TOP 5 BEST PRACTICES FOR SOFTWARE DEVELOPMENT

Breaking down all of software development to simple best practices isn't easy—or even all that possible. The way one engineer approaches work is completely different from the next.

But certain practices and guidelines unite developers and product managers. With these guidelines put in place, software development becomes a smoother process for all!

Here are five best practices in the world of software development. If you keep it simple, commit regularly, and thoroughly test your work, you will have a better time building your code and delivering quality software efficiently and on schedule.

Let's take a look.

Keep it simple

There's a temptation to make code that is overly complicated to read, full of strings which you *might* need in the future, or just a general mess. Writing complex and esoteric code might have been fashionable 30 years ago—showing off all your sophisticated skills—but now code needs to be clear and efficient.

Since Max Kanat-Alexander, software developer for Google, said we should reduce complexity to simplicity, there has been a cultural shift to creating streamlined code. If you ever doubt the quality of your code, remember these two principles:

- DRY

- YAGNI

DRY: Don't repeat yourself

Don't Repeat Yourself. Computers are smart. They're not like humans who constantly need to be told something in order to remember it. Tell your computer once and it will know it until you tell it to forget it.

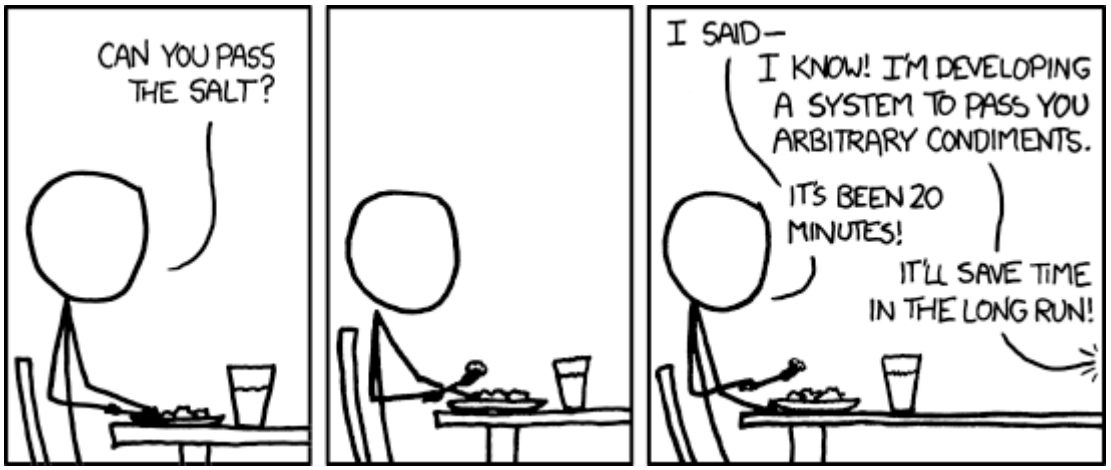
```
class Repeat
  def print_message
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
    puts "I Will Not Repeat My Code"
  end
end
```

Don't repeat yourself for the sake of repeating yourself (Source)

That's why duplication is waste. Whether that's waste lines in code or wasted time in the process, repeating yourself is just wasting time and putting greater strain on the code, your budget, and you. The DRY principle might not apply in every single instance, but it's a great guideline.

In sum: Avoid repetition—instead, look at abstraction, automation, and intelligent implementation of code.

YAGNI: You are not gonna need it



Just because it might save time in the future doesn't mean you need to make it now (Source)

You Are Not Gonna Need It. Would you build a bridge over a small stream in case it becomes a crashing white water river? Probably not, right? Because that's unnecessary, even if it *might* save headaches later down the line.

If you don't need a piece of code now, don't include it. Always focus on the task at hand and don't try to second guess what the future will bring. You're a programmer, not a mystic. Even if you were correct in guessing where the project would eventually lead you, there's a possibility that you'll find

it isn't exactly how you planned it.

Besides, adding code that isn't immediately important wastes time and resources that you *probably* can better spend elsewhere.

