35 What's Next?

The objective of this chapter is to discuss what you should do next to further your understanding and experience in developing software.

35.1 Preconditions

The following should be true prior to starting this chapter.

- You have read relevant portions of this book based on your learning needs.

35.2 What Should You Do Now That You've Been Introduced to Software Design?

There are lots of ways the author can address this question. Below are a few ideas for you to ponder, and perhaps pursue.

35.2.1 Dive Deeper into Book Topics

You should continue to think about and apply concepts covered in this book. From a programming perspective, the three program design criteria and six software design criteria, listed below, should be strongly considered whenever you are developing a software solution. If you get nothing else from this book, hopefully these nine criteria will be an integral part of your software development practice.

D. P. Voorhees, *Guide to Efficient Software Design*, Texts in Computer Science,
https://doi.org/10.1007/978-3-030-28501-2_35

- Program design criteria
  - Separation of concerns
  - Design for reuse
  - Design only what is needed
- Software design criteria
  - Simplicity
  - Coupling
  - Cohesion
  - Information hiding
  - Performance
  - Security

Thinking more abstractly when developing software is critical for developers whom would like to participate in the early stages of a software development project. When the need for a large software system is being identified, early efforts focus on describing the scope of the project. A project scope often includes a description of the system and its subsystems. This top-down description may include design models to show both structure and behavior of the system. As discussed in this book, both the data-flow diagram and IDEF0 function model allow you to abstract away many details while describing the behavior and interactions of subsystems. In addition, when you know that the design and implementation will use the object-oriented paradigm, a UML package diagram may also be a useful design model to describe scope.

Using the design models discussed in this book will help you become a better software developer, regardless of the level of design detail you are describing. The list below identifies all of the design models discussed in this book. The first three design models in this list provide lots of flexibility in how you choose to use abstraction and in decomposing a large system into subsystems, components, and subcomponents.

- Data-flow diagram
- IDEF0 function model
- UML statechart
- Data modeling: entity–relationship diagram, logical data model, physical data model
- Object-oriented design models: UML class diagram, package diagram
- Structured design models: Hierarchy chart, structure chart, structure diagram

The five perspectives on software design introduced in Part III are each worthy of further exploration. While this book only has one chapter on quality assurance, this topic is critical in improving software regardless of the design perspective, application domain, or scope of software. Using the techniques described in Chap. 23 will help you create software with fewer defects and vulnerabilities. Go back and read this chapter again, start using these techniques, and convince your peers, colleagues, and project team members to use them!

All developers can improve their understanding and development of secure software. We should strive to develop more robust software; software capable of reacting in a secure way to expected and malicious use, and to accidental misuse. The 13 security design principles introduced in Chap. 24 represent a great start to creating more robust software. While applying the security design principles during design and implementation is critical, it is just as important to think about security while planning, gathering requirements, and testing.

Human–computer interaction design is a challenging and creative process often overlooked by developers focused on the more technical aspects of software development. Creating a great software product starts with the user interface. Developing a user-interface design that is useful, easy to learn, and allows the user to be productive should result in a software product people will want to use. Continuing to study and apply HCI topics allows one to blend the arts and sciences.

Persistently storing data is often a critical feature of software. The introduction to this design topic, including an introduction to XML and relational databases, is an area with lots of potential for further exploration. With XML, you could continue to learn about text-based XML files and start to learn about binary-based XML files. With relational databases, you could continue to learn about DDL and DML, logical and physical data modeling, or learn more about database administration.

The introduction to design patterns included in this book is just the beginning. If you are serious about becoming a better software designer, particularly using an object-oriented paradigm, you should consider diving much deeper into this topic.

35.2.2 Explore Other Design Topics

You should begin to learn and apply other design topics. These include

- Metrics
  Can we measure the quality of a software design? For example, can we describe the amount of coupling between design components using an easy to understand metric? When these measures exist, do they look only at program code or do they evaluate design models? Is a quantitative or qualitative measure more useful to us in describing the quality of a software design? Can any of the existing measures be automatically *computed* in a software development tool, regardless of whether the measure uses code or models?
- Layers of design abstraction
  What does it mean to be a software architect? How much abstraction is too much when creating a very high-level description of a software system? When creating a conceptual design, how much design is needed to convey intentions and to prevent misinterpretations? When creating a detailed design, at what point is the design too detailed? That is, when should someone stop creating a detailed design?

- Design within a software development processes (SDP)
  How does an organization maintain design artifacts for software that evolves over many decades? What are the benefits and limitations associated with maintaining design artifacts?
- Design within an agile software development processes (SDP)
  How is design done in an agile SDP? How much design should be documented while using an agile SDP while staying true to the agile manifesto? [1]

35.3 Post-conditions

The following should have been learned when completing this chapter.

- You have a deeper appreciation for software design and realize there are many ways to further your understanding of this complex subject area.

Reference

1. Beck K et al (2001) Manifesto for agile software development. https://agilemanifesto.org/. Accessed 30 Jul 2019