CHAPTER 2

SOFTWARE LIFE-CYCLE MODELS

The central idea of this chapter is the interplay between theoretical life-cycle models like the waterfall model and the rapid prototyping model and real-world life-cycle models like the iterative-and-incremental model. My objective in writing this chapter was to enable the student to select an appropriate life-cycle model for a given project.

The concepts of iteration and incrementation are so central to the object-oriented paradigm that it is absolutely essential for every student to understand both concepts before proceeding. If this means devoting an extra lecture to Chapter 2, so be it — there is no point in proceeding until the two concepts are thoroughly understood by the whole class.

I realize that Agile Processes (Section 2.9.5) are extremely controversial at present. I have included this section so that students will be introduced to the topic.

I have included the material on Microsoft's synchronize-and-stabilize model (Section 2.9.6) because Microsoft is currently the world's largest software organization, and virtually every student has had experience with a variety of Microsoft products. Also, many of my students have either done internships at Microsoft and/or want to work for Microsoft. As a result, the class is always extremely interested in this section.

The spiral model (Section 2.9.7) is still somewhat fashionable. As a result, it is being used in domains for which it is inapplicable. It is important for the students to know both when and when not to use each life-cycle model.

PROBLEM SOLUTIONS

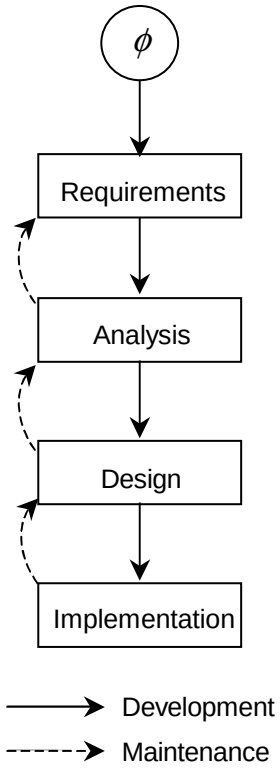

Figure 2.1. Waterfall model representation of the Winburg mini case study.

2.1: Figure 2.1 shows the waterfall model representation of the Winburg mini case study. (This figure is identical to Figure 2.3 of *Object-Oriented and Classical Software Engineering*, Seventh Edition.) The problem is that the figure does not show the sequence of events, that is, which artifact follows which. It is therefore far less effective than the evolution-tree model.

2.2: Episode 2 falls away, and Figure 2.2 shows the resulting evolution tree.

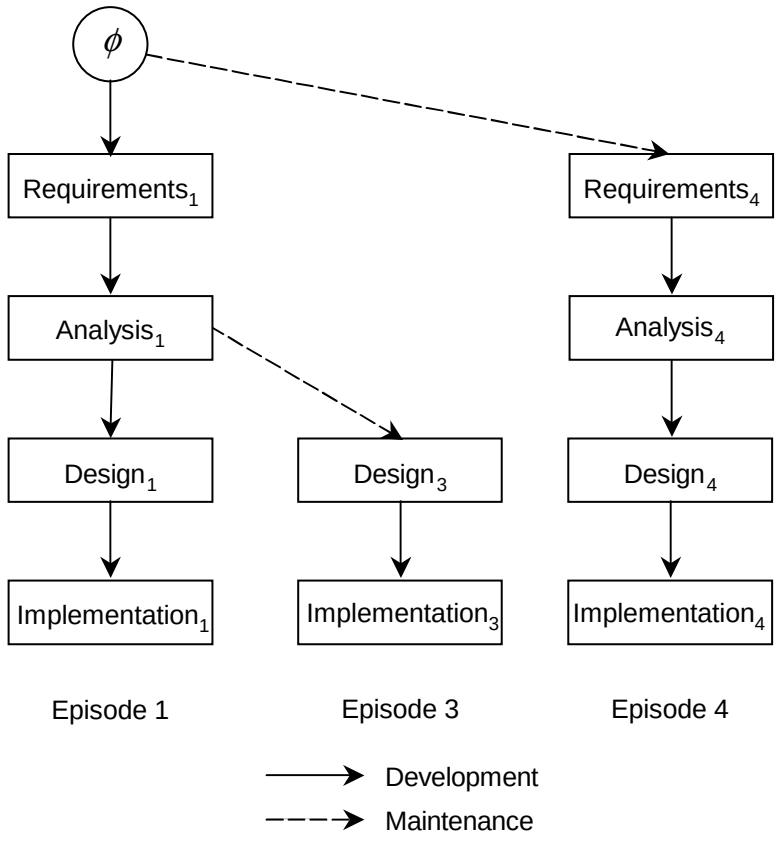

Figure 2.2. The evolution tree for the Winburg mini case study with single-precision numbers used from the beginning.

2.3: Because of Miller's Law we cannot develop a software product in a single step, and therefore we need to use stepwise refinement.

2.4: Incrementation.

2.5: A *workflow* is a set of activities.
An *artifact* is a constituent component of a software product.
A *workflow* creates or modifies one or more artifacts.
A *baseline* is a set of artifacts.

2.6: The iterative-and-incremental life-cycle model is equivalent to a sequence of waterfall life-cycle models.

2.7: Code-and-fix. There is nothing to be gained by using a more sophisticated model.

2.8: Experience and skills of the development team; computer literacy of the client; extent to which the client seems to appreciate his or her real needs.

2.9: The product may not be what the client really needs, so construct a rapid prototype. The design may not permit future development as the corporation grows or changes the way it does business, so ensure that the design is as open-ended as is reasonable. There may be cost and or time overruns, so estimate carefully (see Chapter 9). The users may not be comfortable with the product, so a rapid prototype of the user interface is needed; also, involve the purchasing clerks, factory supervisors, sales clerks, and so on, in the development loop. A competitor may produce off-the-shelf software before the product has been delivered — there is no ethical way to resolve this risk. A critical member of the development team may leave, so keep management of the development organization abreast of major decisions being made, thereby making it easier to integrate a replacement into the team. The development team may not be properly managed, so ensure that managers are competent and well-trained.

2.10: The key requirements here are achieve portability, maintainability, and generality. Excellent documentation is essential for this, so the waterfall model appears to be a strong candidate. However, the waterfall model is a theoretical model, so the iterative-and-incremental life-cycle model is probably the life-cycle model of choice here.

2.11: Infrastructure software that is likely to be extremely widely used by a broad variety of users. For example, an operating system or web browser.

2.12: Software to be used in only one organization. For example, a software product to control a machine of a unique type.

2.13: A small product, particularly one in which the requirements are vague.

2.14: A medium- or large-scale product.

2.15: A large-scale, in-house project. For example, a flight-control system that is developed in-house.

2.16: A small product, or any form of contract software. For example, a database conversion program developed under contract.

TERM PROJECT

2.17: The iterative-and-incremental life-cycle model should be used. First, it offers multiple opportunities for checking that the software product is correct. Second, the robustness of the underlying architecture can be determined relatively early in the life cycle. Third, risks can be mitigated early. Fourth, there is always a working version of the software.