



80

TIPS FOR WRITING BETTER CODE FROM REAL DEVELOPERS



Collaborator

Contents

Selecting the Right Tools	5
Understand the Role of Testing	7
Writing Maintainable and Sustainable Code	10
Starting with the Requirements	13
Learning, Education, and Career Development	15
Code Review and Profiling	20
Collaboration on Development Teams	24
Revisiting Your “Writing” Process	29

Developers understand the need for collaboration. Collaboration isn't limited to the work you do with other members of your development team.

It's also not limited to the work you do with other people within your organization — including: QA, business analysts, and UX designers.

Developers cannot afford to code in a bubble. One of the things we constantly find when speaking with our developer customers is that some of their most important lessons come from working with others. They find that knowledge transfer and onboarding happen more quickly and easily when working with others.

That was the inspiration for this guidebook.


Earlier this year, we reached out to hundreds of developers, across a wide variety of industries, and asked them to share advice for other developers. The question we wanted to answer was, “What is the best piece of advice you have for a developer or a team of developers to write better code?”

We reached out to current SmartBear customers, and

even posted the question on sites like Hacker News, where more than 100 people commented and shared their best advice. As you may expect, the responses were varied and provided a unique perspective for developers across industries, whether you work in a start-up or an enterprise organization.

To help you find the advice that's best for you, we organized these tips into 8 chapters, with additional resources to help you put this advice into action.

- Selecting the Right Tools
- Understand the Role of Testing
- Writing Maintainable and Sustainable Code
- Starting with the Requirements
- Code Review and Profiling
- Collaboration on Development Teams
- Revisiting Your “Writing” Process
- Learning, Education, and Career Development

After reading this guide, we hope you'll join @SmartBear and share your own tip with the hashtag #SBDevTips. 


Selecting the Right Tools

As a developer, you already rely on a number of tools and systems that enable to do your job effectively. Whether you're selecting tools for configuration or defect management, bug tracking or code review — you depend on these tools to help you write better code. Earlier this year, SmartBear Software conducted a survey of more than 600 software professionals, which looked at how teams maintained software quality throughout 2015. We also collected new data about the top tools being used by developers today.

Here is a list of some of the top tools:

- **SCM:** Git
- **Repository Management:** GitHub
- **IDE:** Visual Studio
- **Bug Tracking:** JIRA
- **Requirements Management:** JIRA
- **Code Review:** Collaborator

Tools were also a major theme we heard from developers when we asked them to share their best tips for writing better code. Here are the top tips we received:

1. *“Be sure to dedicate some time to sharpening your saw; researching new tools, technologies etc. It’s easy to do something just because it’s the “way I know to do it” rather than putting in the time to learn a better way.”* 

Mhomde answered on HackerNews

2. *“Use Git Workflow — or similar workflows — in a team of more than two people.”*
3. *“Use Lombok library to generate getters, setters, equals/hashcode, toString, builders, logger variable etc. The less code you write, the easier it is to maintain.”*
4. *“Daemon can run all affected tests for you in the background upon every change, you can quickly commit without waiting for tests to finish.”*

5. *“Be mindful of what you are exposing into your DI container. Exposing lots of low-level objects without real business function can lead to complex problems and unnecessary aliasing. Always prefer exposing interfaces over classes.”*

6. *“I prefer using Docker or similar technology guaranteeing production-like environment locally from day one. Saves from a lot of “WTF” moments.”*

Kirill Frolov

Senior Technology Specialist/ AEM SME, Telstra

7. *“I like to think about cohesion and coupling for non-trivial projects. Recently, I’ve been using a tool to show the dependency graph of my program and I’ve found it to be quite helpful. Perhaps a bigger point is to find and use tools for your environment that really help! I, like others, also favor minimizing changing global state, unit testing (increases odds your code’s interface is more well designed), and code reviews.”*

Mayur Pawashe Answered on HackerNews

Bit Slice Developer

Twitter: @Zorg_

Additional Resources:

- **eBook:** [A Guide to Developer Collaboration with GitHub](#)
- **Report:** [The State of Code Quality 2016](#)
- **Tools:** [See how Collaborator integrates with the tools you trust.](#)
- **Blog:** [Getting That New Tool Past The “Gate Keeper”](#)



Understanding the Role of Testing

The continued growth of Agile methodologies — such as Continuous Integration, Continuous Delivery and Continuous Deployment — has changed the way we think of the relationship between development and QA teams. Specifically, we see that testing is shifting more and more “to the left,” with developers getting more involved in the testing process.

In addition, the increased influence of test driven development has also reshaped the way development teams think of the role of testing. Test Driven Development is a unique way to develop software by starting the process by collecting a requirement, developing test cases for the requirement, followed by the coding process. This can feel like a backwards approach to developers on more traditional development teams, but implementing testing earlier in the development process has proven to reduce defect density and ensure that teams don’t have to sacrifice quality in an effort to move faster.

