Software Engineering for Absolute Beginners

Your Guide to Creating Software Products

—

Nico Loubser

Apress®

Software Engineering for Absolute Beginners

Your Guide to Creating Software Products

Nico Loubser

Apress®

Software Engineering for Absolute Beginners

Nico Loubser
London, UK

ISBN-13 (pbk): 978-1-4842-6621-2 ISBN-13 (electronic): 978-1-4842-6622-9
https://doi.org/10.1007/978-1-4842-6622-9

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Spandana Chatterjee
Development Editor: James Markham
Coordinating Editor: Divya Modi

Cover designed by eStudioCalamar

Cover image designed by Pixabay

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, Suite 4600, New York, NY 10004-1562, USA. Phone 1-800-SPRINGER, fax (201) 348-4505, email orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-6621-2. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

This book is dedicated to Kim,
who keeps me going when I feel like stopping.

Table of Contents

About the Author

Nico Loubser is a software engineer by trade, with 16 years of experience in various industries and technologies. As an experienced team lead, he has mentored numerous developers and has developed a passion for it, which was the inspiration for writing this book. He believes that the so-called software crisis[1] can be alleviated by proper mentorship, but that mentorship is not always available. He currently lives in London, where he seeks exposure to an even greater variety of ideas, methods, and technologies in today's software development industry. He holds a post-graduate degree in software engineering from the University of South Africa.

[1] https://en.wikipedia.org/wiki/Software_crisis

About the Technical Reviewer

Andy Beak is an experienced technical manager with an extensive development background and sound decision-making skills. He is the author of a cybersecurity microdegree course for the EC Council and the author of the Zend PHP study guide published by Apress. He's naturally entrepreneurial and able to zoom in on implementation details while retaining a "30,000 feet" overview of the organizational strategical context in which development occurs. An evangelist for agile working practices and delivery automation, he follows the entire development process and has a high degree of ownership for the quality of the finished product.

Acknowledgments

This book would not have been possible without all of the junior developers I have mentored over the years, as this is where my inspiration for this book originated.

I would also like to thank Andy Beak for reviewing this book, and in doing so, improving the quality of it.

Lastly, to the team at Apress who helped me produce and publish this book, thank you very much.

Introduction

Writing software is a multi-disciplinary exercise. This makes it especially difficult for people who want to learn how to create software but are without someone guiding them and helping them navigate their way between all of the technologies and methodologies there are to learn. The aim of this book is not just to teach, but also to guide the newcomer, showing where the learning efforts should be concentrated, what is good practice, and what are some of the industry standards in the current software development industry. This book bridges the divide between just writing code and creating software systems.

About This Book

This book is not just for the complete newcomer. It is also for someone who can already write code, but is interested in creating complete software projects, from inception to delivery, as well as software design practices.

As a software developer, I can wholeheartedly tell you that writing code is only a part of today's software development paradigm. In today's world, you need to have learned, and in some cases mastered, a set of specific tools, skills, and methodologies that will help you achieve your goals as a creator of software. Whether that goal is to become a hobbyist developer, whether you want to create a startup or work for a corporation, good software engineering skills are very important. Most people will pick up a book about programming, or go on the Internet and start learning how to write code. Very few people will read a book on software engineering principles, and not everyone is so lucky to start in a job where serious engineering principles are followed and enforced and where proper tools are used.

Why Are Good Practices Essential?

Certain principles in software development remain the same, regardless of which company you work for. If you consider a company with 200 employees and a 1,000,000 clients, and compare it to a company with 2 employees and 150 customers, you should notice two things. The bigger company has different problems to solve with regards to infrastructure, scalability, and keeping their code base clean with a potentially large development team. The second thing you should notice is that both companies also have similar problems to solve, such as security, keeping the code base clean, deployments, writing clean code, and using proper software engineering principles to design good code. Even if you build a website for your cousin's brake skimming business, security, proper coding principles, and architectural principles are very important.

A company that serves 100 customers a month should not have its software written in an ad hoc, shoot-from-the-hip fashion. If your complete user base, whether it is 100 people or 1,000,000 people, depends on a software product, then it means buggy code will affect 100% of your client base. No company can afford to have their client base affected to this extent.

