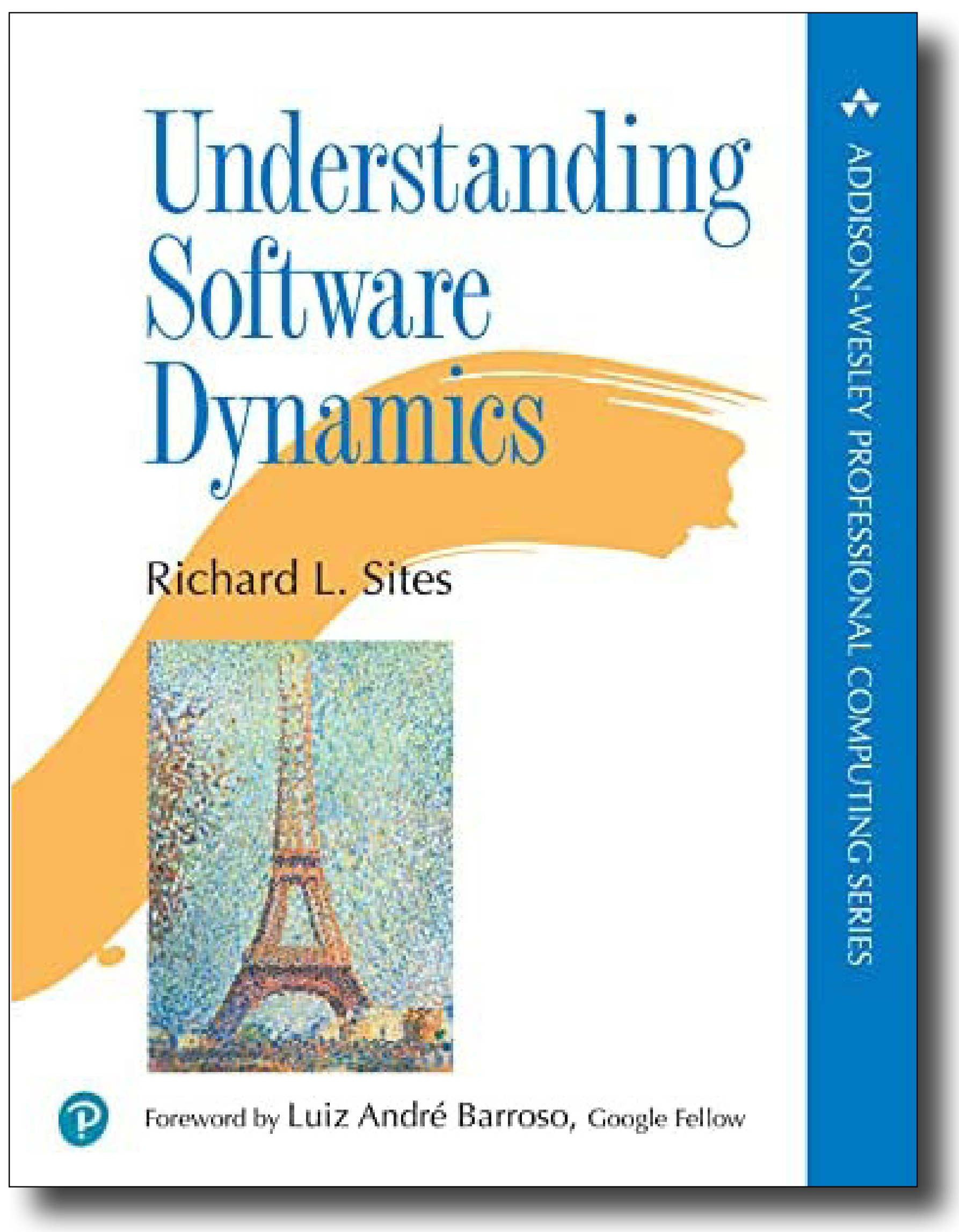
BOOK REVIEW

Understanding Software Dynamics by Richard L. Sites

REVIEW BY TOM JONES



Performance analysis is a difficult subject — a field where any claim must be backed up with a serious amount of rigor, detail of the work done, and where you must be very confident in your results.

In computer science we are almost dissuaded from worrying about performance issues. the frequently taught Knuth quote,

Premature optimization is the root of all evil

is used to keep junior developers on track to not worry too much about the small things.

Most of the time, this advice is given with good intentions. When you are learning, it is better to finish projects — to progress — rather than getting lost in minute details. But this has had the opposite effect of making information on improving performance of real systems hard to come by.

The full Knuth quote,

We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.
Yet, we should not pass up our opportunities in that critical 3%.
A good programmer will not be lulled into complacency by such reasoning, but he will be wise to look carefully at the critical code; but only after that code has been identified

tells us there are times where we should really care and our efforts spent on investigation will be rewarded.