The developers we reached out to offered a variety of perspectives on the role of testing in writing better code:

8. *“Tests are your friends. For new code, use test driven development. It automatically gives you >90% coverage. I find Spock framework to be the best there is on JVM.”*
9. *“In tests use real data objects (not mocks) wherever possible.”*
10. *“Majority of tests must be really fast (<3-5 msec each). Only use spring test or similar “create-the-world-before-every-test” for module/ integration tests or where it’s absolutely necessary. Avoid a heavy approach for unit tests.”*
11. *“Prefer formal factories when constructing complex object hierarchies over component injection: it allows easy mock unit-testing without heavy-weight Di frameworks.”*


Kirill Frolov

Senior Technology Specialist/ AEM SME, Telstra

12. *“My advice is to develop in binomial and start with unit test developments. It helps a lot when we have a second developer beside to discuss*

better ways to implement user needs. When we start with unit test development, we define the perimeter of the business unit; we can correct or complete it with the product owner or study team who defines the business unit. So actually, I guess that the best way today to improve code quality is to adopt TDD, XP but before we have to help teams to implement Agile project management approaches.”

Mohamed Boustta
Web Developer

13. *“My advice would be to embrace and practice TDD, and have colleagues review your code frequently.”* 

Ben Joyce
Senior Software Engineer at QVC

14. *“Use Test Driven Development. With that, better quality and better structure comes automatically.”*

Girts Linde
Senior Software Engineer, Blue Coat

15. *“Test your code every which way. Think of edge cases, security holes, do research on how code fails and internalize those lessons, and constantly test, test, test, test. Manual tests, unit tests, etc. Code that works reliably is (IMHO) better than beautiful code that doesn’t work at all, or falls apart under strain.”*

King_magic answered on HackerNews

Additional Resources:

- **Article:** [Getting into Test Driven Development](#)
- **Article:** [What is Unit Testing?](#)
- **Blog Post:** [The Great Debate: Code Coverage on Legacy Apps](#)

Writing Maintainable and Sustainable Code

Writing better code isn't just about writing code that works in the short term. Rather, teams need to be continuously thinking about how the code they write will be maintained over time. And whether or not they are adding technical debt as new code becomes legacy code. Writing maintainable code will ensure the long-term quality of the applications you help build and will reduce the “bus factor,” that can occur when a team is dependent on an individual, or group of developers who wrote the code, in order to make changes in the future.

When you build software, you stack the new atop the old and everything comes along for the ride. There may be the occasional new module that stands all by itself or plugs in with the rest, but that's the exception. The new code interacts with the old stuff, calling into it, relying on it, and running beside it in production. A code base with high technical debt has a higher total cost of ownership because changes take longer to implement and the risk of introducing new defects while adding new features is high.

How do you build a more maintainable codebase? Let's take a look at what a number of developers have to say.

16. *“Time is the most valuable resource that exists, so think how to save it by creating code that can be easily maintained. This will ensure that you will spend time with your family.”*

Edson Lima

17. *“Write readable and maintainable code. This is very important for new joiners and it reduces a lot of time for them to get an understanding of the existing code.”*

Asim Israr

18. *“I would have three pieces of advice... Increase maintainability at the expense of speed and size. Always test, test, and test, and when you think it works, test some more. And as Donald Knuth once said, “Premature optimization is the root of all evil.”*

Gerald P. Wright

19. *“Write clean and maintainable code.”*

Pradeep Kumar

20. *“Refactor code continuously to improve modularity and maintainability.”*

Bill Mullen


21. *“After writing a method I ask myself: Are there any unnecessary lines of code that I can remove? Can I make the code more readable for other people? Are there any potential problems I am leaving for the future? Is this the code’s final form or will it need changing in the future?”*

Brook Grewcock
QA Test Engineer
CloudCall Group, PLC

22. *“Never forget that reading code is always harder than writing it. Explicitly state the technical debt that you will incur. Doing a quick and dirty for business is okay but just be upfront about cost. So, as a trivial*

example, do this: `obj.DoSomething()`; instead of this: `obj.InvokeMethod(“DoSomething”)`; because if you remove the `DoSomething` method, the first way will cause a compiler error while the second will just stay broken. Obviously, this is too simple, but there are lots of ways to exploit this to add enormous type safety to your code.”

