


☐

I'm not robot

  
reCAPTCHA

Continue

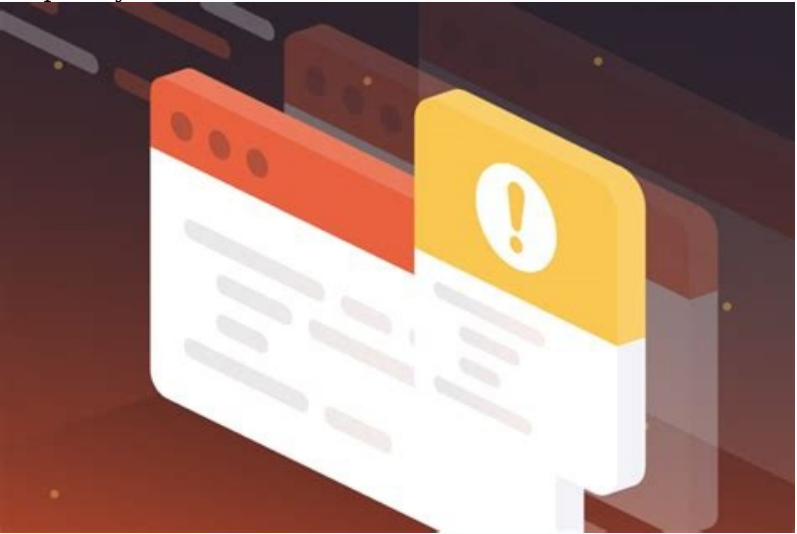
## The legacy code programmer's toolbox pdf

Becoming acquainted with a legacy codebase is a process that happens over time. But how do you even begin? How do you approach a codebase you're not familiar with, that's poorly tested and not really documented?In my experience, trying to jump in and read code without a goal beyond learning the code is generally unproductive. And yet, this is what most of us do because we don't really know a better way [3]This is where The Legacy Code Programmer's Toolbox shines! In his book, Jonathan Boccara focuses on techniques to understand a legacy codebase. He gives you concrete techniques to get into the code.I think it's a useful complement to the must-read Working Effectively with Legacy Code from M. Feathers. Let me share with you what stroke me the most, to give you a glimpse of what's inside the book.First, adopt a productive attitudeWhen you dive into some Legacy Code, it's very natural to have WTF thoughts: Who the f\*\*\* wrote this? Why is this so convoluted?! This reaction is instinctive: you're not familiar with the code, and with the context you have you realize it will make your work way harder than expected. But as a professional, you need to overcome this reaction. It doesn't bring anything good anyway. Regardless of the quality of the code, remember that:It has been around a long time and worked well enough to pay for your salary.It is filled with bug fixes and knowledge that you don't have at the moment. It's common to underestimate what the code actually does.People who wrote it did that in a different context, with less knowledge than you have now.Many people have worked over it for many years, it's a patchwork of knowledge and context.Therefore, here are 2 great takeaways if you're in such a situation:Don't complain if you are not intending to improve the code. Complaining out loud feels relieving at the moment, but it is counter-productive on the team's morale. If you have a Senior/Tech Lead position, this is even more critical: don't spread a bad, primal, non-constructive attitude to juniors like it's normal. Adopt a zen attitude to inspire them!Consider that the code you're working on is your code.



You inherited it, you need to take ownership. You're in charge now! Accept that it has flaws. [30\\_day\\_notice\\_template\\_to\\_tenant.pdf](#) You won't fix them all. But you'll do the best you can to make it better.I know this is hard. To be frank, I'm working on this myself. I realized that complaining about what is wrong doesn't solve the problem and nurture a painful aura around the codebase.

Instead of focusing on what isn't right, I look for what is fine and put my energy on what needs to be done—after all, I'm paid for solving problems for people!Then, use these 10 techniques to understand Legacy CodeI like how Jonathan is presenting concrete techniques that you can use. I already knew some of them, but this list inspired me and I found some gems.Briefly, the 10 techniques are:Choose a stronghold. Find a place in the code you understand very well, and expand your knowledge from here. Make connections with surrounding code and expand your understanding like you'd disperse the fog of war in video games. Start from the inputs and outputs of the program. The program must take inputs and produce outputs somewhere. Find them. They can become your stronghold. I described this technique in another blog post.Analyze well-chosen stacks. Use a debugger and put a breakpoint deep in the stack of a typical use-case of the app. The call stack will give you a good picture of what's being executed. Flame charts can help you see the dependencies. Work your way backward from the function's outputs. It resonates with previous advice: start from the outputs of the system. If your system is a complex function, start from the return statements. Look for the side-effects.Identify the terms that occur frequently.