By no means does this book suggest to over-engineer your software solutions. If you are writing software that reads the temperature in your garden into a database every minute of every day, emails you a graph every month, and your brother can log in online to check the temperature in your garden, then you don't need a supercool Kubernetes cluster on AWS. You do, however, need a clean code design that is easily modifiable, secure code, and version control. You may think no one will hack into your system, and you will be dead wrong. Not all hacking is for financial gain. Some hacking attempts are for bragging rights, which is more than enough incentive to deface your website. If a simple input field is left unprotected, like a telephone number input field, your whole database can be trashed, stolen, or corrupted.

Why Did I Write This Book?

A while ago a friend started to learn how to program. He struggled initially because some of the concepts that were covered were intended for someone with a coding background. Reviewing the literature he was using, I noticed that they also basically all excluded a comprehensive approach to creating software. I saw this as two problems. Firstly, some of the beginner material out there caters to people with some knowledge about programming. This is not the end of the world. You are all intelligent enough to put the pieces together and learn from material that is intended for someone with more knowledge. But it was this aspect that made learning more difficult for my friend. Secondly, there was also the absence of the processes to build comprehensively good systems. So I decided to create this book.

My aim for this book is to show a complete beginner the cornerstones of creating easily readable, maintainable, editable, and releasable software that can be adapted and changed as needed. I wanted to touch on the principles and knowledge needed to create great software products—more aspects of software development than just writing code. As mentioned, software engineering is a vast discipline that requires many technical skills and knowledge to create great software products.

Good software methodologies, tools, and approaches go back a very long time. Having said that, today's software development world is different from what I was generally exposed to when I started out in the early 2000s. Back then, we manually FTP'd our files to the server. Before we FTP'd anything, we would make a copy of that script on the server. It was not uncommon to see files with names such as `index_1.php`, `index_backup.php`, and `index_final_backup.php`. File management is now handled by version control software. Version control systems are not new, but I believe they are now incredibly widespread and more commonplace than ever before. I also believe they are imperative to a programming project.

How This Book Is Organized

Since the intent of this book is to teach you most of the basic aspects of creating software products, it has been designed so that the chapters build on each other.

The first three chapters look at setting up your system. Chapter 1 looks at software editors, Chapter 2 looks at setting up your software environment using containerization, and Chapter 3 looks at setting up your source control system where you can save your project remotely. These first three chapters form the basis on which you create software and what your software runs on.

Chapter 4 teaches you how to write code using Python. The work in Chapters 1, 2 and 3 contribute to this chapter. Chapter 5 builds on Chapter 4, showing you how to write better code. Chapter 6 shows how to design databases.

Taking Chapters 4, 5 and 6 in consideration, you can move on to Chapter 7, in which you build a small project using the skills you learned in the preceding chapters.

Chapter 8 teaches you how to test for code quality, and Chapter 9 looks at design concepts.

Chapter 10 looks at security issues, and we round it all off with Chapter 11, where you look at hosting your software, as well as continuous integration and deployment.

CHAPTER 1

Editors

Creating software is all about solving problems. And your software development editor is a great place to increase productivity and lessen the cognitive load you will experience while solving these problems. Within your editor lies the ability to automate some of your tasks. It will allow you to defer some tasks that would have strained your memory or would have consumed too much time, to your editor. A decent editor will allow you to optimize the layout of your screen so that your database browser, shell, and code editor are easily and readily available within seconds. It will have a large collection of shortcut keys to simplify certain actions. It will also allow you to change the look of your editor, to soften the colors, and to choose a font that is easier on the eyes to lessen eye strain.

In this chapter, we will look at the differences, and benefits, of the different styles of editors available for software development. Selecting an editor may sound like a very trivial issue, but in the end it can lead to bad decisions that can affect your productivity. When we program, we basically create a text file containing different commands. This file will not have a text file extension (`.txt` for example), but rather an extension indicating what language it was written in, for instance `.php` or `.py`. But it is nothing but a text file. These files containing the commands are interpreted (or compiled) and executed by your chosen language's interpreter (or compiler). Because of this, we can, in general, create our programming language's code files in almost any editor we choose, as long as we can give it the right file extension. Because we can choose almost any editor, there are many editors to choose from. This makes the decision more

https://doi.org/10.1007/978-1-4842-6622-9_1

complex. This chapter will highlight some of the editors available and their drawbacks and benefits.

