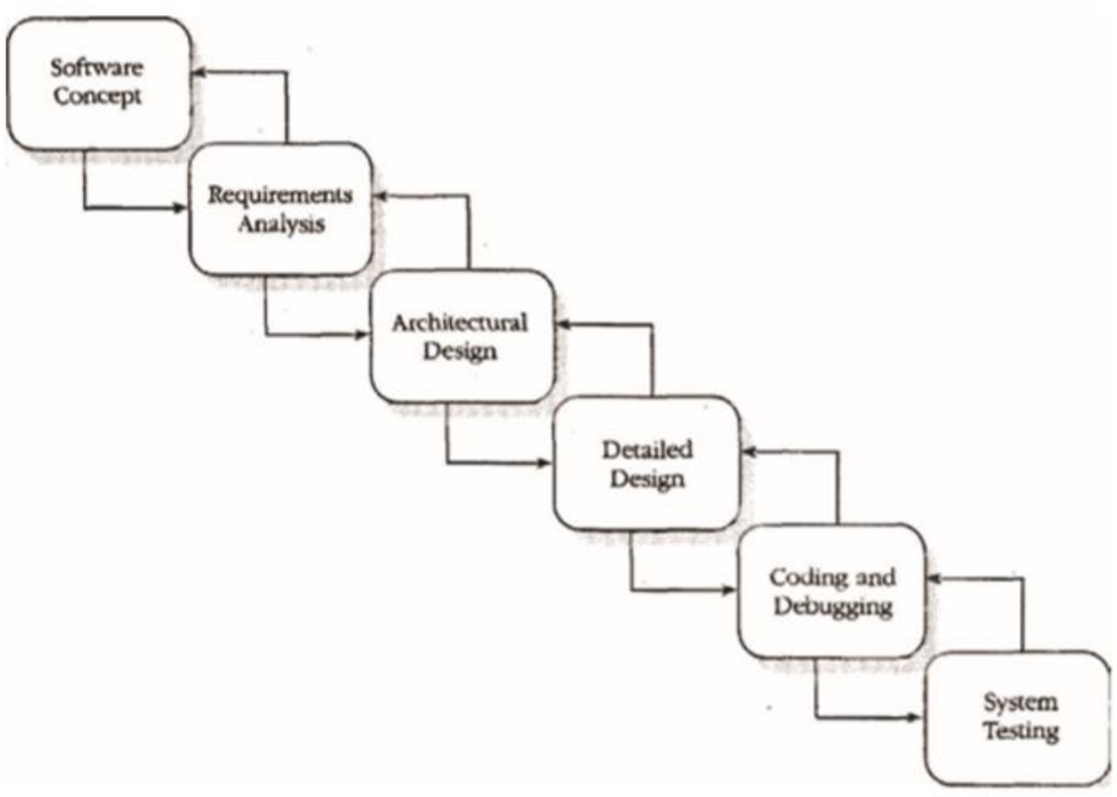


Figure 7-1. *The pure waterfall model. The waterfall model is the most well-known lifecycle model and provides good development speed in some circumstances. Other models, however, often provide greater development speed.*

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.