Joseph Kunk

23. *“Be nice to ‘future you’ — if in 6 months you open up the code and can’t figure out what’s going on right away, you’ve done something wrong. Just because you understand everything now, you won’t when it comes time to refactor down the road. Make sure that you don’t screw future you over by trying to be cute, take shortcuts, or rush something.”* 

Chris Bland answered on HackerNews
Software Engineer
Twitter @ChrisBland

24. *“Here is my most important principle on writing code: Write “defensively.” Assume that you yourself will make mistakes and include verifications and checks against your own errors. Never assume, for example, that a function will return a value always in a certain range, even if you wrote the function yourself and you “know” what it does.”*

Hajo Kirchhoff

25. *“Keep asking yourself constantly: does this code make sense? I think that’s actually a very high bar and most code I’ve seen doesn’t “make sense” in that if some non-developing stakeholder looks at it they won’t go “oh, ok, I see what this is about” but instead “ok this is a bunch of technobabble that I couldn’t possibly understand.” And read, or at least skim, the “Domain Driven Design” book by Eric Evans. His concept of a “ubiquitous language” is necessary for code that makes sense.”*

Mikael Brockman

Mbrock answered on HackerNews

26. *“Always think of the next developer who will maintain your code.”*

Bmancer answered on HackerNews

Additional Resources:

- **eBook:** [A Guide to Approaching Your Development Team When They Write Bad Code](#)
- **Blog:** [What Does Code Quality Really Mean?](#)
- **Blog:** [Who is Responsible for Good Code?](#)



Starting with Requirements

Writing “good code” starts with writing *good* software requirements.

Beyond customer requirements, there is also a varying set of other requirements that need to be considered, including: internal requirements, marketing requirements, performance requirements, and regulatory requirements for specific industries.

Writing effective requirements depends on having the right people involved for input, review participation, and general collaboration on the requirements document.

As **Erik Dietrich of DaedTech LLC** explains:

“Requirements must come to your group in some fashion or another. This could be anything from a cocktail napkin to user stories to reading customer feedback emails to formal, waterfall-style requirements documents/specifications. But they come from somewhere, and they’re almost always written down somewhere. It is absolutely critical that the software development team review these. And, as with the coding standards, review doesn’t mean, “read what’s being done to you.” Requirements should never be a one way street. This documentation absolutely needs

your feedback and push-back if it’s steering the effort on a course to budget overruns, slipped deadlines, or general failure. Just as you might have a business analyst review requirements for making sure they align with business goals, have the development team make sure that they won’t create undue problems with the software or that there isn’t a better, less effort-intensive alternative.”

Here’s a closer look at how other developers are using requirements to write better code:

27. *“Communicate about it. Whether it’s writing internal documentation, user documentation, discussing things with another developer, or just talking to a rubber duck. Just forcing the concepts into sentences can really help you think about them. Several times I’ve been like “this code seems ok” but then when I had to think through it and explain what it does I thought “no, that’s too embarrassing, I need to improve that.”*

Ticos answered on Hacker News

28. *“Firstly, a developer should understand the requirement correctly. Create a flowchart based on the requirement and developer understanding and get it approved from a business analyst or requirement person. Start coding — developers should start writing code, keeping in mind best practices based on specific tools. Get it reviewed with a peer to make sure code is matching the requirement flowchart. Unit testing and then code promotion.”*

Prabhakar Dukka

29. *“We can debate the many techniques for SD, but at the end, having good user and functional requirements along with good test coverage will eliminate the vast majority of the problems during and after the system is in production.”*

Jorge Norabuena

30. *“Throughout the project (requirements, specifications, implementation, and testing) learn to notice what you are not looking for. Also, always*

develop to facilitate the person who will be supporting your project without having you to consult with.”

Ed Boesel
VP, Software Engineering & Development
Fidelity Investments

Additional Resources:

- **Whitepaper:** [4 Popular Document Review Processes that are Costing You Big](#)
- **Article:** [The Case for Reviewing Requirements and Design Documents](#)
- **Article:** [What Technical Documents Should You Review?](#)



Learning, Education, and Skill Development

Life of a developer is a never ending learning process. Developers have the benefit of learning by doing. You learn from every project you work on and every line of code you write. As most developers understand, “best practices” can only get you so far. In fact, that idea was the inspiration for this resource — which is dedicated to offering advice from people who have learned from their own experience.

When we asked developers to share their best advice, we continuously heard the same message: the best way to write better code is to keep writing code and learn from each new project.

As you look to implement the tips outlined in this guidebook, consider how other developers are putting continued learning and education, front and center.



