

# **DevCamp**

## **Coding Foundations course.**

**2024, December.**

**Content extracted and abstracted from:**

**Module 3. Development Skills**

**Module 4. Management Capabilities**

**Module 5. Team-working**

# Module 3. Development Skills

## Index

- Development Skills You Need.
- Are Developers Born or Made? Debunking the Myth of Prodigies
- Can You Lose Your Coding Skills?
- Cramming vs Consistent Study and How to Build a Study Framework
- How to Study Effectively to Improve as a Developer
- Guide to Developer Soft Skills
- Discovering the Tipping Point as a Developer
- What Does it Take to Become a Great Developer?
- How to Learn a New Programming Language
- Getting Past Skill Plateaus
- How to Practice Programming Techniques and Improve as a Developer
- How to Study and Understand Complex Topics
- How to Use Deep Work to Improve as a Developer
- Is Reading Important for Developers?
- Guide to Memorization
- Slowing Down to Learn How to Code Faster
- Mental Models for Learning How to Code and Improve as a Developer
- Practical Ways to Use the Pomodoro Technique as a Developer
- Development Study Tips: Reverse Note Taking
- Task Switching Costs for Developers
- The Power of Making Mistakes – Learning by Failing
- Learn How to Code from Scratch – A Practical Strategy
- Developer Learning Curve – Why Learning How to Code Takes So Long?
- Three Practical Approaches to Task Management

- Speed Reading for Developers
- From Copy Paste to Comprehension
- How to Break Through the Cycle of Procrastination
- Introduction to Asynchronous Method Calls
- An Introduction to AI and How Artificial Intelligent Agents See the World
- Guide to the Model View Controller Architecture
- Object Oriented Inheritance Tutorial for Developers
- Guide to CSS Selectors for Web Developers and Designers
- Best Practices vs Creativity as a Developer
- A Developer's Guide to Content Delivery Networks (CDNs)
- What is Digital Literacy?
- Why do Digital and Coding Literacy Matter?
- Tech Terms and Concepts
- Digitization, Data, and Analytics
- Personality Types and Technical Careers
- What are Tech Careers?
- Tech Trends and Careers

# 3.1 Development Skills You Need.

Development skills are basic competencies that are required in most tech jobs.

Take advantage of these other development skills resources:

- Skills You Need to Get Hired in Tech  
<https://skillcrush.com/2016/01/14/11-skills-you-need-to-get-hired-in-tech/>
- Tech Skills Employers Want Now: More than Development  
<https://skillcrush.com/2016/01/14/11-skills-you-need-to-get-hired-in-tech/>
- The Future Of Jobs: 6 Essential Skills For The Next Decade  
<https://www.forbes.com/sites/ginnyhogan/2024/02/06/the-future-of-jobs-6-essential-skills-for-the-next-decade/>

Technology development skills refer to the knowledge, skills, and traits required to be successful in a tech-related job. Software developers and related jobs require awareness of tools, techniques, languages, and protocols used in professional settings. It also includes traits like persistence, creativity, and interpersonal awareness. Other employability skills like teamwork, problem-solving, accountability and project management are considered important.

##Quick View: Development Skills

[Software Developer Skills](<https://www.liveabout.com/software-developer-job-description-salary-and-skills-2061833>)

13 Technical Skills You Should Have As A Developer](<https://www.geeksforgeeks.org/technical-skills-to-have-as-a-developer/>)

[What Skills Employers Want in a Software Developer]  
(<https://www.informit.com/articles/article.aspx?p=2156240>)

## 3.2 Are Developers Born or Made? Debunking the Myth of Prodigies

In talking with development students I've discovered one topic that constantly arises in conversation. And that topic is the misconceived notion that great developers are born with a special programming gene. So let's walk through the question: are developers born or made?

### Are Prodigies Real?

Before tackling this question, let's take a step back and discuss the topic of prodigies. Because whenever someone thinks that a certain group of individuals are born with super human like talent, they're essentially saying that these special people are prodigies.

### The Mozart Case Study

But are prodigies real? Let's take a look at one of the most famous prodigies of all time, Mozart. At the age of 5 Mozart was playing concert grade music to the royal family. Surely this would qualify Mozart as a prodigy, right?

In his book *Peak: Secrets from the New Science of Expertise*, researcher Anders Ericsson dispels a number of commonly held prodigy myths. He had this to say about Mozart:

*“If you compare the kind of music pieces that Mozart can play at various ages to today's Suzuki-trained children, he is not exceptional. If anything, he's relatively average.”*

In his book Ericsson dedicates a full chapter to debunking the concept of prodigies. And in each case he illustrates that the individuals achieved their respective levels of success through massive amounts of work.

### Are Developers Born or Made?

Extending the Mozart case study, let's discuss how this applies to developers.

Whenever we see a skilled coder it's easy to think that they were born with the innate ability to build applications and that learning new languages and frameworks comes easy to them.

However nothing could be further from the truth. Through the years I've known more developers than I can count and I have yet to find a single one that was a born developer. I know programmers that work for Google and Amazon, and huge companies like that. Along with computer science

professors who specialize in research that boggles my mind to think about. And as amazing as all of these individuals are, each one of them became a great developer through hard work and dedication.

## The Tipping Point

In past guide I've discussed the tipping point for developers. The longer I teach and the more I work on my own coding skills, the more I'm convinced that the key to excellence is as straightforward as focused practice.

If you want to become a skilled developer bad enough, and you're willing to:

- Dedicate the time
- Learn from experienced teachers
- Fight through frustrating challenges
- Continually build projects with features you've never developed before

You're going to wake up one day and realize that everything is clicking and that you've become a professional programmer.

## Why We Love The Prodigy Myth

Before I end this guide I want to address a subtle issue that explains the reason of why we, as humans, love the idea of prodigies.

The concept of prodigies, individuals born with a natural ability to be successful at a certain skill, such as sports, math, or programming can be detrimental to our own success. This belief is dangerous because it causes our minds to have negative responses to failure.

For example, if you're an aspiring developer who thinks that programmers are born and not made, when you come across a bug that you can't seem to figure out or a feature you can't build, your first reaction might be:

- *I guess I wasn't born to be a developer.*

Or

- *I wish I had talent like XYZ programmer, everything seems to come so easy to him.*

If you catch yourself with thoughts like these remind yourself that prodigies aren't real. Developers achieve greatness through smart work and perseverance. The 10,000 hour rule from the Tipping Point book by Malcolm Gladwell may not be exactly accurate. However it does give a general guide for how much work is required to reach a level of mastery.

If you feel like you weren't born with the 'developer gene', go and knock out 10,000 hours of focused practice on building applications. I think you'll be pleasantly surprised to find that you'll become so good, that other people will look at you, and they'll think... that you were just born this way.

## 3.3 Can You Lose Your Coding Skills?

Today's topic was inspired by the [BoingBoing article](#) that tells the story of a QA developer who spent 6 years working for a company and literally did nothing besides play computer games, browse Reddit and waste time.

The Reddit user, FiletOfFish1066, posted his story to the site last week and has instantly become a supreme case study for what happens when you don't continue to develop your coding skills. After not working for 6 years he says that he has completely forgotten how to develop and now he's out of a job. The story goes that he started working as a software tester and realized that, by writing some scripts, he could fully automate his job. So he worked for about 8 months building testing scripts and after that he simply let the tests run. He didn't have to do anything besides kick back and play video games all day.

There are a number of lessons that developers can learn from this story and that's what I want to cover:

I don't really blame the employee in this case. Apparently there was such little oversight in his organization that he was able to get away with not working for 6 years. That tells me that the company he worked for has serious structural problems in order to let that go on for so long.

It should be noted that many organizations are employing people to do work that software can do, but instead are choosing to waste money with archaic manual processes.

This developer started his job doing exactly what he should have done. He recognized that there was a way to automate his job, which was a great first step. If he would have went to management and shown what his automated script did he would have most likely been promoted for his expertise and would have a great career right now.

It's possible to lose your coding skills. If a world class bodybuilder would stop going to the gym, eventually he'd lose his muscles. In the same way if you stop honing your craft as a developer you'll eventually lose your programming skills.

A few years ago I met with a gentleman who was the vice president of software development at a large energy services company. He got the position by selling software that he had built himself, which has become the industry standard and currently processes billions of dollars in transactions each month. Even though this individual used to be at the top of his field as a developer, since he became an executive, he got further and further away from coding the application and he admitted to me that he wouldn't even be able to build a simple program nowadays. I just finished reading the book *Peak* by Anders Ericsson and Robert Pool and it gives case studies

from the medical field, that show that the most experienced general practice doctors are not always the best in their field compared with less experienced physicians. The book explains that, on average, the longer a general practice doctor has been working in industry, the less they focus on learning and therefore they start to lose some of their expertise. Surprisingly it's actually doctors who have recently finished fellowships and gone through extensive training that perform the best. The authors did note exceptions when it came to specialists. For example, cancer specialists that perform surgeries daily and are constantly working on their craft perform better than less experienced surgeons.

I thought that was a great example for developers because I know, from my own experience, that if I'm not daily using my development skills they will atrophy. There really is no middle ground, when it comes to development you're either getting better or worse, you won't stay the same.

So how can you ensure that you're always improving and that you won't lose your coding skills? Here are a few practical tips that I use:

Learn something new about development each day and be intentional with how you learn. In fact, just yesterday I taught myself how to integrate growl notifications into a Rails application, which was something I hadn't done before.

If you simply repeat the work you've done in the past you won't improve. Thinking back to our case study of the general practice doctors, the research showed that the reason why the older physicians skills decreased was because they performed the same work day after day and eventually the only tasks they could perform properly were the things that they had repeated each day. To be 100% honest, this part isn't very fun, which is why so few people do it. Learning new and challenging skills can be intimidating and stressful. In the same Peak book the authors said that a common trait among all top performers, in every field they researched, was that they were willing to deliberately practice skills that they found difficult, because it was only by mastering those skills that they were able to grow in their profession.

A practical way for developers to implement this method of deliberate practice is to write down a list of features that you have never built into an application before. Then spend time each day until you have successfully built each component. After you're done with that list of features, create another list. I personally have a list that I work and study from and it's helped me to feel confident that I'm learning something new each day and that I'm constantly improving as a coder.

On a final side note, I've had multiple people message me about the story from BoingBoing asking how it was possible that the developer created a script for automating his work. Without further details it's impossible to know for sure, however since he was in the testing department my guess is that he built a test suite, using tools such as Capybara to run through the software and generate reports on features. So that would be my guess.



I hope that this guide has helped to inspire you to be deliberate with your practice and that you will continually improve as a developer.

## **3.4 Cramming vs Consistent Study and How to Build a Study Framework**

This guide will discuss the concept of cramming vs consistent study. And don't change the channel if you're not in school, if you're a developer or if you want to learn software development... the learning never ends.

On average I typically am going through over a dozen books at the same time and around 4-5 various online courses because the deeper I get into development the more I realize how much more I really need to understand. With that in mind I think the topic of cramming vs consistent study habits should be beneficial since the way that we study is just as important as the volume of how much we study.

Most of us have been in the situation where we put off studying for too long and before we know it an exam is upon us that we have to cram for. If you can remember back to the last time that you crammed for an exam or project, how much of what you studied can you remember today? If you're like me, probably not much.

While I was in college I was very bad at this and ended up cramming for many of my midterms and finals, with mixed results from a grade perspective. However once I got to computer science graduate school at Texas Tech I ran into a problem – cramming didn't work at all. Software development concepts build upon themselves, so what was taught in the Fall semester would be the foundation for even more complex topics that would be discussed in the Spring. In the Fall I would learn about logic programming and in the Spring I'd have a course where I had to build a production application using the Prolog programming language.

Using cramming as a study technique resulted in me having very poor retention of what I was learning, which meant I had to go back and re-learn the topics that I had already forgotten from the

previous semester. I don't have to tell you how stressful this made my academic life, not to mention the fact that I was working as a full time developer at the same time. So I knew that something had to change and I put together a system for helping me retain what I learned each day through a consistent study pattern.

Much like a function in programming, my system for consistent study takes in a few parameters:

1. Scheduling
2. Fighting procrastination

For scheduling I created a todo list, segmented by day, for what I needed to study, this included academic papers, books, and watching online lectures, I put these in a drag and drop todo list on Basecamp. After I studied a particular item I would drag it up to the next day's todo list so I would have a visual cue that I was done for that day.

For me, I would procrastinate studying because staring at the list of the books I had to read was intimidating and this was mainly due to the fact that I didn't set any practical goals for studying. If you stare at a Discrete Mathematics textbook and tell yourself to "study" it's natural to want to put it off, however if you set small goals you're less likely to put it off. With that in mind I'll put a note, such as read 3 pages of my Information Retrieval textbook, and 3 pages doesn't sound nearly as scary as the vague "just study" mindset.

The interesting result in making small, manageable goals for studying is that not only does it help curb procrastination, but typically I will also read much more than the 3 pages. There have been plenty of times where I set of goal of a few pages of a book and ended up reading a few chapters.

With all of this being said there are times where I plan deep work study sessions. In one of these sessions I will set aside 2-3 hours of time to sit down, without distractions, and work through a complex topic. However I always limit the time to no more than 2-3 hours per day and I will usually not study any other topics on these days since I'm usually mentally drained by the end of them. I already have a post planned where I'll go into detail on how I structure a deep work session, so stay tuned for that in the next few weeks.

I hope that this guide has been helpful and will help you develop your own system for studying so that you can retain when you learn and be able to use it when it matters most.

## **Cramming vs Consistent Study Research**

[UCLA Research Report](<https://www.uclahealth.org/news/release/cramming-for-a-test-dont-do-it-say-ucla-researchers>) \*

## 3.5 How to Study Effectively to Improve as a Developer

When it comes to effective study practices, make sure that you're making the most of your time. Remember that the most important goal with studying is retaining knowledge so that you can use it in real world scenarios. And the best way to accomplish this goal is by following strategies that work with your mind's learning patterns.

The media could not be loaded, either because the server or network failed or because the format is not supported.

Let's imagine that you're back in school and midterm exams are coming up. How would you study?

Some common approaches might be:

1. Re-read the study materials or lecture notes.
2. Highlight and memorize the key terms.
3. Go over your notes constantly until test day comes.

Those all sound like effective study practices. However cognitive research has shown that many of the traditional study patterns that students have followed for decades simply do not work.

I didn't make up that list of study patterns. That's exactly what I used to do in preparing for exams. However I discovered (after failing a number of tests) that these strategies failed miserably when it came to helping me to truly learn new concepts.

### Why Traditional Study Habits Don't Work

This type of approach to studying doesn't work because our minds don't function like computers. A computer can take in information and then spit it back out. However our minds are more relational in nature.

By relational in nature I mean that our brain functions like a graph based network. If new information attempts to enter the brain without being connected to any of our previous knowledge, it will simply be rejected.

For example, let's imagine that you are new to learning programming. If you simply run through a list of programming terms and syntax rules you might memorize them in the short run. However because your brain hasn't been properly introduced to the concepts it will eventually eject the information, viewing it as useless since it's not related to the rest of your view of the world.

However, imagine that you take a different approach. In this new, more enlightened approach, you work with your brain and allow it to connect each of the new programming concepts that you're learning to knowledge and experiences that you already have.

# Effective Study Practices Case Study

Whenever I'm teaching a new programming concept to students I try to draw a fitting analogy to a real world concept. This process is called reification and I view it as one of my most important tasks as a teacher.

Let's imagine that you are learning about the MVC (Model, View, Controller) design pattern in software development. You could take the approach of trying to memorize each of the roles of the Model, View and Controller. However that strategy wouldn't help you answer questions related to how each of the components work together. And if you memorize quiz questions and answers you probably will have issues answering anything that you haven't memorized.

## Reification Example

*A common case of reification is the confusion of a model with reality: "the map is not the territory". Reification is part of normal usage of natural language, as well as of literature, where a reified abstraction is intended as a figure of speech, and actually understood as such.*

What if instead of trying to memorize key terms about the MVC pattern you focused on drawing a real world analogy to the process? My favorite way to understand this type of architecture is comparing it to a restaurant.

- **Model** – the model is the chef in the kitchen. In the same way that a chef prepares the meal for customers, the model works directly with the data for the application.
- **Controller** – the controller works like a restaurant waiter. In an application, the controller's role is based on taking requests and managing communication between the model and the view. This is much like a waiter who takes customer orders, communicates with the chef, and eventually brings the food out to the table.
- **View** – the view is like the table that a customer is sitting at. It doesn't do much besides provide a platform for placing the food on. This is exactly like how the view should operate in an application. If built properly a view should simply be a place where data is shown to users.

Do you see what we just did? We learned about the MVC design pattern in a way that our minds can actually comprehend. I could fall out of bed and recite back the role of each component of the model/view/controller architecture, not because I spent countless hours trying to memorize them, but because I connected the concept to my real world experiences.

# The Hard Way

Through the years I've come to the conclusion that if studying it easy... I'm doing it wrong. Part of the reason why I used to follow study pattern of:

- **Read**
- **Memorize**
- **Repeat**

Was because it was easy. It wasn't mentally taxing to sit down and read through a textbook or my notes. However, research is proving that these types of study habits are not only ineffective, they are also damaging.

## Additional Negative Effects

How are they damaging? If you have followed this type of study system you know one thing: it takes time. This time spent reading and memorizing could have been used in countless other ways that would have proven more effective in the long run. And when it comes to studying, time is one of the most valuable assets that you have, so wasting it is akin to an educational felony.

## Comprehensive Study System

In addition to the process of reification, there are a number of other study strategies that research is showing to be more effective than traditional study practices. In their book "Make It Stick", cognitive psychologists Brown, Roediger, and McDaniel give the following recommendations for studying:

- When learning from a textbook, use the key terms from the back of each chapter to test yourself.
- List out key terms and use each one in a paragraph, this will test to see if you understand a concept outside of the realm of how the textbook or instructor supplied it.
- While reading new material, convert the main concepts into a series of questions and then go back and answer the questions when you're done reading the chapter.
- Rephrase the main ideas in your own words as you go through the material.
- Relate the main concepts to your own experiences, much like the reification process we've already discussed.
- Look for examples of key concepts outside of the text. When I'm learning a new programming language I never rely on a single source. If I come across a concept that doesn't make sense I'll usually review 2-3 other sources that provide alternative explanations to what I'm attempting to learn.

## Summary

In summary, when it comes to effective study practices, make sure that you're making the most of your time. Remember that the most important goal with studying is retaining knowledge so that you can use it in real world scenarios. And the best way to accomplish this goal is by following strategies that work with your mind's learning patterns.

## 3.6 Guide to Developer Soft Skills

I talk quite a bit about improving as a developer. Most of the time I focus on how you can learn new technical skills, such as becoming more proficient in a programming language or framework. However if you limit your knowledge to technical talent you will be decreasing your chances for success in the marketplace. In this guide I'm going to walk through five key developer soft skills that you can utilize to become a well rounded coder.

This list is by no means comprehensive. Instead it represents the skills that I've personally used and had success with. From being the IT Director of a national energy company in my late 20's to the CTO of coding bootcamp with locations around the world, I've seen these skills help me at every level of my career. And as you'll notice, they have very little to do with actual technical ability.

As a caveat, I do not mean for this list to overshadow skills such as practicing clean coding habits or focusing on improving as a developer. Instead these skills should complement your engineering talent.

### Writing

First on the list is the ability to write. In the book *ReWork*. The authors, who are also the founders of Basecamp and the Ruby on Rails framework, wrote that one of the skills they look for in job candidates is their ability to write. This includes positions that you would think writing skill would be pointless, such as developers and system administrators.

Obviously a developer needs to be able to be skilled as a coder. However the book explains that if a developer can write, it is a sign that he or she is a good communicator. Writing skill doesn't mean that each memo you write has to sound like a riveting novel. Instead it means that:

- You can organize your thoughts properly.
- And that you can communicate what you want to say so that others can understand.

If you feel like you lack writing talent, I've included a link to some helpful tips in the Resources section of this guide.

### Conversation

Next on the list of developer soft skills is the ability to converse well with others. Now if you're like me, this is by far the most challenging skill on this list. If I had my way I'd stay behind my desk building applications all day and never interact with another human. It's simply the way I was wired, and I know I'm not alone in that desire.

However, conversational skills are an absolute requirement when it comes to advancing in your career. Whether you are a freelance developer looking for new clients or a software engineer looking to get promoted, you'll discover that the top prerequisite to your success is not technical skill, it's likability. If someone likes you they are going to want to give you a chance to succeed. And one of the most straightforward ways to get people to like you is by becoming a good conversationalist.

## Conversation Tips

Thankfully I've discovered that the system for having great conversations is pretty straightforward. Here are some tips that I've used to improve at this skill:

- Think back through your life and come up with some entertaining stories about yourself. People love stories, especially if they are funny. And I've discovered that telling a few well timed stories has been able to get me in the good graces with CEOs and executives through the years.
- Make the focus of conversation be on the other person. People love talking about themselves, so by simply asking insightful questions you will be considered a great conversationalist... even though you let the other person do all the talking! Like your stories, come up with a list of questions that you can recall at a moment's notice.
- DO NOT COMPLAIN. I have yet to find the person that likes to converse with someone who constantly complains. With that being said, countless people seem to enjoy bringing up every negative thing that has happened in their lives when they meet someone. The good thing about this is that if you can have a conversation without complaining you will stand out as being an upbeat and likable person.

## Management

Moving down the list of developer soft skills, the next item is management. Now if you're an entry level developer don't tune out. Management doesn't have to mean managing people or projects. When I say management I'm referencing how you attack each task you're given.

For example, if you are handed a new feature to build, do you jump right in and start coding? Or are you more organized with your approach?

If a client or managers see that you take a systematic approach to every task you're given they are going to feel more confident giving you more responsibility.



For improving this skill I recommend you reading up on project management books or taking an online course on the topic. The few days that you'll spend learning about management practices will help serve you well the rest of your career.

## Design

Next on the list is design. In Scott Adams (the creator of Dilbert) book *How to Fail at Almost Everything and Still Win Big*, Adams describes how knowing the basic fundamentals of design should be required knowledge for all engineers. I can't tell you how many times I've heard a developer say something like "design really isn't my thing". It's fine if your top skill isn't design. However learning the basics of what qualifies as a well crafted design takes such little work, that anyone who doesn't learn about it is simply being lazy.

As a developer, if you haven't researched what it takes to create a good design, you are going to be quickly bypassed by others who read a single book on the topic. There have been multiple times early on in my career where I neglected design and it cost me dearly. I remember one time where I spent weeks building an incredibly complex feature to only have management spend the entire meeting talk about how much they hated the design, while completely ignoring the actual functionality. If I would have spent a few hours to design the look and feel of the product the meeting and project would have had a more favorable result.

I've included a link to a great book on design in the Resources section of this guide.

## Public Speaking

Last on the list of developer soft skills is public speaking. This may seem like a useless skill for a software developer. However let me give you two scenarios to think about:

- In scenario 1 there is a brilliant developer with poor public speaking skills. When asked to present a project that he built. An incredibly well built project I may add. The developer talks in a monotone voice during the whole presentation and the product demo is filled with him simply moving from one page to another.
- In scenario 2 there is another great developer. But this engineer has worked on his public speaking skills and gives a well organized demo. His time in front of the room is filled with clear language, amusing anecdotes, and analogies for each feature to make the project understandable for everyone in the room. If you were in the room which one of the projects would seem more appealing? It doesn't take a MBA to know that the developer in scenario #2 will win each time.

Note that both of the projects were great pieces of software. Like I mentioned earlier, soft skills are not a replacement for technical skills. They are something to layer on top of programming expertise. You could give a Steve Jobs' level speech, but if the product doesn't work it won't matter.

## Becoming a Better Public Speaker

Public speaking is ranked as one of the most feared tasks to perform. However, I can tell you from experience that you can improve at public speaking quite easily. There are two things that I've done to become a better speaker:

1. I am a member at a local Toastmaster group. Each week I attend a group meeting where I am able to practice getting up and talking in front of a group of people. By simply forcing myself to practice this skill consistently my public speaking ability has improved dramatically.
2. Additionally I listen to one TED Talk each day. The TED conference lectures are given by some of the most skilled orators in the world. By listening to a new talk each day it has helped give me ideas of ways that I can craft my own speeches and it has helped to build a mental model for what makes a great speech.

## The Importance Soft Skills

So now that you know the list. How important are developer soft skills? I can't tell you how many times I've seen an inferior developer promoted to management simply based on their ability to speak well in meetings or converse with co-workers. Remember that the key to each of the skills on this list is that they help people feel more comfortable being around you and that they will be confident that you can get the job done. Likability and confidence are two key prerequisites you'll need in order to gain an edge in the marketplace.

## Resources

- [A Guide to Becoming a Better Writer: 20 Practical Tips](https://www.lifehack.org/articles/work/20-ways-become-better-writer.html)  
<https://www.lifehack.org/articles/work/20-ways-become-better-writer.html>
- [The Design of Everyday Things](https://www.amazon.com/Design-Everyday-Things-Donald-Norman/dp/1452654123)  
<https://www.amazon.com/Design-Everyday-Things-Donald-Norman/dp/1452654123>

## 3.7 Discovering the Tipping Point as a Developer

If you've been programming for a while, a question that has most likely crossed your mind is: Am I a good developer?

Before we go on, let me share a little secret with you... Every developer, even senior devs, have insecurities when it comes to programming. Few individuals like to share that information. Mainly because confidence and even arrogance has become a developer stereotype for some stupid reason.

However, I won't BS you. I can tell you that the more experience I have as a coder the more I realize how much more there is to learn and how far I still have to go.

### Tipping Point for Developers

With all of that being said today I want to discuss topic of defining the tipping point for developers. Which is essentially the point at which a dev goes from a beginner to a pro.

Since this topic is a bit abstract it's not possible to point to a specific point in time and say:

*"Here it is, this is when it all clicks and makes sense."*

There's not a sentinel moment when programming mastery occurs. It's different for every individual.

### My Own Experience

I remember when I was originally learning programming. Understanding the syntax and context did not come easy for me. It seemed like I spent 99% of my time looking things up and copying and pasting code from others just to get my programs running.

### The Doubt Machine

Needless to say, my confidence as a programmer was very low in the beginning. I kept being plagued by nagging doubts, such as:

- Maybe programming isn't for you.
- Even if your code works you won't be able to write your own programs.
- You're only typing in what the book is saying to do, you won't be able to build anything custom.
- And the negative thoughts continued from there.

## **The Painful Process**

If you're a new developer maybe some of this sounds similar to you, or maybe it doesn't and I simply lacked confidence. Either way, I trudged along. Trying everything I could think of to improve as a developer.

- Going through dozens upon dozens of programming books in various languages.
- Trying to build up a portfolio of projects.
- Following online guides. However back when I was originally learning how to code the online resources weren't quite as good as they are today.

## **The Tipping Point(s)**

So what did the trick and pushed me over the edge to become a professional developer? None of those things... and all of those things. I persevered through project after project and I consumed every training resource I could find. And slowly something amazing started to happen: everything started to make sense.

## **The First Tipping Point**

Even though it was a while ago, I still remember the moment my first development tipping point happened. I was sitting in front of my computer at a coffee shop and working on a web application. A few hours went by and I stopped dead in my tracks, realizing that I had just spent the afternoon building a project and hadn't looked up a single code snippet.

It wasn't like I programmed the space station, the project was incredibly basic. However, it was one of the most exciting moments I can remember in my life.

## **The Second Tipping Point**

As great as that was, I still had very far to go. I remember the next moment when I felt like I reached another key milestone. Even though my confidence had increased as a developer, the thought of anyone seeing my code was a very scary thought. However, I had started to build my freelance business and a client (who was also a developer) asked me to perform a pair programming session with him.

He had run into a bug with the program we were building and asked me to jump on a screen sharing session where we could work on the project at the same time.

Honestly, I was scared to death when he asked. I had never coded in front of anyone before and the thought of doing it with this client pretty much gave me a panic attack. However, I didn't really have a choice in the matter so I started the session with him. After a few minutes of nervousness, I

started to relax, and to my surprise not only did I not make a fool of myself, I actually figured out the bug in his code and got the feature working.

## **The Secret**

So what was my secret to getting over the hump and going from a beginner to a professional developer? Unfortunately, there is no easy to follow recipe. However, there is a process that is guaranteed to work. And the process isn't specific to becoming a programmer, it's the same whether you want to be a dev or a professional athlete... it's hard and smart work.

## **The Book**

In the book, *The Tipping Point*, by Malcolm Gladwell, Gladwell gives countless case studies of what it takes for individuals to achieve mastery in a specific field. The key comes down to how dedicated an individual is to a specific skill. The book postulates that it takes around 10,000 hours for an individual to become a true master of whatever they're pursuing.

I'm not sure I believe in the 10,000 hour rule. Mainly because there are a large number of variables when it comes to learning a topic or skill and rarely does a single rule apply for all fields. Also, I think the quality of your practice makes a significant difference. For example, if you're learning how to play the violin: 5,000 hours of practice with a world-class instructor is probably equivalent to 10,000 hours trying to figure it out yourself.

However, with all of that being said, one thing cannot be denied, the key to mastery is hard work.

## **The Solution**

I'm sorry if you were hoping for a quick fix, I can tell you from experience that there are no shortcuts to becoming a developer. You need to learn:

- The fundamentals of coding.
- How to build projects on your own.
- Various processes for working through bugs.

Becoming a great developer is not an easy road. However, be comforted in the fact that you are 100% in control of how skilled you will become. The formula is straightforward: the harder you work, the better you will get. So, get your hands on all the material you can find on the language and framework you want to learn. Work through challenging applications and you will be well on your way to mastery.

And soon you will be able to have the exciting moment of clarity when everything starts to click.

## 3.8 What Does it Take to Become a Great Developer?

Whether you've been programming for years or if you're just now learning how to code, it's natural to ask yourself: what does it take to become a great developer?

I'm going to start off by saying that there is no right or wrong answer to this question. If you ask 100 experienced software engineers this question, you'll get 100 different responses. The reason why there's no clear cut answer is because development is truly an art form. Therefore asking this question about programming is similar to asking what makes a great artist.

Even though the idea of becoming a great developer may seem like a daunting task, there is a practical process that you can follow in order to attain your development goals.

### Tips for Becoming a Great Developer

In preparation for this post I've asked various developers, read blog posts, and listened to a number of podcasts discussing the topic. As I expected the components of becoming a great programming are extensive. In this guide I want to give an overview of the processes and requirements that I've found the most effective. The following are six traits that encapsulate the key characteristics found among great developers. I've also included some practical strategies for working with each of these attributes on a regular basis.

### Working Through Difficult Features

Starting off with one of the most challenging traits, I've found that the only way I improve as a developer is to work through challenging concepts. I find it disturbingly easy to fall into a routine where I only perform the same tasks again and again. I've been working as a developer for a number of years and I therefore have a nice arsenal of tools and features that I'm comfortable building. However I've discovered that if I simply keep building features that I'm already comfortable creating, I won't grow as a developer. It's only when I bear down and dedicate myself to work through a difficult task that I've never performed before that I become better myself.

Having the requirement of working through difficult practice isn't a concept related solely to development. The book *Peak* researched peak performers in music, athletics, and essentially every other skilled profession. The results of the research revealed that individuals only show improvement when working through challenging concepts. This means that if concert violists played the same music day after day and never challenged themselves, their skill would stagnate.

The same concept holds true for developers. If you want to become a great developer you need to work through difficult topics constantly.

If you don't know where to start with finding challenging features to build, visit some of your favorite websites. You could look at Twitter, or AirBnB, or Pinterest. From there you can compile a list of advanced features that you've never built before. Examples would be components such as: infinite scrolling, asynchronous notifications, or multi page authentication.

## Community Contribution

With the growth of the programming industry, the open source community has expanded exponentially. The most popular languages and frameworks in the world, such as Python and Ruby, were created not by corporations, but by programmers interested in the common good.

Depending on your level of experience, community contributions will vary pretty widely. If you're a senior level engineer, you could build an open source code library or build a feature for a programming language. However, even if you barely have any experience you can still contribute. New developers can assist other individuals who are just starting to code.

As great as it is to give back to the developer community, there are also significant benefits to contributing. If you're building a code library that other developers will see, you'll most likely be very careful to ensure that the codebase is properly tested and functions properly. This type of development will make you an even better programmer and will help you in the long run.

## Artistry

When it comes to development, it's easy to get caught up in the day to day minutia of a project and forget that, at its core, programming is an art. In order for code to be artistic, it must be elegant, and for it to be elegant, it must be simple. Some of the most best projects that I've worked on ended up having the most straightforward codebases. However writing simple code is not as easy as you may think. Sandi Metz said this about simple code.

“Novice programmers don't yet have the skills to write simple code”

Einstein said this about simplicity.

“If you can't explain it to a six year old, you don't understand it yourself.”

This may seem like an odd concept, however if you've ever attempted to build a complex project that maintained an easy to follow code design you know it to be true. The more you improve as a developer, the most straightforward your work should be.



# Craftsmanship

Craftsmanship is closely related to artistry, however there is an important distinction. When you're a craftsman you truly take pride in your work. Through the years I've met all kinds of developers. From programmers who simply treated each project like a widget on an assembly line, to developers who made sure that every code file they worked on looked like a piece of art. Personally I've found a cross between the two concepts to be the most effective.

Like many other concepts, craftsman is not isolated to programming. Growing up my Dad, who was a Major League baseball player and is now a coach, always taught me to have what he called a Spirit of Excellence. This meant that no matter what I did or what I was working on, I had to take pride in it. He would tell me that if I was taking the time to perform a task, I might as well do it properly.

While I feel that I take pride in my work, craftsmanship is one of the concepts that I struggle with the most. I find this principle challenging because it can be difficult to find the balance between well written code and perfect code. As the saying goes, "perfection is the enemy of great". Therefore it's important to ensure that you work hard to properly design your codebase. However don't pressure yourself to achieve perfection.

It's also important to have the mindset that no project is ever truly completed. This means that if you attempt to achieve perfection you'll constantly be frustrated. Mainly due to the fact that you will never reach a stage where your codebase will ever be considered done.