Before we delve into our discussion about editors, just some background about why we selected the editors we did in the section below. The language we will use is called Python. It is a very popular and powerful language, plus it's easy to learn. And like most languages, you can use a myriad of editors to achieve your goal of creating software.

The Different Families of Programming Editors

There are three broad sets of editors to use to write your code, and each set is useful in its own way. Using and supporting a specific editor is normally a matter of experience. It may literally take you weeks or months to realize there is something about your editor that you just do not like. A certain editor may give you a slick modern look, but may be slow when it opens files. Or you may be forced to choose one with specific built-in functionality, like support for FTP. Editors often have quirks that will slow you down or start to irritate you as time goes by, and in many cases, you will only learn about these quirks when you test the editors yourself. You should also consider the non-programming aspects of an editor, things like background color, font types, font sizes, and font colors. Staring into a screen for hours on end is very hard on your eyes, and being able to customize your setup to lessen eye strain is important.

Shell-Based Editors

The first set of editors consists of shell-based editors like Vi, Vim, and Nano. Shell-based editors run in a Windows, Linux, or Macintosh command shell. You will get some exposure to command shells in this book, but not with shell-based editors. Shell-based editors have a high learning curve, are purely text-based with no fancy graphical user interface, and are indispensable in certain circumstances. For instance, they're great for fixing code or putting in a temporary code fix on a remote system that has no graphical user interface while someone works on a permanent fix. In a lot of instances, if your career is going the Linux route, you will encounter Vi or Vim. You can also get Vim for Windows but I doubt you will ever need to use it. Vim is quite powerful, but can be made even more powerful if you install the plugin SPF13 for Vim. I believe that these editors have their place in software engineering, but they should not be considered your primary editor.

Text Editors

The second group is text editors like Notepad, Gedit, Atom, Sublime, and Visual Studio Code (also known as VS Code). Text editors are a cost-effective way of getting a GUI-based editor to write your code in (they vary from low-powered to very high-powered). In the case of an editor like the Windows-based Notepad, you get absolutely no features and you cannot add any features. Yet you can create files with the correct extensions which can be interpreted or executed by a programming language. I absolutely do not recommend Notepad, but I include it in the list to prove my point that you can make bad decisions when choosing editors.

Linux's Gedit is good for a quick test script, and it gives some basic features which can aid in development. Just like Notepad, I don't recommend it, unless you need to churn out a 30-line script that does something small.

Under the same umbrella as these text editors, you will also find editors that can be very powerful. Two of these editors are worthy of a mention: Atom and Visual Studio Code. Both come with a myriad of plugins and built-in features, and are customizable to a degree. It may be difficult to choose between the two, and since both are free, I feel there is no harm in you trying out both.

Atom comes across as very modern and approachable, but Visual Studio Code is elegant and in some cases boasts faster startup times than Atom. According to the website `www.software.com`, VS Code leads the race in the most popular free editor for software developers. But both can deliver the power you need for a perfectly free, feature-rich development experience. In this chapter, we will look at VS Code, but I will encourage you to experiment and play around with a lot of editors. This experience will help you notice certain drawbacks or benefits between editors. Personally, I like text editors, but I am not fond of searching for plugins. There are also potential speed issues compared to IDEs, such as when opening large projects or indexing your files to improve searching.

IDEs

A third option is an IDE (integrated development environment) like Pycharm, Eclipse, and Wing (to name a few). They come packed with a debugger, interpreter or compiler, web server, shell terminal, database editor, and fully fledged code editor. An IDE can also syntactically evaluate your code based on the version of the programming language you are using. Some are also integrated with different version control systems, and even keep a local history, just in case you delete something by accident. Some IDEs come at a price, but normally they are well worth it.

By default, your IDE will index your projects, making them instantly searchable. You can follow code from implementation to integration and back again. There are also many shortcuts that speed up certain actions. Granted, some of these features can be added to a text editor via plugins, but speed-wise I have not yet seen editors perform at the same level as IDEs do. On the topic of plugins, IDEs normally also come with a plugin system. I have used many IDEs. At the time of writing, I am using Pycharm, which is really hitting the spot with me. Jetbrains, the company that produces Pycharm, offers a free community edition, which has less functionality than the professional edition (which has a fee) but is still packed full of features.