31. *“There is a fantastic book entitled “Clean Code”: This is my go-to book for writing good software. If you guys incorporated some of the metrics gleaned at within this book into your tool, such as tracking use of comments, ‘spaghetti-code’ severity, etc., these things would be awesome.”*

Michael DeFrancis
Senior Computer Engineer
Harris Corporation

32. *“My advice: Keep learning and growing with each project.”*

Ajoy Bhatia

33. *“Well...more of an academic exercise but for brand new developers I’d recommend they read the book “Code Complete 2”, to learn best practices.”*

Ben Norman

34. *“Know the technology you’re working with! Whatever the programming language is, whatever frameworks or tools you use (or have to use) make sure you take your time to learn how to use them properly. This also involves sharing your knowledge with your co-workers. Most often it’s not easy to find the time to do “just learning”. But management needs to understand that apart from getting the requirements right “knowing the technology” is the most important prerequisite for building better products.”*

Erik Tittel

35. *“Don’t blindly follow “best practice.” Think for yourself and understand the “why”.”*

Joshua Gerrard

36. *“Plan for and make time for learning.”* 

James Starr

37. *“I actually have only one piece of advice and it’s so obvious, I’m not sure it’s even worth verbalizing. All you have to do to write better code is to write bad code. A lot of it. The more the better.”*

Alexander Kalenyuk (Александр Каленюк)


38. *“The best advice I have is not a direct guideline for writing code as much as a broader thing to remember as a developer. When you see a potentially new and better way to do things, “That’s not how we do things here” is a non-answer. As a corollary to this, if you identify a tool or piece of software that will make you more effective, whether it’s an additional monitor, a testing framework, or something like Resharper, consider how long it will take in saved time for the company to make up that expense. It’s generally insignificant. And if it prevents a single bug getting to production, you’ve almost certainly saved hundreds or thousands of dollars in time it would take to fix the bug. Schedules are best guesses because*

each project is unique. Allow for wandering around time. This is the only way that you will discover new things. Celebrate what you learn from failures. If you can't fail, you can't try new things."

Robert Lucente

39. *"Learn to type, so that you are not afraid of using meaningful variable names and including useful comments. Read old stuff: Jon Louis Bentley's "Programming Pearls" and "Writing Efficient Programs"; Knuth's "Literate Programming". The Multics papers. There are few really new problems, and even those that look new are things that people didn't even bother looking at before because they seemed too big. Very smart people have been thinking about computational problems for a long time, often with tighter resource constraints, and failing to learn from them would just be a waste. My favorite aphorisms on programming come from Mark Twain. In the essay "Fenimore Cooper's Literary Offenses" he discussed some "rules governing literary art".*

Harlan Rosenthal

40. *"Practice. You cannot get better at coding by just reading blogs or books. You have to get your hands dirty trying things. Always try to keep things simple and short and choose good names. You may have to come back to your code months later, and being able to quickly grok what you wrote pays dividends. Pick a side project working on something you enjoy. If you work at a corporate job during the day, you may not always have a good opportunity to grow and try new technologies."* 

Tyson Maly answered on HackerNews
Founder, BestFoodNearMe.com
Twitter: @tysonmaly



41. *“Go read open source libraries. I started out barely knowing anything in JavaScript and I had terrible code, as almost any dev could say. However, I was never afraid to follow the debugger through the code of the libraries I used. At the time these were things like Backbone and Marionette, which have incredibly clean and well documented code. Find a library you use and have a feel for how it works, and read through their code. If you run into questions about why something is done a certain way, they’ll tell you! Use Git-blame and see what the commit message was for those lines! Often there will be a description or issue tied to it and you can figure out the thought process they used. Then just start applying similar thinking to problems you encounter.”*

Matt Polichette Answered on HackerNews
Software Engineer at SpinGo
Twitter: @Matt_Polichette

42. *“Learn how to type well (>30wpm). Literally everything else is secondary. You don’t have to be the fastest or most accurate typist, but not having to look at the keyboard while typing pretty quickly and*

accurately is foundational. Not having to use the mouse for common tasks (closing a window, switching applications, etc.) is another core skill. Everybody else’s comments focus on important stuff but I’ve seen programmers absolutely hobbled by not having these basic skills.”

Answered on HackerNews

43. *“Study the source code of major open source projects & try to contribute — you will see techniques for better code in their natural habitat, and through code review.”*

Bahamut HackerNews Answered on HackerNews
Web Developer

44. *“The number one thing you can do to write better code is get together a group of people and do code reviews on each other’s code constantly. Have people in the group point out things they like and dislike about the code,*

along with “why.” Then create a “cheat sheet” where you try to summarize the principles behind all the suggestions people make for your code. Beyond that, try to find “good” code; look at how it is structured. Then try to emulate the practices in regards to language, feature, use, layout, etc.”

CuriouslyC Answered on HackerNews

Additional Resources:

- **Resource Center:** [Code Review](#)
- **Resource Center:** [Code Profiling](#)
- **Blog:** [SmartBear Developer Blog](#)



Code Review and Profiling

Just like writers need editors to catch their mistakes, developers depend on code review to ensure code is defect free. A code review involves one or more developers examining source code they didn't write and providing feedback to the authors, both negative and positive.