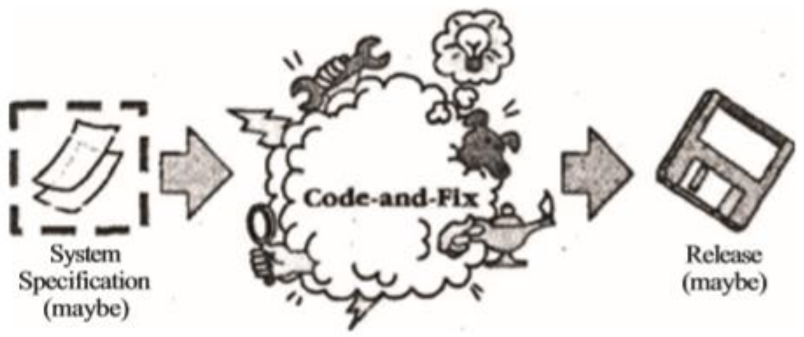
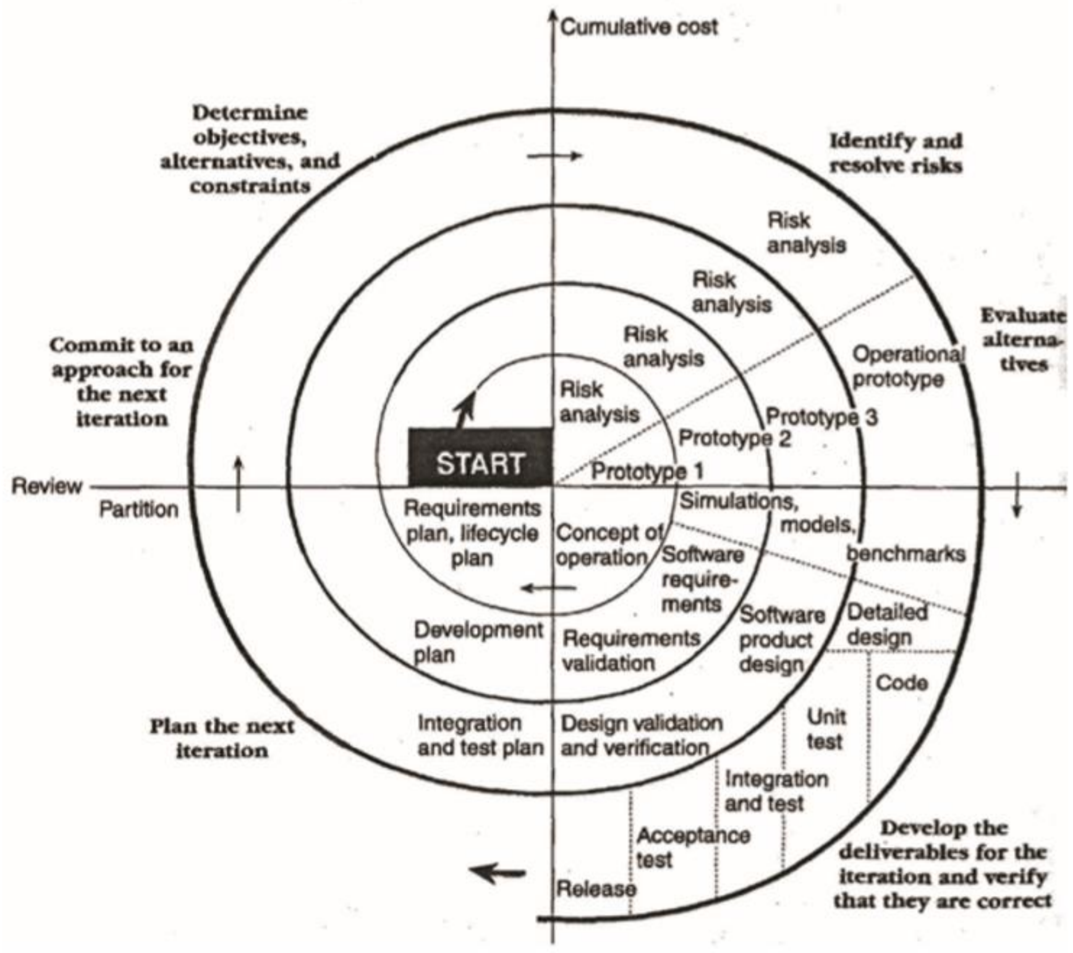


Figure 7-3. *The code-and-fix model. Code-and-fix is an informal model that's in common use because it's simple, not because it works well.*

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.

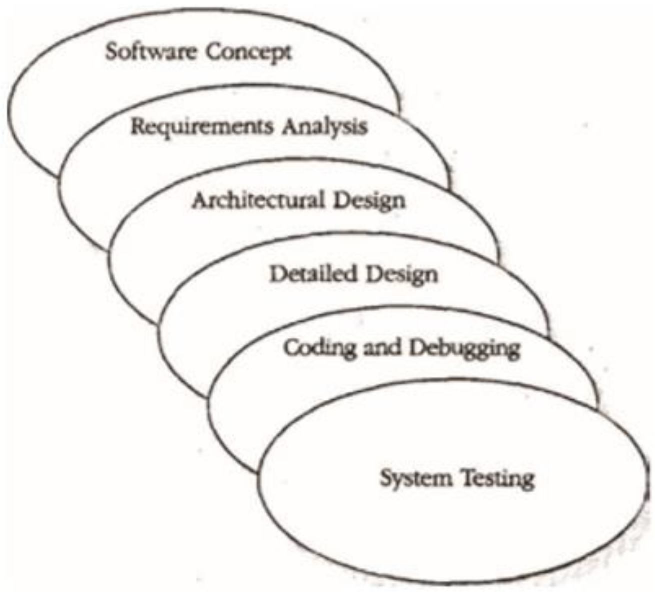


Source: Adapted from "A Spiral Model of Software Development and Enhancement" (Boehm 1988).