There have been several performance books written in the last few years that I hold in high regard and have recommended many times. Brendan Gregg's most recent books *Systems Performance* (which acts to replace the long favored DTrace performance book) and *BPF Performance Tools* are well known and come from one of the world's leading experts in investigating and designing tools to better understand real-world, system performance. The lesser known, but excellent (and open source) *Performance Analysis and Tuning on Modern CPUs* by Denis Bakhvalov, dives deeper into why CPUs are slow and how computers can the best make use of them.

The Gregg books act as excellent introductory material for the performance expert to be. They offer a high-level view of looking at systems and using tools. *BFP Performance Tools and Systems Performance* offer methodology for the performance analyst and very high-quality reference material that should be your first port of call when trying to debug an issue. With these two books and reading the source for the accompanying tools, you can learn how to build your own performance tools, but the books don't go into depth on what makes a good performance analysis tool.

Bakhvalov's *Performance Analysis and Tuning on Modern CPUs* shows how the perf set of tools can be used on Linux to explore programs. It introduces the core fundamentals a developer would need to know to understand the output of perf and how different factors influence software performance as analyzed with perf. The book is — again — a great introduction, but lacks solid examples (which are available as free course material from the author) and lacks the next steps for the developers who need their own custom tools to debug their problems.

Understanding Software Dynamics (USD) by Richard L. Sites stands out from the other recent works. It contains introductory material into how computers work and the factors that determine their performance, and it expands on this by practically describing the limitations that performance tools must meet to be useful to use on real systems.

The author is a veteran of the computer industry. He developed the first CPU performance instructions while at DEC and has had a long career investigating performance issues from CPU level bottlenecks up to entire data center wide application stacks while working with Google and Tesla.

USD acts as a practical introduction to exploring the performance of large systems as their dynamics change. In large systems, there are a lot of transactions moving through at a time — the performance of individual transactions is normally inconsequential. *USD* looks at the distribution of these transactions. Problems live with the outliers: Total systems performance is made of both the fast and the slow transactions and *USD* offers guidance to discover what the best- and worst-case performance should be in a system and how to explain what is happening when the 90th percentile is outside these bounds.

Understanding Software Dynamics (USD) by Richard L. Sites stands out from the other recent works.

USD follows a data center RPC system as its core example and builds tooling to discover where performance issues can appear. The first 7 chapters look at measurement and how different system components contribute to final results. Here *USD* digs into very fine detail, offering guidance to understand why CPU instructions, memory, disk or network access might be the cause of performance bottlenecks in a system.

Throughout these explorations, the lesson is that looking at individual subsystems is not a good model for thinking about the performance of the entire system. Sites shows his thinking for establishing what the best- and worst-case performance of any computer subsystem should be and demonstrates how to test these estimates in the real world.

The second part of *USD* covers how to observe and measure the performance of real systems while maintaining acceptable levels of overhead. In this part, the reader learns different ways that the tools we use today are implemented and the opportunities for observation they provide. Here we are introduced to the design criteria for observation tools.

The third part builds on the lessons of the first two introductory parts of the book and shows a real implementation of the tracing ideas by introducing the KUTrace framework kernel — userspace tracing framework.

KUTrace is an example of a high bandwidth logging framework and has to be implemented with low overhead. It offers an extra source of information for debugging overly long running transactions.

KUTrace is a patch set for Linux that adds a framework for adding small ~64byte log entries to points in the kernel and userspace. The tracepoints are `__predict_false` branches by default. Once the KUTrace kernel module is loaded, they become active and start being saved into a buffer in the kernel.

USD goes into detail explaining the design of the log entries that KUTrace uses, the system interface and the operation of the kernel module.

Sites rounds out the text with 9 final chapters, each of which practically walks through a problem in one of the performance domains that book has examined. There is an example problem for a case of too much CPU execution, executing slowly and waiting for the CPU, Memory, Disk, Network, Locks, Time, and queues. Each of these chapters is a lesson acting on information presented earlier in the book.

Understanding Software Dynamics is an excellent addition to the library for anyone interested in a practical understanding of performance issues in real systems. It does a great job of introducing required background understanding without going into such depth that it is difficult to follow. Several chapters come with examples that help reinforce and expand on the lessons provided in the preceding text and they are focused on improving understanding.

It does a great job of introducing required background understanding without going into such depth that it is difficult to follow.

Sites is clear in his explanation throughout, there are high quality figures taken from real systems at Google and from the example systems. Added color comes from the author's own experiences working on performance.

USD is a well written and carefully constructed book. It supports itself well and is approachable for a reader who hasn't previously delved into performance analysis. For a reader with a lot of experience, there is still much information to learn from here about the specifics of performance of different computer components and the new tooling the author introduces.

TOM JONES, FreeBSD Developer and co-host of the BSDNow Podcast, wants FreeBSD-based projects to get the attention they deserve. He lives in the North East of Scotland and offers FreeBSD consulting.