Have a backout plan

Experimenting and changing things as you go is an easy way to cut down on revision time later. But it is also a surefire way to take your code far in the wrong direction, with no easy way back.

Instead, commit your work often and regularly. This way you'll always be able to rewind to a point where you hadn't got lost in the weeds.

This is a habit, one that is very much something you don't really appreciate the value of until it's too late. If you don't keep committing, you could be adding days or weeks to your projects as you try to find the error, correct it, and then start again on a different path. Even there, there's no guarantee that the new path will be the right one—that's the perfect moment to make a commit, anyway!

(Learn how to set up a CI/CD pipeline.)

Test, test, and test again

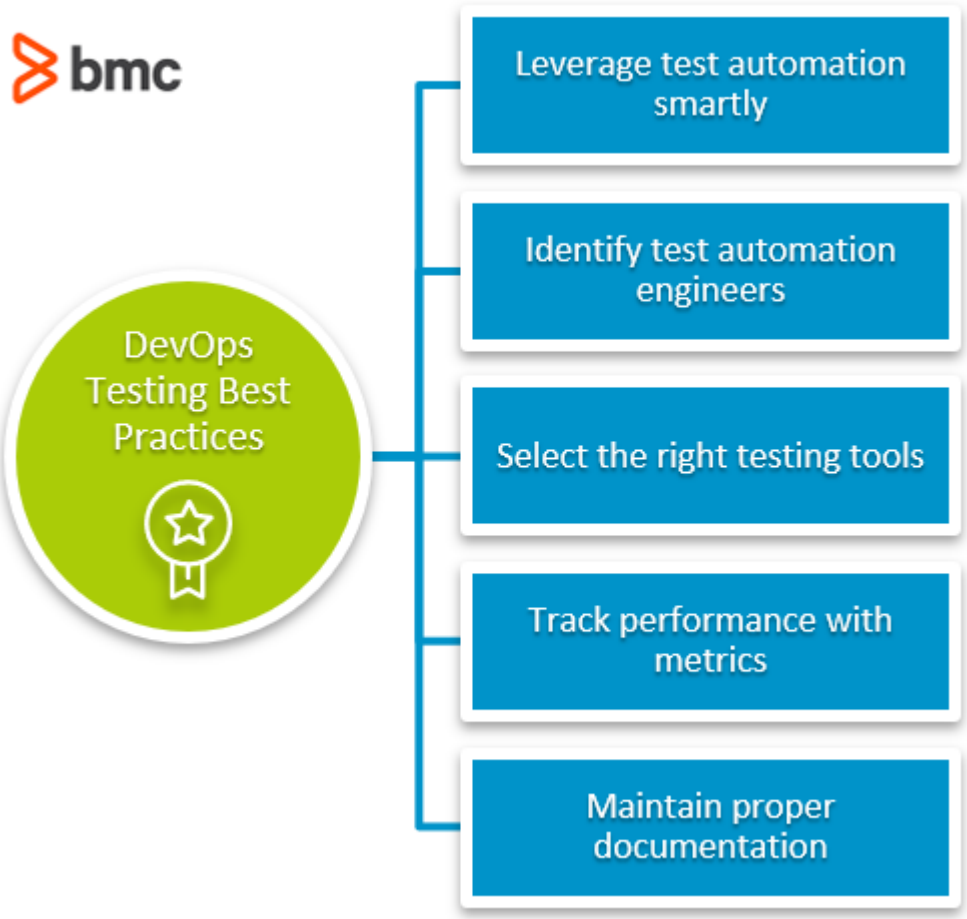
Big software companies don't wait until all of their code is in place before they put it through rigorous testing—neither should you. Continuous testing over the long-term will give you a better understanding of:

- The code you've already created
- What you still have to do

Few things are more difficult than looking at what you've written and trying to pinpoint the single error that is causing a nasty bug in your system. Even if you are prone to three-day, coffee-fueled writing sessions, you need to build testing into your workflow.