The Benefits of an IDE or an Editor like VS Code

Having your work environment set up in the best possible way has quite a few benefits. You will definitely increase your productivity if you get to understand aspects like code navigation and debugging. A few IDEs allow you to query your database and run your shell commands in different panes next to where you are coding. This may not seem like a big deal but it does increase your productivity. Source code navigation and code completion will absolutely increase your productivity, while the ability to step through your code line by line during execution time, and injecting data into it at runtime, are incredibly powerful tools.

Let's go back to code completion. Code completion is such a simple concept. But using code completion frees up your mind so that you do not have to worry about remembering all of the different keywords you find in programming languages, for instance, or even how to implement different program-specific aspects, since the IDE can remind you how to do them. It is great to know these aspects by heart, but remember that writing software can be very taxing on your cognitive system, and breaking your train of thought while solving a complex problem can be problematic, especially if you had to do so just because you could not remember a specific keyword. Having code completion alleviates that burden.

This is the premise on which this whole chapter hinges. Choose an editor, whether it is a text editor with the correct plugins or an IDE that takes the strain off of you having to remember simple things that the editor can just remind you about, and do automatically (or a 100 times faster) the things you did manually, and you can get on with what creating software is all about: solving problems.

Installing Visual Studio Code

I suggest that you install Visual Studio Code because it is a great editor with great features. After installing it, you will take a quick look at some of the features of VS Code. At time of writing, VS Code can be installed from the following location: `https://code.visualstudio.com/download`.

The default layout of VS Code has two sections. The left-hand pane contains the structure of your folders as well as files. The pane on the right-hand side has the code editor in it. The left-hand pane also contains your workspaces, as explained below.

To create a file, just click the File menu item and select New File. When you save this file, remember to save it with a `.py` extension in order for VS Code to recognize it as a Python file.

Workspaces

A workspace shows a project's contents. The files and folders that make up the project are visible in the workspace in the left-hand pane of the editor. It is not mandatory to have workspaces. You may just reopen the directory with your project's code each time. However, workspaces give you a convenient way to organize your projects. Creating a workspace is easy.

1. From the File dropdown menu on the top menu bar, select Open.
2. From there, open the project directory, with your code in it.
3. Once that is open in the left hand-pane in VS Code, open the File menu once again, select Save Workspace As, and give it a name.

You will see that in the left-hand pane, you have created a workspace with the name you gave it and (Workspace) after it. To reopen a workspace, you have two choices:

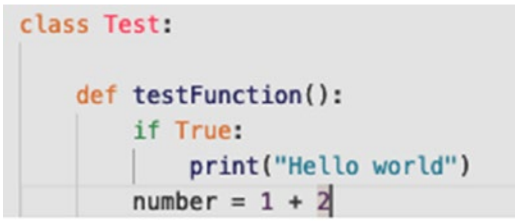
4. From the file menu, select Open Recent.
5. From the file menu, select Open Workspace. From there, look for your project directory, and click the file inside that directory with this name: *the-name-you-gave-it*.code-workspace.

What happens in the background is that VS Code creates a file in the directory your project is in, and now considers that directory your workspace. Inside this file you will find the following text. This file is a skeleton and can be filled out to be more complete, but we are not concerned with that. It is noteworthy, though, that the path value in this file points towards your workspace. You can move this file to another location as long as you update the path value. This is good to know, but not something we are going to do now. The default values will do just fine.

```
{
    "folders": [
        {
            "path": "."
        }
    ],
    "settings": {}
}
```

Built-In Features

VS Code comes with some handy built-in features, such as syntax highlighting. Syntax highlighting is when an editor presents different words, which have specific meanings in a programming language, in different colors. These words can also be grouped by color; for instance, specific keywords belonging to Python, even though not the same word, will have the same color because they belong to the same group called keywords. See Figure 1-1.