## Steve Jobs Craftsmanship

When it comes to craftsmanship, few have taken the same level of pride in their work as Steve Jobs. This is what he had to say about craftsmanship.

"When you're a carpenter making a beautiful chest of drawers, you're not going to use a piece of plywood on the back, even though it faces the wall and nobody will ever see it. You'll know it's there, so you're going to use a beautiful piece of wood on the back. For you to sleep well at night, the aesthetic, the quality, has to be carried all the way through." – Steve Jobs

## Adapting to Change

If you've worked on any real world code projects you can attest that there is only one true constant: change. Great developers set themselves apart from novices by how they adjust to changing requirements for an application. There are two ways that new coders struggle with change.

1. No flexibility with the code design. This means that when a new requirement is added to the project, they will need to completely reconfigure the code in order to allow for the additional functionality.
2. Planning for the wrong future. A developer may have developed a mental model of what the end project will look like, however that estimation rarely matches reality. Imagine that you're building an accounting application and you think the client is going to eventually ask for the system to be completely project based. You will make design decisions based on the workflow hierarchy that you have in your mind. However if you're wrong, you will be forced to reconfigure the entire application. Both of these pitfalls are normal to come across on a coding journey. However a great developer finds the balance between no design and premature design. By building well constructed codebases, the great programmer writes modules that have flexible interfaces that can adapt to change. They also understand that project requirements change and that the code they write has low coupling. This means that changes to one feature in the application should have little to no impact on other parts of the program. For example, back with the accounting application. If a change is required to the payroll module, it shouldn't require you to rewrite the personnel management feature.

## **Tireless Learning**

Lastly, in answering the question of “What Does it Take to Become a Great Programmer?” I’m going to discuss the importance of tireless learning. One of the most important factors in reaching your development goals is having a thirst for knowledge. Thankfully you have 100% control over this requirement. Regardless of how much experience you have as a programmer you won’t ever reach a stage where you should stop learning. There will always be improved processes, new frameworks, and new languages to learn.

I’ve asked some senior developers that I work with how they organize their learning methods. They gave the following recommendations.

- Learn one new language or framework each year. This should also mean that you’re building a production application during that year. It’s easy to follow tutorials and build “hello world” applications. However when you create a real world program you’ll be forced to work through challenging constructs.
- Read multiple books daily. I personally have over a dozen books that I read daily related to development. In fact, many of the topics that I discuss on this show come out of what I’m reading during a particular week.
- Follow advanced tutorials. Many of the developers that I work with admitted that they prefer to learn new coding techniques by reading blogs from other programmers. There are a

number of guides available online that you can go through that will teach you how to build advanced features into your applications.

- Subscribe to newsletters. I subscribe to a number of newsletters that are sent to me each week. This includes newsletters on Ruby, Rails, and JavaScript. These types of newsletters are a great way to stay up to date with changes in a language. They curate some of the best blog posts and tutorials from around the web.

## Summary

I hope that this has been a helpful guide and will help you answer the question of “What Does it Take to Become a Great Programmer?”

## 3.9 How to Learn a New Programming Language

Over the years I've been hired by organizations such as Learn.co and AppDev to write programming curriculum for:

- Ruby on Rails
- Ruby programming
- Python
- Java
- Several JavaScript frameworks

The only language that I really build applications in is Ruby, which means that I've been forced to become proficient in a number of language that I really didn't have much experience working with, sometimes in a very short period of time. And over the years I've developed a system for learning a new language or framework and that's what I'm going to walk through today.

When I'm learning a new programming language I follow these steps:

1. Watch a full tutorial series on the language, when I'm watching I don't try to follow along, I simply watch what the instructor does in the demos so I can get a high level view of the language syntax and flow.
2. Create a hello world application, I'll incorporate a few basics, such as running a loop, creating and instantiating a class, and any other high level concepts I remember from the tutorial.
3. Pick out a sorting algorithm and implement it in the language. It's fine if the sorting algorithm is a basic one like selection or bubble sort. Sorting algorithms force you to use: data structures, loops, variables, and functions. Combining each of these elements will give you a good handle on how the language works.
4. Go through an advanced tutorial on the language and this time follow along and build the programs with the instructor.
5. Go through coding interview questions for the language. Being able to confidently answer these questions will give you a good idea if you have a solid understanding of the language.

I've used these five steps for a number of languages and I can also tell you, once you've become proficient in a single language you'll find it's much easier to pick up new programming languages since most of them have quite a bit of shared processes and all you'll need to do is learn the difference in syntax.

## 3.10 Getting Past Skill Plateaus

A common pattern I see with students learning how to code is:

- Quickly learning a massive amount of information.
- Followed by running into a seemingly insurmountable wall. In this phase the student typically feels like they've reached the zenith of what they're going to be able to accomplish in regard to development.

This second phase is called a plateau. In this guide we're going to walk through strategies for getting past skill plateaus.

I've spoken before about learning curves. And it's important to understand that everyone follows a similar pattern when it comes to learning a new skill. This means that you will experience times where it seems like every day you're soaking in a wealth of new information. But it also means that you will run into times where it feels like your mind will limit you from learning anything new.

### What is a Learning Plateau?

When it comes to hitting a learning plateau, it's important to look at the potential root causes for why it's occurring. It's been my experience that no two plateaus are the same. And until you've diagnosed why you're not learning you won't be able to move on to your next level of skill.

### False Ceiling

And before I continue I want to reiterate something. You will never reach a point where your level of skill is maxed out. Maybe if you're a professional athlete and getting older, then your body is naturally going to decrease in performance. But when it comes to concepts such as understanding development, if you continue to dedicate yourself and if you're willing to listen to experts, your skill will never reach a peak.

### Getting Past Skill Plateaus

Through the years I have witnessed a few key reasons why individuals (and myself) run into skill plateaus.

#### Proper information/resources

When a student lacks access to proper information it makes learning a more arduous process. Imagine a talented developer in high school who had been relying on her teacher (who had limited

skill). In cases like this the student will need to find additional resources, such as online courses that will help teach her concepts she's never been taught before.

## **Best practices**

During a phase of the learning cycle when best practices are the focus, students may feel like they are hitting a learning plateau. I remember when I was first learning about test driven development. The concept seemed counterintuitive. I would spend 2-3 times the amount of time on a feature. And this became incredibly frustrating. It felt like I wasn't learning anything new because my new knowledge wasn't affecting anything on the screen. However this phase isn't actually a skill plateau.

There are many times where developers need to take a step back and focus on quality over quantity when it comes to building applications. My advice for going through this phase is to embrace it. Be patient. As soon as you have a firm understanding on how the best practices can be utilized you'll be able to move onto learning new concepts. The only difference is that now you will be able to leverage your new skills, the result being that you'll be a more refined developer.

## **Challenging/New Tasks**

In my experience, the main cause of students hitting a skill plateau is when they stop challenging themselves. If you remember back to when you were first learning development it seemed like your knowledge was skyrocketing each day. The reason for this was because each of the concepts you were learning were completely new to you.

However, after a certain period of time it seems like it's natural for us to want to become comfortable. Instead of trying to learn something new each day, we simply try to duplicate the work that we've done up to a certain point. This approach is less taxing mentally. However it has the nasty side effect of limiting how we improve.

Whenever I feel like I'm getting into a rut I will look at popular websites and I'll start to put together a list of features that I want to learn how to build. From that point I can put a plan together for what concepts I need to learn in order to implement them.

## **Frustration = Skill**

One of my favorite illustrations of getting past skill plateaus was made by the calligrapher Jamin Brown.

Notice in this illustration how the learning process is filled with plateaus? This is a natural component when it comes to improving at any skill. But also notice that the key to overcoming a

plateau is called the Frustration Zone. I think that's a great name for it. Learning complex topics is not easy. Like you've probably heard countless times, "if it were easy, everyone would do it".

Becoming a developer can be one of the most rewarding experiences that someone can have. And part of what makes learning how to code so fulfilling is how many challenges you'll need to overcome in order to succeed.

## **Summary**

I hope that this has been a helpful guide and that you now have some practical strategies for getting past skill plateaus. And good luck with the coding.

# 3.11 How to Practice Programming Techniques and Improve as a Developer

Whether you are new to programming or have been at it for years, practice is important. The more you practice your programming skills, the better you will be. You have various options to practice programming techniques. These options will help you brush up on your skills and continually improve as a developer.

## Engage in Pair Programming

Programming doesn't have to be a solitary activity. Instead of taking in on by yourself, engage in pair programming. Since pair programming has people working together from a single computer, it is a great way to learn different strategies for tackling problems and approaching the process. This type of practice will quickly make you a better programmer, as long as you choose a good partner.

## Utilize Open Source Software

Open source software is a great way to practice your programming techniques. Start by reading code from various open source projects. This will help you understand how the programmers managed to create such a successful project.

Then participate in various open source projects. As you work, people will give you immediate feedback. It might be hard to hear the criticism from time to time, but it will help you fine-tune your skills, which will make you a better programmer.

## Visit the DailyProgrammer Subreddit on Reddit

Reddit is a community full of people who share ideas and help one another. You can get in on the action with the [DailyProgrammer subreddit](#). This subreddit posts three programming challenges each week. The first challenge is relatively easy, and then they increase in difficulty. The community reviews the solutions and provides feedback. Use this subreddit to improve your skills while having some fun.

## Take Online Courses

Sometimes you don't need to go back to school to develop your programming skills. Massive Open Online Courses (MOOCs) are an excellent way to brush up on your skills. You can learn at your



own pace and practice the techniques that you need to work on without getting rushed. Best of all, you can get feedback during the course, which will help you become a better programmer.

## **Code Katas**

The term kata was first introduced by Dave Thomas in the book *The Pragmatic Programmer*. He borrowed it from martial arts and applied it to the programming world. In order to code katas, you need to take a small requirement and create the code. Then do it over and over again, improving it until it is perfect. This is an easy way to practice coding while making your code better.

## **Practice Programming Techniques Summary**

Don't make the mistake of thinking that you don't need to practice programming. You should practice as often as possible. You can leverage these recommendations to practice programming techniques to improve your skills so that you can take your career or your hobby to the next level.

# 3.12 How to Study and Understand Complex Topics

When I was younger I used to struggle learning a new or difficult subject, and through the years and about a decade of university and grad school have helped me put together a strategy for how to study and understand complex topics. Typically I apply this learning system to subjects such as algorithms and software engineering, however it can be applied to any topic.

While there are a near infinite set of study straggles out there, I like this approach because it utilizes a divide and conquer strategy, focusing on breaking a complex topic into easy to understand components, and putting the pieces back together at the end to see how they all work together.

## System for How to Study

Let's take a case study example: Understanding how logarithms work. Logarithms are used throughout the fields of mathematics and computer science, however unless you use them regularly it's easy to get rusty on them.

### Step 1

The first task that I will do is take a piece of paper and write Logarithm in the center and circle it.

### Step 2

Next I'll go to a comprehensive post on the topic, such as one on Wikipedia. In reading the first sentence I come across a few terms that are a bit fuzzy:

Inverse operation and Exponentiation

I will stop reading the logarithm article and go and read those two articles until I feel comfortable with what they represent. After I feel good about those two items I write them as their own circles that connect to the Logarithm circle. I will also add any examples that will help me understand what the terms mean if necessary.

### Step 3

Next I'll go back to the original Logarithm post and keep going through the article repeating this process until the entire page is filled with a mind map that explains each component that makes up logarithms and how they work together. This may include base case examples, such as:

$64 = 2^6$  is the same as  $\log_2(64) = 6$

If this seems like a dead simple approach to study... it is. The goal of studying is to learn a topic, and one of the easiest ways to understand a complex subject is to break it into easy to comprehend components.

For example, if you're trying to understand an advanced algorithm in computer science from scratch you may feel a little intimidated. However if you break the algorithm down into small enough components you'll see that it's essentially a process of steps made up of connecting simple modules such as loops, manipulating variables, and using conditionals.

A problem is only hard when you try to think of it as a whole, however any concept can be understood if you simplify it down to easy to comprehend pieces. Obviously the more complex the topic, the longer it will take to deconstruct, however I am a firm believer that anyone can understand any topic assuming they dedicate themselves and put the work in.

I hope that you can leverage this mind mapping process to understand complex topics and that it will help you learn how to study properly and truly learn.

## 3.13 How to Use Deep Work to Improve as a Developer

Standing on the podium, Michael Phelps stares at the American flag and listens to the National Anthem after winning Gold once again. After watching Phelps win 21 gold medals (at the time I'm writing this), it's natural to ask: "Was he simply born for greatness?" I don't think so. Yes, his body type has helped him take advantage of physical elements of swimming. However there are millions of individuals with his height and wingspan who watch him at the Olympics from their couches every four years.

There is no magical swimming gene that Phelps was born with. Instead the secret to his success can be found in his discipline to a practice called Deep work. Muscle Prodigy research claims:

"Phelps swims minimum 80,000 meters a week, which is nearly 50 miles. He practices twice a day, sometimes more if he's training at altitude. Phelps trains for around five to six hours a day at six days a week."

If Malcom Gladwell's 10,000 hour rule is even close to being accurate, Michael Phelps surpassed this benchmark years ago.

In case you're wondering how this applies to coding, don't worry, I haven't forgotten that this is a show for developers 😊

### Definition of Deep Work

If you have watched this show you may remember that one of my favorite books is Deep Work by Cal Newport. (The fact I've referenced the book a few dozen times may have given it away). So what exactly is Deep Work?

A dead simple explanation of deep work is:

Deep work is the ability to focus without distraction on a cognitively demanding task.

Whether you believe that swimming is cognitively demanding or not, I believe that Phelps's example is fitting. If you have ever attempted to train with the level of intensity that Phelps does, you can attest to the mental toll that training takes.

So essentially deep work can be simplified by saying that it has the following characteristics:

1. It's a real world action. It's not a theoretical concept, it's something that you can practically implement.
2. In order to work properly you have to eliminate 100% of your distractions.

3. The task has to be challenging.

## **Deep Work Strategy for Developers**

Let's dissect the definition of deep work and build a practical strategy for how it can be implemented from a developer perspective. Let's imagine that you want to learn about the computer science topic of asymptotic analysis. If you've never heard of asymptotic analysis, don't worry, you can trust me that it qualifies as a challenging topic.

### **Taking Action**

Let's start with the fact that deep work is an action. With that in mind you will need to make a clearly defined time slot. If you have never practice deep work studying before, I'd recommend limiting the slot to around two hours. As you'll discover deep work is a draining task. For our example let's imagine that you have designated 9am to 11am as when you're going to study asymptotic analysis.

### **Removing Distractions**

With your time slot set, now it's time to remove any and all potential distractions. Let me be 100% explicit, this means:

- You cannot check your email
- No phone calls. In fact, turn your phone on airplane mode to ensure no one calls or text messages you.
- Don't even think about checking Instagram, Facebook, Twitter, or Tinder. All of your swipe rights will have to wait for a few hours.

If I missed any distractions you can add them to the list. It may also help to listen to classical music to block out any potential sound distractions while you study.

### **Study Hard and Smart**

Now that you have dedicated a few hours to studying asymptotic analysis and have removed all of your distractions, it's finally time to get down to business. If you think that now you can simply start reading a few Wikipedia posts, I'm sorry, that won't earn you a deep work badge.

In order for deep work to be truly effective, it has to be difficult. If I was learning about asymptotic analysis for the first time and wanted to practice deep work while studying it, I'd take the following approach:

1. I'd begin by reading a number of online resources on the subject.
2. Next I'd watch an online lecture while taking notes.
3. I would then find practice exercises where I would attempt to figure out problems from scratch.
4. Next I would write a blog post or record myself teaching the concept.
5. Lastly I would have another student or instructor review my teaching and exercises to ensure that I understood the concept properly.

Do you see how much more comprehensive this type of studying is? Even if you had never heard of asymptotic analysis before your deep work study session you would be fluent in it after you were done.

## Multiple Sessions

When I mentioned earlier how you should limit your deep work sessions to around 2 hours, I don't mean that you can understand any topic in that period of time. Some complex topics may take days, weeks, months, or years to properly understand. So it is completely fine to spend a number of sessions working through the same concept. If you are going to do this I recommend that you make notes for what you were doing when you stopped. This will allow you to pick up right where you left off.

## Summary

I hope that this has been a helpful introduction to what deep work is and how you can practically implement it as a developer. If you want to learn more about the topic I suggest that you pick up Newport's book. It will give you a great set of tools for learning how to use deep work to constantly improve as a developer. When it comes to learning, deep work is the closest thing you can get to steroids. Good luck with the coding!

# 3.14 Is Reading Important for Developers?

I talk quite a bit about improving as a developer, specifically discussing various ways to study from a practical perspective. However in this guide I want to specifically answer the question: is reading important for developers?

The short answer to the question is: yes! However as computer scientists it's poor form to simply take someone at their word. So let's dive into why reading is critical to improvement.

## Why is reading important for developers

Let's analyze a few key statistics in regard to reading.

### CEOs and Reading

How many books do you currently read a year? If your answer is that you're too busy to read entire books, let me ask you another question: are you busier than the CEOs of the world's most successful companies? Probably not.

However research from the Rype Academy shows that CEOs such as Elon Musk, Mark Cuban, and Peter Thiel read around 60 books a year! That's 4-5 books each month.

### Compounded Learning

So why do some of the most successful individuals in the world take the time to go through so many books? At a high level it may seem excessive, however if you truly believe that knowledge is power, wouldn't it make sense to dedicate whatever time is needed in order to attain more knowledge?

If you look at reading like a form of linear learning, than yes, reading would be a waste of time. Linear learning would be a 1 to 1 transfer of knowledge. For example, if it took the author of the book 10 years to research a topic and it took me 10 years to go through the book, that would be pretty pointless. At the end of the day this type of reading would be pointless.

However I look at reading like it's compounded learning. What is compounded learning? Good question! Compounded learning is the process of taking the knowledge from an individual, but not having to spend the same amount of time that it took that individual to research the topic.

# Compounded Learning Case Study

For example, imagine that you read a book on How to Become a Better Developer. The author of the book had to spend years researching the topic (assuming that it was a well written/well researched book). However if you go through the book in a few weeks, that means that you were able to gain years worth of knowledge in a few weeks!

Research shows that top authors will spend a minimum of two years researching a book. And that research time doesn't take into account the fact that authors draw on their entire lifespans' to write a book. All of this means that each time you read a book it's as if you were able to gain a lifetime's worth of experiences and wisdom from the author.

## The CEO Who Didn't Have Time to Read

A few years back I was offered a CTO position for a startup in New York City. The job had a good salary, great stock options, and an excellent product.

However during a dinner meeting with their Founder/CEO I asked him about a book I had finished reading that discussed best practices for tech startups. He said that he had never heard of the book. This wasn't a problem, there are millions of books and I don't judge someone for having different literary tastes than myself. However the CEO followed this statement up by saying that he didn't have time for reading. He was too busy building the business.

The CEO's view of reading resonated with me during the job consideration process. And I ended up turning down the job. If the CEO didn't dedicate time to read and learn from others. That means that he would be relying solely on his own knowledge and life experiences. And even the most brilliant business person will fail if they think that they already have all the right answers.

## My Reading System

It's one thing to say that reading is important, it's another thing entirely to go through a large number of books on a regular basis. With that in mind I've developed my own reading system. This system also takes into account a number of complaints that I've heard others say about reading.

## Reading Schedule

First and foremost I schedule a set amount of time each day for reading. Usually this equals around 1-2 hours, however on weekends this number can be double that number. At any one time I'm usually going through a dozen books ranging from mind/skill hacking through technical programming books.



## **Audio Books Are Books Too**

I'm not sure where the stigma of audio books came from. However with my travel schedule I've discovered that audio books are an invaluable tool in my learning arsenal. Obviously you can't go through programming books via Audible. However you can go through skill and business based books. And I personally have hundreds of books in my Audible account, many of which I've gone through multiple times.

In fact, many of the books I've discussed and quoted from were books I listened to.

## **Books are Too Expensive**

One of the top complaints I hear from students is that books are too expensive. My response is always: if you're not willing to sacrifice to improve, then you're not going to attain your goals. And that includes sacrificing financially.

With that being said, there are ways that you can go through a large number of books, even if you're on a budget. To start off, your local library has countless books that you can learn from each day. And assuming that you bring the books back on time, a library is a completely free option. I have a library within walking distance of my home in Scottsdale, AZ, and I will visit it a few times a week to discover new books.

Additionally, you can sign up for book memberships. Safari Books Online offers an All You Can Read package. I have this membership and have gone through a large number of technical programming books in their database through the years.

## **Summary**

In summary, is reading important for developers? I believe that it is. Reading enables you to activate compounded learning. And if you have the chance to gain years worth of knowledge and experiences in a few weeks, it seems insane to pass up on an opportunity like that.

## 3.15 Guide to Memorization

This lesson introduces a guide to memorization, with a focus on how to create a system for memorizing code when first learning a new programming language.

During a recent bootcamp teaching session where I was walking through a number of front-end development techniques, a student asked a great question. Referencing the CSS styles, she asked:

“What is the best way to remember all of the specific style names and properties?”

This is a vital question to answer, especially for new students. For example, if you look at the CSS documentation you’ll find thousands of potential style options. If you’re learning these styles for the first time that list can be pretty intimidating. And that doesn’t even bring in the idea of learning how the styles work together with applications as a whole!

Obviously, this issue does not only apply to CSS styles. When it comes to learning development, whether it’s a programming language or framework, you will be greeted with a large amount of information that you’ll need to memorize, or at least know where to reference it.

### Guide to Memorization

At first glance, this may seem like a daunting task. And many aspiring developers have given up on their learning journey because it seems like an insurmountable challenge. However, I’m here to tell you that it’s completely realistic for you to learn how to work with a large number of complex concepts. And if you follow the system I outline in this guide you’ll be amazed at how quickly you pick up on memorizing more information than you ever thought possible.

### Repetition

Before I go into the memorization system I have used through the years, it’s important to say that repetition is the key to memorizing large amounts of information. None of the techniques I will give you are going to help if you don’t take the time to work through them consistently.

### Smarter, Not Harder

With that being said, it’s important to know that, by itself, repetition is a slow and naive memory training technique. As a development student, imagine that I hand you a list of a few hundred method names and tell you to memorize them. If you were to simply stare at the sheet of paper and try to memorize the names, how do you think you’d do? If you’re like me and the majority of the world, probably not good.

The reason why dry repetition isn't a great way to memorize names is because it doesn't give you a frame of reference for the names.

## Visual Mental Mapping

In the first memory technique we're going to walk through visual mental mapping.

Our minds are incredible at memorization. However, at the same time, our minds are also picky with how they store information. Let's run a quick test. If I show you 15 random digits, such as:

234  
348532  
984  
234523  
34534  
35234  
234  
25345  
234  
985  
553  
37434  
740  
423  
9812

And I give you 5 seconds to look at each number. How many of the numbers will you repeat back to me? Unless your name is Dustin Hoffman (Rainmaker movie reference... might be dating myself a bit on that one) then you probably won't be able to name very many off.

However, what if I showed you the pictures of 15 celebrities?

Now if I give you the same test as with the numbers. Do you think you'd do a better job remembering the list of celebrities or the random numbers? Assuming you know who the celebrities are you'd be able to repeat back a significantly larger number the celebrities than the numbers.

The reason for this difference is because you have a frame of reference for the celebrities and in this exercise you had a visual reference. By combining these two things your brain was fully prepared to recite back a larger number of items from the second list.

With this knowledge in mind we can apply the same principles for memorizing anything.

## Short Term vs Long Term Memory

Because our brains are efficient machines they naturally sort information based on priority. You are most likely aware that you have short term and long term memory. This concept is the reason why you can instantly remember your second grade teacher's name decades later, but may forget a new acquaintance's name 30 seconds after hearing it.

Typically the brain doesn't log knowledge into our long term memory bank unless it thinks we're going to need it in the future. This is kind of like how a computer works. If you add text to a document and save the file on the hard drive, that's like storing information in the mind's long term memory. However if you run a calculation in the terminal the computer processes the information in memory and then discards it, which is like our short term memory system works.

## Implementing Visual Mental Mapping



So when it comes to implementing the Visual Mental Mapping technique, we're essentially tricking our brains into thinking that it needs to move a piece of information into long term memory. In this process we associate a visual image with the term that we want to memorize. A key prerequisite for this to work is that the visualization needs to be relevant to the term (or the behavior of the term).

Getting back to the developer's initial question. Let's see how we can use visual mental mapping to memorize a CSS style. I'm going to use the text-decoration property as a case study. In the world of CSS, the text-decoration element allows you to add or remove an underline style to a piece of text.

With this in mind I would create an image in my mind that would look something like this:

So in this example I have an image filled with decorations. And on top of the image I have some text that is underlined. And it's sitting on the decorated fireplace mantle.

By creating this visual image, I've mapped:

- Decoration to underlined text
- A familiar image to something abstract

And with this mental image in place, I don't have to think about the term text-decoration, instead I will think of a decorated fireplace with underlined text sitting on the mantle. This visual is much easier for my brain to accept into long term memory because it has a direct frame of reference. The text-decoration word is no longer a foreign element trying to invade my memory. Instead it's catching a ride on an image that already has a home in my long term memory.

## Taking a Real World Example

Sticking with our celebrity theme. Imagine that you wanted to go to a private, VIP party in Hollywood. If you just try to show up the bouncer at the door most likely won't let you in. However if you're friends with Brad Pitt and you walk in together, you won't have any issues attending the party.

Visual mental mapping follows the same principle. Our brains guard our long term memory to ensure that our mind doesn't get cluttered with useless information. For example, what if you logged every piece of information into your long term memory that you come across each day? As you drive down the street to work your brain captures millions of data points such as: street signs, people walking, etc. If your brain didn't guard against useless information entering your long term memory bank, all of this information would be treated with the same priority as your parent's names. Obviously this wouldn't be a good idea!

So our brains are like the guard in the VIP Hollywood party. And when we attach a new piece of information to something already logged in long term memory, it's like we're having Brad Pitt escort us into the party.

## Finding Patterns

So Visual Mental Mapping seems like a great idea. However the idea of creating thousands of visualizations isn't very practical. Which is why, when I'm learning a new programming language, I also focus on picking up on patterns. Returning to our case study of memorizing CSS elements. Let's take a look at the border attributes available in CSS3.

1. border
2. border-bottom
3. border-bottom-color
4. border-bottom-style
5. border-bottom-width
6. border-color
7. border-left
8. border-left-color
9. border-left-style
10. border-left-width
11. border-radius
12. border-right
13. border-right-color
14. border-right-style
15. border-right-width

- 16.border-style
- 17.border-top
- 18.border-top-color
- 19.border-top-style
- 20.border-top-width
- 21.border-width

As you can see there are 21 available attributes. And that's just for managing border styles on a webpage! As you can imagine, it would be pretty intimidating to memorize this list, especially when you realize that it's only a very small percentage of the available CSS styles needed for development.

However if you start to analyze the list you'll notice a number of trends. For example there are a number of styles that simply reference: top, bottom, left, and right. These styles are simply ways for giving a border style to a specific side of an element. Additionally you may also notice that each side also has a set of options for color, style, and width.

So practically, if you know that these elements are all available to the border set of elements. This list can be shrunk down to 5 items:

- 1. border
- 2. border-color
- 3. border-radius
- 4. border-style
- 5. border-width

Which is more manageable.

## Copy/Paste is the Enemy

In addition to creating Visual Mental Maps and using patterns, I'm going to finish off the list of memorization techniques with the recommendation to not copy and paste new concepts that you're trying to learn.

I first heard this advice from Zed A. Shaw, the author of the Learn Hard programming book series. He instructs his readers to not even look at the book at the same time that they're implementing the code. He postulates that by forcing yourself to type in the code without referencing the documentation while typing, it forces the mind to actually think through each keystroke.

In my personal experience as a developer and with teaching. I've discovered a significant difference between the students that copied and pasted code or simply followed along with a tutorial, compared with the students that attempted (even unsuccessfully) to implement the code by themselves.

## **Not Everything Has to Be Memorized**

On a final note, I want to dispel a common fallacy. As a developer you don't have to memorize every class and method in order to build a project. Even professional programmers constantly lookup documentation on a regular basis. Instead of feeling like you have to memorize everything, focus on memorizing the terms that you use the most. This will make the memorization process more practical and natural.

## 3.16 Slowing Down to Learn How to Code Faster

Nowadays it seems like everyone wants to do things faster. We want to pay without taking out a credit card or cash. Social media lets us share images and videos from our lives in a split second. And we get frustrated if Netflix takes more than 3 seconds to start streaming our latest TV show series binge. However if you want to learn how to code faster I'm going to present an odd idea: go slower.

This may seem like a counter intuitive concept. Don't coding bootcamps, even devCamp where I teach, tell you how you can learn how to code in a few months? Yes, and research shows that 8 weeks is a powerful number when it comes to learning. The Navy Seal training program specifically chose 8 weeks as its time frame for conditioning candidates. And if you google 8 Week Training programs you'll find courses ranging from running 10ks to speaking Spanish fluently.

So I'm huge believer that individuals can learn an incredible amount of information in a short period of time. But what I'm talking about today is becoming more deliberate when it comes to learning new information.

### Learn How to Code Faster

If you're like me when you learn a new topic the first thing you'll do is either move onto the next topic or repeat the concept as quickly as humanly possible.

For example, when I learn a new Ruby or Scala programming method I'll usually jump right into using it in as many different situations as possible.

However I've discovered that this may not be the best approach because it's very short sighted.

### Our Default Mind

When it comes to learning how to code one of the most challenging requirements is moving knowledge from our short term memory to our long term memory.

Remember back to the last time you learned a programming technique. Do you remember how easy it felt when you repeated what the instructor taught? The syntax seemed straightforward and it probably seemed like there was no way you would forget how to implement the feature.

But after a few days if you try to rebuild the component, is it easy or hard? If you're like me the concept that seemed incredibly easy only a few days ago now causes you to draw a blank.



But don't worry. This doesn't mean that we're incompetent. Instead it means that this piece of knowledge wasn't given the chance to move from our short term to our long term memory.

## Hacking the Mind

So if our default mindset is to forget what we've learned after a few days (or a few minutes), how can we learn anything? This is where our brain's default programming comes into play and where we can hack the way that we learn.

I'm going to take it for granted that you want to learn how to code faster. I've messaged with a number of the readers of this blog and so far I haven't found anyone that was going through the posts because they were bored. So one technique I've been using to learn a new programming language has been to slow down.

I'm currently teaching myself the Typescript programming language. Typscript is the language that is recommended for Angular2 development so I thought it would be a good next language to learn. However instead of taking my default approach which is to slam through training guides and tutorials, I'm taking a more methodical approach.

## Slowing it Down

Through my learning path I'm going through a number of books and video series. And as I follow along with the guides, as soon as I learn a new topic I completely stop. I'll stand up. Write the new component on one of my whiteboards. And actually write the program out by hand.

After that I type the program out on the keyboard... very slowly. So slowly that I know I could go around 4-5x faster. But by taking this approach I'm forcing my mind to think about the new concept instead of rushing through it.

When it comes to working with our long term memory this approach is more effective than simply flying through a concept because it forces our minds to think through each keystroke.

## Bend it Like Beethoven

I didn't learn this technique from another developer. Instead I heard about how one of the most successful classical music institutions in the world, the Meadowmount School of Music in New York, taught students new music compositions.

As a game the school gives out portions of the sheet music. So where most schools will give each student the full song, Meadowmount splits the music up into pieces. From there it hands each student a single piece for them to focus on. From that point the student will only learn to play that single piece of music. They will start out very slowly. They won't rush through notes because they

don't even know how they fit into the song. This approach teaches them how to concentrate on learning a new song one note at a time.

From that point the students trade note cards and then focus on learning another piece of the song. They continue with trading cards until each student has been able to work through the entire set of cards.

By forcing the students to break a song into pieces they no longer will have any weak points in a song. Instead the students will have focused on the notes themselves. From this point it's trivial for all the students in the class to combine their knowledge and learn how to play the song all the way through.

## **From Classical Music to Coding**

So can this approach help you learn how to code faster? I think so. The research shows that by slowing down and breaking concepts into small pieces it's easier for students to transfer information from the short term to long term memory.

## **A Practical System**

So the next time you are learning a coding concept take a step back. Instead of simply copying what the instructor is teaching, write it down on a piece of paper. Walk through exactly what is happening in a program.

If you take this approach you will discover that you're not longer simply following a teacher's set of steps, but that you'll actually learn how the concepts work. And if you get to the stage of understanding, you will be ready to transfer that knowledge to your long term memory and remember it for good.

## 3.17 Mental Models for Learning How to Code and Improve as a Developer

I talk quite a bit about what it takes to become a great developer. In order to achieve a level of mastery I've discussed a number of criteria and today I want to add a new pre-requisite to the list. Let me begin by asking you a question. If I showed you some code, would you be able to tell me in a few seconds if it's good or not? The world of software development is incredibly complex. However I've discovered through the years that the best developers have the uncanny ability to instantly judge the quality of someone's code. I've spoken at length that the concepts of prodigies and savants are a myth. But if this is the case how can expert developers be able to analyze programs so quickly? To answer this question we need to go back to Fake Ancient Greece.

### Mental Models for the Kouros

I said Fake Ancient Greece because my favorite illustration of mental models was discovered alongside one of the greatest forgeries in modern art history. In Malcolm Gladwell's book *Blink*, he tells the story of the Greek Kouros. In 1985 the Getty Museum purchased a Greek statue called the Kouros, for over \$9 million dollars.

Initially the museum was hesitant to purchase the statue because there was a significant fear that sculpture was a fake. Kouros pieces were so incredibly rare, the chances that a legitimate and well cared for piece had been discovered were slim to none. However the museum was willing to take the risk and embarked on a fact finding mission. They put the statue through every scientific test available at the time. And the Kouros passed with flying colors. After going through the full examination the museum purchased the Kouros for \$9 million dollars.

Art historians from all over the world were flown in for the unveiling of the Kouros. But something went terribly wrong. The moment that these specialists saw the statue they knew the Kouros was a fake. Interestingly enough they couldn't give any actual reason. They simply knew that something was not quite right.

Their suspicions turned out to be correct and the Kouros ended up being proved to be a hoax. But how were these individuals able to do what countless scientific studies could not? It all comes down to mental models.