Among the types of bugs found during code reviews:

- Missed test cases: “You tested for an empty string, but not for null.”
- Typos
- Un-optimized code
- Unnecessarily complex code
- Re-implemented code: “We already have a function that does what you're doing here, and does it better.”
- Lazy documentation

It shouldn't come as a surprise that when SmartBear surveyed developers earlier this year, we found that code review is looked at as the #1 way to improve code quality. We received similar feedback when we asked developers to share their best advice.

Or, as one developer put it, “code review is a must.”

Implementing a code review process starts with selecting the right tools for your development team. For some projects, teams may rely on the pull request functionality of an open source tool like GitHub. Larger organizations, with development teams working across different locations, often need more robust functionality, as well as the ability to integrate the tool with their current environment. This is where a collaboration tool like SmartBear's Collaborator can come in handy.

In addition to implementing tools for code review, it's also important to analyze, debug, and profile code to determine the most effective way to make it run faster. How can software developers and quality engineers ensure that their code is quick, efficient and ultimately seen as valuable? The solution lies in using a profiling tool to examine an application's code and locate and eliminate performance bottlenecks. These tools can quickly diagnose how an application performs and enable programmers to zero in on areas of poor performance. The result is a streamlined code base that performs at, or exceeds, customers' expectations.

Here's a closer look at how other developers are using code review and profiling to write better code:

45. *“Perform code reviews which will not only find bugs/issues faster but it will also help them to learn productive programming skills.”*

Asim Israr


46. *“If I had one piece of advice, it would be to write comments that reflect intent, not what the code does, but what its purpose is. Whomever stumbles across this code in the future has one question in his mind – should I look closer at this code? Comments are potentially forgotten as the code changes. But the intent and purpose of the code usually do not change. Write clear code, and comment it with purpose. Let us know the purpose, so we know whether to spend time looking at it or move on. Read other people's code. It's hard to learn in a vacuum. 2) Have your own code reviewed. Learn from others. 3) Test your code. This forces one to write simple, non-dependent code.”*

Mike Wilson

47. *“Engage in meaningful code reviews. There is a simple process to holding code reviews for the reviews to be fruitful and not argumentative. You code review a single module/sub-system, a constrained focus. Optimally you are reviewing exactly one member's, or exactly one pair's, work. You select the teammates who will participate in the review in advance.”*

48. *“The participants are given several hours, or a day, in advance to prepare. Each member independently reviews the code and physically notes (pen & paper, word doc, etc) anything they see worthy to discuss. At the code review, each member brings a physical paper to the review with their notes. Each participant discusses their notes, aspects that are called out by multiple reviewers are then discussed in more detail and are specifically provided a course of action to either remedy or to take no action. Points that are documented by only one or two members are most likely personal code style opinions and are generally not acted upon. Holding a code review in this manner ensures that each*

person truly did the review. It's too easy to go to a code review and allow one or two people to speak and just sit back and zone out. The physical documentation also showcases if a person did not take it seriously."

49. *"Every member shows up with a full page or two pages and another brings two sentences. It enshrines accountability to the team. It eliminates code reviews being arguments over wholly subjective aspects. Finally, it aids a personal feeling of improving the team as opposed to changing your code solely because "you were told to." It's hard to feel ill will towards your team when multiple members all point out the same items."* 

Chris Marisic

Senior Software Engineer Consultant

Twitter: @dotNetChris

50. *"Developers should always code thinking about the worst case scenario. I tell my team that you should code first for the error scenarios and then happy path, which actually fall in place by then. Code review is a must."*

Sreekumar Kadali

51. *"Optimized code is a one-way street. Usually, there is no going back or extending the code. Readability, maintainability, and reliability is what should get truly optimized. My teams run a profiler when we do have performance issues and it's often very surprising what the culprits are. When we do optimize code, we retain the simpler, less optimized version and test the two against each other for results, the retained readable version is used to both test against optimized versions and sometimes switched back in when we suspect bugs. As such, our code typically has many performance switches that can be flipped for debugging purposes."*

Peter Pupalaikis
VP, Technology Development
Teledyne LeCroy

Additional Resources:

- **eBook:** [10 Things Developers Wished Their Bosses Understood About Code Review](#)
- **Case Study:** [How Whirlpool Does Streamlines Its Code Review Process](#)
- **ROI Calculator:** [Understanding the Value of Code Review](#)
- **Article:** [How to Choose a Code Profiling Tool](#)