Figure 7-4. *The spiral model. In the spiral model, you start small and expand the scope of the project in increments. You expand the scope only after you've reduced the risks for the next increment to an acceptable level.*

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

7.4 Modified Waterfalls



Source: Adapted from *Wicked Problems, Righteous Solutions* (DeGrace and Stahl 1990).

Figure 7-5. *Tlje sashimi model. You can overcome some ofthe weaknesses ofthe waterfall model by overlapping its stages, but the approach creates new problems.*

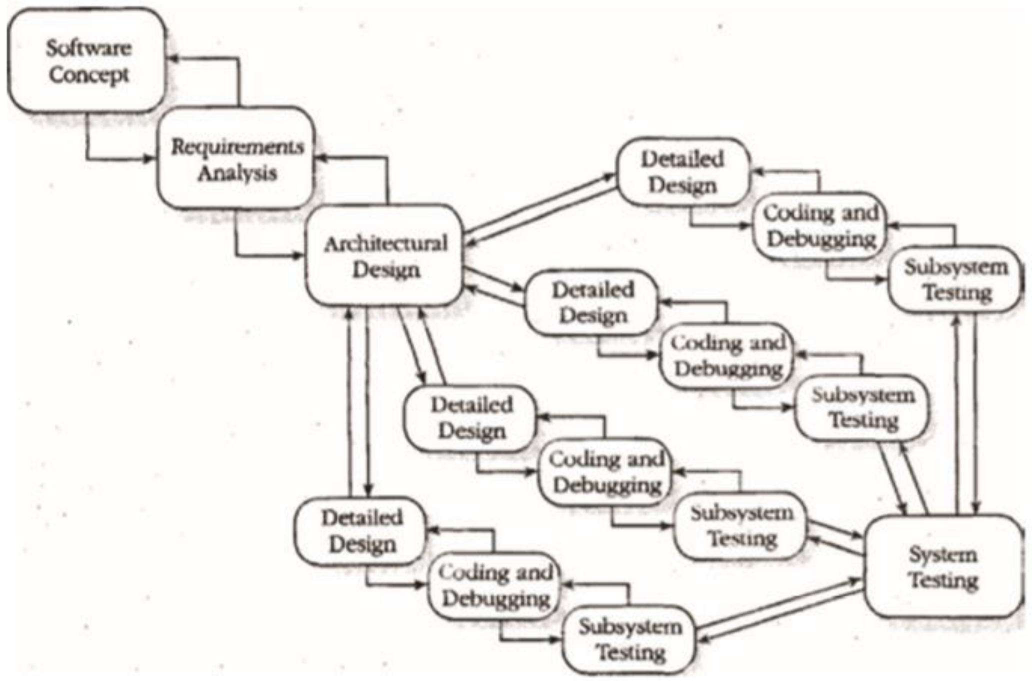


Figure 7-6. *Tlie waterfall model with subprojects. Careful planning can allow you to perform some of the waterfall's tasks in parallel,*

