books kim
book summaries for easy reading

# The Pragmatic Programmer: From Journeyman to Master

By Andrew Hunt, David Thomas

# Book summary & main ideas

*MP3 version available on www.books.kim*
*Please feel free to copy & share this abstract*

## Summary:

The Pragmatic Programmer: From Journeyman to Master is a book written by Andrew Hunt and David Thomas. It provides guidance for software developers on how to become better programmers, as well as advice on how to work more effectively with other members of the development team. The book covers topics such as debugging, refactoring, design patterns, testing strategies, version control systems, and project management techniques. It also includes tips on communication skills and career development.

The first part of the book focuses on developing good programming habits. This

includes understanding the importance of writing clean code that is easy to read and maintain; using source control systems; learning new technologies quickly; being aware of potential problems before they arise; taking responsibility for one's own mistakes; and staying up-to-date with industry trends. The authors also discuss ways in which developers can improve their productivity through automation tools.

In the second part of the book, Hunt and Thomas provide advice about working within teams. They emphasize collaboration between different roles within a team (e.g., designers vs coders) in order to create successful projects faster than if each role worked independently from one another. They also discuss methods for dealing with difficult people or situations that may arise during a project's lifetime.

Finally, Hunt and Thomas offer guidance about career advancement opportunities available to software developers who are looking to move up in their field or switch careers entirely. This section covers topics such as networking events, job interviews, resume building tips, negotiating salaries/benefits packages etc.

Overall The Pragmatic Programmer: From Journeyman To Master provides valuable insight into becoming an effective programmer both technically speaking but also when it comes down to working collaboratively with others in order achieve success together.</

## Main ideas:

***#1. Understand Your Tools: Take the time to learn the tools you use, and use them to their fullest potential. This will help you to be more efficient and effective in your programming.***

Understanding your tools is an essential part of being a successful programmer. Taking the time to learn how to use them properly and efficiently can save you hours of frustration and wasted effort. Knowing the ins-and-outs of your programming language, development environment, debugging tools, and other software will help you write better code faster. It also allows you to take advantage of features that may not be immediately obvious or intuitive.

For example, if you are using a text editor such as Vim or Emacs, learning all the keyboard shortcuts available can drastically reduce the amount of time it takes for you to edit files. Similarly, understanding how different libraries work in your language can make coding much easier by allowing you to quickly access functions without having to look up their syntax every time.

In addition, taking the time to understand what each tool does best will allow you to choose which one is most appropriate for any given task. This means that instead of spending extra time trying out multiple solutions until one works correctly, you'll already know which tool is best suited for the job at hand.

***#2. Donâ€™t Live with Broken Windows: Fixing small problems quickly can prevent them from becoming bigger problems later on. This will help you to maintain a high quality of code and keep your projects on track.***

The idea of Dont Live with Broken Windows is an important concept for any programmer to understand. It suggests that small problems should be addressed quickly and efficiently, before they become

bigger issues down the line. By taking care of these smaller issues as soon as possible, you can ensure that your code remains high quality and your projects stay on track.

This idea is especially relevant in todays world where technology moves at a rapid pace. If you dont take the time to address small problems right away, they can quickly snowball into larger ones that are much more difficult to fix. This could lead to costly delays or even complete project failure if not handled properly.

By following this advice from The Pragmatic Programmer: From Journeyman to Master by Andrew Hunt and David Thomas, programmers can save themselves a lot of headaches in the long run. Taking care of minor bugs early on will help keep their projects running smoothly and prevent them from becoming

major roadblocks later on.

***#3. Plan to Throw One Away: Donâ€™t be afraid to scrap a project and start over if itâ€™s not working. This will help you to avoid wasting time and resources on a project that wonâ€™t be successful.***

Planning to throw one away is an important concept for any programmer. It means that if a project isnt working, its better to scrap it and start over than to waste time and resources trying to make something work that wont be successful in the end. This can be difficult because of the amount of effort put into a project, but its often necessary in order to create something truly great.

When planning out projects, programmers should always keep this idea in mind. If they find themselves stuck on a problem or

unable to move forward with their current approach, they should consider scrapping what they have and starting from scratch. This will help them avoid wasting time on something that wont turn out well.

Its also important for programmers not to get too attached to their projects. They need to remember that sometimes throwing one away is the best option for creating something amazing. By being willing and able to let go when needed, they can ensure their projects are successful.

***#4. Donâ€™t Panic: When faced with a difficult problem, take a step back and think through the problem logically. This will help you to find the best solution and avoid making mistakes.***

When faced with a difficult problem, it can be easy to panic and make mistakes.

However, the best way to approach any problem is to take a step back and think through the issue logically. This will help you identify the most effective solution without making costly errors. The Pragmatic Programmer: From Journeyman to Master by Andrew Hunt and David Thomas emphasizes this idea of "Don't Panic" when dealing with complex problems.

The authors suggest that taking time to analyze the situation before jumping into action can save valuable resources in terms of both time and money. They also point out that it is important not only to consider all possible solutions but also their potential consequences before deciding on an appropriate course of action.

By following this advice, you can ensure that your decisions are well-informed and

based on sound logic rather than rashness or emotion. Taking a few moments for careful consideration may seem like an unnecessary delay at first, but in the long run it will pay off as you avoid costly mistakes while finding more efficient solutions.