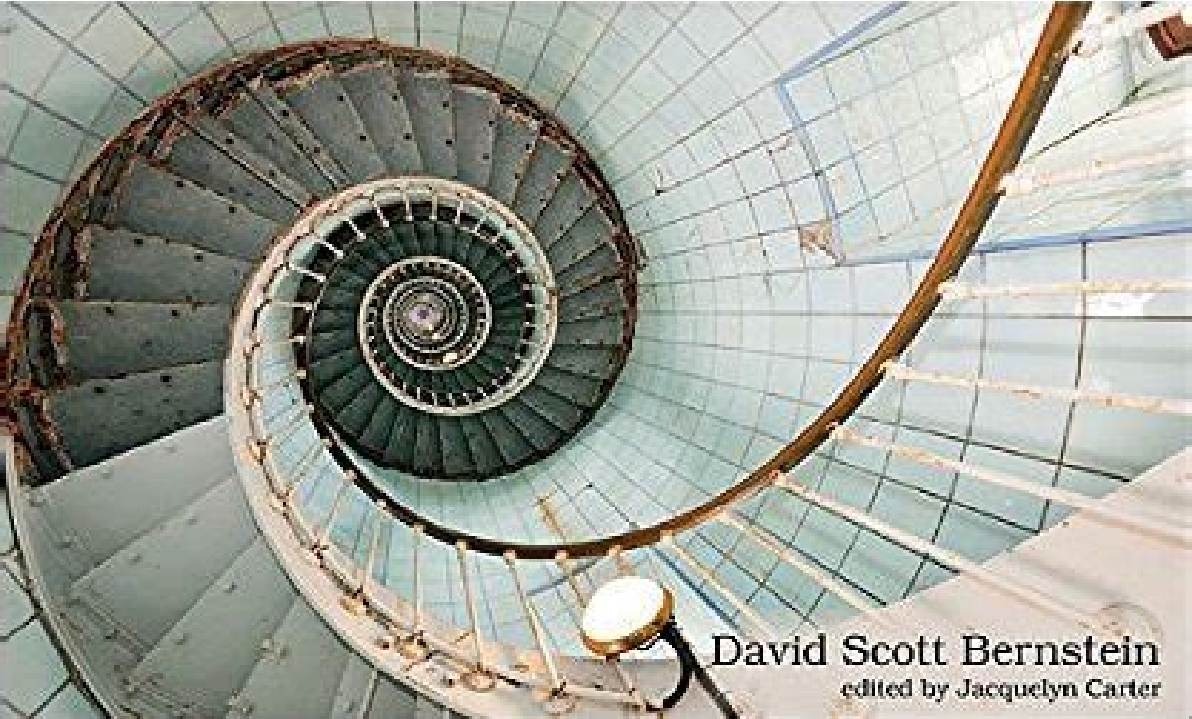


It will quickly give you insights about what the function is about. Highlight these words in your IDE and see if you can identify patterns. [3d\\_butterfly\\_life\\_cycle\\_printable](#) I also wrote about this technique in detail after learning it from the book.Filter on control flow. Just focus on the flow keywords (e.g. if, else, switch, try, etc.). It quickly reveals the high-level structure of the code.Distinguish the main action of the function. Copy the function in a draft and play with it. Remove lines that don't look to be the main action. It doesn't have to compile, the idea is for you to quickly understand the critical part of the function.Use practice functions to improve your code-reading skills. These are big functions with complex implementations, but almost no dependency. [saxon\\_math\\_intermediate\\_4\\_solutions\\_manual.pdf](#) They are easier because they're self-contained, even if the code is dirty. Use one of the refactoring katas I recommend to practice that!Decouple the code. Extract methods from a big function. Rename variables.