### What are Mental Models?

In preparation for this guide I was discussing the topic of mental models with a friend and was surprised when she looked at me confused. After informing me that she'd never heard of mental

models I decided to add in this section to explain what mental models are. And after that we'll get into how we can build them to learn development.

A mental model is a mental representation of a specific topic or skill. You can't create a mental model overnight or with 'cram' sessions. Mental models are developed through years of repetition and countless hours of honing a craft.

My Dad is a major league hitting coach for the Houston Astros. Throughout my life I've been able to watch him instruct hitters on how they can improve their swings. And I'll never stop being amazed by the fact that he can watch a new hitter's swing and within a split second pick out multiple ways that the player can improve. I can tell you that he did not develop this skill in a short period of time. He has spent more time watching hitters and film than anyone I know. And through the years he has developed a mental model of what the perfect swing looks like.

## **Mental Models for Developers**

Ok, so we've talked about art historians and baseball coaches, but how can we create mental models as developers? You may or may not like the answer, however it doesn't really matter because it's the truth. Mental models are made through repetition. However repetition by itself isn't enough. For example if you built an identical program everyday for 10 years you would get really really good at building that one application. However you wouldn't improve as a developer.

I've talked before how medical research shows that doctors who have spent years practicing on the same types of patients are less proficient than doctors fresh out of residency. In the same way, as developers we improve when we're stretching ourselves each day.

You can stretch yourself by doing things such as:

- Learning a new programming language or framework.
- Teaching others how to learn programming.
- Creating an open source code library and allowing other developers to use it.

Einstein said it best when he said:

“The only source of knowledge is experience.”

## **Summary**

If you dedicate enough time each day improving yourself as a developer you will be able to truthfully answer 'yes' to the question I posed at the start of this guide. You will be able to have the ability to look at a piece of code and instantly know if it's good or bad. And you'll know that it's not some type of coding super power, instead it's a skill that you earned through your constant pursuit of improving as a developer.

# 3.18 Practical Ways to Use the Pomodoro Technique as a Developer

As we continue to work through ways to hack the developer's mind, the focus of this guide is going to be on increasing productivity. Specifically, we're going to analyze practical ways to use the Pomodoro Technique.

I'm constantly researching new ways to improve my personal productivity. And through my journey as a developer, a popular approach that I've discovered is the Pomodoro technique. This is a process that I've utilized and I credit it with allowing me to focus on a large number of tasks each day.

## Pomodoro Technique Definition

Don't let the weird name scare you away. The Pomodoro technique is a dead simple productivity system that focuses on splitting tasks into timed intervals throughout the day.

## Practical Ways to Use the Pomodoro Technique

## Pomodoro Technique Process

One of the greatest strengths of the Pomodoro technique is how easy it is to implement. The process that I follow is:

1. Each morning I pick out the tasks that I want to accomplish that day.
2. I then decide how long each task will take. The Pomodoro technique works on a point system. Each time you work through a 25-minute task you earn a point.
3. Typically I try to earn 10 Pomodoro points each day. This means that if I have 3 tasks that I know will take an hour each, I will earn 6 points for those tasks. And it means that I have 4 additional 25-minute slots available for the rest of the day.

## Taking a Break

Did you notice how I kept saying 25-minute time slots? There is a reason for the odd number. The Pomodoro technique places a high priority on taking scheduled breaks. After completing each 25-minute task, you take a 5-minute break. During this free time, you can do anything you want. You

can get on social media, you can take a walk around the block or anything that you want to do. Just make sure that your break does not exceed 5 minutes.

Also, after you've completed 4 tasks it's recommended that you take a 15-minute break. However, you can tailor your breaks and intervals to what works best with your schedule.

By planning breaks throughout the day, you will decrease your chances of burn out. And I've noticed that I no longer feel bad about doing things such as checking my Instagram account or Hacker News throughout the day because I can fit my guilty pleasures into my scheduled free time.

This is one of the aspects that I truly love about the Pomodoro technique. Many of the other productivity systems I've tried in the past tend to lead individuals towards burning out. However, the Pomodoro approach allows you to have a sense of balance.

## **Lifestyle vs Fads**

Have you ever tried dieting before? When I was younger I struggled with my weight and to help fix it I tried a number of intense diets. This included nutrition strategies such as dramatically decreasing calories, or killing off carbs. However, I noticed that I'd stay true to the diet for a few weeks or even a few months. However, eventually I would fall back into poor eating habits. Once I recognized this trend I moved to having a balanced approach to eating. I stopped trying nutritional fads and I transitioned my focus into eating in a way I felt I could eat the rest of my life.

I made this change in my nutritional approach a few years ago and it's completely stopped my roller coaster dieting and weight loss and weight gain.

## **A Lifestyle of Productivity**

In the same way, when I was younger I fell into the same pattern with working on tasks. I'd get excited about working on a project or learning a new programming language. And I would spend countless hours working on what I wanted to accomplish. However, this approach inevitably led to burning out and large stretches of time where I didn't want to work at all.

I look at the Pomodoro technique in the same way that I look at having a balanced diet. By limiting the number of tasks that I work on each day and by implementing planned breaks between each task, I no longer burn myself out. Additionally, after I have finished my work for the day and have earned my 10 Pomodoro points, I feel a sense of accomplishment that I never felt before. And after work, I don't feel guilty spending time with my family and friends, because I know that I completed every task that I set out to work on that day.

## Practical Implementation

So how can you implement the program? There are a few ways.

To start off you can simply use the timer on your phone and then count up each of the tasks/points that you achieved each day. That's how I started off working with the Pomodoro technique.

Additionally there are a number of smartphone apps that have pomodoro timers and even allow for creating a task list that you can use as a pick list for your tasks each day. I like these types of apps because they also give you historical analytics so you can see how many tasks you've completed each day. The pomodoro focus app is my personal favorite (and it's free).

# 3.19 Development Study Tips: Reverse Note Taking

In this guide we're going to go back in time and walk through when I developed the system of reverse note taking. A quick Google search will show that I have coined the term, however I did not invent the process.

Back when I started computer science grad school at Texas Tech I was struggling with one of my classes. It had been about a decade since I had been in a classroom environment and I was having a difficult time paying attention to the 1.5-hour lectures.

## The Problem with Traditional Note Taking

During this time I spent quite a bit of time meeting with Dr. Richard Watson. And during one of our meetings, I brought up the issues I was having. His first question was based on how I was taking notes for the course.

I showed him my notes and he instantly told me that I was taking notes completely wrong. He pointed out multiple places in my notes where I had missed key concepts that were unifying elements. And without noting these items I wouldn't understand the topics at all.

In reviewing the notes I realized he was completely right. I spent my time writing down facts and what I thought were key terms. However, I regularly failed to articulate how everything worked together.

For example, for my notes on tree data structures, I outlined each of the key elements of binary search trees and B-Trees. But I failed to describe the innate differences of the three components from a behavior perspective. This would be similar to taking notes in a history class and writing down the names, dates, and locations for Napoleon's loss at the Battle of Waterloo without describing the critical differences between his old armies with the one he lost with.

## Reverse Note Taking

Finding out that I was taking notes wrong was great. But it wouldn't have been too useful without learning an alternative approach. So Dr. Watson asked me to try a different type of note-taking technique.

He said to put my pen and paper away during class. And instead of taking notes during class, he recommended that I simply listen to the lecture. He instructed that as soon as the lecture was over I



should find a quiet place and THEN write down all of the topics that I remembered from the discussion.

Initially, I was skeptical of this approach, mainly because I was afraid that the important concepts would go in one ear and out the other. He added that I should tape-record the lecture so that I could use the recording as a safety net for the topics that I failed to remember.

Despite my negative perspective on the approach I decided to give it a try. (Obviously, my natural note-taking approach wasn't effective, so I didn't have much to lose).

I followed Dr. Watson's advice to the letter. And I was pleasantly surprised to discover that I remembered much more information using this reverse note-taking approach compared with simply trying to write down concepts during the lecture.

## **Benefits Reverse Note Taking**

I started following this reverse note-taking process years ago and I still use it today. Through this time I've noticed a number of key benefits to this approach.

### **Narrowed Focus**

First and foremost, having the knowledge that I will have to recite back the key components of the lecture forces me to have an increased level of focus. This is the opposite of how I used to take notes. My old way of taking notes would many times distract me from the concepts being discussed. I would hear a concept that I felt was important and I would take my focus away from the speaker and focus on writing down the topic.

Many times this would inadvertently result steal my focus away from another important concept, or a description on how the topic I was writing down worked at a high level.

Additionally, as a naturally competitive person, I would make a game of how much I could remember from each lecture. If I was able to remember enough to write down two pages of notes on Monday, I would try to write down two and a half pages on Tuesday. By making a game of the practice it forced me to narrow my focus even more on the content.

### **Story Based Mindset**

Another benefit to reverse note-taking is that it forced me to think of the lecture as a unified story instead of a series of facts. Let's go back to our illustration of Napoleon's battle at Waterloo. If you listen to a lecture about the battle and take notes during the class you'd probably do things like write down:

- General names
- Cities where battles took place

- Dates
- Etc

However, if you simply listen intently to the lecture and recite it back afterward you won't repeat dates and locations. Instead, you will naturally remember the battle in story form. You'll discuss the struggles that the Duke of Wellington had to overcome in order to lead the charge against the French army. And because it's a story, your retention of the topics will be considerably higher compared with attempting to memorize facts and figures.

If I were to ask you to remember back to a high school history class and to a movie you saw in high school, would you have a better chance of remembering the plot of the movie or the history lecture?

So getting back to my computer science grad school experience. By leveraging the reverse note-taking strategy I forced myself to think of the topics discussed during the lecture as a story as opposed to a bunch of theories and math equations.

## **Forced Repetition**

Lastly, the reverse note-taking approach made it easier to review the lecture material compared with my old style of note-taking. Previously I rarely would listen to a lecture recording. Even if I had the intention to listen to the recording, other priorities always seemed to override the task. I mainly attribute this failure to the fact that I, for some reason, trusted my notes.

However, when I started reverse note-taking I would always listen to the lecture a second time to fill in any items that I missed during my post note writing session. I discovered this single benefit to be critical to my success since it became an automatic habit to reinforce my knowledge. In contrast to my old approach where I trusted my untrustworthy notes, with reverse note-taking, I didn't trust my memory, so I knew I had to reinforce my memory. And the consequence was that I always would listen to a lecture twice, with the final result being a dramatic increase in retention.

## **Summary**

This approach is not for everyone. I know students who excel with a more traditional note-taking strategy. However, if you find yourself in a situation like me I highly recommend you giving reverse note-taking a chance. You may be surprised how effective it can be.

## 3.20 Task Switching Costs for Developers

Task switching, commonly referred to as multitasking, can be detrimental to your performance as a developer and can even lead to errors in your projects.

Our world has changed dramatically over the past decade, whether for good or bad is not a topic we'll discuss today, however one thing is sure: we are constantly bombarded with distractions. As I was researching this post I received over a dozen emails, 7 Snapchat messages, 30 notifications on Instagram, 7 Twitter notifications, 5 Skype instant messages, and surprisingly only 9 text messages. If you were counting that's around 72 various notifications that were pushed to me in the past two hours. Beyond that, I researched this post at a coffee shop filled with potential distractions.

So exactly how bad are distractions? Research from Gloria Mark, who is a Professor in the Department of Informatics at the UC Irvine shows that it takes, on average, 23 minutes and 15 seconds to get fully back on task after being distracted. That's a very very bad thing when it comes to productivity, however, I've seen it myself, I've lost track of how many times I'll be in the middle of a development project and receive an email on a completed unrelated matter and instead of ignoring it and continuing to work, I'll read it and then spend time working on another task before returning to the project. This may not sound like a major issue, except that when I come back to the project I don't pick up from where I left off. Instead, I have to re-familiarize myself with what I was working on the moment that I was distracted. If the problem was complex it may take me even longer than the 23 minutes in order to get back in the zone and work on the project.

So in a world filled with emails and social media distractions how can anyone get any real work done? After reading Cal Newport's book "Deep Work" I started to put together some practical ways that I can work efficiently and still stay in touch with the world.

### System for Decreasing Task Switching Costs

If I'm working on a project I set aside a specific amount of time that morning. For example, if I'm working on Project X for 2 hours, I will put it on my calendar and say that from 9-11 am I'm working on Project X.

---

I remove any and all negative distractions during that time. That means I'll usually put my phone on Airplane mode so I don't receive any social media notifications. Notice how I said 'negative' distractions? I made this distinction because in the same research report from UC Irvine it revealed that not all distractions are bad. If the distraction is related to the task that you're working on, it can actually be beneficial. For example, if I'm working on the routing engine for a web application and the client messages me to discuss the application, what they say may actually influence the work that I'm doing or

give me an idea on how to refine it. That's a good distraction and it's why I typically will keep my email and instant messenger on while I'm working. However, if I see that the Skype message or email is coming from another client or is completely unrelated I'll simply ignore it. I do know many Deep Work proponents who would say that 100% of your distractions have to be eliminated, however, that's not always practical.

---

Have a clear conclusion for whatever you are studying or working on. If you don't establish an end for the task, your mind is going to be prone to wander in the same way that a runner without a finish line won't be able to effectively compete in a race. The research around task-switching costs also reveals that even planned distractions are harmful, so if you are planning on working for 2 hours straight on a project don't plan any breaks in the middle of the task. Maintain your focus throughout the allotted time and then you'll be free to relax afterward.

I hope that this has been a helpful overview of task switching costs and that you now have some practical methods for staying on task.

## 3.21 The Power of Making Mistakes – Learning by Failing

Let's take a step back in time back to my first semester of Computer Science grad school. Stepping into my first class I was filled with nervous excitement. The class was taught by Dr. Gelfond, one of the most respected individuals in the artificial intelligence sector.

As the class progressed I witnessed a disturbing trend. Instead of simply lecturing us like our other professors, Dr. Gelfond constantly called students upfront to write programs on the chalkboard or to describe a concept he discussed.

This wouldn't be a big deal, except that he made a habit of calling us upfront specifically when it was clear that we did not understand the concept.

Was he cruel? Did he want to make us look ignorant in front of the entire class?

### **Secret Weapon to Mastery: Making Mistakes**

Actually, the opposite was true. Instead, Dr. Gelfond cared enough about us that he imparted to us the secret weapon to mastery: making mistakes.

Wait, making mistakes is the opposite of what our mind tells us to do, right? Making mistakes is embarrassing. Mistakes tell the world that we don't understand a concept.

However, making mistakes also provides a number of powerful tools that anyone interested in learning should be aware of.

### **Making Mistakes: Memory Steroids**

First and foremost, when you make mistakes, especially publicly, you're going to feel like you're taking memory steroids. How so? When I think back to Dr. Gelfond's class I still remember every mistake I made when I was called in front of the class. The memories generated by making mistakes are so vivid that they can be recalled, even years later like mine.

Now obviously simply remembering the mistakes by themselves would be pointless. However in addition to remembering what I did wrong, more importantly, I remember what I had to do to correct my mistake. It's been over three years since I took that class, but I can still remember each of the key concepts that he taught us. And I can tell you from experience that I cannot say the same thing about all of the classes I've taken.

## Mistakes Force Learning

Another benefit of making mistakes is that they force you to learn. No one likes being wrong. So assuming that you have a passion for knowledge, you can use the memory of making mistakes to help motivate you to learn a concept properly.

If Dr. Gelfond would have simply stood in front of the class and lectured for the entire semester I most likely would have studied enough to do well on the tests and leave it at that.

However, because I constantly had the thought in the back of my mind that I may have to be called up in front of the class to write a program or describe a concept it forced me to study harder than I would have for a test. This healthy fear took me from simply being able to remember a concept to truly mastering it.

## Mistakes Kill Pride

Lastly making mistakes helps to kill pride. Proverbs 16:18 says:

“Pride goes before destruction, a haughty spirit before a fall.”

One of the largest obstacles to learning is pride. Anyone puffed up with pride will find that their learning progress will come to a halt. When someone is filled with pride they can't see beyond their own limited knowledge.

Thankfully, if you embrace the process of learning by making mistakes, pride will never be able to stake a claim in you. By their very nature mistakes force you to realize that you don't know everything and that you have more to learn... which we all do.

## Summary

So whether you are just learning to code from scratch or if you're a seasoned developer, never be afraid to make mistakes. Mistakes reveal that you're traversing into new territory that you've never been before, which is what you need to do to go from mediocrity to mastery.

# 3.22 Learn How to Code from Scratch – A Practical Strategy

Becoming a developer is a rewarding yet challenging task. One of the greatest blocks for people to understand programming is simply having a plan and deciding where to start. In this guide, I'm going to walk through strategies to help you learn how to code from scratch.

I've been a developer for a number of years. I taught myself how to code and I've witnessed a wide variety of educational techniques for learning programming over the past decade. Some of the strategies I've seen are good, others are a waste of time. This list contains the strategies that have stood the test of time and will help you launch your coding journey.

## Learn How to Code from Scratch

### Small Bites

First and foremost on the list of tips to learn how to code from scratch is the principle of small bites. I have a friend who trains professional and Olympic athletes for Adidas, named Mark Verstegen. Back when I used to train at his institute he would always say something that really stuck with me. When any athlete presented a tough goal, such as qualifying for the Olympics or making it to the big leagues he'd ask them:

“How would you eat an elephant?”

After the athlete would look at him with a confused look he'd follow by saying:

“It's not a trick question, the only way to eat an elephant is one bite at a time.”

This is great advice for many aspects of life. However, I've discovered that it's an especially important concept for developers to understand. When I think back to when I learning development, my greatest came when I tried to do too much.

For example, when I was trying to build a new feature I would attempt to code the entire feature at one time. Most of the time this would end up with the program not working and then I'd had to go through every line of code until I figured out what was wrong. However, the more experienced I've become as a developer the more I realize the importance of breaking concepts down into small, easy to manage chunks. Let's imagine that you are building a connection to the Twitter API. Instead of trying to build the entire feature, focus first on connecting to the API. Then print the values returned from Twitter. Finally you can format the data so that it looks nice.

By breaking what you're learning into small components, you'll discover that you will have a better understanding of the processes going on. You will also be able to remember how to implement the features later on in real-world projects because the concepts will be more tangible.

## **Tutorials**

Over the past few years, the online educational space has grown exponentially. Whether you are looking to learn Java or Ruby you'll be able to find countless tutorials that will help you understand programming. These types of tools most likely won't turn you into a professional developer by themselves. Since achieving a professional level of skill takes years and typically requires you to work on a wide range of real-world projects.

However, tutorials can be a great introduction to programming. In addition to giving step by step guides for how to build applications, screencasts are also great for showing you what types of apps a specific language or framework can build. When I'm learning a new language I'll watch a full series of tutorials without even trying to type in the code. I do this so that I can familiarize myself with the capabilities of the language.

One of the weaknesses with tutorials is that it's hard for them to replicate your own environment. For example, if you're working on a Java programming language tutorial from a few years ago, there's a good chance that the instructor will have a different language version than you do. This will cause some confusing bugs and without any assistance, many individuals have quit their programming dreams out of frustration.

But don't let that scare you away from using tutorials. I credit a number of tutorials with helping me teach myself development. And I highly recommend them as a great place to start. Especially when you want to learn how to code from scratch.

## **Reading**

Next on the list is reading. Libraries could be filled to the brim with the number of programming books that are on the market. I have even written a few! I like going through coding books because they allow me to go at my own pace. When I go through tutorials it usually means that I need to dedicate a specific amount of time to go through the videos each day. However, with a book I can read a few paragraphs or I can go through a few chapters.

When you have a full-time job and you're learning programming on the side books are a great resource. This is because they allow you learn at your own pace. Books can also be a good resource later on when you need to reference a specific topic.



Also, when you go through a programming book I highly recommend you write and run the code from the book. This will help you remember the programming language syntax much better than simply reading it. Remember that reading retention is incredibly low in most individuals. However if you combine reading with actually writing the code as you're going through the content you'll see much better results.

Another trick to use when reading programming books is to not look at the book when you're writing the code. For example, if you are reading my Ruby programming book you'll see a code snippet like this when you're learning how to use object-oriented programming. If you force yourself to type the code without looking at the book the entire time, you'll discover that your retention will increase dramatically.

## **Real World Projects**

Last on the list to learn how to code from scratch is building real-world projects. After you've gone through a number of tutorials and read a few books you'll be ready to try your hand at building applications. A natural question to ask is: "What types of projects should I build?"

There's really no right or wrong answer to this question. If you have an idea for a business then could start with trying to build it with your newfound coding knowledge. You could also look at re-building current applications. Such as creating a Pinterest clone.

I've found this technique of creating cloned sites very beneficial since it allowed me to focus on building functionality instead of having to waste time on coming up with ideas. For example, when I learned the Swift programming language I built an Instagram clone. Years ago when I was learning HTML and CSS I re-created the Google homepage from scratch.

The most important factor to remember about building real-world projects is to stretch yourself. No developer ever improved by duplicating functionality they are already comfortable building. Instead, make sure you are challenging yourself to implement features that you've never created before.

## **Determination**

### **Coding is Hard**

On a final note, don't let anyone tell you different: coding is hard! From setting up a development environment to building functional applications, programming will greet you with challenges at every stage.

## **But You Can Learn Programming**

However, with that being said, you can become a developer. There's not a magical programmer gene that coders are born with. It simply comes down to:

- How determined you are
- If you're willing to work consistently
- How good your strategy is when it comes to learning

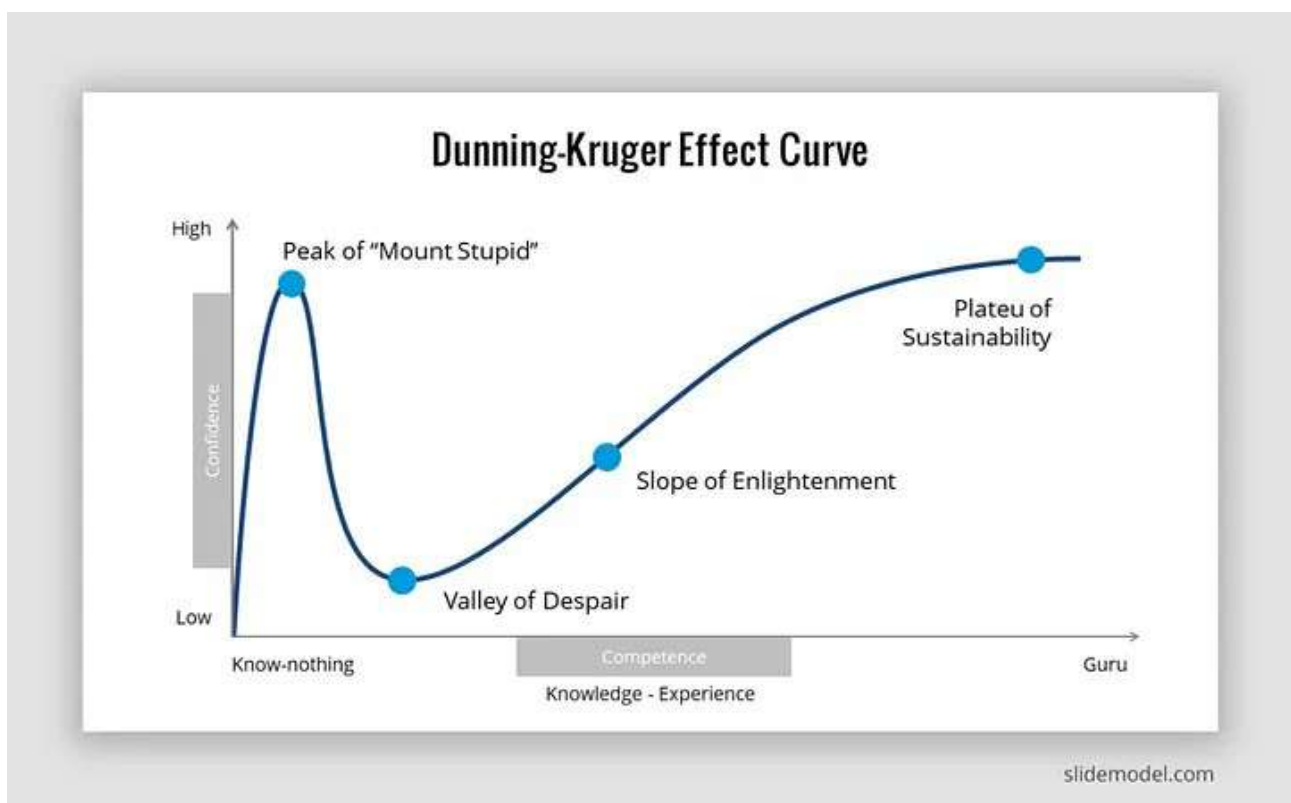
## 3.23 Developer Learning Curve – Why Learning How to Code Takes So Long?

When it comes to becoming a developer, one of the questions I get asked the most is: Why does it take so long to learn how to code? I've discovered the answer can be found in research related to learning curves.

### What is the Learning Curve?

The concept of learning curves has been around since 1885. Typically the research has been performed in the psychological and cognitive sciences. However, the concept can be clearly utilized when it comes to learning development.

### Developer Learning Curve



In this image is the standard learning curve. This was generated by a big data analysis algorithm that analyzed the learning patterns of individuals in a number of industries. The curve is smooth because it takes the average learning process and averages the process.

Later in this guide, we'll take a look at what a learning curve looks like for a single person.

Over the years I've had the privilege of teaching students how to become developers. I've witnessed this learning curve play itself out again and again. And in this guide, I want to examine the three stages that all developers go through. Additionally, I'll discuss about how long it takes to traverse from one stage to another.

The three stages that I'll discuss are:

1. Know-Nothing
2. Knowledge/Experience
3. Guru

## **Know-Nothing: The lift off / The Stupid Zone**

Let's start off by taking a look at the liftoff stage. This is an exciting time for new students. During this time students are immersed in learning skills that they've never seen before.

Because all of the topics that students learn during this stage are new, their expertise skyrockets. I like to call this the liftoff stage because it's easy to visualize a new student's expertise like a rocket ship soaring into the sky into places it has never been before. During this time a student will learn how to:

- Configure a development machine
- Learn a programming language
- Work with various frameworks
- Build functional applications

This stage usually lasts for the first 250-300 hours that a developer is learning how to code. This estimate is based on what I've seen with the DevCamp bootcamp students and equals around 6-8 weeks of intensive learning.

As fun as this stage is, it has drawbacks. One of the key problems is that it can give students false confidence. When they see themselves building applications that actually work it's natural to believe that they can dive right into building production apps for clients.

However, they don't realize that they're about to enter... the twilight zone of learning how to code.

## **Knowledge/Experience: The Twilight Zone / From Valley of Despair to Slope of Enlightenment**

After the exciting liftoff stage of the developer learning curve, aspiring developers will enter the twilight zone. This is a challenging time for students and many students decide to quit programming entirely during this stage.

So why is this time so challenging? After seeing countless students go through it I've discovered that there are a number of contributing factors.

- While in this stage, many of the core concepts and commands haven't cemented themselves in a student's long term memory. This results in them having to constantly look up documentation, query StackOverflow, and things like that.
- During this time the novelty of simply having an application work has worn off. Now students are asked to perform advanced tasks such as: Working with legacy applications, Debugging defects, Improving performance, and Building features that they don't have a step by step tutorial for
- Additionally, while working through the twilight zone, students are expected to start implementing best practices. In the launch stage, the primary goal was to get applications functional. During this next phase, students start learning how to build applications that can be used in real-world scenarios. This means that a student may spend five times longer to build an application with the identical feature of something they created during the launch stage. This can be frustrating, however, the increased time spent implementing best practices allow the applications to be scalable and flexible enough to be used in production. This is in stark contrast to the apps created during the launch phase that don't adhere to industry standards.

## The Zone

There is good news though. If a student persists through the twilight zone of learning they will enter The Zone of the developer learning curve. This zone is entered usually after around 1,000 hours of study and work. During this stage, developers have a wide range of features they can build without having to look up the documentation for.

In this stage when you visit StackOverflow you'll be answering as many questions as you ask. And thankfully learning new concepts will come easier. The reason why learning is easier at this stage is because you will have developed a mental model of development.

For example, I recently started working with the Scala programming language. I've been able to pick up on how to build applications in Scala dramatically faster than when I started learning C or PHP a decade ago. This is because I have a decade of knowledge in the development space that allows me to frame the new concepts. When I read the documentation and see what it says about data types I don't have to wonder what a data type is. Instead, I can skip ahead to learning the syntax.

As you'll notice in the developer learning curve, the growth pattern in this phase is less than the other two stages. As you've heard me say countless times, learning never ends for developers. However, learning does change. During this phase, a developer focuses on learning topics such as:

- Incremental performance improvements
- Building helper code libraries
- Refining how application code flows
- And overall best practices

## **Guru: Literally "The Zone" / Plateau of Sustainability**

Throughout this guide, you may have noticed that the developer learning curve was smooth. However, that's not reality. The reason why the curve was smooth was because it averaged out the learning path of a large number of individuals.

When it comes to a single student, the learning curve looks more like this image. There are ups and downs throughout the learning cycle. As a student, you may decide to switch programming languages after a few years (like I did when I switched from PHP to Ruby around 5 years ago). Even though you don't have to start back from scratch, it will still take time to learn a new language or framework. And throughout your development journey, you'll discover plenty of ups and downs. So don't get discouraged if you aren't satisfied with your skill level. Because I have a secret to tell you: good developers never feel like they've arrived and are done learning.

## **Summary**

I hope that this has been a helpful guide to understanding the developer learning curve, and good luck with the coding.

## References:

[The Dunning Kruger effect for programmers](<https://udenna.medium.com/the-dunning-kruger-effect-and-programming-c5a74071bce3>)

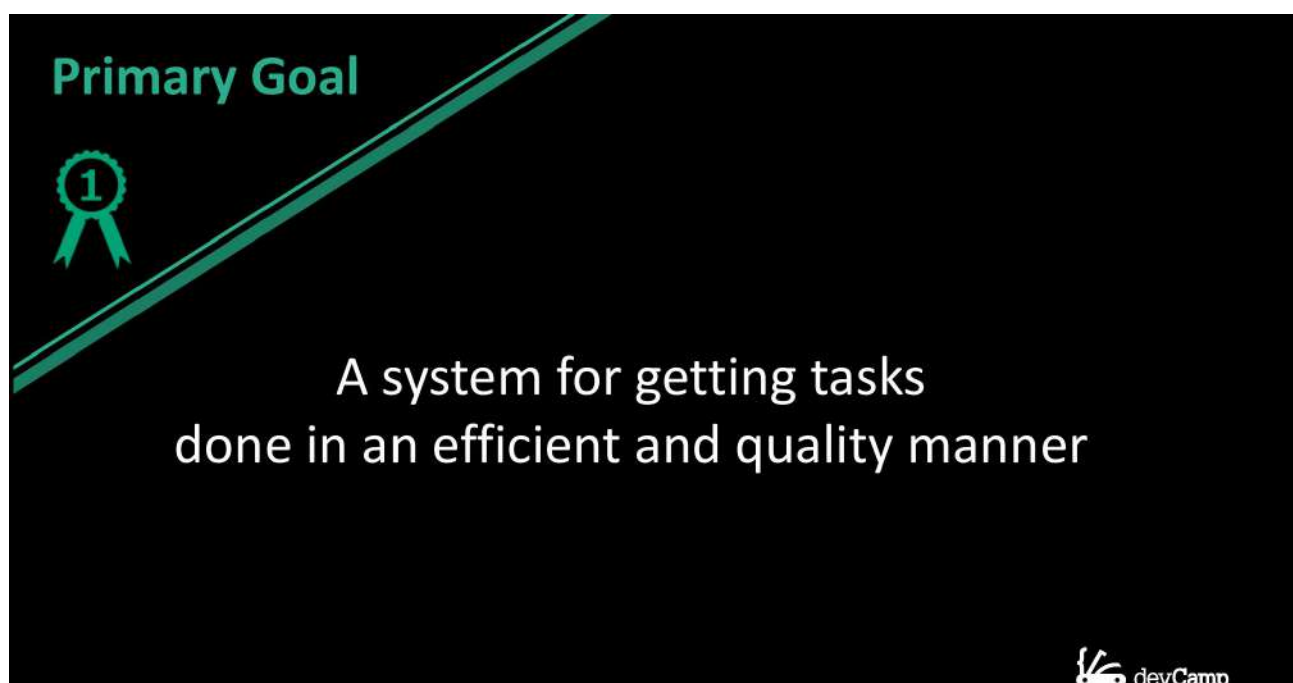
## 3.24 Three Practical Approaches to Task Management

One of the most common questions that I get asked is how to properly manage tasks on a day to day basis and this can come from the perspective of a developer an employee an entrepreneur or simply a lifelong learner.

All of us have a wide ranging set of tasks that we have to get accomplished on a daily basis and being able to properly manage those can be quite challenging. And so what we're going to walk through in this guide are a number of potential solutions for how to build out a task management system.

The reason why I'm going to give you multiple options is because it's been my experience that I might go with one approach for a few months or even a few years and it works well. But I'm continually adapting it because everything in our lives change and we ourselves change and we adapt and so, therefore, it makes sense that the way that we manage our day to day tasks should also adapt over time.

Now the very first thing we're going to do is we're going to talk about the primary goal of being able to manage tasks.



Then, we're going to walk into the formal definition and see what this represents and that's going to help lead the way for what our system is going to look like.

So the primary goal for task management or building this type of system is that it is a system for getting tasks done in an efficient and quality manner. And I picked out the wording for this in a very intentional way and it's because there are three components that we can analyze here.

First is that tasks have to get done. So that may seem like a common sense kind of approach, however, how many tasks do you have that have simply been sitting in your inbox to get done for a long period of time and you've procrastinated through them and they simply haven't gotten done. I know that that is something that I'm definitely guilty of. And so the very first item on the agenda with any task management system is that it helps you achieve this first goal of going from start all the way through finish on a task.

Now some of these tasks might be very simple it may be as easy as clearing out your inbox or it could be a task that is actually something that lasts for six months such as building out an entire web application. So the task size can vary but the goal is always the same and that is that you should be able to take it from start all the way through finish.

