APPENDIX A
The Laws of Software Design

This appendix summarizes all of the actual laws discussed in this book:

1. The purpose of software is *to help people*.
2. The Equation of Software Design:

   $$D = \frac{V_n + V_f}{E_i + E_m}$$

   where:

   D
   : Stands for the *desirability* of the change.

   V_n
   : Stands for *value now*.

   V_f
   : Stands for *future value*.

   E_i
   : Stands for the *effort of implementation*.

   E_m
   : Stands for the *effort of maintenance*.

   This is the primary law of software design. As time goes on, this equation reduces to:

   $$D = \frac{V_f}{E_m}$$

   Which demonstrates that *it is more important to reduce the effort of maintenance than it is to reduce the effort of implementation*.
3. The Law of Change: The longer your program exists, the more probable it is that any piece of it will have to change.

4. The Law of Defect Probability: The chance of introducing a defect into your program is proportional to the size of the changes you make to it.
5. The Law of Simplicity: The ease of maintenance of any piece of software is proportional to the simplicity of its individual pieces.
6. The Law of Testing: The degree to which you know how your software behaves is the degree to which you have accurately tested it.

That's it. Many more facts and ideas were discussed in this book, but these six items are the *laws* of software design. Note that of all of these, the most important to bear in mind are the purpose of software, the reduced form of the Equation of Software Design, and the Law of Simplicity.

If you wanted to sum up the most important facts to keep in mind about software design in two simple sentences, they would be:

- It is more important to reduce the effort of maintenance than it is to reduce the effort of implementation.
- The effort of maintenance is proportional to the complexity of the system.

Armed with only those two statements and an understanding of the purpose of software, you could very possibly re-evolve the entire science of software design, provided that you also understood that the complexity of the system actually comes from the complexity of its individual pieces.

APPENDIX B

Facts, Laws, Rules, and Definitions

This appendix lists every single major fact, law, rule, and definition covered in this book:

- *Fact*: The difference between a bad programmer and a good programmer is *understanding*. That is, bad programmers don't understand what they are doing, and good programmers do.
- *Rule*: A "good programmer" should do everything in his power to make what he writes as simple as possible *to other programmers*.
- *Definition*: A program is:
  1. A sequence of instructions given to the computer
  2. The actions taken by a computer as the result of being given instructions
- *Definition*: Anything that involves the architecture of your software system or the technical decisions you make while creating the system falls under the category of "software design."
- *Fact:* Everybody who writes software is a designer.
- *Rule:* Design is not a democracy. Decisions should be made by individuals.
- *Fact:* There are laws of software design, they can be known, and you can know them. They are eternal, unchanging, and fundamentally true, and they work.
- ***Law***: The purpose of software is *to help people*.
- *Fact*: The goals of software design are:
  1. To allow us to write software that is as helpful as possible
  2. To allow our software to continue to be as helpful as possible
  3. To design systems that can be created and maintained as easily as possible by their programmers, so that they can be—and continue to be—as helpful as possible

- ***Law***: The Equation of Software Design:

$$D = \frac{V_n + V_f}{E_i + E_m}$$

  This is the Primary Law of Software Design. Or, in English:

  > The desirability of a change is directly proportional to the value now plus the future value, and inversely proportional to the effort of implementation plus the effort of maintenance.

  As time goes on, this equation reduces to:

$$D = \frac{V_f}{E_m}$$

  Which demonstrates that it is more important to reduce the effort of maintenance than it is to reduce the effort of implementation.
- *Rule*: The quality level of your design should be proportional to the length of future time in which your system will continue to help people.
- *Rule*: There are some things about the future that you do not know.
- *Fact*: The most common and disastrous error that programmers make is predicting something about the future when in fact they cannot know.
- *Rule*: You are safest if you don't attempt to predict the future at all, and instead make all your design decisions based on immediately known present-time information.
- ***Law***: The Law of Change: The longer your program exists, the more probable it is that any piece of it will have to change.
- *Fact*: The three mistakes (called "the three flaws" in this book) that software designers are prone to making in coping with the Law of Change are:
  1. Writing code that isn't needed
  2. Not making the code easy to change
  3. Being too generic
- *Rule*: Don't write code until you actually need it, and remove any code that isn't being used.
- *Rule*: Code should be designed based on what you know now, not on what you think will happen in the future.
- *Fact*: When your design actually makes things more complex instead of simplifying things, you're overengineering.
- *Rule*: Be only as generic as you know you need to be right now.
- *Rule*: You can avoid the three flaws by doing incremental development and design.

- ***Law***: The Law of Defect Probability: The chance of introducing a defect into your program is proportional to the size of the changes you make to it.
- *Rule*: The best design is the one that allows for the most change in the environment with the least change in the software.
- *Rule*: Never "fix" anything unless it's a problem, and you have evidence showing that the problem really exists.
- *Rule*: In any particular system, any piece of information should, ideally, exist only once.
- ***Law***: The Law of Simplicity: The ease of maintenance of any piece of software is proportional to the simplicity of its individual pieces.
- *Fact*: Simplicity is relative.
- *Rule*: If you really want to succeed, it is best to be stupid, dumb simple.
- *Rule*: Be consistent.
- *Rule*: Readability of code depends primarily on how space is occupied by letters and symbols.
- *Rule*: Names should be long enough to fully communicate what something is or does without being so long that they become hard to read.
- *Rule*: Comments should explain *why* the code is doing something, not *what* it is doing.
- *Rule*: Simplicity requires design.
- *Rule*: You can create complexity by:
  - —Expanding the purpose of your software
  - —Adding programmers to the team
  - —Changing things that don't need to be changed
  - —Being locked into bad technologies
  - —Misunderstanding
  - —Poor design or no design
  - —Reinventing the wheel
  - —Violating the purpose of your software
- *Rule*: You can determine whether or not a technology is "bad" by looking at its survival potential, interoperability, and attention to quality.
- *Rule*: Often, if something is getting very complex, that means there is an error in the design somewhere below the level where the complexity appears.
- *Rule*: When presented with complexity, ask, "What problem are you trying to solve?"
- *Rule*: Most difficult design problems can be solved by simply drawing or writing them out on paper.

- *Rule*: To handle complexity in your system, redesign the individual pieces in small steps.
- *Fact*: The key question behind all valid simplifications is, "How could this be easier to deal with or more understandable?"
- *Rule*: If you run into an unfixable complexity outside of your program, put a wrapper around it that is simple for other programmers.
- *Rule*: Rewriting is acceptable only in a very limited set of situations.
- ***Law***: The Law of Testing: The degree to which you know how your software behaves is the degree to which you have accurately tested it.
- *Rule:* Unless you've tried it, you don't know that it works.