Decoding Software Design

By Blaine Willhoft and Rob Willhoft

Software Design Decoded: 66 Ways Experts Think by Marian Petre and André van der Hoek is a collection of short, one-paragraph observations on the habits that successful engineers (“experts”) have developed or learned over time. It doesn’t directly teach how to learn any of the skills it points out, so it isn’t an effective resource to pick up to directly grow as an engineer. However, it could be a valuable resource as a teaching tool or conversation starter when teaching more complex subjects such as working in teams and communicating with users. In this role, it could be used by educators or mentors to provide a framework to teach younger developers more quickly how to effectively contribute to mature, complex software projects.

Our review is a bit unusual in that it combines our perspectives as two people active in these two mentoring disciplines: one as a manager in the software engineering field (Blaine) and one as an educator of aspiring software engineers (Rob).

Blaine: Some of the early advice in the book encourages developers to first simplify by tackling the largest, most fundamental problems before moving onto increasingly granular problems afterward. When working with inexperienced developers, I find that they’re much more successful when tackling much smaller subsets of larger systems, but when called on to work on larger problems, they often struggle. In many cases, this is because they haven’t yet learned how to break down the problem into a series of subproblems. Learning this skill is also what enables senior engineers to estimate accurately, as they can more easily see the components, large and small, that make up a larger project.

It doesn’t directly teach how to learn any of the skills it points out, so it isn’t an effective resource to pick up to directly grow as an engineer. However, it could be a valuable resource as a teaching tool or conversation starter when teaching more complex subjects.

Rob: Students often struggle with the more abstract parts of problems. There’s a great desire to just jump in and start writing code. Early in the book, Petre and van der Hoek talk about designing abstractions. This short chapter can be a great encouragement to students to spend time on projects thinking deeply about the problem in terms that are separated from the code where they’re comfortable. Not many textbooks emphasize the importance of this step, which often seems like a waste of time to students. The book follows immediately with the importance of using metaphors to bring all members of the team into focus on the key design ideas.

Blaine: Another valuable takeaway from this book is its emphasis on dealing with uncertainty in the realm of software engineering. Many of the large software projects the engineers I mentor undertake are consulting projects that developed directly for a customer, so the ability to estimate and meet a plan are vital to the success of both the company and our clients. Uncertainty compromises the ability to estimate. Petre and van der Hoek accurately assess that there are essentially two kinds of uncertainty: uncertainty due to things you can’t foresee, and uncertainty due to knowledge you don’t (yet) have. The authors correctly assess that you must take steps as early as possible to address the latter problem by learning whatever you don’t yet know to enable the project’s success, whether that’s a new technology or simply old or complex code in your own system. The former problem, however, as the book points out, can’t be eliminated but must be acknowledged and planned around.

Rob: Most of us spend our entire careers learning how to deal with uncertainty. The authors give great advice to the beginning programmer on a very practical level to remove some of this uncertainty—in short, work with others. Much of the learning environment encourages students to be a shining star. But, as this little book is quick to point out, experts reach out and learn everything they can from others on the team. They’re quick to seek advice and discuss ideas with others on the team.

Blaine: The book does a fantastic job encouraging the soft skills that are often overlooked by young engineers. For instance, Petre and van der Hoek point out the value of user-focused development, from simply involving them to using prototypes to accurately assess what user interfaces will meet their needs and delight them. They also correctly point out that continual curiosity (play, as they call it) and the ability to know when to step away from a problem and do something else for a while are essential.

Rob: Two of my favorite sections talk about experts being skeptical and playing the fool. Some of the greatest advances in computer science have come from people who are unwilling to accept that the established best is really the best. It’s great that the authors talk about how you should be skeptical, not only of others’ ideas but also your own. You’ll have to read the book to find out about playing the fool.

Software Design Decoded is a fast read, and we both resonated with the topics it covered. It offers quick advice, grounded in real-world observations, that’s beneficial for developers of all experience levels to remember often.

Blaine Willhoft is an engineering manager at Vodori, a hybrid product and consulting firm focused on content and regulatory management for the life sciences and wellness industry. His passion is to use the perspectives he’s gained to enable teams to simultaneously foster their curiosity while also maintaining the highest level of productivity. Contact him at blaine.willhoft@gmail.com.

Rob Willhoft is a professor of computer science at Roberts Wesleyan College, where he teaches the capstone course in software development that all CS majors take. Willhoft has a PhD in computer engineering from Syracuse University. He’s a member of ACM. Contact him at Willhoft_Robert@roberts.edu.