Get started with continuous testing by:

- Understanding the key concepts in DevOps testing.
- Considering shift left testing, which helps find and prevent detects early in the software development lifecycle.
- Automating testing at regular intervals throughout the process will help you get much needed feedback on your possibly buggy code. Let bots point out your mistakes and then you can focus on correcting them and moving your project forward.
- Exploring the growing world of testing as a service (TaaS). If automated or large-scale testing isn't possible, TaaS is particularly useful for small companies or teams with too much on their plates.

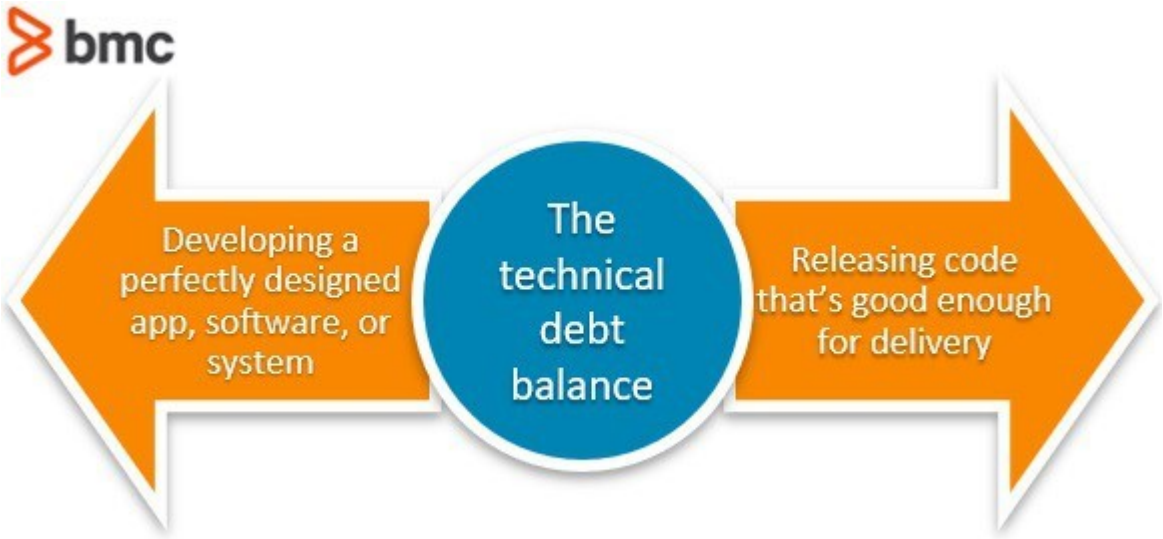


Understand how to estimate

When you are managing a software project, whether as a part of a development team or as a freelancer, you need to have realistic goals about your time requirements and budgeting. Making quality code isn't just writing the code—it's taking the time to write it well and revise it to be even better.

If you are rushed into finishing jobs, you will write less-than-standard code. Unrealistic time constraints do no one any favours and they lead to compounding technical debt.

Make agreements that allow you the time to develop properly. Both you and your client will be happier as a result.



Make it rugged

Remember our first best practice: Simplicity is king in software development. But you don't want your software to be so simple that it breaks. There needs to be an element of ruggedness to the code you write. We want it to be both:

- Difficult to misuse
- Kind to human errors

If your software is unforgiving in these ways, it will be much more difficult for end users to implement properly.

Creating code that is user friendly and unlikely to break (or be misused and confusing) is key for successful rollouts. After all, if users can't figure out how to use it, where's the value?

The importance of best practices

Bringing in best practices will help you make the most of your development process without reinventing the wheel. Make your code simple to read, simple to implement, and simple to use.

Every business needs best practices in order to ensure efficiency of time and money, and this is certainly true of developing software products.

Related reading

- BMC DevOps Blog
- Managing Containers & Code for DevOps, part of our DevOps Guide
- Deploying vs Releasing Software: What's The Difference?
- DevOps Engineer Roles & Responsibilities
- How & Why To Become a Software Factory
- Python vs Go: What's The Difference?
- Top DevOps Conferences To Attend