**Chapter 2: Software Engineering**

Software Engineering in Practice (pp 19-20)

1. Understand the Problem
   1. Who has a stake in the solution?
   2. What are the unknowns?
   3. Can the problem be compartmentalized?
   4. Can the problem be represented graphically?
2. Plan the Solution
   1. Have you seen similar problems before?
   2. Has a similar problem been solved?
   3. Can subproblems be defined?
   4. Can you represent a solution in a manner that leads to effective implementation?
3. Carry out the Plan
   1. Does the solution conform to the plan?
   2. Is each component provably correct?
4. Examine the Result
   1. Is it possible to test each component?
   2. Does the solution produce results that conform to the data, functions, and features that are required?

**Chapter 3: Software Process Structure**

Identifying a Task Set (p 34)

For a small, relatively simple project, the task set for requirements gathering might look like this:

1. Make a list of stakeholders for the project.
2. Invite all stakeholders to an informal meeting.
3. Ask each stakeholder to make a list of features and functions required.
4. Discuss requirements and build a final list.
5. Prioritize requirements.
6. Note areas of uncertainty.

For a larger, more complex software project…

1. Make a list of stakeholders for the project.
2. Interview each stakeholder separately to determine overall wants and needs.
3. Build a preliminary list of functions and features based on stakeholder input.
4. Schedule a series of facilitated application specification meetings.
5. Conduct meetings.
6. Produce informal user scenarios as part of each meeting.
7. Refine user scenarios based on stakeholder feedback.
8. Build a revised list of stakeholder requirements.
9. Use quality function deployment techniques to prioritize requirements.
10. Package requirements so that they can be delivered incrementally.
11. Note constraints and restrictions that will be placed on the system.
12. Discuss methods for validating the system.

**Chapter 4: Process Models**

The Waterfall Model (pp 42-43)

Among the problems that are sometimes encountered when the waterfall model is applied are:

1. Real projects rarely follow the sequential flow that the model proposes.
2. It is often difficult for the customer to state all requirements explicitly.
3. The customer must have patience. A working version of the program will not be available until late in the project time span.

Incremental Process Models (p 44)

When an incremental model is used, the first increment is often a core product. The core product is used by the customer (or undergoes a detailed evaluation). As a result, a plan is developed for the next increment.

Evolutionary Process Models (p 45-49)

The prototyping paradigm begins with communication. You meet with other stakeholders to define the overall objectives for the software. A quick design focuses on a presentation of those aspects of the software that will be visible to end users. The prototype is deployed and evaluated by stakeholders, who provide feedback that is used to further refine requirements.

…

Both stakeholders and software engineers like the prototyping paradigm. Users get a feel for the actual system, and developers get to build something immediately. Yet, prototyping can be problematic for the following reasons:

1. Stakeholders see what appears to be a working version of the software, unaware that the prototype is held together haphazardly.
2. As a software engineer, you often make implementation compromises in order to get a prototype working quickly… and the less-than-ideal choice (accidentally) becomes an integral part of the system.

…

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.