Mind Mapping Your Way to Better Code

A simple tool delivers big benefits in design and implementation of code, especially on personal projects.

by Andrew Binstock

January 10, 2020

For much of the last year, I have been discussing tools and techniques to make coding clearer, more reliable, and more consistent with your immediate goals. In this series, I have discussed the beneficial use of comments to design and plan code and taking notes as you code. At times, though, you encounter the need to do some simple, back-of-the-envelope kind of design, in which you track what you need to do and where you are in relation to the undone tasks. Moreover, as you work, you regularly add other tasks to be done that integrate with the existing list of remaining work. I find that this kind of recording arises particularly on personal projects, where I can hold the entire project in my head as I code. That capacity is a great benefit in that it helps me to anticipate dependencies, breaking changes, and so forth. But it quickly becomes a heavy habit when I need to track reminders to myself and coding notes all in my head. Having an effective way to diagram and track the tasks and their important traits inside a large project map is a huge benefit. Mind maps represent an effective and easy way of capturing just this kind of data—allowing you to reclaim the mental space it otherwise occupies.

If you're not familiar with mind maps, you've surely seen one or two in presentations or documents. They take various forms, but the one in Figure 1 gives a representative example. This sample is crafted more for presentations than you will need for coding. But it shows the main features, which consist of a series of trees that bring together one feature set and to which you can add branches and leaves that contain new features, opinions about features, to-do items, and essentially any information of value.
In Figure 2, I show a more typical mind map for a personal project. It shows considerations for a feature that adds HTML5 output to a typesetting system I have worked on.

As you can see, the main box names the feature (support for HTML5), and the map has three primary considerations: How the support is triggered (at startup or by a config file?), how the feature is delivered (an HTML5 plugin with various functions hanging off it), and then how the HTML is verified. By itself, this mind map is too high level to provide much information except for identifying some basic things I'm thinking about when it comes to designing this feature.

The little circles at the end of some lines indicate collapsed subtrees that can be expanded to reveal additional information. Figure 3 shows that expanded view.
You'll notice more entries and that some of them have a small blue question-mark icon, which I use to indicate an aspect that needs much more evaluation before proceeding. And at the top of the chart you can see a call out with a question I need to address. In Figure 1, you can see other helpful icons showing benefits (the circled +) and drawbacks (the circled -). Mind maps are infinitely modifiable. You can take a whole branch and move it to a different branch, for example.

With this tool, you can sit down and design features, identify the issues and the aspects that need to be coded, make comments, solicit feedback, indicate priorities and status, and do whatever else you need to do. This is much more useful than a to-do list, while being much less complicated than UML diagrams.

There are several good mind mapping tools available. FreeMind is an open source version written in Java that works well. Among commercial products, I like and use XMind, which has many more options and is especially easy to use. However, there are plenty of products to choose from, including Microsoft Visio.

Developers who use mind maps to plan development and organize their thoughts, use comments to lay out their code a priori, and take notes to track their decisions as they code are much more capable of getting the code right the first time and hunting down errors quickly. They can also move faster and more adroitly (in the same way that developers with a full set of tests can modify existing codebases) and with security, knowing that decisions are thought out well, easy to correct, and easy to document. For the small overhead these tools require, I believe these benefits are very much worth it.

A personal note: This is my last issue of Java Magazine. After five very enjoyable years at the helm, I'm ready to take on other challenges, including getting back to working on my preferred coding projects. I will surely pop up here and there with articles and reviews (likely even in this magazine). If you've enjoyed my work, I invite you to follow me on Twitter (@platypusguy) or to reach out to me on LinkedIn, where I accept all invitations. At the moment, I am currently participating in interviewing prospective successors and I'll make sure that Oracle has a good person in place to carry on. From the bottom of my heart, thank you all for being readers; and to many of you, I send additional gratitude for your thoughtful comments and suggestions over the years. It's been truly an honor.
Andrew Binstock

Andrew Binstock (javamag_us@oracle.com, @platypusguy) is the editor in chief of Java Magazine. Previously, he was the editor of Dr. Dobb's Journal. He co-founded the company behind the open-source iText PDF library, which was acquired in 2015. His book on algorithm implementation in C went through 16 printings before joining the long tail. Previously, he was the editor in chief of UNIX Review and, earlier, the founding editor of the C Gazette. He lives in Silicon Valley with his wife. When not coding or editing, he studies piano.