Now we have two other components here. The next is that we want to do things in an efficient manner. So this means that we limit the amount of wasted time and resources and our system should go and it should help us achieve that goal. And then lastly these items should be done in a quality manner. So with these three components completion efficiency and then quality if you can build your system to be able to help you achieve each one of these goals then you're going to be in very good shape regardless of what the tasks are, those are three components that you always want to get done.

Now I'm going to go into the definition from Wikipedia not to bore you but instead to help give us a framework for what a task management system really should contain.

## Definition



*"Task management is the process of managing a task through its life cycle. It involves planning, testing, tracking, and reporting. Task management can help either individual achieve goals, or groups of individuals collaborate and share knowledge for the accomplishment of collective goals. Tasks are also differentiated by complexity, from low to high."*



So it says that "Task management is a process of managing a task through its lifecycle. It involves planning, testing, tracking, and reporting. Task management can help either individuals achieve goals or groups of individuals collaborate and share knowledge for the accomplishment of collective goals. Tasks are also differentiated by complexity from low to high."

Now I want to highlight a few key words here

## Definition



*"Task management is the process of managing a task through its **life cycle**. It involves **planning, testing, tracking, and reporting**. Task management can help either **individual** achieve goals, or **groups of individuals collaborate and share knowledge** for the accomplishment of collective goals. Tasks are also differentiated by **complexity**, from low to high."*

Those are life cycle, planning, testing, tracking, reporting, individual goals, group goals, collaboration, sharing knowledge, and complexity. And the reason why I picked out these words is because they go into all of the different elements that a task management system should really contain.

Now a few of those are optional. So if you do not work with a team then collaboration and sharing knowledge may not be a big priority and so whatever system you use won't have to have those components. But in general each one of these key words is necessary in order to build out a fully functional task management system. So now that we've taken a high level view of task management.

Now let's talk about some of the pros and cons. So in the pro category a properly structured task management system is great because it helps to provide structure and organization. It allows you to

## Pros

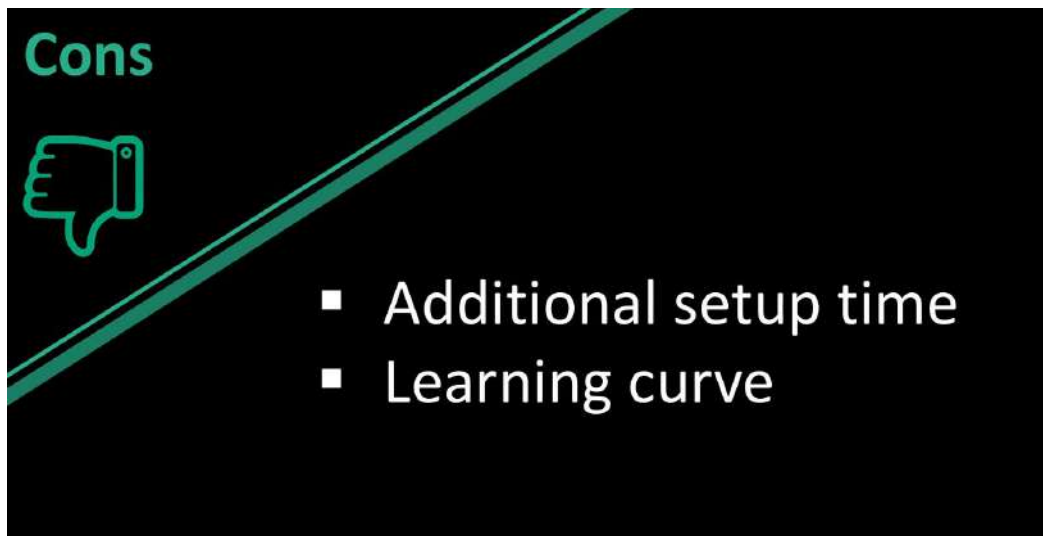


- Provides structure
- Tracking/reporting
- Collaboration
- Decreased stress

perform tracking and reporting. It allows for collaboration and one of my favorite pros is that it decreases stress. Research shows that one of the top causes for stress and anxiety in individuals is uncertainty.

And so if you as a individual have a high level of uncertainty on the tasks that you're working on that is going to be a very significant cause of stress in your life. And so what a task management system should do. One of the top goals is to provide a level of structure and also to decrease uncertainty. So when ever you have been able to properly outline and structure your goals on a daily basis that should give you a level of certainty because it tells you exactly what you want to get done on a day in day out basis.

Now that we've talked about the pros let's mention a few of the cons.



One is that there is additional setup time there is no way around this. If you want to properly implement a task management system you're

going to have to dedicate and allocate the time needed in order to build it properly and then also to maintain it. I personally spend anywhere between 10 and 30 minutes a day working through and planning out my activities. And this is very common in pretty much every task management system I've ever implemented. And so you are going to have to have that time sliced out of your day. Now also there can be a learning curve. So depending on the type of system that you want to implement there are various things that you're going to have to learn in order to build it out properly.

I do not think that either of these cons detract enough in order to say that you shouldn't have some form of task management system and what we're going to do is we're going to go through three options and these are three systems that I have personally use throughout the last few years and each one of them has proven to be very effective.

The three that we're going to talk about are going to be



daily task journaling, Kanban and then a traditional to do list in the first case study we're going to go with is daily task journaling and the example that I'm going to use is from a company called [Best Self Co](#) I've use their service for a while and it is a system where they print and structure an entire journal. This small book and they send you a new one every 12 weeks. It is a paid service. But each one of the components that are included can be something that if you do not want to pay the money but you still want to implement daily tasks journaling as your system then you can do all of this yourself completely for free.

I'm simply going to walk through their structure as because there were a number of benefits and I got quite a bit out of it when I was going through their system and every daily task journaling system is a little bit different and so what I want to do is include some of the more popular elements and then you can always pick and choose which one of these items you can include into your own daily routine.



**Tools**

- Start the day: gratefulness
- Time box tasks
- Primary goal
- Daily targets
- Improvements
- Wins
- End the day: gratefulness

What they had is that you always start off the day listing off what you're grateful for. Now this is not something that they invented. In fact I've heard a number of popular entrepreneurs such as Aryana Huffington who does this exact thing every day when she wakes up. She lists off three things that she's grateful for before she has opened her phone to look at e-mails before she's done anything else in her description of the process was that it allowed her to start her day with the right mindset instead of diving right into any tasks that could potentially increase stress and anxiety.

It instead allowed her to start with the mindset of being happy and being able to be thankful for the best parts of her life. And then if you look down at the very end of the list you also do the same task. So at the very end of the day you list off three things that you are grateful for. Research has

So this is something where you can brag a little bit to yourself on the things that you were happy with. And so the way that the Journal actually looks and this is a verbatim screenshot or a picture of my journal when I was going through it.

On the left hand side there you can see that all of my tasks for the day were time boxed and it went from 6:00 a.m. all the way through 9 p.m. So right there I listed out every single task that I had to do and this includes your entire life. So if you are in school or if you are working in a job



you don't simply list the elements that are related to your school or work, instead you list everything out.

As you notice I included my workout, I included my general tasks, I included lunch, and all of those elements and as you can see every moment of the day is spoken for and that is something that is very important because one thing that I've experienced that if I have a portion of my day that I have not allocated for some task then when I get to that time then there can be some confusion or it can even lead to procrastination or me not spending my time in an effective manner.

Now one process that I found incredibly effective was the notes section on the right hand side there. As you can see that lined up with a number on the tasks side. They have a task on the left hand side and then they have some notes on the right hand side. I started to notice a trend where I would work through various projects and then when it came to the next day it would take me a little while to pick up where I left off. So say it was a code project I may have had a fantastic day the day before and building out a feature but I didn't know where to start on the next feature. And so what I use the notes section 4 is I simply give myself a few key words there.

As you can see at the very top I was building out the daily smarty web application and on this particular day on the 24 of 2017 the next feature that I knew I was going to have to build out was the Twitter authentication system for daily smarty. What that was enabling me to do was when I started to write out the tasks for my very next day. I instantly knew where I needed to start with. So when I woke up the next morning and I started working on that daily smarty application I didn't have to go through and look at the code that I pushed up the last day and look at where and then try to decide where I was going to start.

Instead I simply was able to look at those few little notes I wrote down and I instantly knew where to start on the new application and so that is something that's very helpful. And I've seen that help not just myself but other developers that I've worked with. Being able to help yourself out because as much as you may think at the time that you're going to understand exactly where you left off and you're going to be able to just start the next day running. If you don't give yourself that information and give yourself some hints on where to start off then you're going to end up wasting time simply going back and trying to figure out what you need to do for that next day. So make sure that what you're trying to achieve is specific in that you're also helping yourself out as you go through and complete the tasks.

Now on the bottom on the left hand side and the right hand side you can see that that is where you list off what you're grateful for both when you wake up and when you go to bed. Moving to the top right hand side. Every day has an overarching main goal for the day. So you try to pick out your one top item that simply needs to get done something that is very popular in the goal setting community is making sure that you properly prioritize your goals. So if you're working for a company and you have one feature that you need to build that day make sure that that is your top goal and also make

sure that you start that day by attacking the most important item. So make sure that you're prioritizing not just your goals but you're also prioritizing your time in a way that reflects that.

Now moving down a little bit you can see that the other item is the target list. So you have your goal you can think of that as your strategy, it's what you want to get done that day. Now your targets, this can be a tactical approach. So this can be a set of to do items that could list out the steps that are needed in order to achieve that goal. And the book gives a few little hints such as right there in the parentheses where it says what we'll make today a win for you?

That is very important. There are many times when I haven't been good with setting up a task management system where I didn't properly establish what a win for me that day meant and so I could have had an incredibly successful day by most standards. But because I didn't establish that in my mind and I didn't write down what a win looked like for that day when the end of the day came. I would be a little uncertain and therefore stressed and anxious on if I was good that day or not. And so that is important to list that out. So at the very end of the day you can look at it and see if it was a good day or is a day that you need to improve on.

Moving down the page you can see the next group of content list out the lessons that you learned that day. So say that you procrastinated on a task don't simply say oh I procrastinated on this task list out why that occurred. Imagine that you are coaching somebody else so imagine that you have a student and you are trying to help them be as efficient and productive as possible. And you saw that they procrastinated that day walk through the causes of the procrastination. Was it a topic that was intimidating for you? Was it something you needed more information on? Analyze the cause of it and then that can help lead you to improving on it.

So in lessons learned don't simply say oh I was bad at x, y, and z this day instead say I struggled with these tasks for this reason and this is how I think I could improve on that tomorrow.

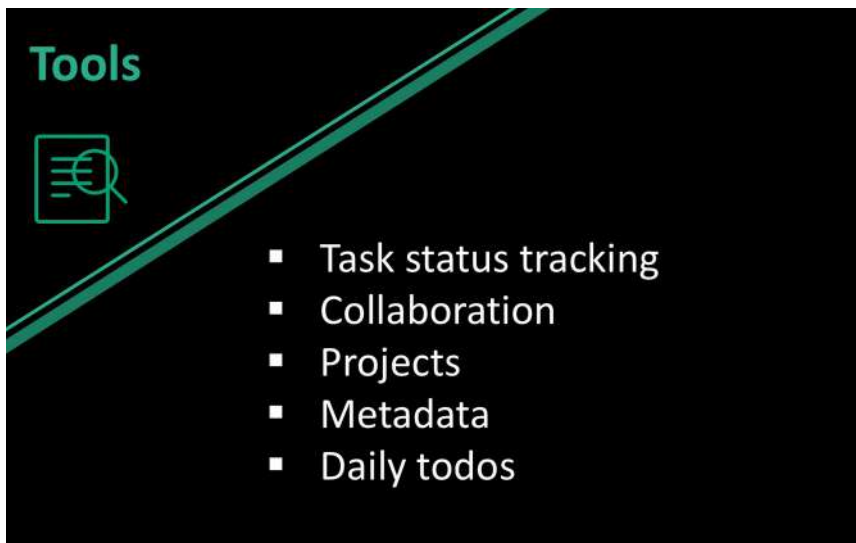
Moving down to the very last item. This is the fun one, this is the set of wins for the day. Make sure that you always list out the best parts of your day and how you were successful. Because that's going to help give you confidence and it also gives you a nice log where you can look back and see all of the wins that you had. So any time that you built out a feature successfully or you understood a topic that you're studying this is where you should list that in your daily journal. And that's my approach for using daily tasks journaling.

If you are someone who really enjoys working with pen and paper this can be a great option for you. One item that you may note is that this is not the most collaborative approach. So if you're working with a team on projects this by itself is not going to be the best option for you. This is something that is more focused on helping you as an individual.

Now we're going to move on to a task manager system that you can use as an individual or in a group in collaboratives setting. And that is a Kanban approach and a Kanban is a project



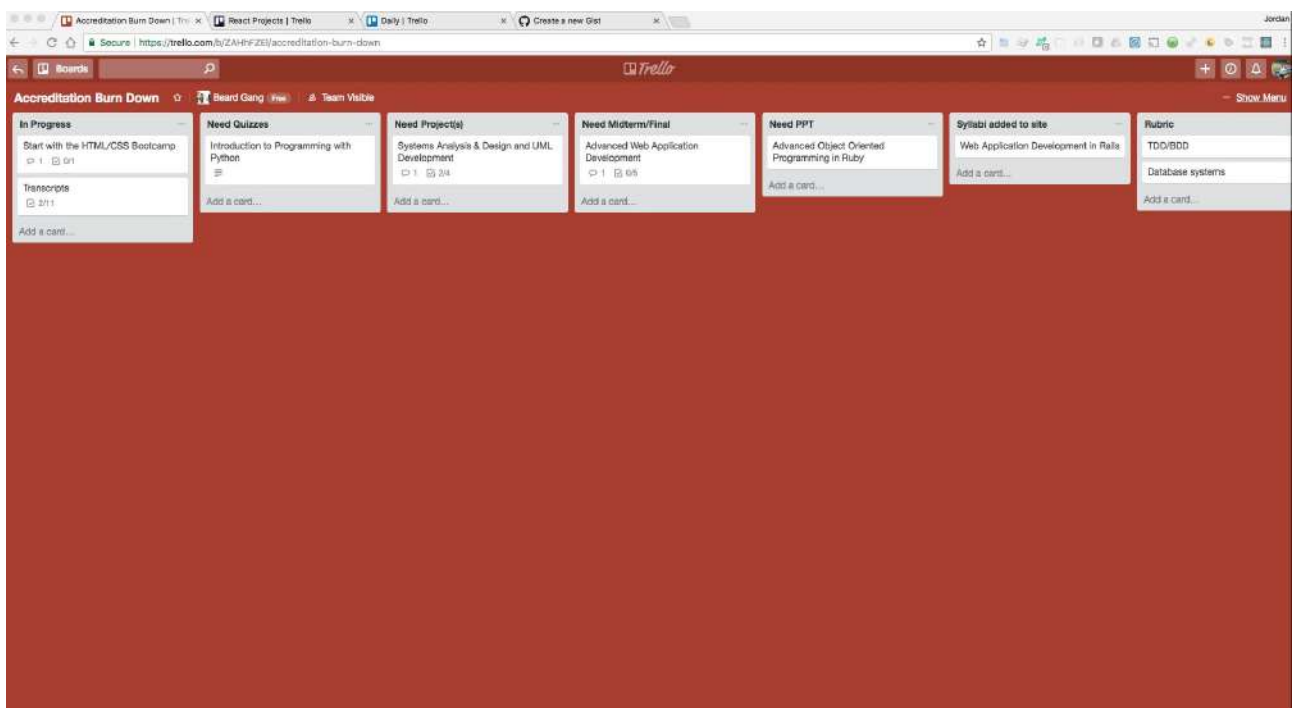
management methodology that's been used for several decades and has become incredibly popular in a number of different industries and so we're going to walk through a few variations of that.



We're going to see how we can use it for daily status tracking, for collaboration, how we can use it for projects both for yourself and for groups, and also how you can leverage this kind of tool for adding metadata and other kinds of associated content, and helpful items like that into your daily workflow.

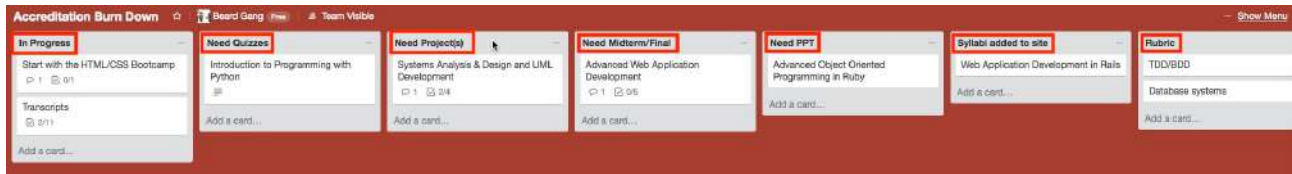
For this walkthrough we're going to jump into the browser and we're going to work with the [Trello](https://trello.com/) software.

And so Trello is a free tool to use, they have paid options. If you're working on a small enough team then it is free and especially if you're just using it for your own purposes then it's completely free and you can see a very different approach here compared with the daily task journalling.



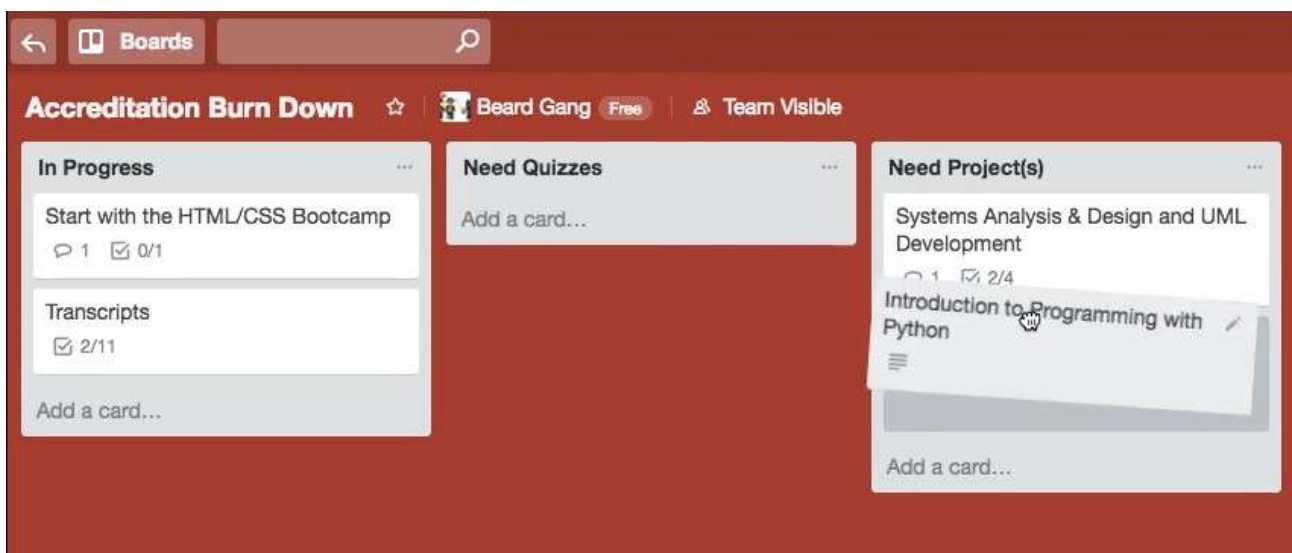
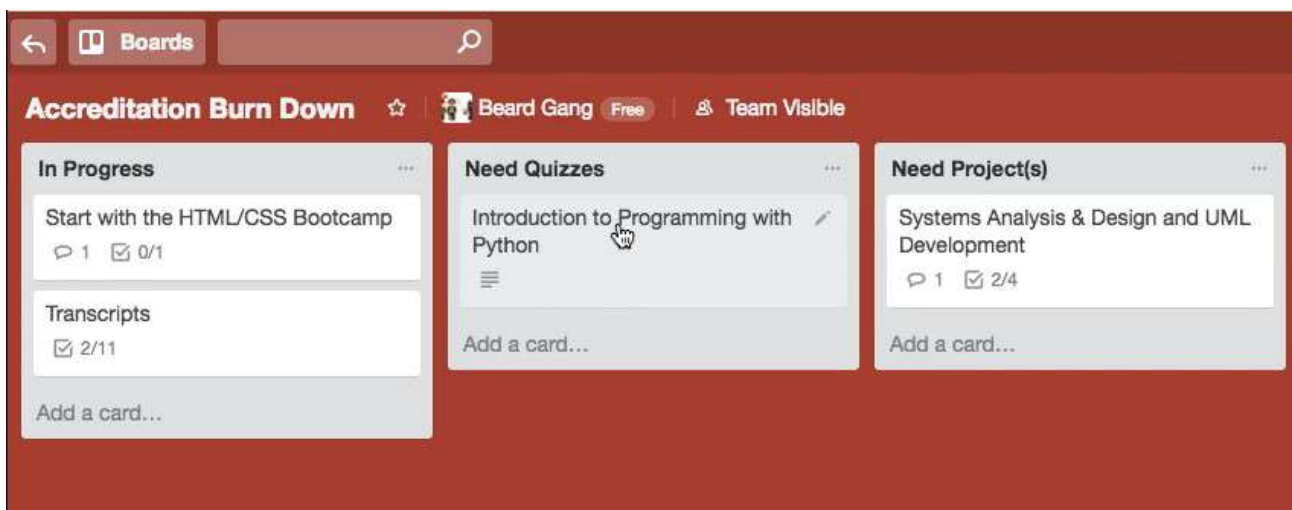
And in addition to that we're going to go through three examples. I have a number of Kanban boards that I work with individuals on and also that I work with just for my own personal use. And so I'm going to show you the approach because it is different for each one of these boards.

The first one I'm going to show you is the traditional Kanban approach and so what Kanban is, is it gives you the ability to track states of a specific project. So on the top of each one of these cards here is something that's called in progress so this is a state. Then Need Quizzes, Need Projects, and then each one of them have a state that they are in.



And so the way I've set this up is this is sequential and if you're wondering what this is for, this is a process that I use with the curriculum team for Bottega. This is how we track to see if a entire course is done and also how we can track and then have reporting on the status of each one of our projects.

Take for example our introduction to programming with python. Right here we can see that it needs quizzes and so that can tell someone on the curriculum team who's been tasked with writing the quizzes that if they're looking for something to do then right here they can come to the course and then start and add that. After they've added all the quizzes they can click on this and then drag this task over to the next card.



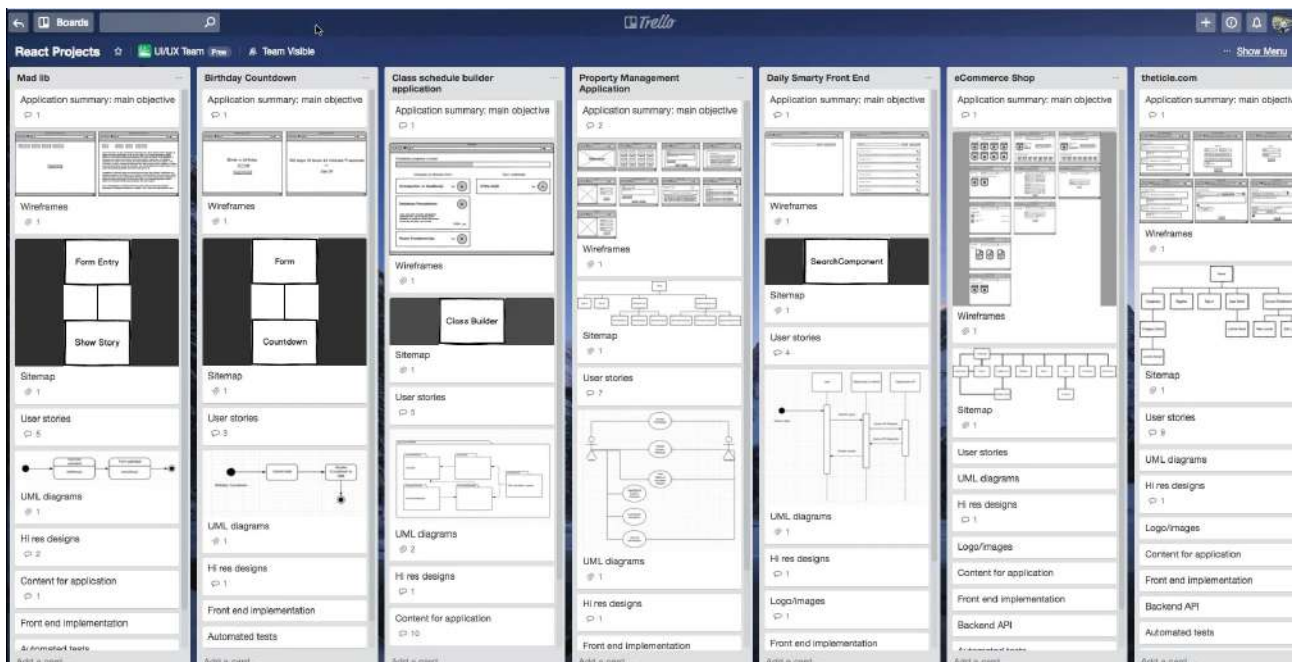


So in this case every one of our courses has projects and so with that in mind we know that when the quizzes have been added it's on to our task where we need to start adding projects.

Obviously, this is very specific to my and my team's use case. But if you imagine any kind of project that you are working on it probably has various states of being in progress too. All the way down to having a quality assurance team looking at it and you'll have a custom set of states and processes and you can simply take each one of these tasks on a journey.

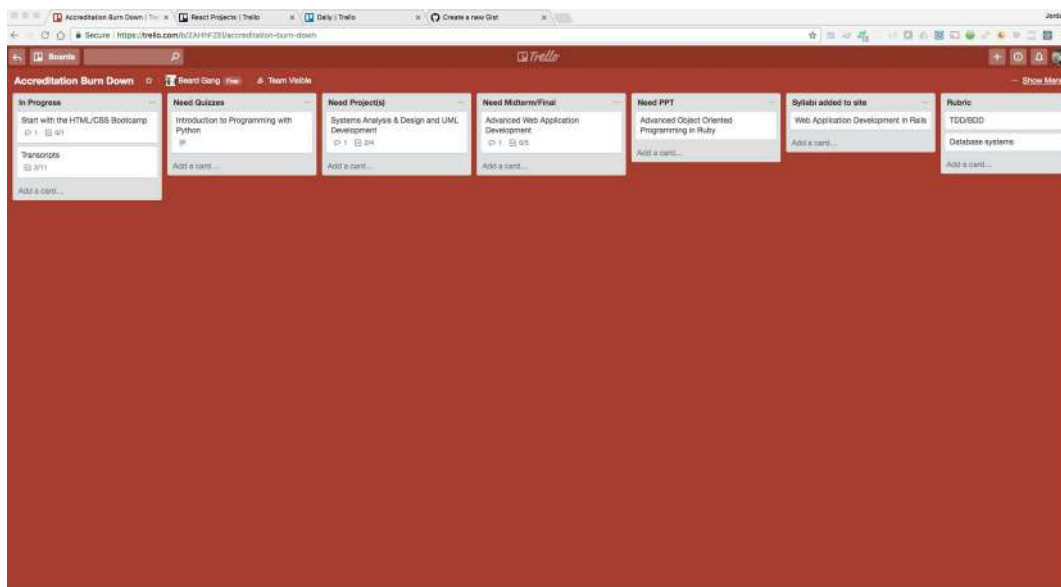
Now, this is a full project management solution. So this goes even beyond a daily task management. This is going into a tool that you could use if you're working with a large team of people where everyone can get reporting they can track the status of each item and then they can collaborate and be able to assign tasks to other people.

Now I'm going to go to a another board that functions very differently.

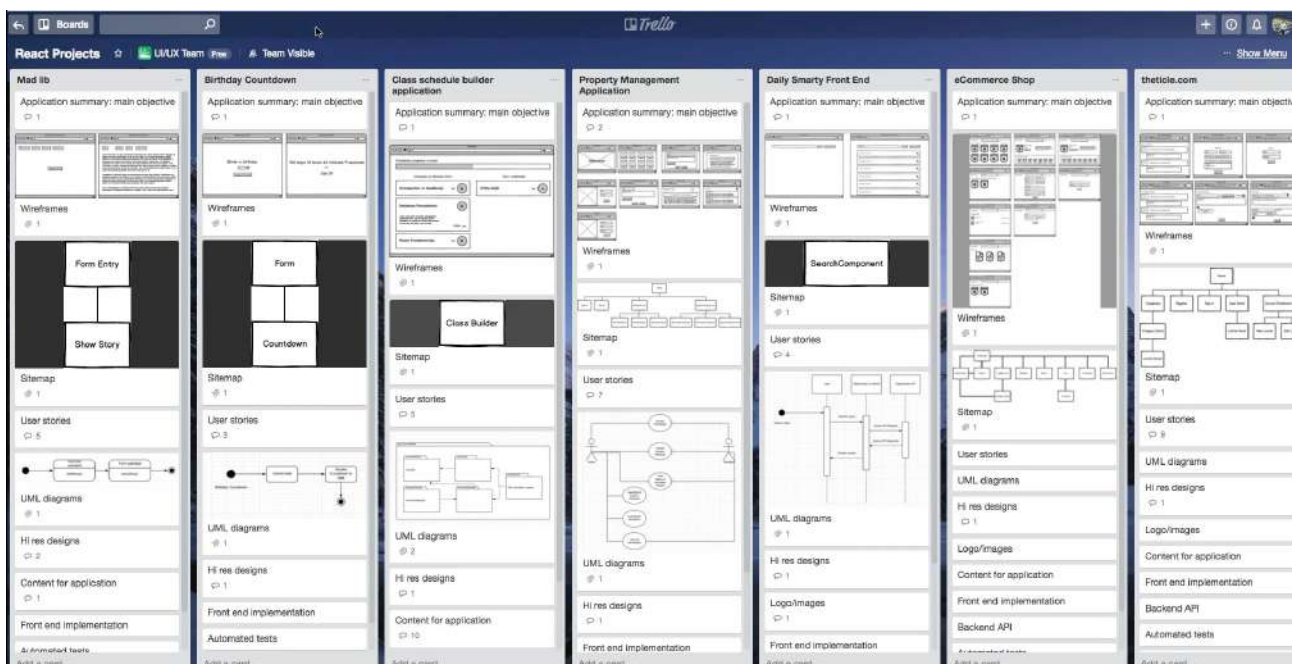


So this is a specific course board and it has a number of different collaborators. I have worked on each one of these cards. And then we also have designers and engineers that have also contributed in one spot or another. So we are not treating this like a traditional Kanban board.

If you look back at the course one that we had Here... :



... we set up essentially we sent a journey. So we set up each one of these cards to be a state. And then you would take a task and then drag it to the very next state.

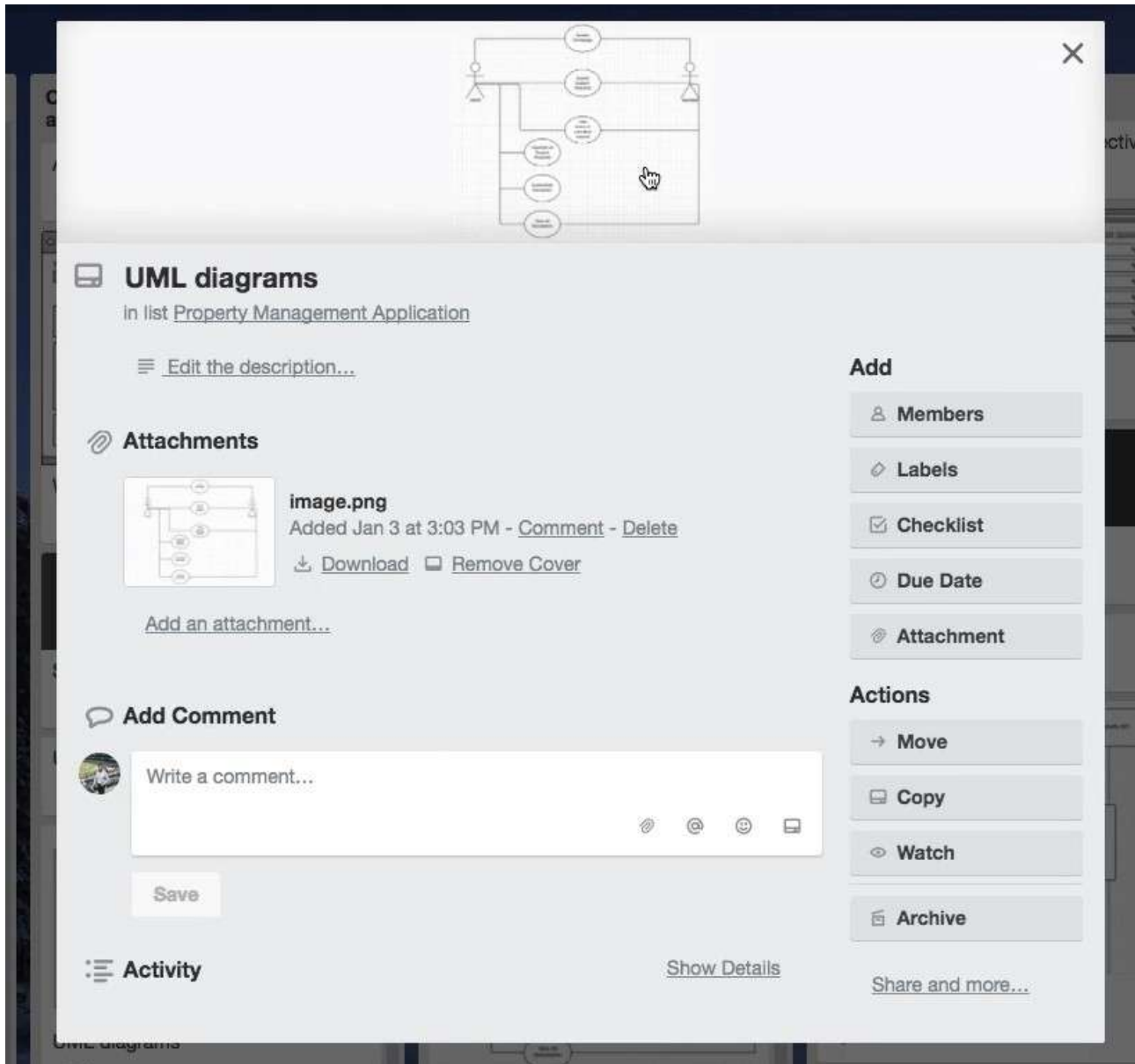


So, what I've done is I've treated each card as its own project and so the goal of this is not to take one of the tasks and then move it over to another board because this is a one project and this is a completely different project.