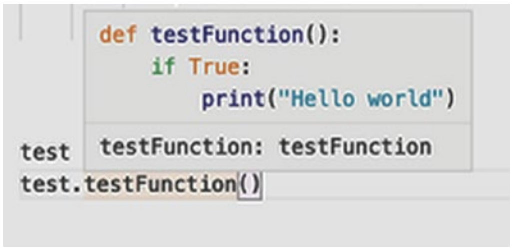
```
class Test:

    def testFunction():
        if True:
            print("Hello world")
        number = 1 + 2
```

Figure 1-1. *Keywords*

In Figure 1-1, you can see that specific keywords (in this case, `class`, `def`, and `True`) are in blue. The numbers (1 and 2) are in light green, and words indicating function names (we will get to functions later in this book) are in yellow.

Taking the above code into consideration, when you implement the code, you won't have the function written in front of you. You will only use the names, in this case `Test` and `testFunction`. This means that if you need to see how `testFunction` works on the inside, you need to browse to the page where test function is written. But with a decent editor like VS Code, this becomes a lot easier. The following may all be a bit hard to envision at this very moment, but once you start writing code, it will all start to make sense. See Figure 1-2. With VS Code, you can view the code written even though you are in another script by hovering your mouse over the word `testFunction()` and pressing Shift + Ctrl. A popup will appear with the code in it. Holding Shift down and clicking the name of the function will actually take you to where the implementation is written. These two small portions of functionality make it a hundred times easier to navigate a codebase and take the task of browsing out of your hands.

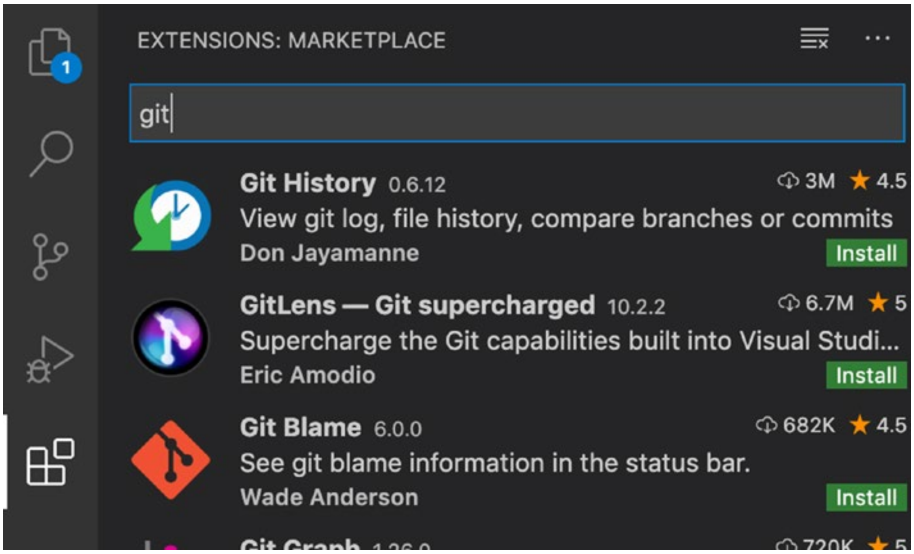


```
def testFunction():
    if True:
        print("Hello world")
test  testFunction: testFunction
test.testFunction()
```

Figure 1-2. Code popup

Features to Install

You may find that some of the features you want are not built in. They are called extensions and they are installable via the Extensions Marketplace. To install an extension, click the four squares in the left-hand shortcut menu, as seen in Figure 1-3. This will open the Extensions Marketplace, allowing you to search for extension that can make your life even easier. In this example, I searched for "git" in the search text box, and underneath it, all of the potential extensions appeared.



EXTENSIONS: MARKETPLACE

git

Git History 0.6.12 — 3M ★ 4.5
View git log, file history, compare branches or commits
Don Jayamanne — Install

GitLens — Git supercharged 10.2.2 — 6.7M ★ 5
Supercharge the Git capabilities built into Visual Studi...
Eric Amodio — Install

Git Blame 6.0.0 — 682K ★ 4.5
See git blame information in the status bar.
Wade Anderson — Install

Figure 1-3. Searching for extensions

Summary

This was an easy chapter, but the subject is no laughing matter. Choosing an editor that is right for you is important, but may take some practice. I still remember how I thought the editor I used back in 2003 was the best PHP editor ever and that I would not need anything else in my life. Now, many editors and many years later, I can reflect on that simple choice and clearly see the error of my ways. You took a quick look at VS Code, but it should be enough to get you going and a good first step as you learn how to create software.