Collaboration on Development Teams

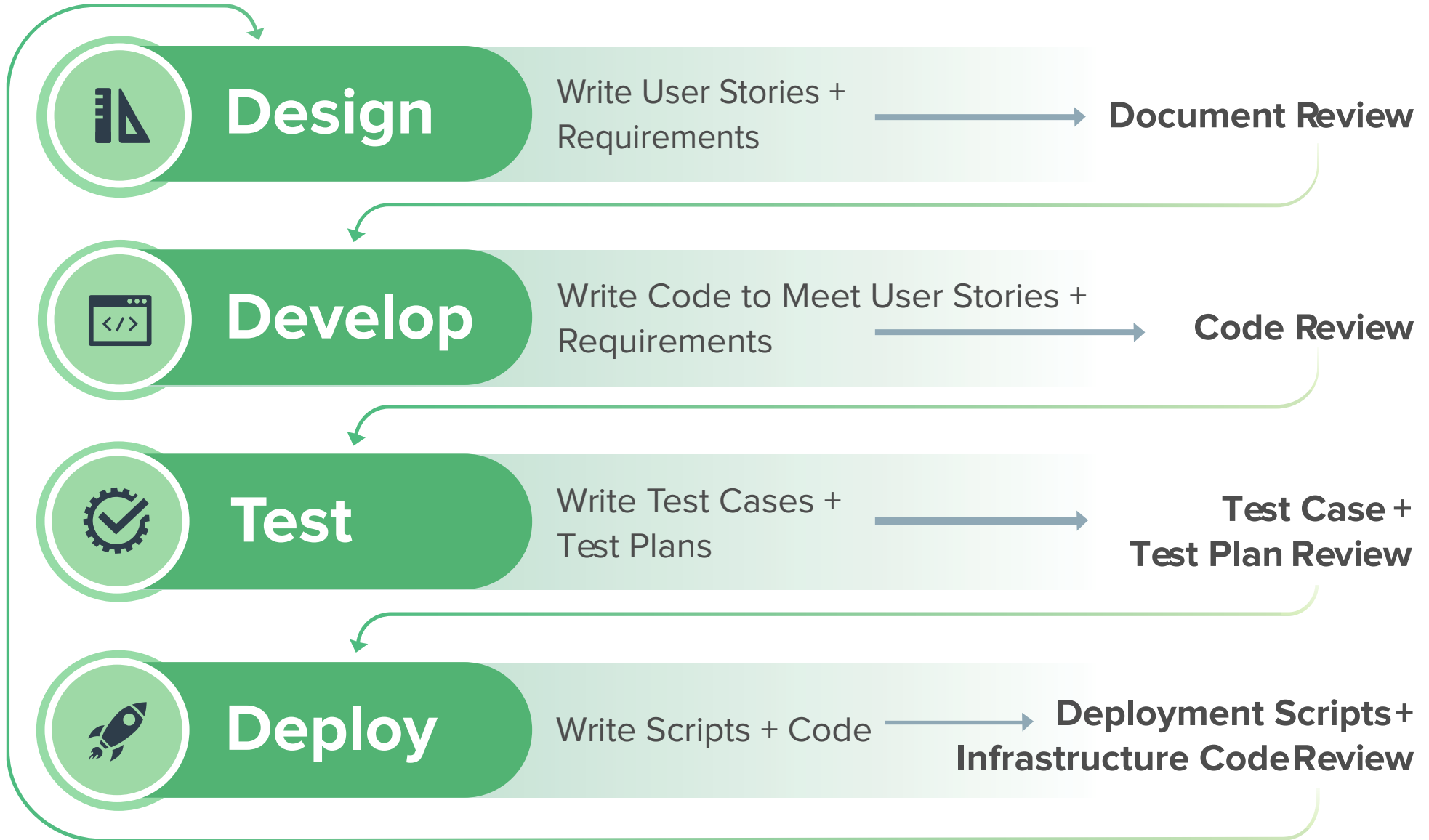
Software quality is driven by collaboration throughout the entire development lifecycle. At each phase of the development process, collaboration is critical. You may already be doing some form of collaborative software development, but improving the efficiency of this process brings added speed, quality, and agility.

Developer collaboration can encompass the following:

- Idea creation
- User stories
- Design documents
- Product requirements
- Source code
- Test artifacts
- Deployment

In the idea generation phase, a good room and a white board might be all that is needed. But as user stories and requirements are written, the ability to provide feedback and track changes becomes even more important. As you can see from the graphic on the next page, collaboration truly spans across all stages of the software delivery lifecycle — from design to development, and test to deployment.





Here's what developers had to say about the importance of collaboration.