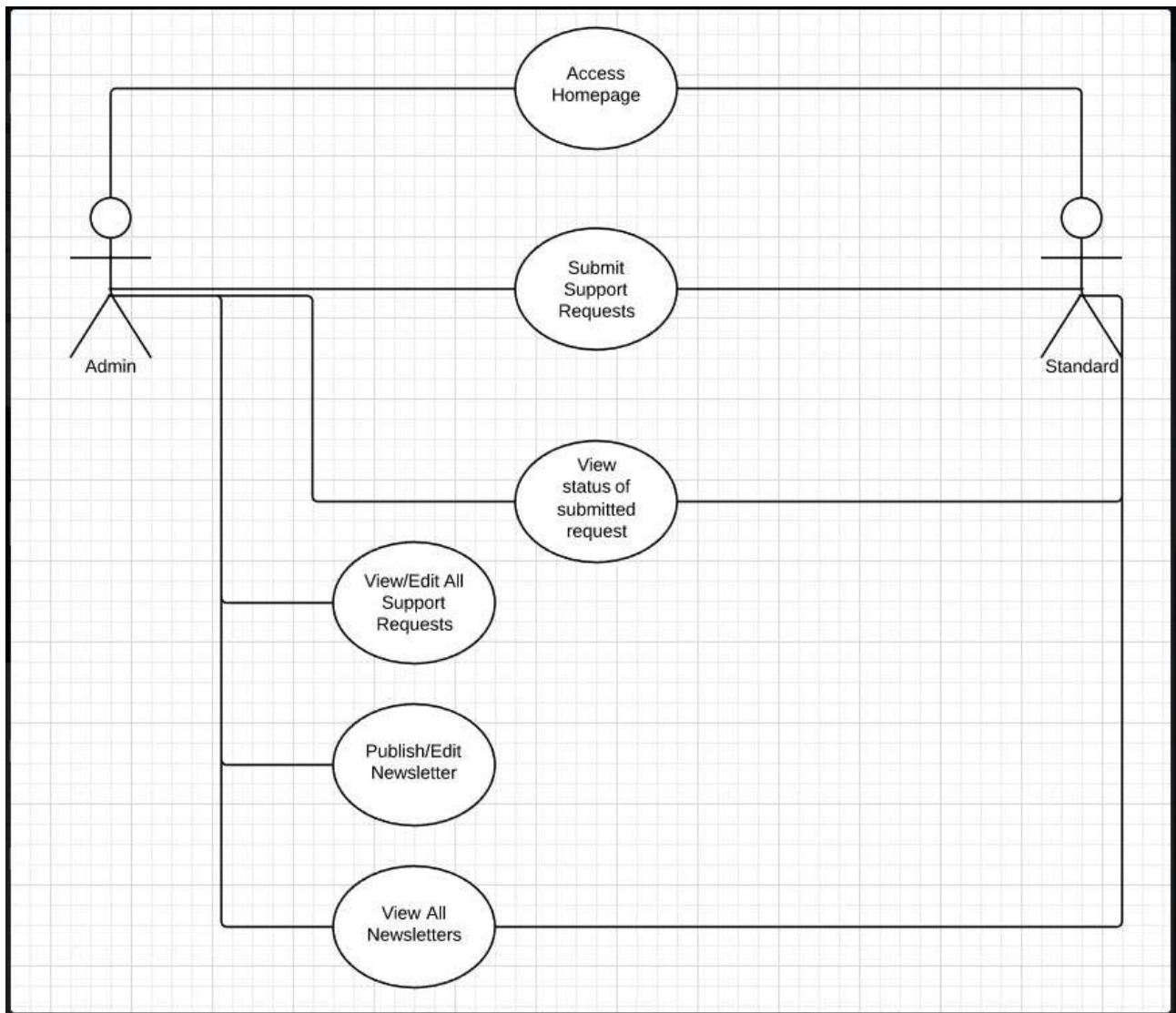
And so here I'm treating it very different than the first example. But what I wanted to show you here was the ability to add quite a bit of metadata. So this is incredibly helpful for having a dashboard where you can look at a project and get all of the information you need all in one spot.

So this is fantastic for collaboration and also tracking the progress of each one of these projects. So for example I have to add to this property management application a uml diagram.

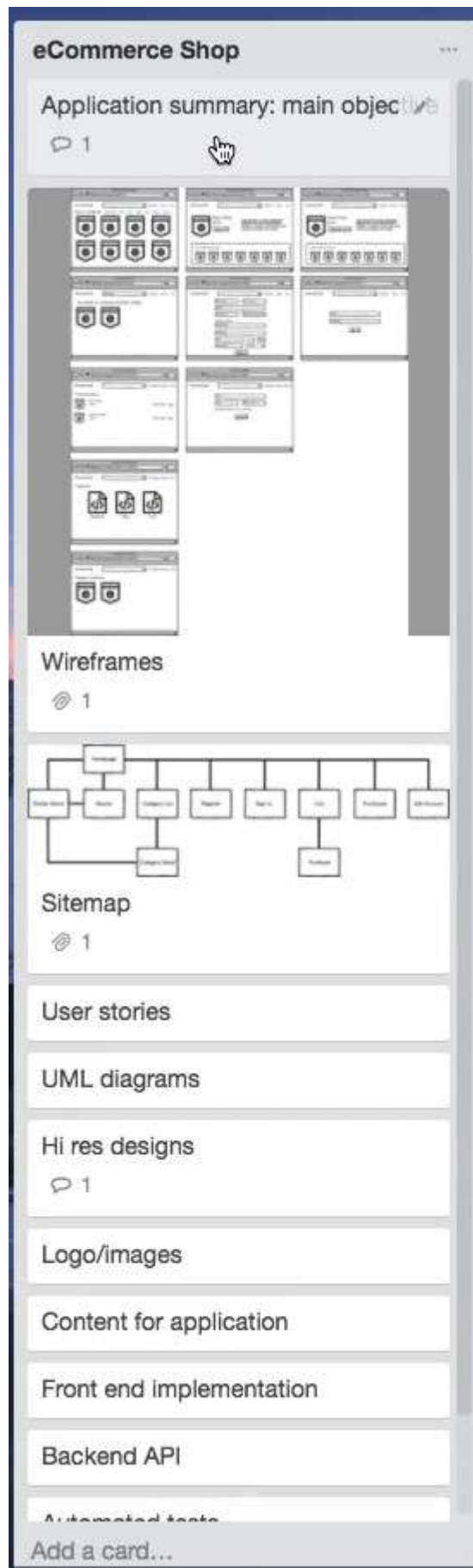
And so before there was no little diagram here but if you click on this it will actually pop up the diagram that I added.



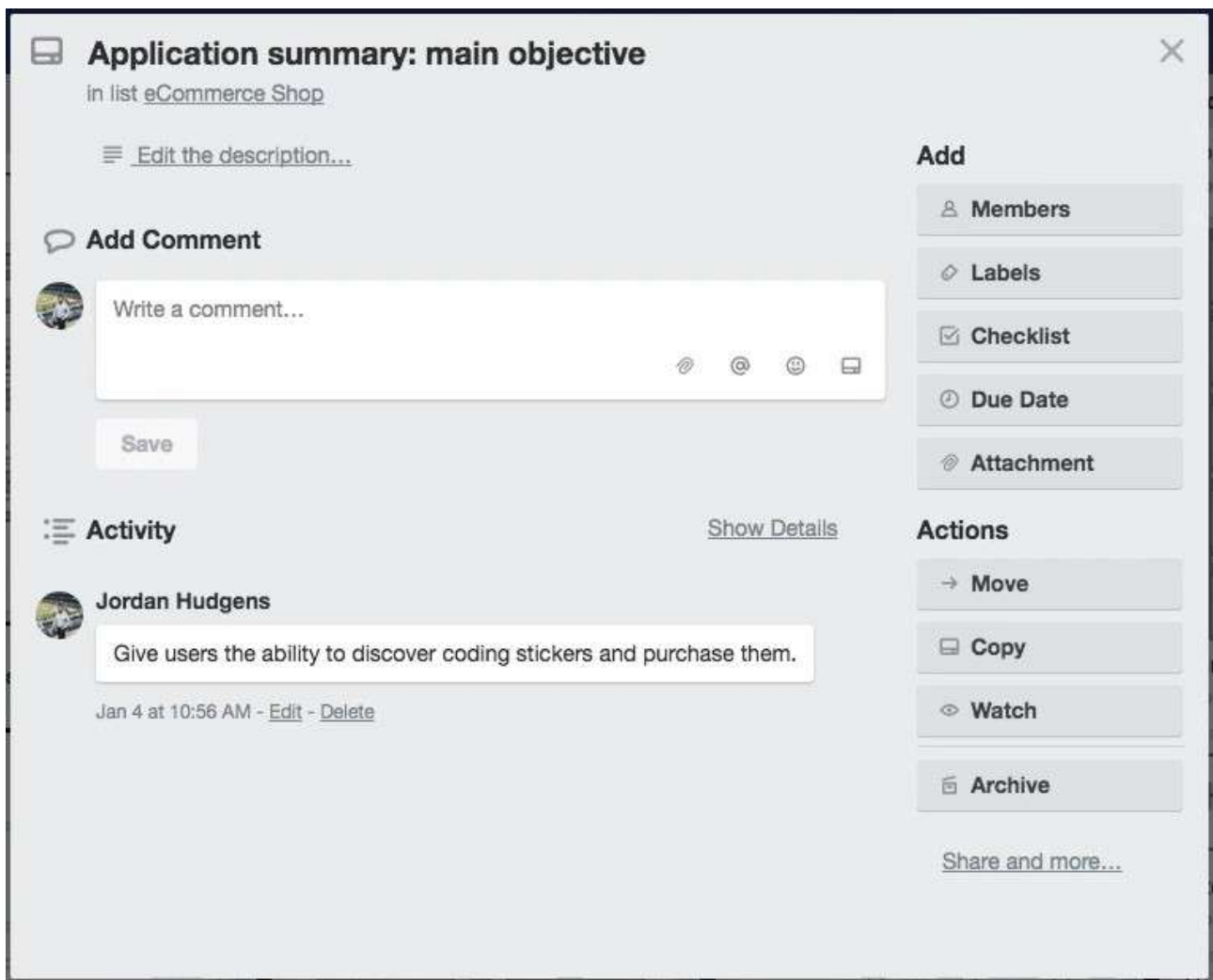
So in the beginning this was completely blank it didn't have the image. But now after I finish that uml diagram I was able to upload it and now everyone on the entire team can come to this dashboard and they can all see what the diagram looks like and then they can work on it from there.



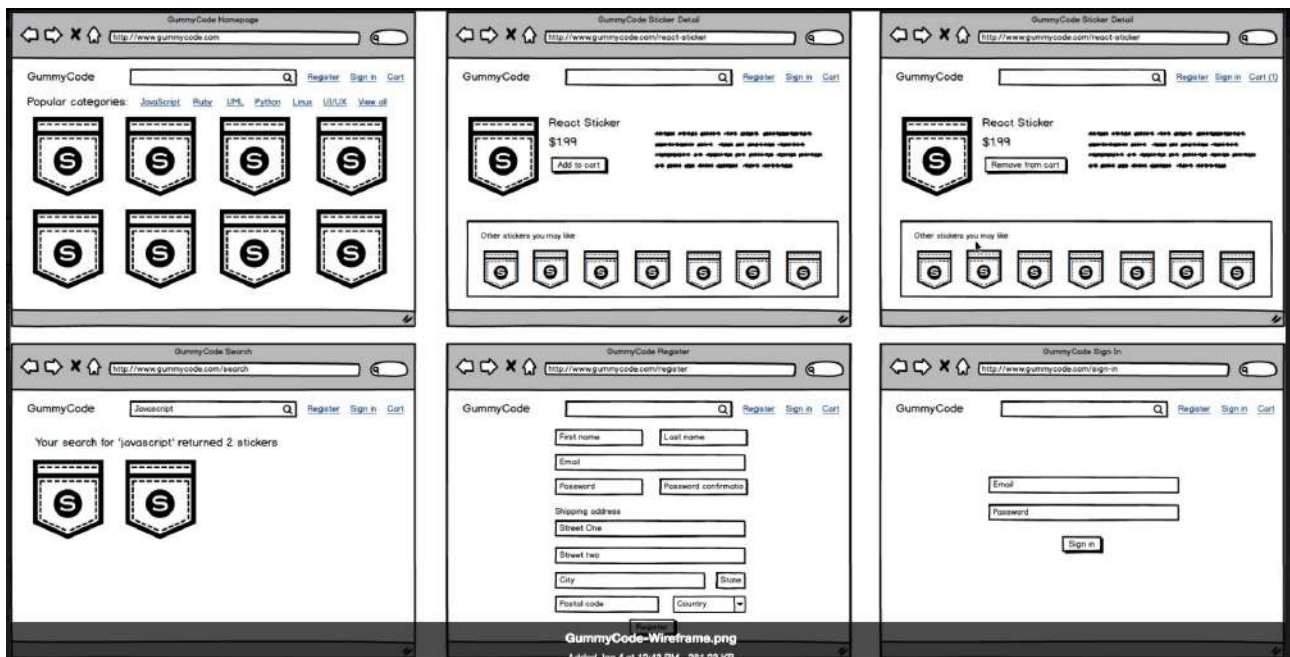
So for example if I come down to one of these other projects like this e-commerce shop right here



I worked on this with a number of designers and engineers where it started with listing out the goal for the system, which is just a short sentence right here.



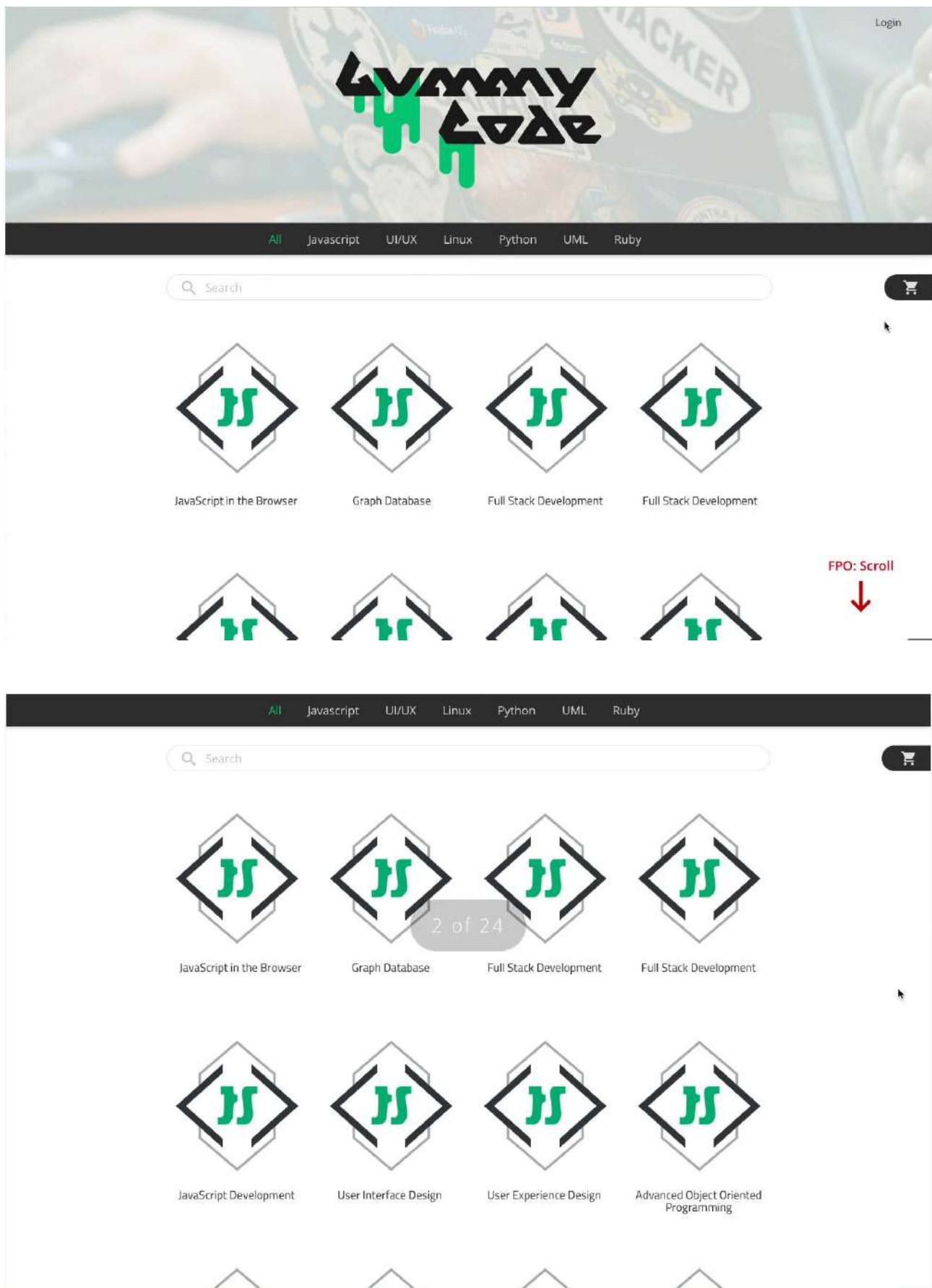
Then from there I created wire frames and then site maps that showed the high level behavior of the system



and as I was doing this the other members of the team were able to see this and we didn't even have to talk on the phone or have meetings they simply were able to see everything that I added. And then they were able to go and perform their own tasks.



So for example the designer was able to go and create a full design for the entire system.





## Cart (4)



Web Application Development  
in Rails

1	+
	-

[Remove](#)

\$ 1.99



Redis Database

1	+
	-

[Remove](#)

\$ 1.99



Scss

1	+
	-

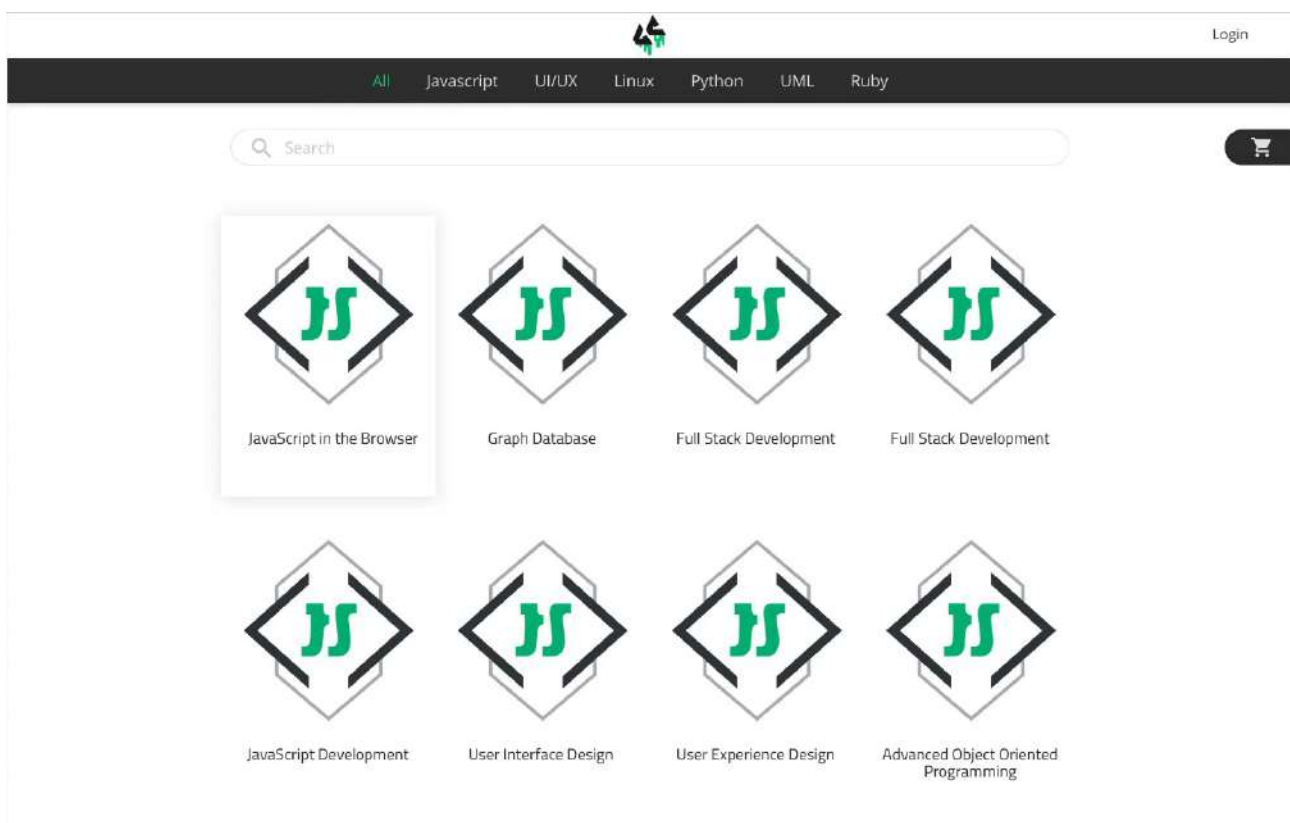
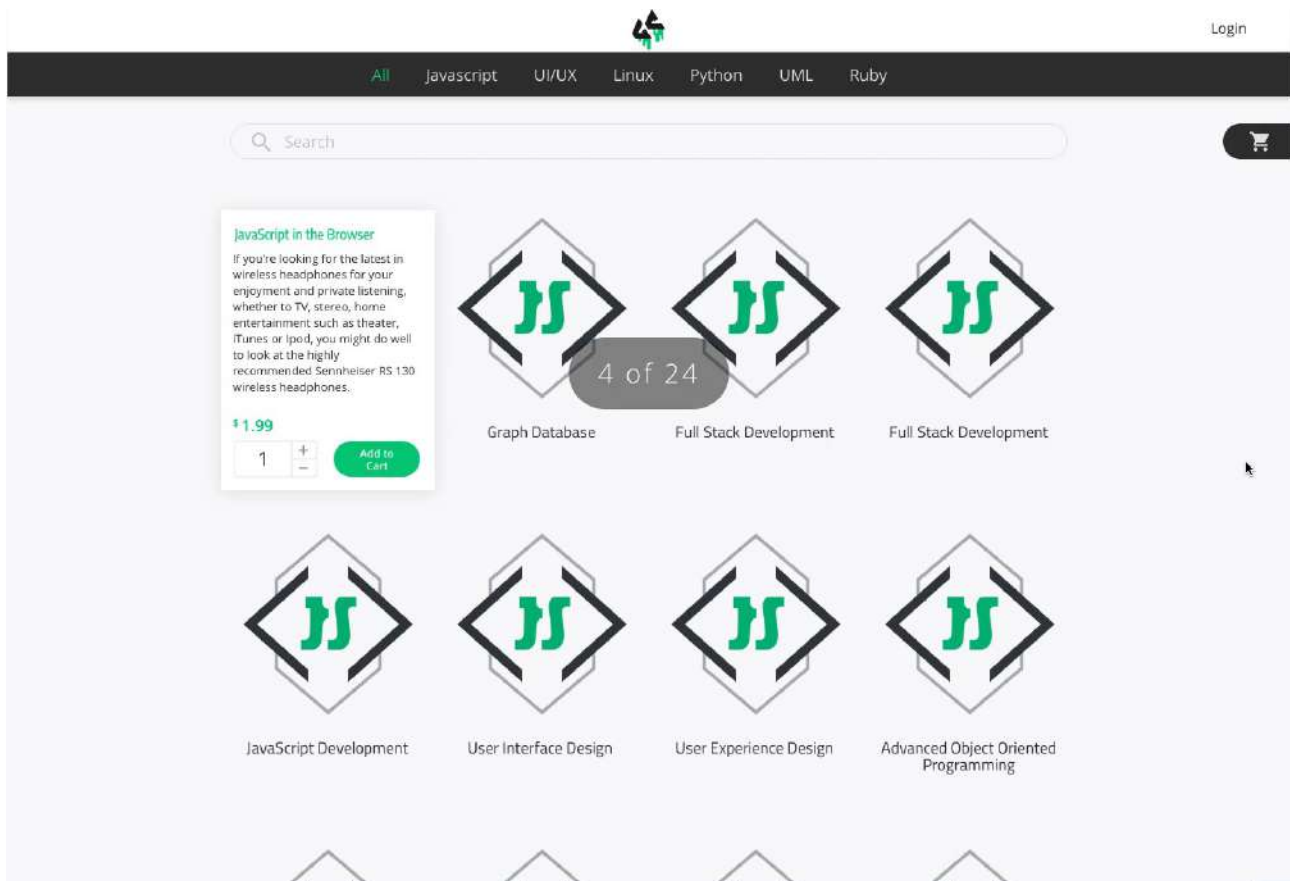
[Remove](#)

\$ 1.99

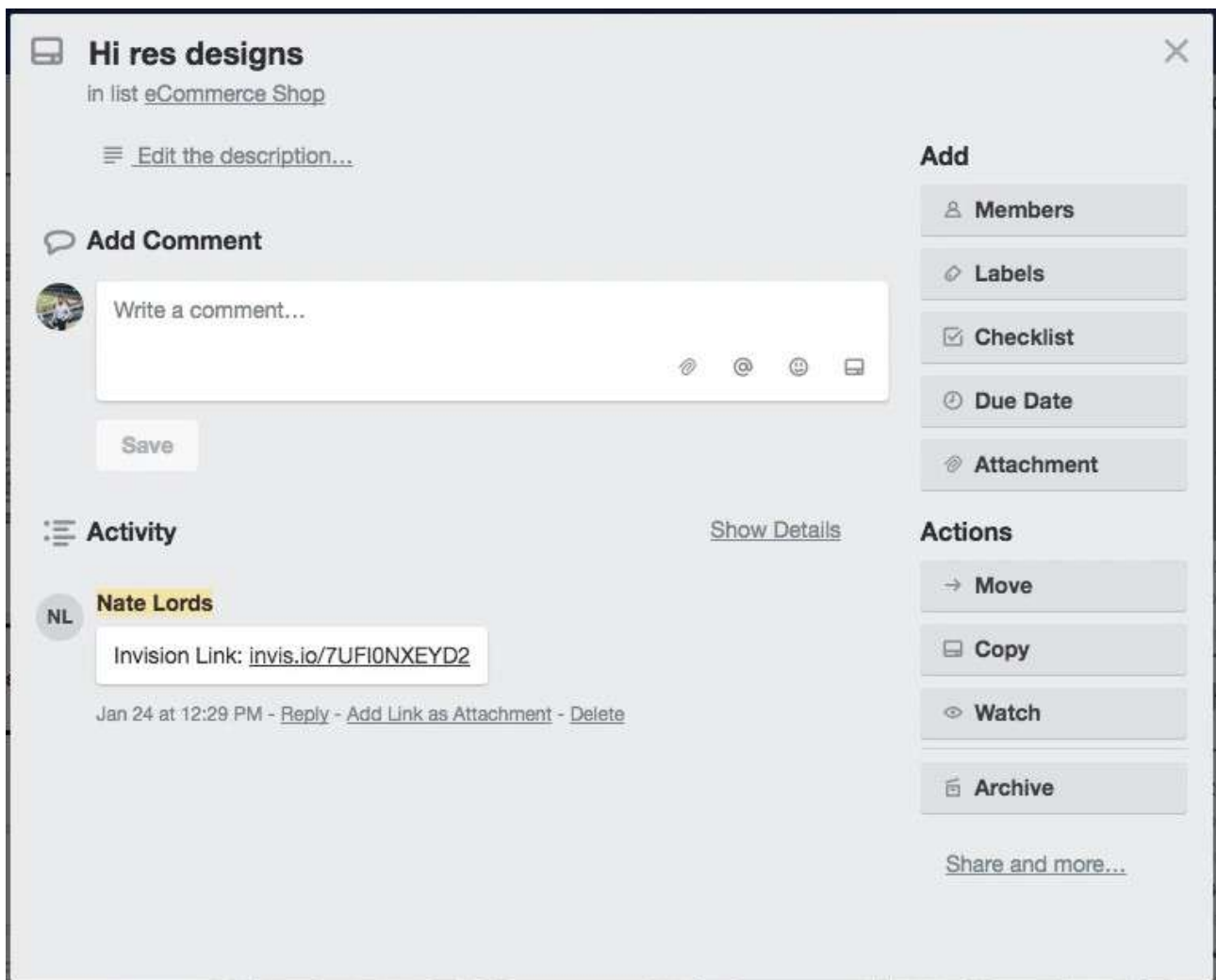
Checkout

Subtotal \$ 7.96



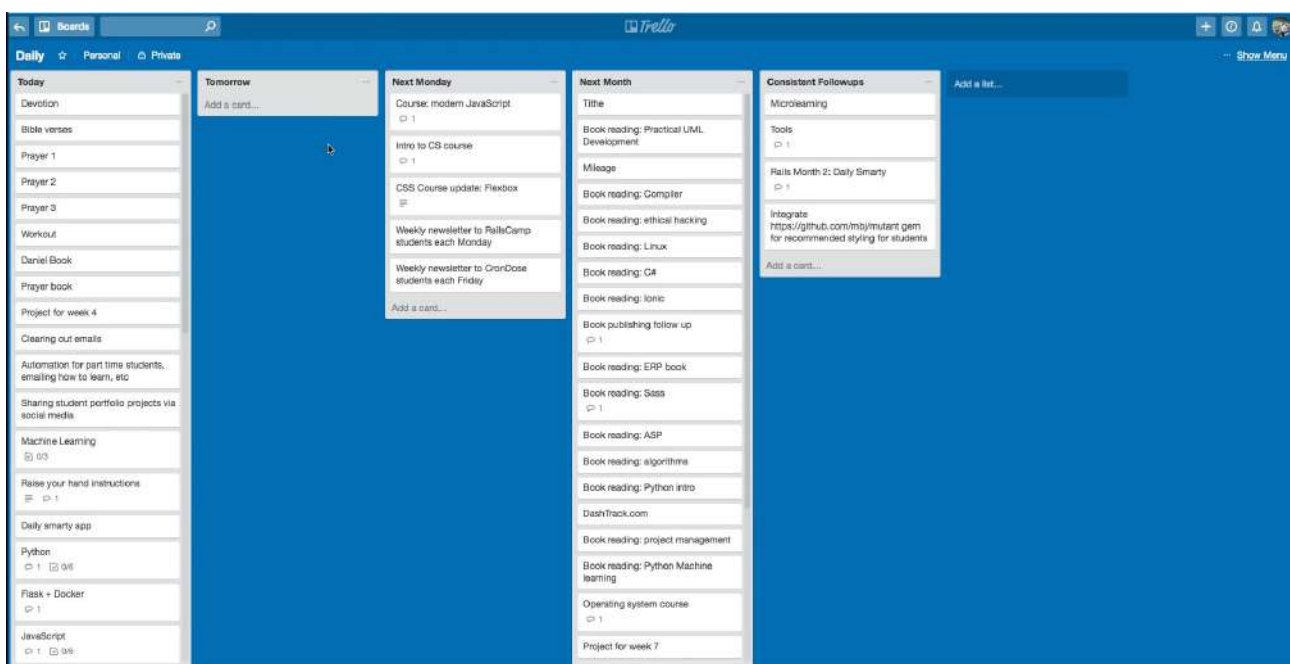


And all he had to do was track each one of the items that I was adding and then that gave him an idea on exactly how he needed to structure his own tasks. And then when he was done as you can see right here this was done by our designer Nate Lords. And he simply pasted in the link so that I and the development team were able to come to it and see the completed design.



And so we did that for each one of these projects and it proved to be incredibly efficient for us.

The last Trello example I'm going to show you is my daily Trello.