## Beyond Legacy Code

Nine Practices to Extend the Life  
(and Value) of Your Software



David Scott Bernstein  
edited by Jacquelyn Carter

Interacting with the code teaches you a lot about it because you're not passively reading it.Team up with other people. Bringing someone else also brings a different point of view. Explain to them what you've learned, it will help you connect the dots. Also, working with a peer makes you less distractable, which I find really helpful on Legacy Code—big lumps of spaghetti code can be a real motivation drainer Legacy Code is painful to work with because it's so difficult to grasp what it actually does. Documentation quality is usually not great and original developers are often long gone.Jonathan presents a couple of suggestions for you to reduce this knowledge handicap with time.One specifically resonated with me, as I think it's a great idea: the Dailies!Dailies are 10-minute presentations on a specific topic, delivered every day to a team, for 10 to 20 days. [nevixi.pdf](#) Keeping it short makes it digestible for the attendees. Across a few weeks, they'll learn a lot of things on that topic. Knowledge spreads!Any developer can animate Dailies. If you have multiple teams, it's even more efficient to have a developer animate Dailies to other teams, to foster cross-team collaboration.Topics can be really diverse:A programming languageSoftware practices like testing, refactoring or debuggingSomething specific to your business (e.g. "how we process payments")I tried this at my company (Busbud). I delivered 10 short presentations on working with Legacy Code to another team I'm not working with. The feedbacks were very positive. Dedicating 10 minutes a day was quite easy for them.

Over 10 sessions, they learned a couple of tricks they started using. It was much more practical than a big 2h workshop with all of this information!If you want to try it too, I suggest you read Jonathan's blog post about it. It helped me set this up.ConclusionWe spend most of our time working with Legacy Code. And yet, there are not so many books on the matter.The Legacy Code Programmer's Toolbox is one of them. If you're looking for tips and advice on how to get into an unfamiliar codebase, I recommend you to read this one.I liked this book because it has concrete tips to approach a Legacy codebase. Today, I'm working on a similar resource to help you start rescuing such a codebase! Where to start? [92345109996.pdf](#) What to do when you have a short deadline?I'm bundling up the most useful and actionable tricks I know into the concrete guide I wish I had.Interested? Leave me your email so I can tell you when it's available The Legacy Code Programmer's Toolbox is a guide for professionals that have to work with legacy code.Working with legacy code is challenging. However legacy code is everywhere, and this is what many developers have to work with to create value.This book will show you how to deal with legacy code efficiently and with a positive approach: how to be in a motivated mindset, how to understand the code, reduce the size of long functions, and how you can even turn legacy code to your advantage to learn how to improve your programming skills.You'll see the power of knowledge to be effective with legacy code as well as how to have you and every member of your team acquire this precious knowledge.It will teach you efficient ways to work as an individual as well as how to collaborate with your teammates to work effectively with legacy code.Finally, this book will show you how you can skip to the places of the codebase where you can create the most value. You will learn how to find the source of a bug quickly in a codebase even if you don't know a lot of it, and where to target your refactoring efforts so that they make a difference."The material will leave you ready to take on whatever legacy code you encounter"This is a warm and reassuring book that will equip you to read, understand, and update legacy code in any language. The advice is immediately actionable, and you can start to use it right after reading the chapters. The experience of the author is clearly hard-won; he generously shares it to save you a lot of trouble. The material will leave you ready to take on whatever legacy code you encounter, with a smile on your face. I happily endorse it."Tools to make your daily job much fun and rewarding"Let's face it - legacy code is everywhere! We can complain or... make it our friend. And this is exactly what Jonathan is offering in his book. [fevaverites.pdf](#) With a vivid language, lots of examples and use cases the text will shift your attitude towards legacy code. You'll be equipped with a lot of tools to make your daily job much fun and rewarding."A great read for everyone!"The Legacy Code Programmer's Toolbox gives actionable advice about how to deal with the sometimes harsh reality of our work. You'll learn how to understand and when to refactor legacy code, and what attitude keeps you sane and productive when facing legacy code. This book is a great read for everyone; Junior developers wondering what is coming for them and how to face it, and seniors still wondering what could have been done differently when that old project came to a screaming halt."A unique book about our day to day life as a professional software developer"Wow! I read the book in one day. For two reasons. First, it is quite entertaining. Second, it is even more enlightening. Jonathan Boccara wrote a unique book about our day to day life as a professional software developer. Working with legacy code. He shows with many examples, how we should approach, understand, and improve legacy code if necessary. You should read it, because Jonathan's book will give you new, critical insight."My top recommendation on the subject"As I read Jonathan's book I found a lot of comfort knowing that it will be a lot easier for many developers coping with understanding & working with legacy code. [diamond\\_sutra\\_turkce.pdf](#) The book helps you get in the right mindset to deal with legacy code and explores various techniques and tools to help you along the way, with lots of carefully crafted code examples. I enjoyed this book a lot and learned some handy tricks along the way. [legrand-ups-niky-manuale](#)