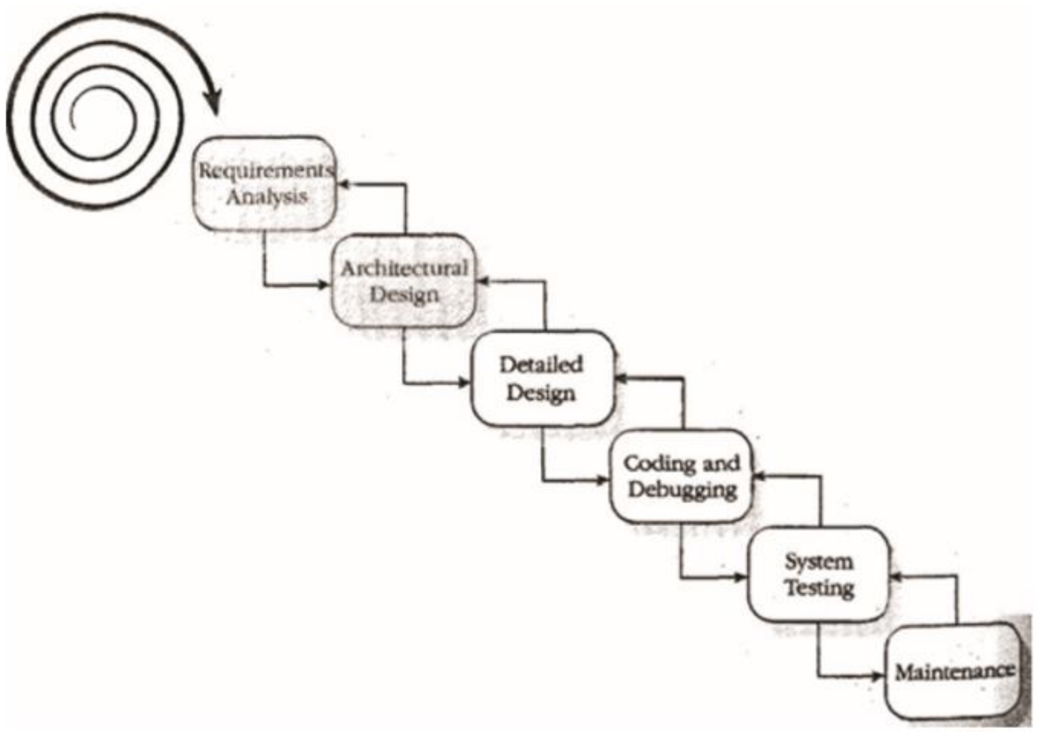


Figure 7-7. *Risk-reduction waterfall model. To overcomeproblems associated wiht the waterfall model's rigidity, you canprecede a waterfall with a risk-reduction spiralfor requirements analysis or architectural design.*

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

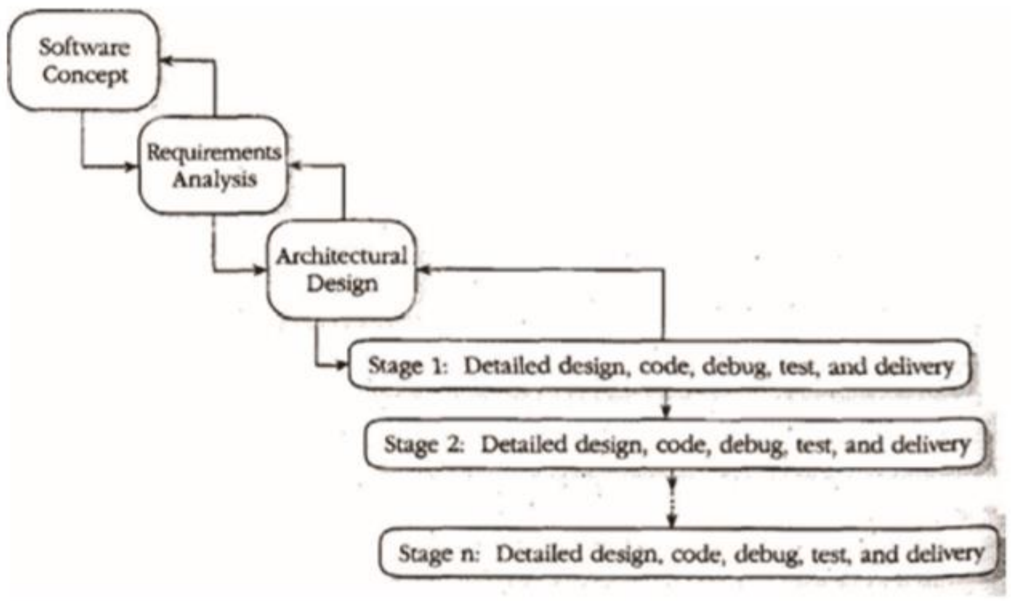

Software Concept
Requirements Analysis
Architectural Design
Stage 1: Detailed design, code, debug, test, and delivery
Stage 2: Detailed design, code, debug, test, and delivery
Stage n: Detailed design, code, debug, test, and delivery