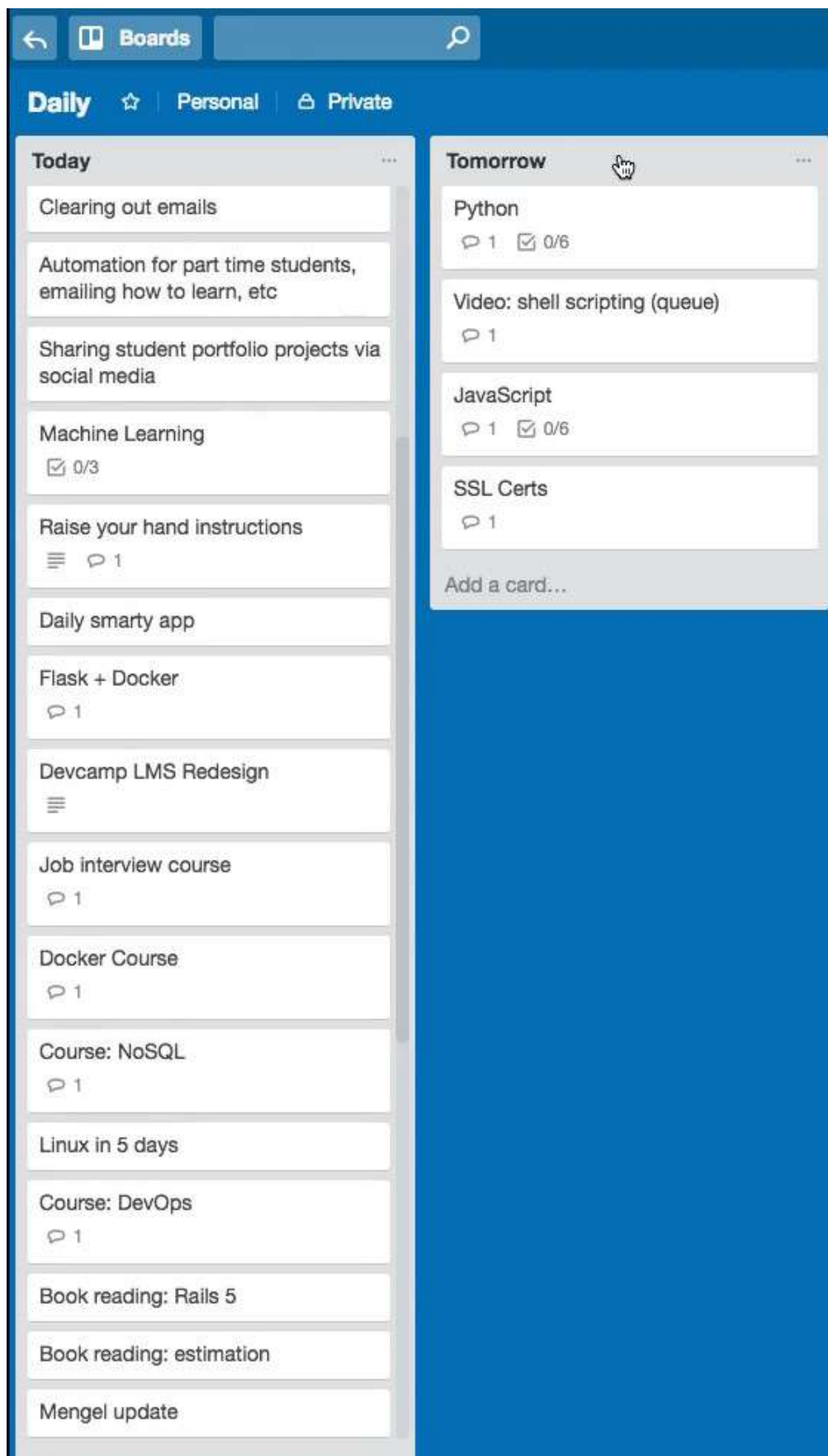


So right here I've set up a Kanban board and this is just for myself to work on as you can see you can also have them where they're private versus having team collaboration. And here I've set up the

And just like we saw with the last task board you able to add any metadata that you want. So for this task you can click on it and this has all of the content that I need to add to a page of devcamp

which is just a really nice easy way of keeping all of the details for something I need to work on in one location.

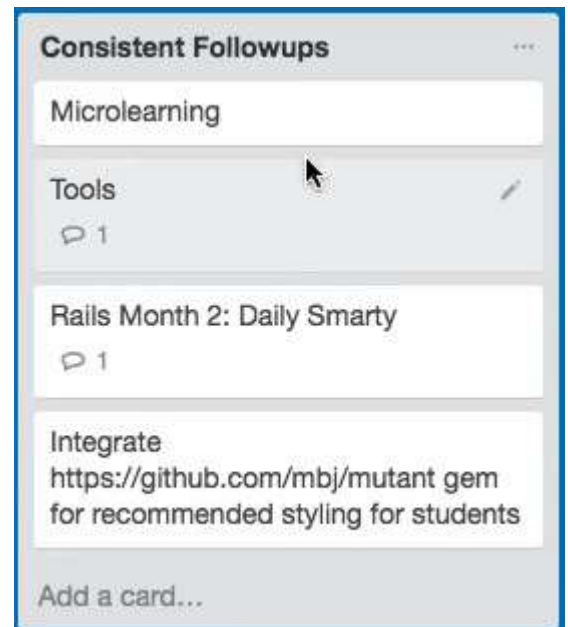
So this is an effective way of doing it. And then what I do is as soon as I'm done with one of these tasks those am done writing the python curriculum and everything I wanted to do on this task for the day I simply come to this tomorrow task and I simply drag it over to this card. And by the end of the day I should have each one of these cards moved over and this entire channel of Today here should be empty. And then what my process is is I simply change the name.



So this would be today and then I change this channel to be tomorrow and then the next day I simply repeat it and I dragged each one of these cards over to the other channel. I also as you can see have other ones set up because there are tasks that I perform on a daily basis.

But then there are also tasks that I perform on a weekly basis. So I had this channel set up for these all all the items that I want to do starting the following week and then I also have tasks that should only be performed monthly and so they are right here.

And then I also have a final item which is this set of items that I just have here for consistent follow ups.



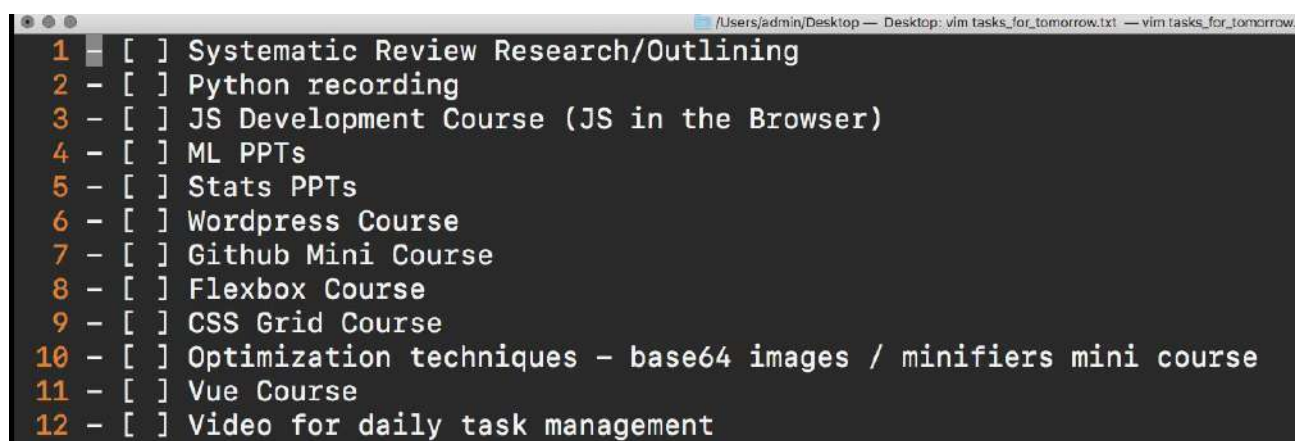
It's just here essentially so that I will not forget about them and so I'll continually look over here to make sure that I am performing each one of these tasks.

So we're almost done we have completed two of the three walkthrough's and so switch back here and now we're going to look at the third option which is the to do list.

Now this one is probably one of the most basic options that are out there but because of that it has the lowest learning curve. So if you think back to the daily tracking journal when you did that you needed to learn a number of options you needed to learn a number of different things you needed to learn the process you needed to build out your own system. And it takes not a lot of time but it does take some time to learn.

And the next option in working with Kanban boards and working with tools like Trello that takes quite a bit longer especially if you've never used a tool like that before. But with this type of system there is pretty much no learning curve is a basic to do list and that may seem like it's not the most effective approach but as I'm going to show you it actually can be quite helpful when done properly and it is very flexible.

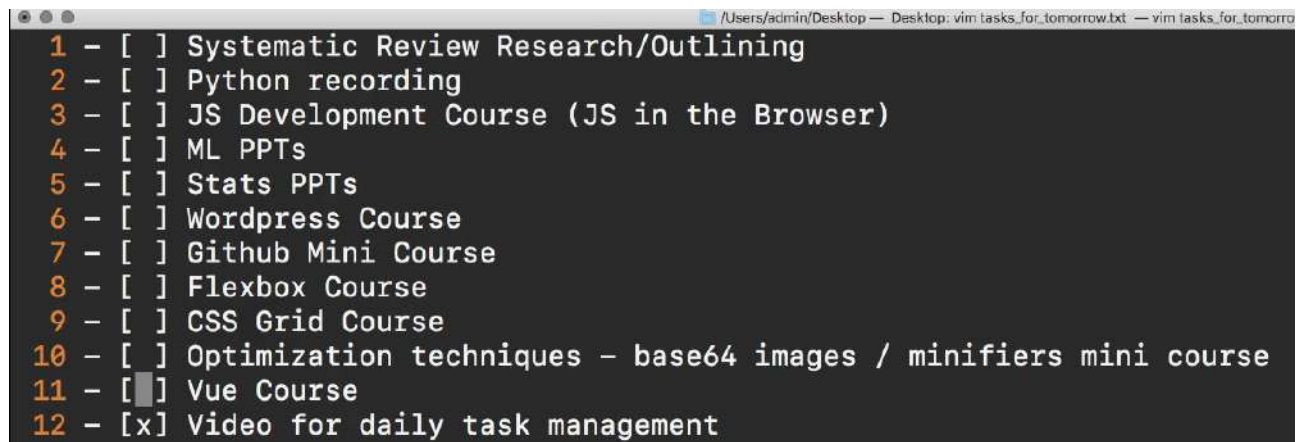
And in order to walk through it and open up a text editor. And as you can see right here I have a list of 12 tasks





and if you're curious on which of these three systems that we've walked through which one I'm personally using I'm actually using all three in some form or another right now because I've found that the combination of those three is a way that allows me to be as efficient as possible.

I'm continually changing and evolving myself so that is going to adapt. Now if you're curious if I'm being truthful you can come down even to number 12 here and you can see my task is video for daily task management so I'm actually doing this as we speak. So it's going to be really nice. At the end of this I'm going to be able to put a little x here and that is going to be crossed off.



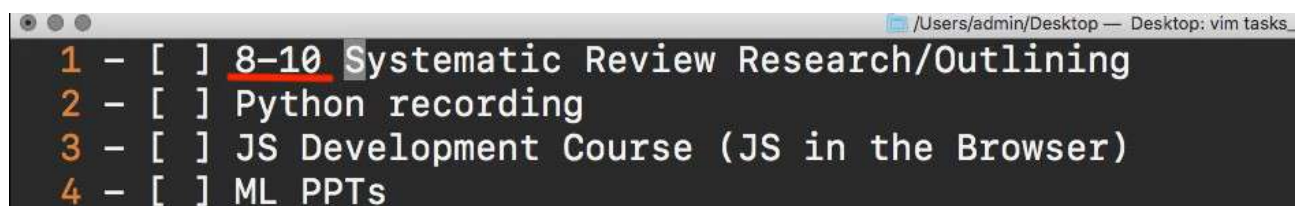
```
/Users/admin/Desktop — Desktop: vim tasks_for_tomorrow.txt — vim tasks_for_tomorrow.txt
1 - [ ] Systematic Review Research/Outlining
2 - [ ] Python recording
3 - [ ] JS Development Course (JS in the Browser)
4 - [ ] ML PPTs
5 - [ ] Stats PPTs
6 - [ ] Wordpress Course
7 - [ ] Github Mini Course
8 - [ ] Flexbox Course
9 - [ ] CSS Grid Course
10 - [ ] Optimization techniques - base64 images / minifiers mini course
11 - [ ] Vue Course
12 - [x] Video for daily task management
```

Now if these symbols look weird to you it's because I'm using a language here called markdown that is completely optional. If you are not a developer and you don't want to use tools like markdown you can have a list that looks exactly like this

- 1.
- 2.
- 3.

and that is perfectly fine.

Now you also can time box these and that's is something that I do quite often so I might do something like say from 8 a.m. to 10:00 a.m. I'm going to perform this task.



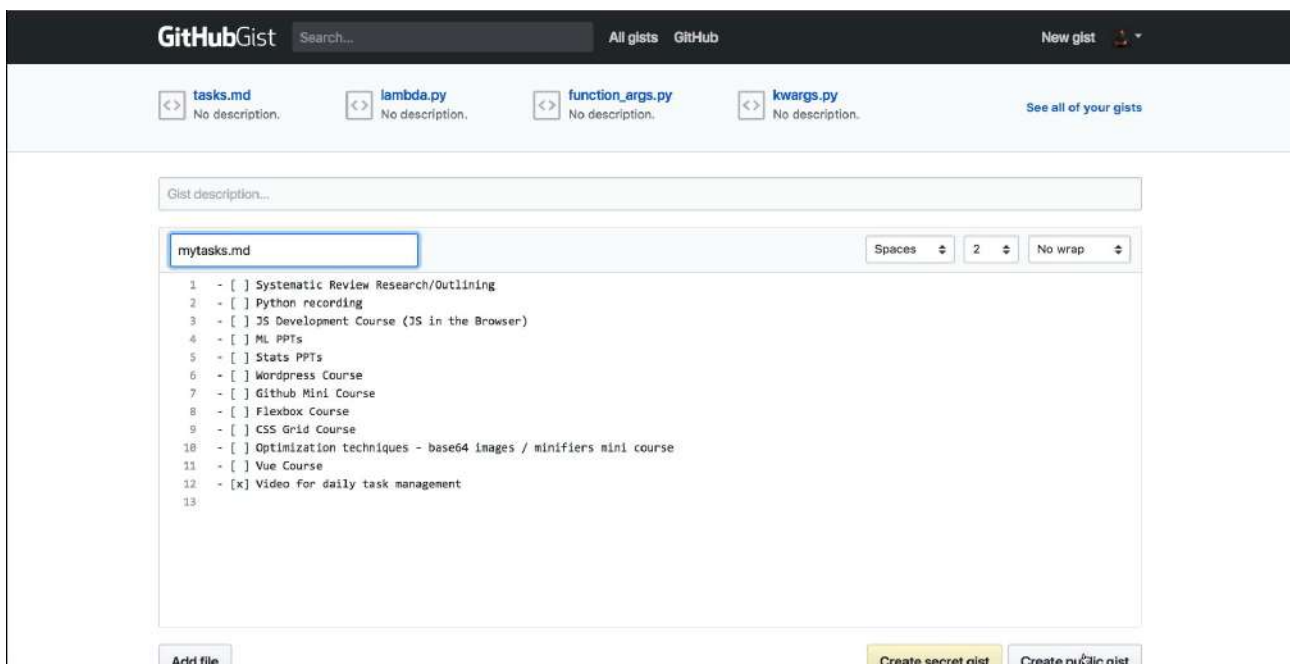
```
/Users/admin/Desktop — Desktop: vim tasks_for_tomorrow.txt
1 - [ ] 8-10 Systematic Review Research/Outlining
2 - [ ] Python recording
3 - [ ] JS Development Course (JS in the Browser)
4 - [ ] ML PPTs
```

So that is a way where I can be a little bit more structured with it. I don't have them right here

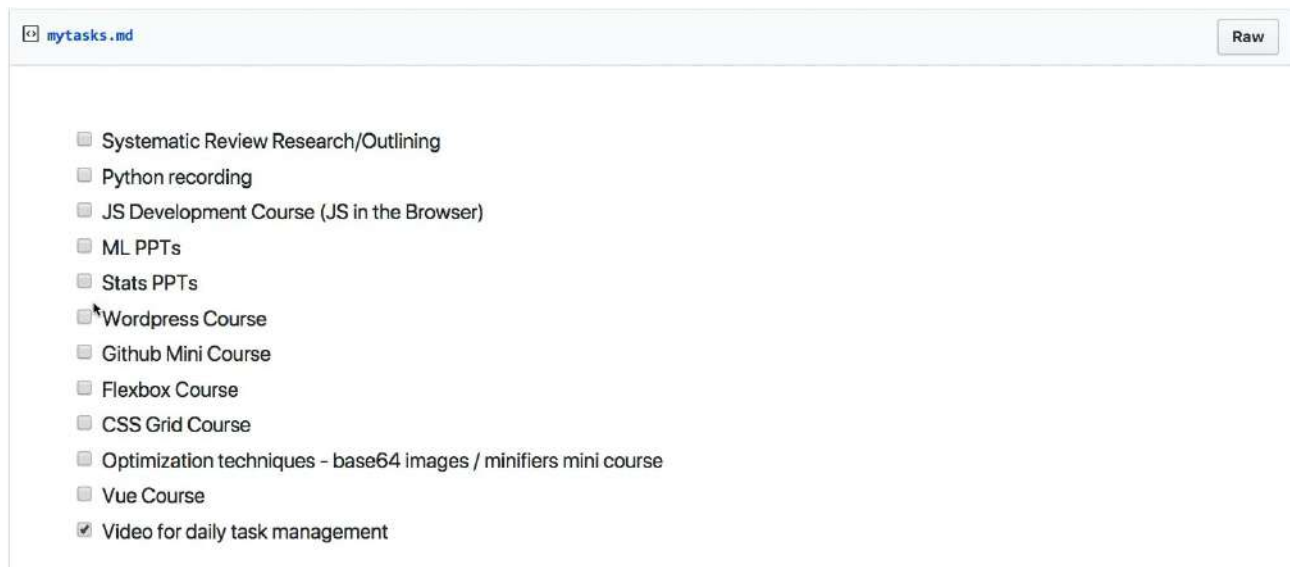
```
1 [ ] Systematic Review Research/Outlining
2 - [ ] Python recording
3 - [ ] JS Development Course (JS in the Browser)
4 - [ ] ML PPTs
5 - [ ] Stats PPTs
6 - [ ] Wordpress Course
7 - [ ] Github Mini Course
8 - [ ] Flexbox Course
9 - [ ] CSS Grid Course
10 - [ ] Optimization techniques - base64 images / minifiers mini course
11 - [ ] Vue Course
12 - [ ] Video for daily task management
```

because one I already have that my journal and so I'm not a fan of wasting time. So there's no need for me to put the Times right here. I'm simply keeping this list so that I don't have to always go back directly to the Journal and I can simply mark items off right here.

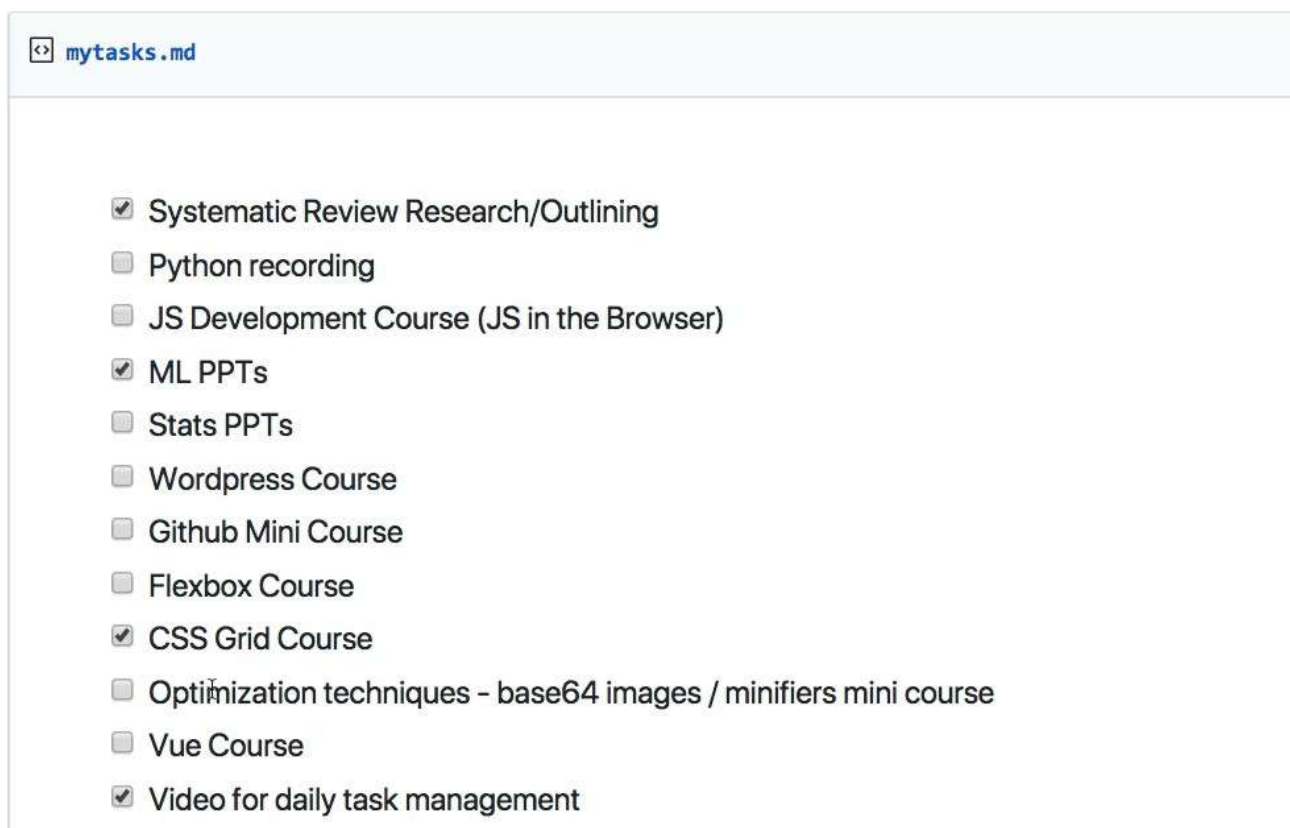
Now the other reason why I like using markdown is because of something I'm going to show you it's a way that you can leverage a tool like a to do list and actually make it collaborative. So if I save this I can copy this and to put it on the web so that markdown language I'm going to show you the reason why I'm using it here in a second if I paste this and I can just say mytask and I'm using GitHub's gist engine right here so I can say mytask.md



You notice here this is just what I had in the text editor and if I create public gist what that's going to do is see how it actually converted this into a To Do List automatically?



so I can share this link with anyone else on the team I'm working with and they can see exactly what I am working on and also what the status is so I can come here and say okay this is done for the day and so's this and so is this item and then they can track that.



And so if you don't want to use a tool like Trello you simply want to use something that is dead simple to implement then this is a another option.

So in review we've walked through three different systems that you can utilize. Daily tasks journalling, we walk through Kanban boards, and just walked through the dead simple to do list kind of system.

My recommendation is for you is to try out all of them and see which one allows you to be the most efficient to decrease stress and anxiety and help you organize your goals and achieve them on a daily basis.



## 3.25 Speed Reading for Developers

As a developer, the sheer amount of potential content that is available for you to read whether it's on the web or in books can be very intimidating. And so when it comes to being able to go through large amounts of content I've found that utilizing the spritz technique is incredibly helpful.

What spritz is, is it's a speed reading technique that allows you to quickly go through large volumes of content and it leverages focus. And so what I mean by that is that it actually compiles an entire book or blog post and then it shows all of the words in a sequential form in the same exact spot on the screen. And so in this guide, we're going to walk through exactly why I use this as a tool to see if it's something that you might find helpful.

I started doing this primarily when I was getting into my Ph.D. research because the sheer amount of content I had to go through was very large and I will show you right here my reading list simply to research a single paper was sixty-three publications.

★ ●	Authors	Title	Year	Published In	Added
★	Kouet, J. F.	Managing cognitive load during document-based learning	2009	Learning and Instruction	Feb 8
★	Baker, Ryan S.J.D.; Corbett, Albert T.; Alevon, Vincent	More accurate student modeling through contextual estimation of slip and guess probabilities in bayesian knowledge tracing	2008	Lecture Notes in Computer Scien...	Feb 8
★	Pang, Bo; Lee, Lillian	Opinion Mining and Sentiment Analysis	2006	Foundations and Trends® in Infor...	Feb 5
★	Chen, Chih-Ming; Liu, Chao-Yu; Chang, Mei-Hui	Personalized curriculum sequencing utilizing modified item response theory for web-based instruction	2006	Expert Systems with Applications	Feb 7
★	Malyusz, Levente; Pem, Attila	Predicting Future Performance by Learning Curves	2014	Procedia - Social and Behavioral S...	Feb 8
★	Yannakakis, Georgios N.	Preference learning for affective modeling	2009	Proceedings - 2009 3rd Interna...	Feb 8
★	Gagné, Robert M.; Briggs, Leslie J.; Wager, Walter W.	Principles of instructional design	2005	Performance Improvement	Feb 8
★	Manouselis, Nikos; Drachaler, Hendrik; Vuorikari, Riina; Hu...	Recommender Systems in Technology Enhanced Learning	2011	Recommender Systems Handbo...	Feb 5
★	Baker, Ryan S.	Stupid Tutoring Systems, Intelligent Humans	2016	International Journal of Artifici...	Feb 5
★	Moore, Alex	Teaching and learning: Pedagogy, curriculum an culture	2000	Practice	Feb 6
★	Sträffling, Nicole; Fleischer, Ivonne; Polzer, Christin; Leut...	Teaching Learning Strategies with a Pedagogical Agent	2010	Journal of Media Psychology: The...	Feb 5
★	Goodyear, Peter; Retalls, Symeon	Technology-Enhanced Learning	2010	Sense Publishers	Feb 5
★	Stankov, Slavomir; Rosić, Marko; Žitko, Branko; Grubiš...	TEX-Sys model for building intelligent tutoring systems	2008	Computers & Education	Feb 8
★	Srivastava, Ashok; Sahami, Mehran	Text Mining, classification, clustering, and applications	2009	Computer (Long Beach, Calif.)	Feb 7
★	Shiban, Youssaf; Scheihorn, Iris; Jobst, Verena; Hörnlein, ...	The appearance effect: Influences of virtual agent features on performance and motivation	2016	Computers in Human Behavior	Feb 5
★	VanLehn, Kurt	The Behavior of Tutoring Systems	2006	Int. J. Artif. Intell. Ed.	Feb 5
★	Markham, Kimberly M.; Mintzes, Joel J.; Jones, M. Gail	The concept map as a research and evaluation tool: Further evidence of validity	1994	Journal of Research in Scie...	Feb 8
★	Verpoorten, Dominique; Poumay, Marianne; Leclercq, ...	The eight learning events model: A pedagogic conceptual tool supporting diversification of learning methods	2007	Interactive Learning Environ...	Feb 6
★	vanLehn, Kurt	The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems	2011	Educational Psychologist	Feb 5
★	Popescu, Elvira; Trigano, Philippe; Badica, Costin	Towards a Unified Learning Style Model in Adaptive Educational Systems	2007	Seventh IEEE International Con...	Feb 5
★	Folsom-Kovarik, Jeremiah T.; Sukthankar, Gita; Schatz, Sae	Tractable POMDP representations for intelligent tutoring systems	2013	ACM Transactions on Intelligent Sys...	Feb 8
★	Ramesh, Vyshnavi Malathi; Rao, N. J.	Tutoring and expert modules of intelligent tutoring systems	2012	Proceedings - 2012 IEEE 4th Int...	Feb 5
★	Jones, Vicki Jo; Jun H.	Ubiquitous learning environment: An adaptive teaching system using ubiquitous technology	2004	Beyond the comfort zone: Pr...	Feb 5
★	Davis, Brent; Simmt, Elaine	Understanding learning systems: Mathematics education and complexity science	2003	Journal for Research in Mat...	Feb 5
★	Brusilovsky, Peter; Millán, Eva	User Models for Adaptive Hypermedia and Adaptive Educational Systems	2007	The Adaptive Web	Feb 5

And some of these were entire books and they are definitely things that would take a long time to read if I simply went word by word and read in a traditional manner but I actually had to go through this amount of content typically in a few weeks, and so I knew I had to have a better solution. And that's when I started researching some of the best speed reading techniques that are out there.

Many of them did not work out very well for me, but the spritz technique was one that actually was effective and so that's why I wanted to share it with you. So if you go to [spritzlet.com](https://spritzlet.com) this allows you to add this little bookmark bar here.

And so you can simply add it follow the instructions here. Just like it's showing you in the little animation. And what this is going to allow you to do is any page that you go on. You are going to be able to have the spritz technique implemented.

So I'm going to show you right here. This is a Python programming book and it is a decently long or this chapter is decently long. And so this might take a few minutes to go through. If I come and press on the little spritzlet bookmark bar you can see what it is doing.

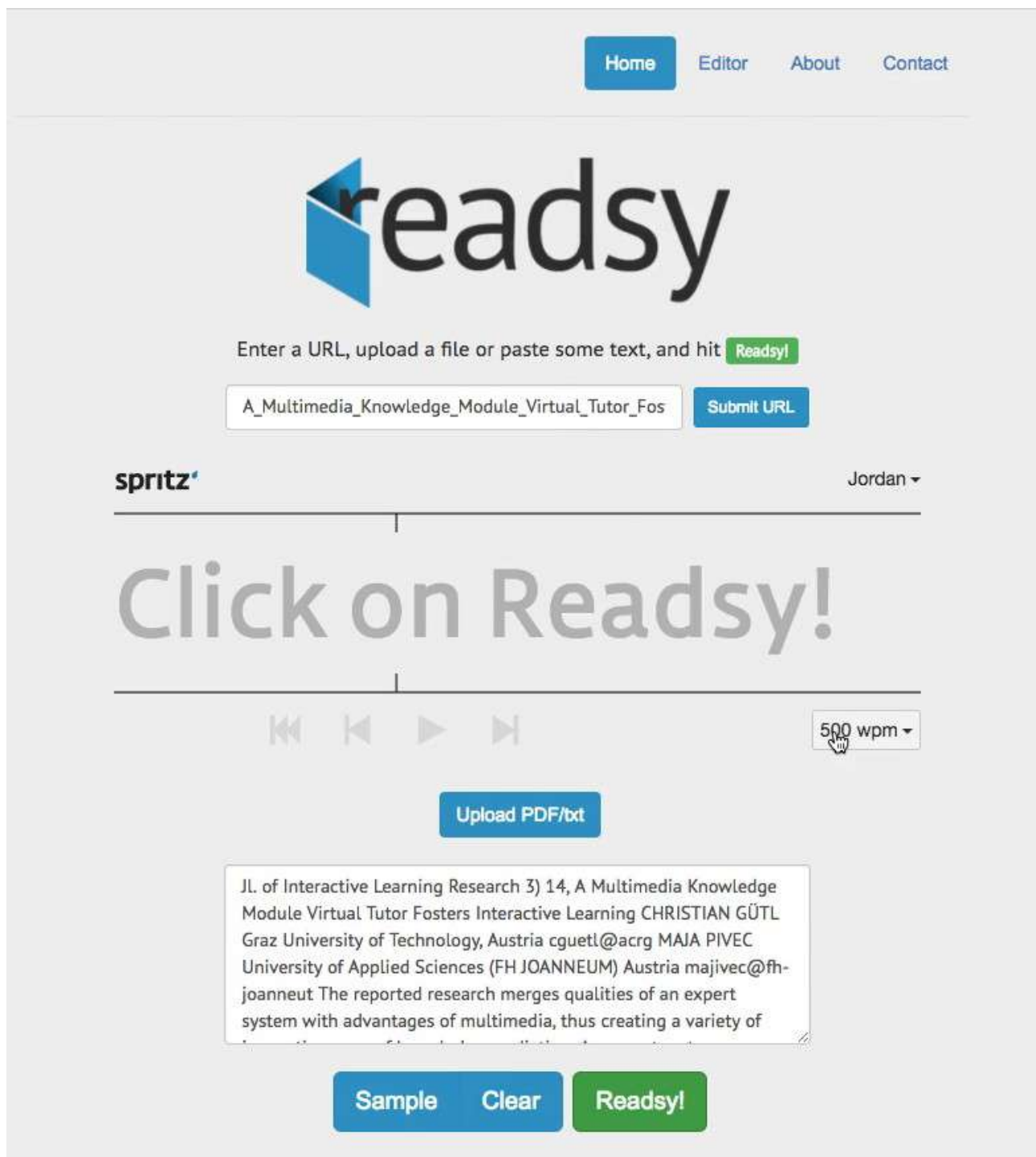
This is exactly what I was explaining at the very beginning, the way that spritz works is instead of having your eyes move from left to right. So if you were to read this content you would say "How do we compose a complex string?" and you'd go all the way down and one of the more challenging kinds of skills to learn is to remain focused when you're reading and it can be very hard. I'm not sure about yourself but if I am reading content like a book or article it's very easy for my mind to wander to get distracted, sometimes I'll read the same exact paragraph or sentence a few times until I'm actually paying attention.

What I discovered with the spritzlet technique is that it completely removes any of the distractions because this is moving so fast I'm really forced to focus specifically on the words as they're going across and I have it right now at four hundred and ninety-five words per minute. When I started I was much less I started around 150 words per minute just to start to get used to this kind of focus because this is a completely different way of reading content and then I moved to 200 than 250 words.

Now I'm close to 500 words per minute which allows me to go through this kind of content in a very short period of time. So that is how you can use the little spritzlet bar. Now if you have a large amount of content that isn't in web form. So for example, if you want a PDF and this is something that I had to do quite a bit with my research for my Ph.D. is most of the content that I showed you so everything that I have here all of these published works 100 percent of these are in PDF.

So the spritzlet bar does not work on PDF documents even when you open them in the browser. And so what they recommend is to use this tool called [Readsy](#) and what you can do with this is you can come here and it's free to use and click on the file that you want to upload. And so I'm going to pick this one out right here. Press upload it will go through the PDF document. Convert it into just a stack of words and then you can perform the exact same task.

So you can see right here



And so if you were to come and see me when I am researching some of those published works you would find me right here staring directly at this line. So staring at this vertical bar for minutes on end going through each one of these items and this is something that if you've never tried it before it may seem very weird or it may seem like something that simply wouldn't work.

And so what my advice to you would be is to simply try it, go and add the spritzlets bar to your browser and then say go to a Wikipedia page and make it be a topic that is something that you have some kind of understanding with so that you know what is going to be reviewed and then go through and go through the entire article.

And I think you'll be shocked by how much you retain and it's because this may seem like a very different way of reading content but because it changes your entire focus because you are now

completely zoned in on all of these words that are flowing through right here. You actually are paying even more attention than you typically would if you were reading in a traditional manner from left to right and all the way down.

And so what my approach usually is, is I use the spritz technique the first time that I'm going through a large piece of content and if it's something technical or something mathematical that has formulae then obviously I have to go back and go through those formulae and then I have gone through it twice but it's much faster by doing it that way and I get a very good idea for the high-level concept of what is being discussed and then I can go back and review it again and then I have mastered that specific article.

So in review, that is the spritz reading technique. I hope that you find it helpful and that it helps you in your development learning journey.

## Resources

- - [Spritzlet @ Spritzinc.com](https://sdk.spritzinc.com/js/1.2/bookmarklet/index.html)
- - [Readsy App - Highlight the important](https://readsyapp.com/)

## 3.26 From Copy Paste to Comprehension

There are a number of ways that developers are learning how to code nowadays.

You have developers that are self-taught, developers who went to a boot camp, and then you have developers who went through a more traditional approach. Where they went to a university, got a computer science degree, and then they learned development after that. You have all kinds of different ways that you can learn.