52. *"Discuss and agree on what "BETTER" means to the team. Some teams (moon lander) want 101% perfect code. Some teams (???) want anything running now."*

53. *"Writing Better is an iterative learning process, that writers go through."*

Paul Gerken

54. *"The short answer: help each other out."*

55. *"If it's an individual developer, I'd keep asking on StackOverflow or posting on any developer forums, or possibly even learning from YouTube videos on a module to code. If it's a team of developers, it's possibly even better since they all can collaborate and learn about some things (An example is Developer A and Developer B working on two separate things. Possibly Developer B is working on some module that Developer A has written before. Developer B asks Developer A about how to implement this similar module in order*

to save time. Then, Developer B will have two options with the extra amount of time: 1 - work on something new that will benefit the software 2 - if the code works, but is not optimized, find a better optimization technique... which will definitely benefit Developer B, and possibly Developer A, in the future.) Coding is a craft that takes practice, and each time something functional is written, there will be some sort of bug, always, non-avoidable. It doesn't matter how good the developer is at the end of the day. It all depends on how much the developer is willing to learn and grow from the experience."

Brian Pham


56. *"Try to enhance your work experience by trying to do other developers' work. If you are a structured code writer, be a OOP coder and vice versa. If you are a Web developer, try to write a desktop application. If you are a team leader, try to be a member of your team for one day. If you didn't write SQL, OLAP, warehousing code, and such things then you are missing a lot of fun. If*

you are a Web administrator, try to write a trojan, that will expand your experience. If you didn't work with a QA team then you can't be professional developer. Try to get involved a little, make some decisions, have an opinion, try to be the one in charge for a change and don't tell me that you are afraid of going big..."

Shomaf Saheb

57. *"If you start going down the road to create a new form (or page) that has a new nifty or changed UI behavior, be sure to have a plan that updates all other forms or pages that need to behave the same way. Users get frustrated if they use buttons, keystrokes, gestures (as examples) a certain way on one form only to go to a different form and learn a different method that actually does the same thing. It's not only frustrating, but a user's quality of work goes down as well. The bottom line goal is to give users ways to improve the quality of their own work."*

Arlan Privett
Software Developer
nQativ

58. *"Find your strength, nurture it and learn to collaborate by embracing the mixing of others bringing their strengths. I started coding when I was in high school and have been doing tech jobs through now being over 50. So many times people want tech guys to know anything tech. While we may be the most qualified on the team in some cases, that doesn't mean we know everything about tech any more than doctors know everything about health. (Even if they are the most qualified health person in the room.) Be content to be honest about your limitations and don't lose one for the team being the best man for the job when you are not the right man for the job."* 

John Farrar
CEO, SOSensible Group LLC
Twitter: @SOSensible

59. *“My personal pet peeve, don’t assume the person before you did it wrong or was an idiot until you can prove it. Generally devs do something, even if it is not ideal, for a reason. Be it, business changing requirements at the last minute, or schedule crunch or a lack of domain knowledge. But before you call them an idiot or assume they were wrong, try to figure out what they did and understand why they might have done it. I have seen a lot of code get changed by new comers only to find out it was written that way for a specific use case that they just didn’t understand yet. Then stuff breaks, people get upset and the whole team gets frustrated.”*

Mark Davis answered on HackerNews
Software Architect
Twitter: @davismwfl

60. *“Work with other developers, ideally next to them, on the same project. Learn from them no matter you think them “good” or “bad”.”*

Guillaume Piolat Answered on HackerNews
Founder, Auburn Sounds
Twitter: @p0unce

Additional Resources:

- **eBook:** [A Guide to Improving Code Quality & Collaboration](#)
- **Blog:** [Defining Developer Collaboration](#)
- **Blog:** [How Collaboration Humanizes the Enterprise](#)
- **Tool:** [Learn more about how Collaborator streamlines collaboration throughout the SDLC](#)



Revisit Your “Writing” Process

Throughout this guidebook, we’ve look at different areas of the software development process and provided insight into the different areas where you, and your team, can invest to improve the quality of the code you write on a daily basis.

In the final section, we wanted to take a look at tips that specifically address how you can adjust your approach to writing code. Some of these tips will focus on how you structure code. Others will discuss areas for you to focus when it comes to optimizing code.

61. *“Spend some time to structure your code before coding. It helps to avoid rewrites shortly afterwards.”*
62. *“Model behaviour in interfaces and data in classes. Always prefer immutable data classes.”*
63. *“When designing a class/interface, try to separate different concerns. For example, if you read file from disk, parse it and create an object from it in a single class. It’s probably worth considering three classes and two interfaces instead.”*

64. *“Do automated deployment right from first build.”*
65. *“If something takes more than 30 minutes a day 5 days in a row it’s worth automating.”*
66. *“Always favour constructor injection over field injection (except in tests).”*
67. *“Interfaces terminate dependencies. According to research, using interfaces results in a statistically-significant correlation between minimising ripple effect when changing the code (ripple effect refers to number of changes you have to do elsewhere to accommodate/adjust after changes in current class). Therefore always prefer injecting interfaces over classes.”*
68. *“Always isolate external resources (file system access, network access etc.) with something mockable. This way testing is simplified.”*
69. *“Metrics and logging are important as there is no other way to troubleshoot production.”*