Figure 7-9. *Staged-delivery model. Staged delivery avoids the waterfall model's problem of no part of the system being done until all of it's done. Once you've finished design, you can implement and deliver the system in stages.*

7.7 Design-to-schedule

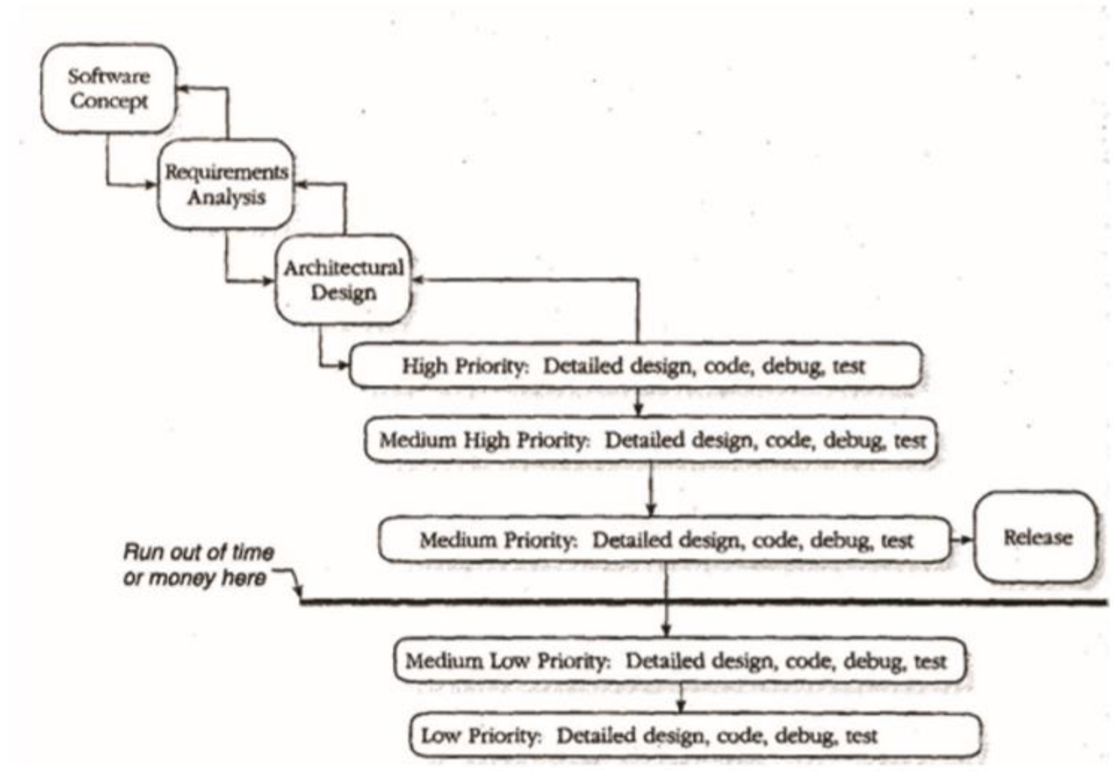

Software Concept
Requirements Analysis
Architectural Design
High Priority: Detailed design, code, debug, test
Medium High Priority: Detailed design, code, debug, test
Medium Priority: Detailed design, code, debug, test
Release
Run out of time or money here
Medium Low Priority: Detailed design, code, debug, test
Low Priority: Detailed design, code, debug, test

Figure 7-10. *Design-to-schedule model. Design-to-schedule is similar to the staged-release model and is useful when your system has a drop-dead delivery date.*