One of the common issues that I've heard from students through the years, is that they say they feel like if they're learning from a boot camp experience or if they're learning from online tutorials, it feels like they're walking through tutorials and they know how to copy and paste what the instructor is telling them. Then afterwards if they were asked to build out a real application, they'd have no idea where to start.

That's what I want to talk about in this video. The title of this video is going "From copy and paste to comprehension". The first thing that I want to say is that if you feel like you're at a stage where you're just copying and pasting code. So you are copying and pasting code from `Stack Overflow`, from tutorials, from anything like that. Do not worry.

That's actually perfectly natural. That is a very natural stage of the learning process. I want you to first feel that that's a normal. Someone like myself, or if you see any other developers, and you think that we just learned it magically, that is not the case.

If you feel like you're just copying and pasting right now, and you're not really understanding the concepts do not worry. That's all part of the learning process. I want to take a look at a real world analogy.

This is something that happened with me years ago, and it helped me understand a little bit more about learning in general especially as it related to coding.

When I was growing up I did not have a real handy kind of experience. I couldn't go and repair something in the house, I couldn't repair sheet rock, I didn't really have a kind of building experience whatsoever. I didn't grow up in a family where that was prioritized.

I had no idea how to do anything really when it came to being handy around the house. Then I bought a house, this was in my mid 20s or so, and it needed to pretty much be redone completely from scratch.

My father-in-law was a very successful architect, and he walked through and he showed all of the different things that had to be done. It was like he was looking at the house, and he already saw what the end goal needed to be and all the steps that needed to be taken in order to get there.

I didn't see it at all. What we did as a family is we just started rebuilding the house and started renovating it. Now I started with no knowledge of how to do that whatsoever.

What I would do is I'd be given some kind of task, it was usually pretty menial tasks, as they didn't really trust me to very much which they shouldn't. What they would do is my father-in-law would show me how to frame a wall, he would first do it himself, and then he told me to come over and do it.

I didn't have any idea on any of the details. I didn't know what was going on. I didn't know how foundations worked in the house, I didn't know the way that electricity had to work, or the way the wall had to be formed. All I could do was watch him, see exactly what he did, and then go and try to duplicate that myself.

That's how it started. That went on for months and months, and that entire time I still had very little knowledge on what was actually happening behind the scenes. What started to happen is the more time I spent remodeling that house, the more time that I spent practicing and following his example, all of a sudden things started to make sense.

I started to see how different parts of the house were all connected, and then I didn't even need to follow him in step-by-step kind of approach. I'd still go ask him for advice many times, but I started to build up a mental model for understanding how it worked.

Now with coding, it's the exact same thing. If you feel like you are in a stage where all you're doing is you're watching a screen cast, and three quarters of the things that they're saying don't even make sense, and you're just following along.

Sometimes it's hard to even follow along and you'll make typos, and what ends up working on their screen doesn't work on your screen. That is perfectly fine. That is exactly what happens in any kind of thing that you're ever going to learn.

Just like it happened to me when I was learning how to remodel a house. It took time, and it also took a lot of repetition. What my main encouragement for you to be is:

## **1. Don't get discouraged**

If you feel like you're not learning the underlying concepts right away, that will come. That comprehension will come. That's a reason why this guide is called from copy and paste to comprehension. You're not going to be able to build up that mental model right away. That is going to take time and it's going to take repetition.

It's the same reason why I recommend to all the students have gone through any of my courses that they need to be coding every single day. They need to be building out new applications, and learning the topics from different angles.

## **2. Watch Several Tutorials on topics you don't understand**

That also is something that I really try to encourage. If you watch a tutorial or you're going to a boot camp and you hear one concept, and it doesn't make any sense to you, that's fine. Go and see several different tutorials from other instructors on that topic. Keep going until it makes sense.

There's so many times where I've had a concept that I wanted to learn, and the first time I heard it it sounded like a foreign language to me. I've been doing this for over 15 years, and there are still things that make no sense to me the first time I hear them.

What I've learned what helps me out the very most is by going and hearing it from some other angle. I'll go, and just say it's a new feature in `React`, and I don't understand it the first time I read it in the update. So then I'll go watch a tutorial on someone who discussed that topic.

Maybe they gave an example, many times that example didn't make any sense to me either. So I go and I hear it a third time, and sometimes a fourth and a fifth, and I keep going until it actually makes sense, until I can repeat it myself.

If that's a stage where you're in, where you feel like you're just copying and pasting things, you're not understanding the concepts. Do not worry. Do not lose heart. What you're going to be able to do is be able to take those concepts, repeat them over and over again, and sooner or later they are going to start to make sense.

What I've found is that the very best developers, what really marks them is they refuse to give up. They will continue to repeat and to find out and discover new ways of learning things. They refuse to not understand a concept.

Make sure that that's the attitude in the mindset that you have. That you don't get discouraged when you hear something and it doesn't make sense to you. If that was the way I lived, I would just live in depression all day every day.

Every single day I hear something that I've never heard before and it doesn't make any sense. What my approach is, is if it doesn't make sense the first time I just make sure that I dive deeper into it until I do start to build up that mental model and until it starts to make sense. That would be my top advice for you.

If you feel like you are locked into that stage where you don't really understand the concepts yet, you're simply going through and repeating what others are doing. That's natural. That is the first

step. My advice would be work through that go through it, repeat it, learn from other people who were discussing the same topics until it makes sense.

Then the one thing I will promise you is after you've done that, and you've done it enough times, you'll look back at that concept that at first looked like a foreign language and didn't make any sense to you. It's all going to click, and you're going to be surprised it took you that long to understand it in the first place.



## 3.27 How to Break Through the Cycle of Procrastination

I want you to imagine sitting in front of your computer, staring at a blank screen. You know you have to work on a code project, but it feels like you're frozen. The task before you is so intimidating that you don't even know where you begin. It feels as if you'd rather be doing anything else in the world besides that task that's staring you in the face. This scenario is the ugly and all too common face of procrastination that programmers are forced to fight constantly.

If this situation sounds familiar you're in good company. But if you want to become a professional developer you'll need to implement a system for hacking procrastination. And that's what we're going to walk through today.

As the lead instructor for Devcamp, I get asked questions from students around the world. However one of the most prevalent inquiries I get from aspiring coders is how to overcome procrastination.

### Root Causes of Procrastination

Before we walk through a system for hacking procrastination we first need to dive into the root causes for this negative habit. Everyone is unique, however, through the years I've seen procrastination is typically caused by three thought patterns:

- Perfectionism
- Fear of success
- Lack of planning

In order to overcome procrastination and get back on track, we'll need to address each one of these issues. Because if you let any of these mindsets control the way your mind operates, you will never be able to reach your potential.

### Hacking Procrastination

I called this guide hacking procrastination because I think that hacking is the most appropriate term for what needs to happen in order to achieve success. Developers hack applications in order to build features or fix bugs. In the same way, we need to hack our thought patterns so that our brains function properly.

Before we go through the system I want to make one concept clear. As humans, we were made for action. Procrastination is a negative habit that we've learned through fear driven thought patterns. In

order to be successful at anything in life, whether it's development or business, overcoming procrastination is a requirement.

## Hacking Perfectionism

Starting off the list of the causes for procrastination is perfectionism.

Have you ever watched a baby trying to stand up for the first time? Babies, who haven't learned that failure is a "bad" thing will spend countless hours trying to stand up. Each time they fall down it doesn't seem to phase them in the slightest. But you won't find a baby that lets perfectionism get in the way of achieving their goal. Instead they will keep trying until they can stand up and eventually walk by themselves.

However somewhere between the time that we're babies and adults we develop the thought pattern that we're not supposed to fail. So instead of trying and failing until we succeed, we simply try to only perform tasks that we know we can do properly.

In order to hack perfectionism we have to remove the component in our brain that is afraid of failing. If you are a developer learning how to build a new feature that you've never worked through before? Let me clear something up. You are going to do it wrong the first time! And that's 100% fine. If you think that by waiting you are magically going to learn how to perform the task perfectly you are sadly mistaken.

So step one is: embrace failure and remove the requirement of perfectionism.

## Hacking the fear of success

Next on the list is hacking the fear of success. If you're overcome the trap of perfectionism, congratulations. However I've seen just as many developers get stuck due to the fear of success as the fear of failure.

This concept may seem odd since success doesn't seem like something that you should be scared of. However I remember back to when I was first learning development. When I was walking through a coding book I would get so excited when I discovered a new concept. However then I would freeze. My mind's first response was:

"If you learn this, then what are you going to do?"

For example, when I first learned how to build a connection to a database, I put the book down and didn't pick it up until weeks later. By learning the database concept it opened up a new and scary new world of all of the new topics I had to learn after that. All of a sudden I had to understand:

- SQL queries

- How to build relationships between database tables
- SQL injection requirements
- And the list seemingly went on infinitely in my mind

In order to hack the fear of success we need to quiet our minds. The fear of success is really rooted in the fear of the unknown. So whenever you're feeling this fear, simply take a step back. Be happy that you have learned a new topic. And then move onto the next feature or topic. Don't let your mind run wild with all of the potential, unknown concepts that you'll need to learn in the future. Like learning anything else, you need to take it one step at a time.

## Hacking the plan

Last on the list for hacking procrastination is creating a practical plan. When I recognize that I'm procrastinating I now tell myself to look at my plan of attack. Usually I'll discover that my plan is too general.

For example, if I'm building a payroll application I may have an item on my todo list that says: Build reporting engine. That's a scary feature! That's the type of item that will stick on my todo list for weeks without me taking any action. In order to fix this, I've learned that if I break the requirement into a series of very small tasks that I can break the cycle of procrastination. For the reporting engine feature I can create a series of much smaller, more manageable tasks, such as:

- Create page for users to access reports.
- Implement a database query for pulling the reports from the database.
- Build a file downloader for reports.
- Etc.

When I break a large and scary feature down into small pieces I instantly feel better. The feature is no longer scary and I no longer feel like pushing it off until tomorrow. Instead I am able to simply follow a set of small tasks each day until the feature is complete.

## Summary for Hacking Procrastination

I hope that this has been a helpful guide for helping you break the cycle of procrastination in your own projects and that you will be able to use it to become a more effective developer. I'll leave you with a quote from the book *The Five Elements of Effective Thinking*:

*“Being willing to fail is a liberating attribute of transformative thinking.”*

So put yourself out there, create a practical plan, and stop procrastinating and start coding!

## 3.28 Introduction to Asynchronous Method Calls

Whether you are a mobile or web developer, the topic of asynchronous method calls has probably come up at some point on your coding journey. So what exactly are asynchronous method calls and why should you care about them? Since technically that was two questions, let's take them one at a time.

### What are Asynchronous Method Calls?

In order to understand what asynchronous method calls are, it helps to go through an example. Imagine that you are posting a picture to Instagram.

If you use Instagram you know that after you create a post:

1. It shows you a loading icon. While your image is being uploaded to the Instagram servers you can view other posts.
2. If your post is created successfully it gives you a success notification.
3. Whereas if your post has an error it says that the photo couldn't be uploaded and asks if you want to try again.

Believe it or not, this workflow will teach you everything (at a high level) that you need to know about asynchronous method calls as a developer. Let's take a look at the process step by step.

#### Step 1

During step 1, when Instagram shows the loading icon and allows you to view other posts, the app is making what's called an asynchronous request to the Instagram server. It's considered an asynchronous request because it's performing an action. In this case, it's uploading a file, while not blocking other method calls, such as using other parts of the app.

#### Step 2

In step 2 Instagram is sending a response back to the application. When the app sent out the request in step 1 it created what's known as a promise. The promise expects some type of response from the server. Typically either a success message or an error. In the case of step 2, the promise is ready and waiting to catch the request as it comes back from the server.

And since the response contains a success message Instagram proudly lets the user know that their post is now live.

### **Step 3**

In step 3 the application needs to know what to do when a failure occurs. Imagine how bad of an experience it would be if you uploaded an image to Instagram but it didn't tell you if there was a failure.

In order to ensure that the user and the application know when issues arise, the server response in step 3 would contain an error message. From that point, it would be the app's job to parse the response and give the user feedback.

## **Recap**

As a recap, asynchronous method calls are tools you can use to communicate with outside services while not stopping the flow of the application. Typically asynchronous method calls work with promises. This means that they wait for server responses to confirm that the process completed properly or an error message if there was an issue.

## **Why are Asynchronous Method Calls Important?**

So now that you know what asynchronous method calls are, why are they important? Well, let's imagine a world without asynchronous behavior.

## **A World Without Asynchronous Method Calls**

Unlike our streamlined example of Instagram, let's imagine that YouTube didn't allow for asynchronous method calls. And let's follow that up by imagining that you had thousands of videos to upload to the site.

In the dark ages of the web, before the asynchronous method calls became popular, this scenario would have been very ugly. Because the system wouldn't allow you to perform two actions at once, you'd have to open a new page for each video you want to upload. From there you'd need to start an upload from scratch.

Instead of being able to upload multiple files at a time, like YouTube lets you do now, you'd be forced to take days to upload all of your videos. And then you'd have to monitor each of the open pages in case an error occurred. Needless to say, this would not make for a good user experience.

# Mobile Apps

When I was teaching myself how to build iOS applications my knowledge of asynchronous method calls really came into play. If you are building mobile applications that communicate with outside servers a thorough understanding of asynchronous methods is required.

## Summary

I kept this guide very high level. The main reason is because each language and framework has a different configuration for how asynchronous method calls work. In the show notes, I've included links for some popular frameworks and how to implement asynchronous behavior.

## Resources

[Asynchronous methods and threads](#) .

<https://stackoverflow.com/questions/21122842/whats-the-difference-between-synchronous-and-asynchronous-calls-in-objective-c>

- [Swift: Asynchronous](#)
- <https://www.youtube.com/watch?v=qzFQXE6Y44M>
- [Promises in AngularJS, Explained as a Cartoon](#)
- <https://www.andyshora.com/promises-angularjs-explained-as-cartoon.html>

## 3.29 An Introduction to AI and How Artificial Intelligent Agents See the World

This guide discusses how artificial intelligence programs see the world.

This post was inspired by an article I was reading from the MIT Technology Review.

<https://www.technologyreview.com/2016/06/30/159029/ai-is-learning-to-see-the-world-but-not-the-way-humans-do/>

### Article Review

The article was titled “AI Is Learning to See the World—But Not the Way Humans Do”. This title caught my eye right away because it described what I’ve seen throughout my graduate school studies.

While I don’t pretend to be a domain expert when it comes to artificial intelligence. I have taken a number of courses on the subject and I find it fascinating. This article specifically struck a chord with me because it focused on an element of AI that is oftentimes overlooked.

So many people have the movie version of artificial intelligence in their minds.

Images of human-like machines that act exactly like humans, except for the lack of emotions may be scrolling through your thoughts right now.

However, it’s been my experience that the future of AI will be much different than what science fiction suggests.

### Plane vs Bird

Before we go further into my opinion on how artificial intelligence programs see the world, let’s scroll back in history and look at a different type of technology.

Before the Wright Brothers went on their initial flight, humans had been attempting to fly for thousands of years. However, the early flying technology failed. This was mainly because humans were attempting to replicate how birds fly in order to allow humans to take to the skies.

It wasn’t until humans realized that conquering flight was going to be accomplished via a different route were we successful. As you may have noticed, airplanes do not resemble birds in the least bit, in form or function. There are some commonalities that both share, however at the end of the day planes and birds are able to accomplish the same goal, flying through the air, while using very different processes.

This analogy is how I like to think about artificial intelligence vs humans. Computer scientists have been trying for years to build computers that mimic human intelligence and behavior. However, I believe that true AI will not resemble humans at all, but instead will be tools that allow humans to become more skilled at whatever we do.

## **AI's True Purpose**

With so much discussion on whether or not artificial intelligence programs will replace humans and take over the world, a view held by Elon Musk and Stephen Hawking, the true goal of AI gets lost in the debate.

In analyzing the true purpose of AI, it helps to discuss what computers can do better than humans, and vice versa.

Computers can process vast quantities of data in a very short period of time. This processing power is far better and more efficient than humans.

On the other hand, humans have the ability to make common-sense decisions and learn how topics are related to each other.

Therefore the goal of AI, in my mind, should be to combine the decision making power that we have as humans. And augment that with the processing power found in computer systems. This type of application would be very effective.

## **Practical Application of Artificial Intelligence**

Let's consider how AI can be applied to learning how to code. Right now universities and coding boot camps are trying their best to teach students how to become developers. However, even the best schools have to use a one size fits all approach in some form or another.

This approach to teaching development is the best we can do right now. However, by leveraging AI, schools will be able to analyze learning trends. Which will allow them to customize curriculum based on student behavior. Imagine having curriculum that dynamically changes based on how you performed on an exam or homework to fit your needs? That's what is possible with the proper application of AI.

This type of behavior can be developed, not by space-age robots, but instead by practical machine learning algorithms that track students' performance.



# Turn Off the Computers!

On a final note, I want to address the idea that artificial intelligence will one day take over the world.

Eric Schmidt, the former CEO of Google had a great take on AI, specifically addressing the doomsday scenarios that people are discussing. He said:

“The scenario you’re just describing is the one where the computers get so smart is that they want to destroy us at some point in their evolving intelligence due to some bug. My question to you is: don’t you think the humans would notice this, and start turning off the computers? We’d have a race between humans turning off computers, and the AI relocating itself to other computers, in this mad race to the last computer, and we can’t turn it off, and that’s a movie. It’s a movie. The state of the earth currently does not support any of these scenarios.”

## Summary

In summary, AI is one of the most critical sectors in computer science today. The services that effectively combine human expertise with artificial intelligence will be the most valuable companies in the world and will help take humanity to a new stage of evolution. However, it’s important to have a clear understanding of how computers work compared with how Hollywood likes to depict AI. In the show notes, I’ve includes links to some resources if you want to further study artificial intelligence and the topics I’ve discussed today.

## Resources

<https://www.businessinsider.com/eric-schmidt-dismissed-the-ai-fears-raised-by-stephen-hawking-and-elon-musk-2016-6>

<https://www.brookings.edu/articles/understanding-artificial-intelligence/>

## 3.30 Guide to the Model View Controller Architecture

Today I'm going to discuss the Model View Controller Design Pattern, also known as MVC architecture.

While the Model View Controller design pattern is used by some of the most popular application frameworks, it's still a concept that can be confusing, especially to new developers.

At its core, the Model View Controller design pattern is a workflow that gives a structure to application frameworks. MVC lets you, as a developer, organize your code in a way that is logical and can also be understood by other developers.

### Practical Implementation of MVC

Like other topics, I think the best way to understand the Model View Controller design pattern is to break it down into individual pieces, and then putting them together to see how they all work with each other.

**Model** – Starting off with the model component, this is where the application data resides. For example, if you have an invoicing application, the model files will store items such as the attributes for invoices, custom methods such as how an invoice relates to the rest of the application, along with other features such as database query scopes. The model should only be called by the controller and shouldn't have any interaction with the view.

---

**Controller** – Controller files manage data flow for the application, they take in requests from the routing engine and coordinates how the rest of the application processes that request. For example, in a sample invoicing application if you go to a page where you can create a new invoice. The controller would register that request, pick out what view should be rendered, and call the model for any data that is needed for the page. In a well written application controller files should not contain very much complex logic.

---

**View** – Views should be the most straightforward component in an application. The view should simply render the template and display any data passed to it by the controller, it should not have any direct communication with the model, instead it should only communicate with the controller.

## A Real World Example of MVC

In one of my teaching roles I worked on creating Rails curriculum for Learn.co, and they had a great analogy for understanding the Model View Controller design pattern:

Imagine the the MVC architecture is a restaurant. The model is the chef, the chef's role is to get orders from the waiter and make the meals. The waiter is the controller, they take requests from the customer, informs the chef, and brings the meal back to the customer. Lastly the view is the table, it simply holds the meal, it doesn't have to do any type of complex task besides simply being there.

So why is the Model View Controller design pattern important? Imagine if everyone built applications the way that they personally felt code should be created, if new developers started to work on a project, they would have no idea where to even find features, much less how to extend the functionality or fix bugs. With the Model View Controller design pattern a new developer can quickly find where the data files are, how the data flow works, and where to update the view pages.

I hope that you now have a more clear idea of how the Model View Controller design pattern works and how you can implement it in your applications.

## Resources

- [Model View Controller detailed walk through](#)
- <https://onextrapixel.com/a-detailed-overview-of-the-model-view-controller-mvc-coding-structure/>
- [Example of MVC based application](#)

<https://basque.devcamp.com/trails/learn-ruby-on-rails-from-scratch/campsites/building-your-first-rails-application>

# 3.31 Object Oriented Inheritance Tutorial for Developers

First and foremost, what is Inheritance as it relates to OOP? At a high level inheritance allows you to extend the functionality of classes in your code. We'll walk through a real world code project in this guide that will give an example for how inheritance can help you more efficiently design your codebase.

## Poorly Constructed Code

We'll begin by reviewing a real world example of a poorly written feature. I'm going to use the Ruby programming language for this example, however inheritance works the same across most programming languages. Here we have an Invoice class that, for the sake of this example, simply prints out a summary of the invoice. The class encapsulates a total, customer, and category value. Additionally the class has a summarize method that calls those values and prints them out.

As far as a traditional class goes, this Invoice class works perfectly fine. However, as it usually occurs, what happens if you're asked to build a feature for printing out a shipping label. If you don't use inheritance then your code may look something like this.

With the new ShippingLabel class we need a total and attribute value. Additionally we're printing out a label that contains the values from the class. There are a number of issues with this implementation, the most glaring is that the Invoice and ShippingLabel classes have almost identical values. By creating two separate classes this code is breaking the DRY (Don't Repeat Yourself) principle. If the requirements change where invoices and shipping labels need to both print out an address, city, state, and postal code you'll need to add the identical attributes to multiple places in the application. Eventually this type of application design would lean to a very convoluted codebase that would be nearly impossible to maintain.

Don't worry though, this is where the principle of inheritance comes in to play.

## Object Oriented Inheritance Tutorial

In order to refactor this code we're going to perform two tasks:

1. Use inheritance so that the Shipping Label class will be a child class of the Invoice class.
2. Refactor the Invoice class to it's more flexible and will work better as a parent class.

Using inheritance in Ruby is relatively straightforward. In our class definition we simply declare that the ShippingLabel class inherits from the Invoice class, as shown here.

As you can see, by having the ShippingLabel class inherit from Invoice we were able to remove the initialize method and associated attribute declarations. This implementation has removed the duplicate code we had earlier, so it's looking better. However there's one issue with our program now. If you notice when we create a shipping label we need to add in the category, because the Invoice class requires it. However it would be a poor design decision to require shipping labels to be instantiated with an attribute they don't need. With this in mind let's move onto refactoring the Invoice class to make it more flexible.

Most languages have the ability to declare that some attributes in a class can be optional. In the code below I've removed category from the attribute argument list and now it's simply an optional value.

With this implementation, when we create a new shipping label we don't have to include any values that would be unnecessary.

For further reading on Inheritance I highly recommend "Practical Object-Oriented Design in Ruby: An Agile Primer" by Sandi Metz. I hope that this has been a helpful object oriented inheritance tutorial and will help you improve your code design decisions.

[Practical Object-Oriented Design in Ruby: An Agile Primer](https://www.amazon.com/Practical-Object-Oriented-Design-Ruby-Addison-Wesley/dp/0321721330)  
<https://www.amazon.com/Practical-Object-Oriented-Design-Ruby-Addison-Wesley/dp/0321721330>

Please note that this book is not free and is working with the Ruby programming language. If you would like to check it out, feel free, but it is not required.

# 3.32 Guide to CSS Selectors for Web Developers and Designers

When it comes to building out web applications, a solid understanding of CSS will take you a long way. You can use this post as a guide to CSS selectors. Which are the key component for connecting styles with page elements.

## What is CSS?

CSS stands for Cascading Style Sheets. Your CSS files are where you can place the rules that will dictate the look and feel of your application.

## What is a CSS Selector?

So if you imagine the components of a website being cogs in a machine, your HTML and CSS elements would be separate entities.

Essentially your CSS files contain style rules for the website, but in order for the rules to go into effect, you need to connect the two components. And that's where CSS selectors go.

CSS selectors allow you to specify elements on the page and then apply the style rules to the elements that you list off.

For example, you may want to select a navbar or a table so you can give the components their own styles.

Taking a look at this page you can see a sampling of the page elements that you can select. Including:

- Page title
- Logo
- Navigation bar
- Image
- Sharing Links
- Breadcrumbs
- Etc

# Guide to CSS Selectors

So how exactly can you select elements on a page? Much like the HTML markup language, there is a specific syntax to use for CSS styles. And that's what we'll walk through today.

## Basic Syntax

Starting with the basics, there are two main ways to select items on a page. You can either select:

- Elements – such as divs, h1 tags, etc. For these, the element name itself will select the item on the page.
- Unique IDs – for selecting an ID you use the # symbol before the name of the ID.
- Classes – classes are attributes that you can use multiple times. In order to select a class, you use the . selector syntax.

In this image, you can see each one of the selector options. The div element is selected and given the background color of red. The div with the class name of my\_cool\_class has been given the background color of green. And lastly, the id named my\_cool\_id has been given the background color of blue.

## CSS Selector Prioritization

Notice how even though all three elements are divs, they can all be selected separately. This is accomplished through the concept of CSS selector prioritization. At a high level, it means that the most specific selector wins. This is why the background color red wasn't rendered for the elements that had the class and id selectors since they are considered more specific.

## CSS Selector Nesting

In addition to selecting basic elements, there are many times where you need to select nested elements on a page.

In this example, you can see that we have a set of bullet point elements nested inside of our class. By using this syntax we're able to select the nested items inside of another element and define their own styles.

## **nth-child Selectors**

So what happens when you want to select an item that doesn't have a class or id and simply belongs to a group of elements. For example, what happens if we need to give specific styles to only one bullet point?

That's where the nth-child selector comes into play.

Here you can see a selector that leverages the nth-child mechanism, which allows us to specify which element we want to style. In this case, I passed in 1 to pick out the first bullet point. nth-child selectors are a part of a special group of selectors called pseudo-classes.

## **List of Pseudo Classes**

The list of pseudo-classes is relatively extensive and I've placed a link to the full list, along with a link to the CSS selector documentation that you can use to further your knowledge.

## **Summary**

I hope that this has been a helpful guide to CSS selectors and that it can help you build custom styles into your next web application.

## **Resources**

- [CSS Selector Documentation](#)

[https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting\\_started/Selectors](https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_started/Selectors)



## 3.23 Best Practices vs Creativity as a Developer

This guide discusses how you can find the balance of best practices vs creativity as a developer.

To be honest, this was initially a difficult post to write, mainly because I had a hard time organizing my thoughts on the topic since it's a little abstract. I had the high level concept of the strained relationship between best practices vs creativity as a developer, in the sense that many developers, especially ones new to coding will fall into two camps:

1. Attempting to follow standardized conventions in every way, essentially duplicating code from tutorials and a language or framework's documentation page.
2. Ignoring all common practices and building applications in whatever way makes the most sense to them (at the time).

There are pros and cons with both approaches, and like many other topics, a cross between the two is going to result in the best strategy. Let's look at the pros and cons of following best practices and ignoring the idea of being creative:

### **Pros:**

- Application code will be easier for future developers to pick up and add features to since they'll know how the code is structured and where all of the methods and classes are located.
- Programs should be well tested via unit and integration tests.

### **Cons:**

- Code structure may have more of a cookie cutter approach.
- Code may suffer from the square peg/round hole syndrome.

Before I go on, please don't misunderstand me by thinking that I'm saying that developers who follow best practices have these issues. I'm simply referring to developers who throw creativity out the door and simply try to build programs using standardized methods and attempting to copy code that they see from other developers.

Now let's take a look at the pros and cons of developers who only embrace creativity:

### **Pros:**

- They have fun, this is very important since it keeps them motivated to build interesting projects. They're also able to express their unique perspectives on how applications should be structured.

- They're constantly trying new things and finding new ways to build features. This can result in learning quite a bit about the language or framework that they're using.

### **Cons:**

- Programs that only rely on the developer's creativity can be nearly impossible to manage later on. Even the developer who built the application may have a hard time understanding his own code if he has to go back and add new features.
- Even though programs built ignoring best practices and relying solely on the developer's creativity are fun to build at first, as the codebase grows the level of fun decreases exponentially. In fact, it's common for new developers to kill an entire project and have to start from scratch because the codebase became such a mess following a non-standardized approach.

So if there are pros and cons to both approaches, which is the best way to go? I am a self taught developer, and originally I definitely fell into the second camp of building apps simply using creativity. However I ended up building some horrible applications, I did learn a lot about various languages through the process, so it was a beneficial strategy from a learning perspective. Over the years, since I matured as a developer, I realized that I had to find a balance between following standardized best practices and being able to add my own creative touches into a program.

It's simply ignorant to disregard industry wide accepted best practices. Concepts such as properly structured, object oriented code increase a project's maintainability and also makes it more efficient to add new features in the future. Some of the most brilliant minds in the world have spent the past century refining development procedures, and a good developer should build upon that cumulative knowledge.

With all of that being said, there is still a place for being able to integrate your own creativity into a development project. The more skilled I've become as a developer I've realized that I'm more creative with my code than I ever was before. When I originally started programming, my "creativity", was really just an unorganized attempt to get features to work the way I thought they should be structured in my own mind. However, around a decade of experience has refined how I build programs, and that experience has allowed me to learn how to be more expressive with how I write applications.

The more confident I've become as a developer the more I've been able to explore different ways of building projects and I'm having more fun now than ever I had before.

I'll leave you with this thought from Sandi Metz, one of the software developers I personally admire the most and the author of the book "Practical Object-Oriented Design in Ruby" when she

described the balance between following cookie cutter approaches vs implementing creativity in development, she said,

“Design is not an assembly line where similarly trained workers construct identical widgets, it’s a studio where like-minded artists sculpt custom applications. Design is thus an art, the art of arranging code.”

I hope that this has been a helpful discussion and will help you find the balance of best practices vs creativity as a developer in your own projects.

## 3.24 A Developer's Guide to Content Delivery Networks (CDNs)

In this guide we're going to walk through how to understand content delivery networks. One of the first companies I worked with was a video platform company. That was over a decade ago and CDNs were in their infancy. That was my introduction to the world of content delivery networks and since that time I've worked with a large number of CDNs for various applications.

### CDN Definition

So what is a CDN? CDN stands for Content Delivery Network. Essentially CDNs leverage a distributed set of servers, replicating data across a geographic area to enable faster application load times. If that isn't 100% clear don't worry, let's walk through an example of how CDNs work.

### Content Delivery Network Example

One of the top CDN users in the world is YouTube. With petabytes upon petabytes of data the YouTube streams over 4 billion videos to users across the world each day! In addition to having to play videos for users, the site also prides itself on being able to give instant access to media.

If YouTube took a minute or more to retrieve a video it would make for a very poor user experience. With that in mind the site employs one of the web's more powerful content delivery network.

Instead of storing each uploaded video at a single data center, YouTube takes a distributed approach. By leveraging a CDN the site stores the video files at data centers across the world. This means that when you're watching this CronDose episode from your home in Los Angeles, someone watching the same episode in the UK is being sent a cloned video file from a server closer to them.

This approach decreases the distance that a file needs to be sent to the end user, which improves the performance.

### CDN Diagram

As you can see in this diagram, a CDN leverages servers throughout the world (or a defined geographic region) in order to give end users quicker access to data. This is a powerful tool because it allows applications to deliver the same speed to users anywhere in the world.

### How Does a CDN Work

Let's go through a step by step example to understand how CDNs work.

# Data is Uploaded to a Server

As you may imagine the process is kicked off when a user uploads some type of media to a server. This can be a video, a picture, a website, or really any type of data that will be accessed by users.

This data is uploaded to a single data center. The distribution process doesn't kick off until the next step.

## User Accesses Data

Next in the CDN workflow is when a user accesses the data. Extending our YouTube example, this could be a user watching a video from England. The playback for this video may be a little slower because the video needs to be transferred to the server closest to the user in the UK.

Now the video file will be stored on the server in the UK and any other users accessing the same video will be streaming it from the local server instead of the one on the other side of the world.

## Replicated Process

This process will get replicated across the world. If you're wondering why an uploaded video isn't instantly pushed out to each remote server, that's a good question. The reason is due to data efficiency management. The majority of videos on YouTube aren't watched by users across the world. Therefore it would be a significant waste of resources if the system pushed every video across the network.

## Expiration

Lastly, just like you throw out old milk in the fridge, CDNs place an expiration date on content. This is to ensure that a remote server only stores fresh data. After data has expired it's wiped from the system. If another user requests the same video it will be retrieved from the original data center and it will be given a new expiration date.

# When to Use a CDN

So this is all great information if you have the next YouTube, but when else is it helpful to use a CDN? You may be surprised to learn all of the use cases for working with a content delivery network, here are a few.

## Literally any website

You can have your HTML, CSS, and JavaScript assets stored on a CDN, which will dramatically improve your site's load time. The CronDose and devCamp websites both leverage a content delivery network for assets. This has resulted in a better user experience and improved search engine rankings since Google includes site speed as a SEO requirement.

## Video streaming

If you are streaming your own video you will definitely want to leverage a CDN. Video is one of the most resource intensive features that you can build into an application. Traditional hosting systems such as a VPS or even a dedicated machine can't compare with the performance of a content delivery network.

At devCamp we chose to go with Amazon's CDN since it integrates seamlessly with the S3 storage engine, which we use for hosting our videos. We're currently in the process of integrating the CDN prior to our beta launch.

## Image hosting

Lastly I strongly recommend using a CDN for image hosting. There are a few reasons for this and they're both related to search engine optimization. As I already mentioned, Google has made site speed an important criteria for high page rankings. With that in mind streaming images from a CDN is faster than a typical server. Also, web browsers will load images from another server simultaneously with your website.

## Page Load Visualization

Not clear? Let's take a look at this visualization.

Let's imagine that you have a website that stores your site and images. When a user goes to your page the web browser will load your website and THEN the images, since they're located on the same server. or the sake of working with easy numbers let's say this full process takes 2 seconds.