Kirill Frolov

70. *“I always advise my team: Act in haste, repent forever; program in haste, debug forever.”*

Touseef Afzal

71. *“Developers should always follow modular approach (Don’t repeat yourself approach.)”*

72. *“Use coding conventions (for resource optimization and organization standards.)”*

73. *“Use design principles wherever required but it should not be overkill.”*

Asim Israr

74. *“My best piece of advice is to use a type safe language and structure your code so that as much as possible is strongly typed, so that the compiler will catch errors at compile time. The earlier you catch a bug, the easier and cheaper it is to fix.”*

Joshua Frank

75. *“Optimize Late (and Sometimes Never...) I should preface my statements by saying that I mostly work on embedded, numerically oriented code, like digital signal processing, and applied math applications. When my team and I develop code, we follow a principle of optimization resistance. This is because: Many times code that looks incredibly readable (and therefore to some, sub-optimal in performance) is nearly optimum or optimum enough. And the reverse: Many times code that has been obfuscated through optimization not only doesn’t perform better, it might not even be a substantial contributor to sub-optimum performance of the system as a whole.”*

Peter Pupalaikis

76. *“After 25 years of programming I’ll give you the lesson my first boss gave me: “Think more, type less”. It’s stood the test of time, I’ve passed it on to many juniors.”*

Dazzawazza answered on HackerNews

77. *“Follow the data. Use the tools & processes that enable you to make convincing, correct decisions based on real data. If you’re unsure of which feature to build next, find out what your users want. If there’s a difficult bug, get as much data (crash logs, user reports, screenshots, videos etc) as possible before you start attacking it, there’s always a pattern. Ignorance = failure.”*

Davie Janeway

78. *“Write out pseudo code for what needs to be done. Work on something else for a few days. Review the pseudo code to make sure it makes sense. Begin developing. Stop before being done with the coding and work on something else for a few days. Come back to the development and make sure it still makes sense; refactor if necessary. Finish development and then work on something else for a few days. Review the development and refactor for clarity and efficiency. Sleep on it then review again and refactor for clarity and efficiency. Let a weekend pass then refactor again for clarity* and efficiency. * by clarity, I mean avoiding deeply nested code (>3*


levels), and using variable names that say what they are.”

Louis Bergsagel
Software Developer
SanMar

79. *“Develop a proper inspiration plan and then act upon it so as to attain the desired output results. The more accurate will be the plan the more it provides the desired results. To get the desired amount of appreciation it’s important to must add new features during the development and coding stage.”*

Silvy Novita

80. *“Don’t get cute. Code golf is for competitions, not production code. Even if something takes 10 lines that you can do in 1, if 10 lines is clearer, do it that way (you’ll thank yourself in 6 months when you have to figure out what you were doing). Keep your functions to one or two pages. Keep your nesting to three levels, preferably*

two. If you have multiple nested if/while/for sections, break them up. (See: “cyclomatic complexity”). Follow the coding conventions of the project you’re working on, even if you disagree with them, consistency is more important.” 

Hans Van Slooten answered on HackerNews
Principal Developer, Baseball-Reference.com
Twitter: @cantpitch

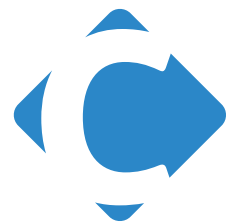
Put these tips to work

From selecting the right tools to investing in continued education and collaboration, there are no shortage of areas for your team to invest if you want to write better code. Share this resource with your development team and see what tips they have for writing better code.

One of the most important components of a collaborative approach to software development will be document review and code reviews. An effective code review process requires a tool that enables collaboration across teams and across locations. A dev collaboration tool like Collaborator has helped industry-leading organizations like Cisco, Everi Games, and Whirlpool improve code quality and can integrate perfectly within your development team, regardless of systems or processes that you’re currently using.

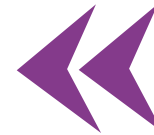
TRY COLLABORATOR FREE TODAY!

With the right tools and best practices, your team can peer review all of your code



Collaborator

TRY IT FOR **FREE** TODAY



SMARTBEAR

Over 4 million software professionals and
25,000 organizations across 194 countries
use SmartBear tool

4M+
users

25K+
organizations

194
countries

[See Some Successful Customers >>](#)

API READINESS



Functional testing through
performance monitoring

[SEE API READINESS
PRODUCTS](#)

TESTING



Functional testing,
performance testing and test
management

[SEE TESTING
PRODUCTS](#)

PERFORMANCE MONITORING



Synthetic monitoring for API,
web, mobile, SaaS, and
Infrastructure

[SEE MONITORING
PRODUCTS](#)

CODE COLLABORATION



Peer code and documentation
review

[SEE COLLABORATION
PRODUCTS](#)



Collaborator