7.8 Evolutionary Delivery

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

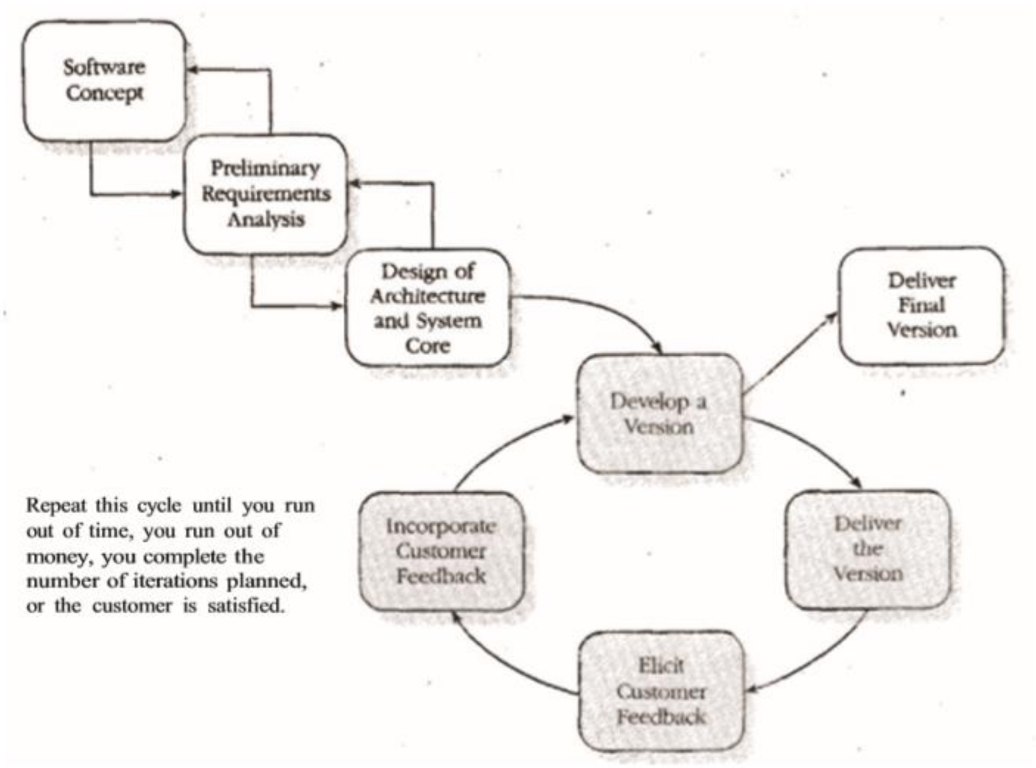

Software Concept
Preliminary Requirements Analysis
Design of Architecture and System Core
Deliver Final Version
Develop a Version
Repeat this cycle until you run out of time, you run out of money, you complete the number of iterations planned, or the customer is satisfied.
Incorporate Customer Feedback
Deliver the Version
Elicit Customer Feedback

Figure 7-11. *The evolutionary-delivery model. Tins model drawsfrom the control you get with staged delivery and the flexibility you get with evolutionary prototyping. You can tailor it to provide as much control or flexibility as you need.*

7.9 Design-to-Tools

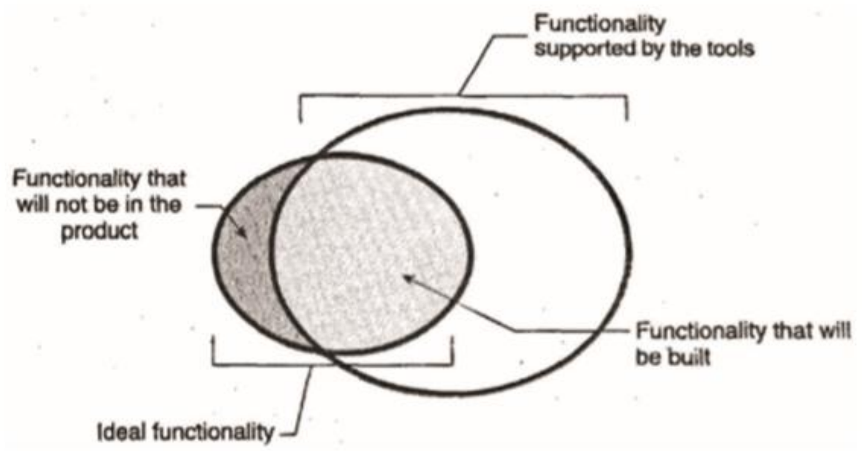

Functionality supported by the tools
Functionality that will not be in the product
Functionality that will be built
Ideal functionality

Figure 7-12. *Design-to-tools product concept. Design-to-tools can provide exceptional development speed, but it typically provides less control over your product's functionality than other lifecycle models would provide.*

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?