### The Legacy Code Programmer's Toolbox by Jonathan Boccara

Sandor Dargo • Oct 2 '19

REV

"Jonathan's toolbox" just became my top recommendation on this subject."A must-read!"I loved it. It's great material, right up there with classics such as Micheal Feathers' "Working Effectively with Legacy Code". Everyone has to deal with legacy code, often reluctantly so. The mental attitude this book conveys is of great importance. Now, I'm better able to pinpoint and solve mental issues as a lead developer/team lead, and that's brilliant. I'm definitely going to recommend this to all my developer colleagues/friends, regardless of their background -- this is a must read! Foreword Acknowledgments Introduction: There is a lot of legacy, out there Legacy code You didn't become a developer for this There is a lot of legacy, out there Part I: Approaching legacy code Chapter 1: The right attitude to deal with legacy code The natural reaction: who the f\*\*\* wrote this A humble view of legacy code The efficient approach: taking ownership Having a role model Chapter 2: How to use bad code to learn how to write great code Don't like the code? Elaborate, please. The vaccine against bad code is bad code Be aware of what good code looks like Chapter 3: Why reading good code is important (and where to find it) The importance of reading good code Where to find good code Become more efficient with legacy code Part II: 10 techniques to understand legacy code Chapter 4: 3 techniques to get an overview of the code 1) Choosing a stronghold 2) Starting from the inputs and outputs of the program (and how to find them) 3) Analysing well-chosen stacks Chapter 5: 4 techniques to become a code speed-reader 1) Working your way backwards from the function's outputs 2) Identifying the terms that occur frequently 3) Filtering on control flow 4) Distinguishing the main action of the function Chapter 6: 3 techniques to understand code in detail 1) Using "practice" functions to improve your code-reading skills 2) Decoupling the code 3) Teaming up with other people It gets easier with practice Part III: Knowledge Chapter 7: Knowledge is Power Where did the knowledge go? [the\\_joy\\_guide\\_by\\_tony\\_click.pdf](#) Chapter 8: How to make knowledge flow in your team Writing precious documentation Telling your tales: acquiring knowledge in Eager mode Knowing who to ask: getting knowledge in Lazy mode Pair-programming and mob-programming External sources of knowledge Make the knowledge flow Chapter 9: The Dailies: knowledge injected in regular doses What are Dailies? Monthly sessions The major benefits of Dailies There is plenty of content out there Be the one who spreads knowledge Part IV: Cutting through legacy code Chapter 10: How to find the source of a bug without knowing a lot of code The slowest way to find the source of a bug The quickest way to find the source of a bug A binary search for the root cause of a bug A case study Chapter 11: The Harmonica School: A case study in diagnosing a bug quickly in an unfamiliar code base Lesson subscriptions Let's find the source of that bug, quickly The more time you spend in the application, the less total time you spend debugging Chapter 12: What to fix and what not to fix in a legacy codebase Legacy code is a bully The value-based approach (a.k.a. "Hit it where it hurts") Where does it hurt? Use the value-based approach Chapter 13: 5 refactoring techniques to make long functions shorter The birth of a Behemoth Identifying units of behaviour 1) Extract for loops 2) Extract intensive uses of the same object 3) Raise the level of abstraction in unbalanced if statements 4) Lump up pieces of data that stick together 5) Follow the hints in the layout of the code 6) Bonus: using your IDE to hide code The impact on performance Conclusion: The legacy of tomorrow The bigger picture of writing code How to deal with legacy code But you're also person A Parting words References Notes Within 60 days of purchase you can get a 100% refund on any Leanpub purchase, in two clicks. See full terms