***#5. Use Tracer Bullets: When starting a new project, use tracer bullets to test out the different components. This will help you to identify any potential problems before they become too difficult to fix.***

Tracer bullets are a useful tool for testing out the different components of a project before it is completed. By using tracer bullets, developers can identify any potential problems early on in the development process and address them before they become too difficult to fix. This helps to ensure that projects are

completed efficiently and with minimal issues.

Tracer bullets work by allowing developers to test out each component of their project individually. This allows them to focus on one part at a time, ensuring that all aspects of the project have been tested thoroughly. Additionally, this method also allows developers to quickly identify any areas where further attention may be needed.

Using tracer bullets when starting a new project is an effective way for developers to ensure that their projects will run smoothly and without issue upon completion. It also helps save time and money by identifying potential problems early on in the development process.

## *#6. Prototype to Learn: Use prototypes to explore different*

***solutions and learn more about the problem. This will help you to find the best solution and avoid wasting time on solutions that wonâ€™t work.***

Prototype to Learn is a concept that encourages developers to use prototypes as a way of exploring different solutions and learning more about the problem. By creating multiple prototypes, developers can quickly identify which solutions are viable and which ones wont work. This helps them save time by avoiding spending too much effort on ideas that dont pan out.

The idea behind Prototype to Learn is that it allows for rapid experimentation with different approaches. Developers can create simple models or mockups of their proposed solution and test it in various scenarios without having to invest too much time or resources into building

something from scratch. This also gives them an opportunity to get feedback from users early on in the process, allowing them to refine their design before committing any significant amount of development effort.

Using this approach, developers can iterate quickly through potential solutions until they find one that works best for their particular situation. Its an effective way of ensuring that the final product meets user needs while minimizing wasted effort along the way.

***#7. Program Close to the Problem Domain: When programming, try to use language and concepts that are close to the problem domain. This will help you to create code that is easier to understand and maintain.***

Programming close to the problem domain

means writing code that is as close as possible to the language of the problem. This helps ensure that your code is easy to understand and maintain, since it will be written in terms familiar to those who are working on or with the project. For example, if you are programming a system for managing customer orders, use words like "order" and "customer" instead of more generic terms like "object" or "entity." By using language specific to the problem domain, you can make sure that everyone involved in developing and maintaining your code understands what it does.

Using concepts from the problem domain also makes coding easier by allowing you to take advantage of existing knowledge about how things work in that particular area. If you know how customers interact with an order management system, then you can write code based on this understanding rather than having to learn

new concepts from scratch. Additionally, when other people come across your code they will already have some familiarity with its structure because it follows conventions used within their field.

Overall, programming close to the problem domain allows developers and maintainers alike greater insight into what a program does without needing extensive documentation or training materials. It also reduces development time by taking advantage of existing knowledge about how things work within a given context.

***#8. Estimate to Avoid Surprises: Estimate the time and resources needed for a project to avoid surprises. This will help you to plan ahead and ensure that the project is completed on time and within budget.***

Estimating the time and resources needed

for a project is an important step in avoiding surprises. By taking the time to accurately estimate how long it will take to complete a project, you can plan ahead and ensure that all necessary tasks are completed on schedule. Additionally, by estimating the cost of materials or services required for the project, you can make sure that your budget remains intact.

Accurate estimates also help to identify potential problems before they arise. If there are any areas where additional resources may be needed or if certain tasks require more time than initially anticipated, these issues can be addressed early on in order to avoid costly delays down the line.

By taking the time to properly estimate a project's timeline and costs upfront, you can save yourself from unexpected surprises later on. Estimate wisely and

plan accordingly â€“ this will help ensure that your projects remain successful!

***#9. Refactor Mercilessly: Regularly refactor your code to make it more efficient and maintainable. This will help you to keep your code clean and make it easier to add new features.***

Refactoring mercilessly means regularly taking the time to review and improve your code. This can involve restructuring existing code, removing redundant or unnecessary elements, and making sure that all of the components are working together in an efficient manner. It also involves ensuring that your code is maintainable; this includes making sure it is well-documented and easy to understand for other developers who may need to work with it.

By refactoring your code on a regular

basis, you will be able to keep it clean and organized while also improving its performance. Additionally, by keeping your code up-to-date with modern coding standards, you will make it easier for yourself or others to add new features without having to rewrite large sections of the program.

***#10. Test Early, Test Often: Test your code regularly to identify any potential problems. This will help you to ensure that your code is working correctly and avoid any unexpected issues.***

Testing early and often is an important part of the software development process. By testing your code regularly, you can identify any potential problems before they become major issues. This will help to ensure that your code works correctly and avoid unexpected errors or bugs.

Regular testing also helps to improve the overall quality of your code by catching small mistakes before they become bigger ones. It allows you to make sure that all features are working as expected and that there are no hidden issues lurking in the background. Additionally, it gives you a chance to refactor or optimize parts of your code if necessary.

Finally, regular testing helps to reduce stress during the development process by giving developers peace of mind knowing that their work is being tested on a regular basis. This can help them focus more on writing better code instead of worrying about potential problems down the line.

*Thank you for reading!*

*If you enjoyed this abstract, please share it with your friends.*

Books.kim