However if you store your images on a CDN a web browser will try to pull both the site and images in at the same time, theoretically cutting the page load time in half.

## List of CDNs

So now that you know what CDNs are used for, here are some of the top CDNs to work with.

- Akamai
- Limelight
- Amazon
- MaxCdn

## CDN Workaround

I'm not going to lie to you, working with CDNs can be time consuming and a bit challenging. With that in mind I also wanted to include a CDN workaround called CloudFlare. I use CloudFlare for the CronDose site and it gives the full benefits of a CDN. However it doesn't require you to perform the manual integration.

Many hosting providers also enable you to install CloudFlare when you create your site, which is what I did with CronDose. I liked it because it lets me spend more time on creating content and less on working with multiple hosting providers.

## Warning to Using CDNs

Before you run out and start migrating all of your sites, videos and images to a CDN I offer a word of warning. One of the greatest strengths of a CDN, its ability to cache website assets, can also make it frustrating to work with.

I have had multiple times where I have pushed design changes to a website for a client and they couldn't see the change. Even after clearing their cache they couldn't see the updates I made. This issue arises because the CDN still had the old version of the website stored on the server closest to them.

When this happens you need to clear out all cached versions of an application. Services like CloudFlare let you do this with one click, however other systems can be a bit more complex.

I typically make the CDN integration one of the last features to add when I'm building a new application. This makes a site's front end easier to work with since you'll know that everyone is looking at the most up to date version.



## 3.25 What is Digital Literacy?

In this lesson, we're going to discuss what digital literacy is.

I think one of the best ways of doing this is to first look at what the term literacy means. If you say that someone is literate, it means that they can read, they can write, but it also is something more than that. It means that they have a mental framework for understanding a specific kind of topic.

So with that in mind, let's talk about digital literacy. There are many different forms of what digital literacy could represent. It could be a traditional kind of concept, such as being able to read or write code, but it actually extends much further than that.

It could be the ability to automate tasks. Imagine that you are an accountant, you may not ever build an actual program, but if you have digital literacy, you may have the ability to understand the way, one, how accounting software works, and two, you could build automated tasks in order to process all of your daily tasks in a more efficient manner.

By the end of this course, one of my top goals for you is that you'll have a mental framework for being able to understand how technology works. This could help you in any kind of career that you're in. If you're going to be a developer, these are the foundational topics that you need to understand in order to start writing code.

If you're going to be a project manager, this is going to help you communicate properly with whatever type of team you're working with. Even if you're not going to be a developer at all, these are going to give you the skills and the mindsets that you're going to need in order to have success in any field.

## 3.26 Why do Digital and Coding Literacy Matter?

Now that we've walked through a high level definition for what digital literacy is. Now let's discuss the importance of digital literacy in all kinds of different fields.

So, first and foremost, if you want to be a developer, digital literacy is an absolute must. You're going to have to understand all kinds of different topics in order to build-out applications. You're going to have to understand how web browsers work. You're going to have to understand the basics to start off of various programming languages and why you'd pick one programming language over another. So, from a developer's perspective, that's what's needed.

But what if you don't want to become a developer on a long-term basis? Maybe your aspirations are to be a CEO or some type of high ranking executive in your company, you still need to have digital literacy. And I want to talk about a tale of two CEO's that I've worked for through the years. So, one was a non-technical CEO and the other one did have a form of digital literacy. They weren't a programmer per se, but they did understand the high-level concepts and skills required in order to build-out applications.

With the first CEO, the non-technical one, every project that I worked on with them was a struggle. They constantly had expectations that didn't match with reality. They wanted outcomes, so they wanted the application to do things that either, weren't possible or weren't really practical. And so, the end result was that we had a number of failed applications, the company wasted quite a bit of money and everyone on both sides of the team, on the executive team and with the developers was just, constantly frustrated.

Now, let's compare that with the second CEO I mentioned, he's actually the CEO of Bottega and he has some experience in programming. He doesn't build any applications, currently, but he has in the past. He definitely has a very strong form of digital literacy. He understands various programming languages, he knows when you'd want to use one versus the other. He understands the way that databases work and all of these kinds of high-level concepts.

Working with him is much more efficient than that first CEO. When we plan out a project, he has the proper expectations on how long that project is going to take. He also has a solid understanding of the resources needed. So, how many developers are going to be needed? How many Project Managers? What the capital expense is going to be for servers? And because of that, each one of the projects we work on has a much higher level of success. And also, each one of the developers on the team really enjoys the process much more.

So far, we've talked about the importance of digital literacy for executives and also for developers. But, what if you don't want to do either one of those things? Having a solid idea of digital literacy is still important. And the reason for that is because the digital world has touched every area of our lives in some form or another. Whether it's financial markets, politics, or even real estate. You can't really escape it anymore, and so whenever you can have a higher understanding of how the Internet works, or how complex data systems work, or anything like that. It's going to give you a leg up in every part of your life.

## 3.27 Tech Terms and Concepts

In this guide, we're going to walk through some key technology terms and concepts.

Now, I don't want you to feel intimidated whatsoever, if you've never even heard of any of these terms before. The goal of this guide is that I want to start painting a picture for you for how development works and give you an idea of how you can build your own mental framework for digital literacy.

The four key terms that we're going to talk about in this guide are going to be **stacks**, **servers**, **clients**, and **APIs**. We're going to walk through each one of these terms because, believe it or not, almost every one of the things that you see, whether it's an application or some type of software system you're working with, comes down from one of these terms or has to use it in some form or another.

Starting at the top what a **full-stack developer** is. If you ever hear someone called a full-stack developer, that means that they work at every stage of an applications development. They will usually build the front-end, that is what you see, it's the look and feel of an application. Then they'll also build the back-end, and then they will connect the two. They'll build databases, and every part of the application that needs to be implemented, they will do.

Now, full-stack developers usually work at smaller organizations where they're needed to build out an entire system, and the resources are limited. Usually when you're working in a larger organization, you have a much more specific set of tasks.

Moving on to the next one, such as a **front-end developer**. A front-end developer specifically focus is, as you may have guessed, on building out the front-end. So that is all they are doing. They are not working very much with servers, instead they are working to implement designs and everything that a user sees when they interact with an application.

Next is a **back-end developer**. A back-end developer is one that doesn't build anything that a user is actually going to see. Instead, they're specifically focusing on building out the code on the server. So they're building out all of the conditional logic, the rules, authorization, all of the kinds of things that the front-end developer is going to need to interact with but that a user won't actually see.

Lastly is a **mobile developer**, so the mobile stack. If you're working on mobile, you're building different smartphone applications. Typically, that's going to be something that is on android or on an iOS device. There are a number of different ways that you can build mobile applications.

Technically, mobile is usually seen as a different form of **front-end development**, and the reason for that is because whenever you're interacting with some application on your phone, it

doesn't typically reside purely on the phone. It usually has to go and communicate with a **back-end server** so that it can have access to **databases**, it can perform various **queries**, and it can access data that may not reside on the phone. So those are the **development stacks**.

Now, let's talk about **servers**, we've already kind of been introduced to them when I mentioned **back-end developers**, but technically every one of the applications that you're going to interact with, whether it's mobile, whether it's on your desktop, or a web application, has to communicate with a **server** in some form or another. Now there are all kinds of servers. There are servers that run Windows, there are servers that run Linux, and at the end of the day, every kind of application is going to have to use it in some form or another.

If you have a database and you want that database to be accessed by anybody else, whether it's someone else in the organization or someone else on the other side of the world, that's going to have to reside on a server and then someone's going to have to build a way of interacting with that.

That's a perfect lead-in to the next key term. If a server is where all of the code and databases and all the data resides, then what a **client** is, is anything that interacts with the server. Popular clients are web browsers, mobile phones, or even other applications.

Lastly, the last key term I want to talk about is an **API**. Now, an API is a little bit of a scary term if you've never heard it, but really it is not as complex as a lot of people try to make it seem. It stands for **application programming interface**, and all that is, is a fancy term for saying that there is a way of communicating or interacting with the program.

Now, if you've never heard any of those terms before, then that could still be a little bit confusing. Let's walk through a real world example that shows how each one of those can be utilized.

Let's say that you want to post a picture on Instagram. You pull out your phone and you load up the Instagram, iPhone, or android application. You take a picture, and you are using a client, that application on your phone is considered a **client**.

Now, when you take the picture and you press send, that you want to post that to your Instagram account, what happens is that it goes and communicates with an **API**. So, Instagram on their servers has an API, and it allows for accepting pictures, taking in things like the title, description, knowing which user sent it. The API wraps up all of that data and then it stores it on the **server**.

The server then goes, it stores it in the **database**. It performs all of its calculations, and any kind of work that it needs to do. It then goes and it sends all that data back to the **client**, and you can see that your image was posted and now all your friends can see it.

## 3.28 Digitization, Data, and Analytics

In this guide, we're going to walk through three more key terms to help you along your digital learning journey.

The three terms that we're going to walk through are **digitization**, **data**, and **analytics**. Now, you're going to see as we walk through each one of these concepts, there is overlap between all three of them. They will rely on each other, and you're going to have to use them in some form or another (that's perfectly fine). So, at the end of this guide, if they all seem very similar, that is natural. I want you to understand what they are, and specifically how they're different from each other.

Let's start with the first one. **Digitization**, I think the easiest way of understanding it, along with each one of the other concepts, is to walk through a real world example. Digitization can be represented by imagining going to a doctor's office. You walk in and they would look you up in a giant set of cabinets. They are full of all these paper records with your medical history. Every time you've gone into that doctor, they took notes and they can see your medical history with those set of paper records.

What digitization did, is it digitized each one of those records, so now instead of going in and them looking you up with paper records, they can perform a database inquiry in their computer and automatically pick those out.

Where this comes in very handy is when they want to transfer those data records. Before, they would have to do things like fax over your full file. Now what they can do is they can share it securely, so that if you go to another doctor, they can then immediately have access to your digital records.

The next concept we're going to walk through is **data**. Now technically, just about everything that you do on your computer is related to data in some form or another. Let's take an example of a sports team.

Years ago when a specific sports team (it could be football, it could be baseball, it could be basketball) wanted to place their players in a specific position on the field, they usually were using their own gut instincts.

If it was baseball, they would try to position the players on the field with where they expected the hitter to hit the ball. Now with data, what they can do is they can go back and look through

historical records and instead of just going with their gut instinct, they can actually use digital proof for how to make a recommendation.

The last term is the term **analytics**. Analytics is a very broad concept, it relies on data very heavily and one of the best examples you're going to see of how to use analytics is actually in the form of how you interact with websites.

Imagine that you go on Facebook and occasionally you may notice that certain items on the page have been moved, so your suggested friends might be on the top right hand side for one day, and then it might be on the bottom left hand side the next day.

Well, what Facebook is doing is they're using analytics, so they're placing different components on the page and then they're tracking to see how you interact with those components. What they can do then is they can leverage your reaction, they can leverage the data, so that they can optimize their application so that people use it in the most efficient manner possible.

## 3.29 Personality Types and Technical Careers

Take a moment to assess your personality type and learn about areas you may enjoy in a technical field.

Go to [Personality Test](#)

<https://www.16personalities.com/free-personality-test>

Take the exam and use your results to find the technical career that may fit your current strengths and traits. Research completed by Evans Data Corporation. (n.d.). Retrieved July 18, 2018, from Evans Data and presented by Stephanie Conley.

Software Developer Personas, a guide to understand and finding the right software development career path for you.

### Full Stack Developers

Description:

**Persona: ENTJ, ESTJ, INTP.**

Let's see if we can make this work! Likes to be able to do more than one function or part of the job. Enjoys being hyper-productive and competes with self and others to be the go-to source for information in multiple areas.

A full stack developer is an engineer who can handle all the work of databases, servers, systems engineering, and clients. Needs to be proficient in a server-side language such as JAVA, PHP, C #, Python, Ruby. Also needs to know Javascript thoroughly and understand API's and how they connect to the front end. Needs to be proficient in frontend web languages and libraries like Bootstrap, HTML, CSS, SASS.

Front-End Developers

**Persona: ESTJs, ESTPs, ENFP's and ENTPs**

These are the Magicians of web development. These individuals are fascinated by human interaction and the driving force behind what we as humans do. Imagination is a key trait, they can take a product design concept or idea scribbled poorly on the back of a napkin or broken down in complete requirements and create an elegant app solution.

Description: Front end developers use JavaScript, HTML, CSS and other tools and libraries such as Bootstrap and SASS to bring the design concept to life.

They also work with other teams to understand how humans interact with technology and how to build software that we as people enjoy using. Example what you see when you look at Apple's homepage.



# Back-End Developer

Persona:

Most common personalities are **INTP, INTJ, ISTJ**. These developers live for logic and love the chance to exercise their conceptual power, analytical thinking, and ability to solve difficult problems.

Think of a master chess player. They love strategy, give them a challenge and they won't disappoint. Loves the more difficult technology, such as manipulating databases, creating REST API's, and working with the business logic portion of the application.

Description:

Very similar to the skills needed for full stack development. Back-end developers are responsible for building out the "client side" which is the internal workings or the "logic in web applications by using a server-side scripting language like Ruby. This is everything that you don't see before the data reaches your browser.

Quality Assurance

## Persona: QA- the INTP, ENTP, ISTJ.

This personality type loves to test limits, how far can I push before the software breaks. Loves to deconstruct projects to understand the internal workings and make improvements. They dive deep into research to solve problems. They can also see possible errors in software before the errors even occur. May even try to create problems to solve if they get bored they have an innate drive to solve problems. An Automated QA Engineer can develop, they simply choose to challenge other developers by breaking their code.

Description:

They test limits of software or web apps, debug and suggest improvements. They are responsible for assuring quality in the final product and for finding all the flaws before it goes public. This can be with manual testing where the user manually uses the program to find flaws. It can also be automated testing where the engineer writes code to do the work for them. Automation is much faster and more thorough. Automated Engineers also need to know how to code in various languages from server-side languages to front-end languages.

Technical Project Manager

## Persona: Project Manager- INFP, ENTPs, ENFJ's

These leaders work for max cooperation and is willing to stick their neck out on a chopping block on another's behalf. These people are organizational problem solvers and prefer to interact with people instead of code. Think of a coach.

Description:

Much like a coach the Technical Project Manager or Agile project manager manages the design and build of IT products and services. They are the proactive management that pushes the team to

complete tasks on schedule. They may also be trained in SCRUM and hold PMP and scrum certificates.

## IT Business Analyst

### **Persona: Business Analyst-INFJ's, INFJ's, ISTP's.**

Can easily spot the root of a problem and seem to find practical solutions to work through the problem. They enjoy the details and tend to prefer

over-communication. They often prefer to work behind the scenes instead of up front as a leader.

Description:

A technical business analyst works on the business side of an organization and facilitate process improvements within the company's information technology department. The business analyst works alongside QA, Software developers, design and Project Management to help the company become more effective by breaking down tasks into smaller more manageable tasks. It is quite common for a business analyst to certify in SCRUM.

Cyber Security

### **Persona: INTJ, ENTJ, ESTJ, ISTJ, INTP.**

Thou shall not pass! Think of intense, these personality types take their job quite seriously. Like secret service, they are very much sucked into their work and are methodical and very detail oriented. They want all the information and often dive into deep research.

Description: A Security Software Developer is a specialized type of Software Engineer. Also known as IT Security Software Developer, Security Developer, Cyber Developer. Their role is to prevent attacks through their expertise and knowledge of databases, networks, hardware, firewalls, and encryption. They keep computer systems running smoothly, prevent the theft of financial and personal information, and block intruders from accessing and divulging proprietary data.

## Machine Learning and Data Science

Persona:

Again we see a plethora of INTP, INTJ, ISTJ. and the love for logic and a deeper level of puzzle. This field allows a deeper dive and ability to exercise their conceptual power, analytical thinking, and ability to solve difficult problems.

Description: This is an overlapping field that covers machine learning, deep learning, AI, statistics, research, and applied mathematics. Machine learning covers statistical techniques to give computer systems the ability to "learn" with data, without being explicitly programmed. Data science is an interdisciplinary field of scientific methods, processes, algorithms, and systems to extract knowledge or insights from data in various forms, either structured or unstructured, similar to data mining.



## 3.40 What are Tech Careers?

Since you're going through this digital literacy course I'm assuming that you have some interest in getting into a tech career in some form or another. One of the most common misconceptions is that people think that the tech careers are pretty much all filled by developers. That is not the case. It is very helpful to have an understanding on programming, coding, and all of the various pieces that make up applications. But there is a vast number of careers and only some of those are related to writing code on a day in, day out basis. In this guide, we're going to walk through eight of the most popular careers in the tech space.

The very first one is to be a developer or a programmer. That's usually the one that most people think is related to a tech career and it is because it is very popular. A developer, programmer, could be somebody that writes back-end algorithms. It could be someone that implements front ends. It could be a full stack developer that builds every kind of application and works at every stage. But whenever it comes to being a developer or some type of programmer you need to be familiar with programming languages. You need to understand how applications work. Then, you need to be able to work in that environment.

The next career is a project manager. A project manager is someone who might have an understanding of coding and development. But they probably are not actually writing code on a day in, day out basis. Instead, they're managing a project. They are working with a team and they are setting the estimations. They are reporting to all the stakeholders. They're ensuring that the project gets built on time and functions properly.

The third career is a product manager. Now, there is some overlap between project managers and product managers. But the key difference is that you can think of product manager almost like the CEO of a specific project. This is something that is very popular and it's a certain methodology that certain companies follow, Facebook being one of the most famous ones. They say that their product managers are the CEOs of those specific set of projects. For example, if someone is a product manager of the Facebook News Feed, it is like they're the CEO of that feature on the application. They're going to coordinate with the advertising team. They're going to work with probably multiple development teams. They're going to ensure that their feature works properly.

The next career is a QA analyst. That stands for quality assurance analyst. This is typically somebody who will work with a development team. They will build out a set of tests and they will constantly check to verify that new features are one, working correctly, but also they're going to make sure that new features don't break old features that were already there in the system. A great example of this would be, and I do this with DevCamp. We actually have a QA team so that when one of the development team members pushes up a new feature, a member of the QA team will go through and they'll make sure that the new feature meets the requirements that it's supposed to do.

But then they also go back and they check to make sure that that new feature didn't accidentally break an old piece of functionality because that is a very common issue that can happen. You, as a developer, could be so focused on building out one feature you don't realize that some of the changes you made in the code actually broke something that used to work properly. A QA analyst is someone who verifies, one, the new features are working properly, but also that the system continues to work the way it's supposed to.

The next career is a data analyst. In large organizations, data analysts are usually individuals who have some level of programming expertise. They can perform tasks such as running database SQL queries or they might be able to automate a few tasks by building a Python script or a JavaScript script that can go and pull data off a website, run calculations. The end goal is that a data analyst should be able to take data, so whether it's data from a database or something from a website or anything like that, and they should be able to help the company make decisions based off of that data.

The next career is a software architect or software engineer. Now, this is usually not a career that you would go into right away. Typically, these roles are filled by individuals who were developers for a number of years, worked on many different applications. Then, they take all of that experience and they use that to help lead entire teams. A systems architect, and this is a role that I've filled multiple times, is where they are consulted before a single line of code has been written. They help decide the programming languages that get picked out, the type of database that is used, how the system is going to be deployed, and all of the various pieces that make up a software system. In many cases, this is usually a more senior developer and it's someone who can leverage all of their past experience to help make a efficient system.

The next career is what is called a DevOps engineer. This is short for developer operations. This individual is usually someone who enjoys working with operating systems. They will be the ones that will configure servers that the applications will run on. They'll build out all of the back-end systems and structure. So they're usually going to be the ones that are going to be interacting with the developers quite a bit because they have to know the requirements of the system.

The last career we're going to discuss is that of the UI/UX developer. Now, many times this is an individual who doesn't really write code. It would help if they have an understanding on how code works and they have a high-level view for all of those kinds of processes. But, usually, they're going to be the ones that describe the way that a user is going to interact with a system. Many times they can be designers but I don't want you to think that you have to be some type of creative genius in order to be a UI/UX developer. Instead, the best UI/UX devs that I've ever worked with were very detail oriented people. They could take a high-level concept and then they could diagram out all of the ways that a user would interact with a system. They'd help make that process as efficient as possible.

# 3.41 Tech Trends and Careers

Careers in technology-related fields are growing faster than jobs in nearly every other industry.

## Tech Trends and Careers

Exciting jobs in software development, digital graphic design, information management, and technology project management are just a few of the options. Explore the possibilities that could be in your future!

*Please be aware that some of these resources include information about specific companies. We do not promote and are not affiliated with these companies in any way. They do provide good foundational information that you may find helpful in learning these concepts which is why we include them.*

**Explore: Accenture, McKinsey, and Gartner are some of the technology think tanks that continually monitor the technology industry.**

Stay up-to-date by regularly visiting their websites.

[Accenture](#)

[McKinsey](#)

[Gartner](#)

For individuals in the United States, consult the [Bureau of Labor Statistics](#) for updated information on technology-related careers.

## You may also be interested in:

[US News. Best Technology Jobs](#)

# Module 4. Management Capabilities

## Index

- How to Discover and Manage Your Strengths
- Agile Project Management
- Learn management skills

## 4.1 How to Discover and Manage Your Strengths

In this lesson, we're going to talk about management. But management is such a broad term. Technically, you can't really learn management until you've walked through the ability to manage yourself.

So that's what this entire video is going to be about. It's going to be focused on managing yourself by finding and then managing your strengths. Specifically, we're going to walk through the feedback analysis process that was originally developed by Peter Drucker.

Now, the process that we're going to walk through is not something that you can complete by the end of this video, or even by the end of this program. This is a management framework for you to start thinking about discovering and then analyzing each one of your strengths.

So with this feedback analysis work flow, what it entails is first by coming up with a few items that you believe are your strengths. What you should do is you should write each one of those specific expectations down. I know that sounds very abstract. So, let's talk about a real world example.

If you'll get out a pen and paper and you write out a few of your goals. These need to be very practical, they shouldn't just be things like make more money, or be successful. They should be something like learn the JavaScript programming language so that I can build out an application, something like that.

Something that is measurable, something that's practical and also something that you feel like you can actually achieve. So what Drucker recommends that you do is you write down these list of action items, these expectations and then you work to try to achieve them.

You look back, so don't throw away the paper after you've written it down. You look back a year later and you look through that full list and you see which one of those actions was I able to successfully accomplish. After you consistently do this, this isn't an exercise that you just do today and then you forget about it.

You consistently follow this process of writing down expectations, taking action and then trying to achieve those goals. Within about two to three years, what you will have done is you'll be able to start noticing trends on which type of actions you're the most successful at and which ones maybe not so much.

For example, if you see a trend where whenever you write down an action or a goal where you want to learn something new and you notice in two to three years that you're consistently reaching that goal, you're consistently attaining it, that means that is a strength.



On the other hand, if you are writing down various goals and you realize that you're not actually achieving those, then that might be more of a weakness. What Drucker recommends is for you to work through this process until you've achieved a certain level of confidence on what your goals are.

Whether it's learning, whether it's project management, whether it's leading, whatever it is, by the end of those two to three years you should have a good idea of really what your strengths are.

And a lot of times, people aren't the best at knowing what their strengths are. That's the reason why he recommends this process is because you're actually using data. You're using real life analysis to see what things you're good at and what things might need some work.

What he recommends is to focus on the things that you're good at, to focus on your strengths because if you take someone who is very good at a certain topic, let's say it is building out front end applications. This is a digital literacy course, so we might as well focus on development.

If you notice that you're consistently hitting your goals and you're getting very good at JavaScript and React and these types of concepts and these skills, then it is much easier and more efficient to take someone who's very good at one thing and to get them to world class in that skill.

But, if you take something that maybe you've seen over a couple of years that you're not the best at, then it takes much more work and it's not as efficient to achieve those goals. That's really what the entire concept I want you to focus on with this process is, is to focus on your strengths because it is a much more efficient process to go from strength to even greater strength as opposed to weakness to strength.

## 4.2 Agile Project Management

In this guide, we're going to discuss Agile Project Management.

If you've never heard of what Agile is, it is essentially a set of values and beliefs on how to effectively manage software projects. One of the main goals of Agile and for teams that follow Agile is that **the processes that it allows for should allow a team to make more effective decisions as they manage and build out their applications.**

That may seem like common sense; however for years before Agile came around, software teams used all kinds of various project management processes. Many of them used the wrong kind of approach, they focused on the wrong topics and they spent too much time that really ended up wasted. And so what Agile came around to do was to provide a high level framework for how projects should be managed.

Now we said in the beginning that Agile is a set of values and beliefs about how to manage a project. That actually comes from the [Agile Manifesto](#) that is their official definition.

Let's walk through what each one of those Agile values are because that's going to help you understand the entire process. The first Agile value is that **we value individuals and communication over processes and tools.**

All that really means is that many times in organizations they spend so much time and resources on trying to build out these tools and these various processes, and they actually ignore if their team doesn't like using a certain set of tools.

I've come across this many times in different projects that I've worked on, where a team really didn't like using a set of software tools, whether it's a project management tool or something like that and it made them less efficient.

If you're really following Agile, you would get rid of that software or whatever process it is that's inhibiting the team, because you're valuing the team as a whole and those individuals, and you want to make it as efficient as possible for them to communicate.

The next value is that **we prioritize working code over comprehensive documentation.** There have been so many projects I've seen where their resources go into writing all of this complex documentation. What usually ends up happening is the system has to change so much that they are constantly updating that documentation and that becomes more of a priority as opposed to simply having a working application.

You still need to have proper documentation and a set of instructions for either how to set up or work with a software system. But you should never lose sight of the goal, that the most important thing is that the application works properly.

The third value is **customer collaboration over contract negotiation**. Years ago what the process was, whenever you were building out software, is you needed to spend an extensive period of time building out the full set of requirements. Then you would go back and forth with the client until they agreed to the exact specifications that you outlined.

The problem with that though is that there were always items that got missed. The process that I saw happen all too often was that you'd spend all this time up front planning out the project, listing all the requirements, and then as soon as something needed to change, then there would be a lot of wasted time where you'd have to go back and forth with the customer and say, "Okay, this wasn't in the original contract. How are we going to do it? How much money are you going to have to pay us in order to update it?", and things like that.

Whereas in the Agile approach, contract negotiation is less important, whereas customer collaboration is more important. You're going back constantly, usually on a weekly or every other week basis to ensure that the system is being built properly.

The fourth Agile value is **responding to change over sticking to a plan**. What that means is as you go out and you build these applications, I can promise you because I've been doing this for over 15 years; I have never had one application where we got all of the requirements perfectly set up in the very beginning.

Software systems need to change on a dynamic basis. This is one of the key reasons why Agile was created, was because people years ago tried to set up all the requirements from day one and then it'd become incredibly frustrating because the application would need to change.

The customer would realize that they forgot a really important set of features that was not put in place there in the very beginning. Then it would become very difficult to change the system because everything was already locked in place from the beginning.

What Agile does is it allows you to be much more dynamic. That's really where the name comes from, it allows you to be agile. So as you're building out the software system, you're able to make more dynamic changes and you're able to be more flexible as you build it out.

## 4.3 Learn management skills

In this lesson, we're going to take a high-level overview of some of the best ways to learn management skills.

Now, this course is not specifically one on management, it's a digital literacy course. But it's still important to understand how management should work, especially as it relates to any kind of digital projects or to technology companies in general.

If you find yourself in a management position where you're working with a team of developers, whether it's project management, product management, or anything like that, one of my top pieces of advice is to focus on the developer's perspective.

Any time that I've seen a breakdown between management and a development team, which also usually leads to a breakdown and failure of the project, it usually came from the management team not properly looking at things from a developer's perspective. They simply saw some type of goal or project, and they didn't communicate properly with the team.

If you remember back to when we talked about the different agile values, remember how communication was one of the top priorities? Whenever you find yourself in a position of management, that is a paramount requirement, it's that you're constantly communicating with the development team.

You're getting feedback from them for new features, for how to fix certain bugs, for anything like that. I promise you the more that you communicate with that team and the more you get feedback from them, the more effective you're going to be as a manager.

When it comes to learning new management skills, make sure that you are also reading constantly. I was really surprised to learn this, but in Fortune 500 companies, a survey was done and the average number of books that these Fortune 500 company CEOs read was a hundred books a year.

Some of the best CEOs and managers always prioritize reading and learning, and so if you're doing that, not only are you feeding yourself a new information, which is always good as long as it's good quality information, but even more importantly what you're doing is you're learning from someone else's experience.

So if someone else has written a book on management, they typically spent years in learning and failing and doing things like that. So if you're able to read that book, you can learn from their experience as opposed to having to learn from your own failures and your own experiences.

# Module 5. Team-working

## Index

- What is teamwork?
- Scrum vs Kanban

# 5.1 What is teamwork?

In this guide, we're going to talk about some practical ways that you can improve and harness teamwork in collaboration, especially as it relates to a development team.

First and foremost, one of the most important things to understand is that if you have a development team, whether you're involved in one, say you're a developer, programmer, or a QA team member. Or if you're managing one. So, if you're a project/product manager, or even a CEO, then it's very important to understand what your team isn't like.

Many times I've heard of development teams and a manager talking about them and treating them almost kind of like a swarm. So, like a worker bee or a swarm of bees, or ants, or something like that. They try to create an analogy between their team and nature like that. And I've discovered that is not even close to being the case.

Development teams, and really, teams in general, are filled with unique team members. They all have their own sets of strengths and weaknesses, and if you do it right, you can utilize each one of those team members and their unique strengths and weaknesses in order to come together and create a very well functioning project. But if you try to treat each one of the members as just one cog in the team, then you're going to end up running into a number of issues.

And I know that may sound a little bit abstract, so let's kind of dive into a real world example. Say you're managing a team of five developers, and you're building out a full application. So, you're going to need a back end, you're going to need a front end, you're going to need a database, and design elements.

Well, if you try to treat the team just like a swarm of bees or like ants, you would simply list off each one of the tasks and then you would randomly assign it in thinking that you're just going to get the project done that way. But in reality, you really need to pay attention to the strengths of each one of your team members.

Find the ones that love front end, find the ones that love engineering or architecting an entire system. Find the things that they love, and also that they do well in, and then assign the tasks that way.

Now, I know that may sound like common sense, however, I've been doing this for a number of years, and all too often I speak with different team members when I come in and I get added to a development team, and I discover that they've been given tasks that either they have very little experience in, or they simply do not like at all and that usually ends up in bugs and in a poor final project.

So, make sure you're always communicating with your team. One of the best ways of ensuring project success is by communicating, having a transparent way of understanding when your

development team may be having issues, when they may be doing something they don't enjoy, and that way you're able to adjust and hopefully catch something before it becomes an issue.

And lastly, when it comes to teamwork, make sure that everyone on your team has a freedom to embrace failure. One of the top slogans at Facebook very early on was Fail Fast and Fail Often. Now they didn't mean that they wanted the entire project to be a failure but, they weren't scared of failure on a day in, day out basis.

Let me give you an example, and this is something that happened pretty recently with a developer that I was managing. This developer was very talented, loved development, loved building out applications.

But I started to notice a trend, and so did a few of the other team members in this group, where we would give him a specific task, and then he would simply ignore it, and, obviously, that's not a good thing.

And so I started to dive into it and communicate with him on why that was happening, 'cause it started to happen a few times, and it was causing an issue. And it turned out that if we gave him a task that he didn't know how to build, he didn't know how to implement, he would simply procrastinate on it.

So, instead of feeling like he could come to one of us and simply say that he doesn't know how to do that, he was putting it off to the side and procrastinating. So, this is something where I needed to have an entire meeting with the team, and say how that is not the attitude we want to have.

It is perfectly fine to not understand a certain concept, or not know how to build out a feature. I've been doing this for a while, there are still constantly features that I come into that I have no idea how to build. And so, I have to ask others around me that maybe have more experience in that specific field.

And so, whenever you're managing a team or working with team members, make sure that they feel comfortable enough so that they can come to you when they don't know how to build something.

That's going to help make the entire development process much more transparent, and everyone's going to be able to feel comfortable, even when they may not know something, and it's going to make it so they're not scared of failure, which at the end of the day, is going to move the entire project along much faster.

## 5.2 Scrum vs Kanban

So far in this course, we've spoken quite a bit about agile project management, but remember that agile is a set of values and beliefs.

So, essentially, it is a high-level set of rules and a mindset that you want to have when you're building out an application, and as important as it is to have that mindset, we also need to now take it a little bit further.

We need to go more low-level and see what does it look like to manage a project using agile methodologies, and so to do that, we're going to analyze two of the most popular implementations. They are more of real-world day-to-day practices that you're going to use. One is called **Scrum** and the other is called **Kanban**.

Both of these approaches are very extensive with all kinds of rules and best practices, so there's no way that we could go over all of them in a short video, and in fact, there are specific certifications where you could spend months learning each one of these approaches.

So, the goal of this guide is simply going to be to give a high-level overview so that when you hear about them in the future if you want to go into project management, then you'll know how they work.

Let's start with Scrum. Scrum is a project management methodology that has a few key unique characteristics. So, for one thing, a Scrum approach means that you break down your new features into what are called **sprints**. Now, these sprints are usually two to three-week iterations, so the entire team will focus on a new feature that you want to implement.

So, say that you're building out some kind of social media application and the next feature is to have the ability to follow friends such as on Facebook or Instagram. In a Scrum approach, the entire team would be focused for those two weeks for that entire sprint to take that feature all the way from the idea to where it's deployed in the application.

Now, a key with Scrum is that by the end of that sprint, there is a fully functioning feature. Another concept that is very popular when you're building out a project and using the Scrum methodology is to have what are called **stand-ups**.

So, remember with agile and the agile management process, communication is one of the biggest keys. Well, with Scrum, you have what is called a daily stand-up, which means that at the very beginning of the day, so right when you get in office usually, the entire team gets around, and everyone talks about a few key concepts.

They talk about what they did yesterday for the feature, they talk about what they're going to do today, and then they also bring up any problems that they might have had.



Now, I find this approach incredibly helpful because, one, it keeps everybody honest. So, if somebody did not work every well the day before, if they were lazy, it's going to become very apparent to everyone else that they didn't do any work, so it keeps everyone going and motivated to work. The other thing I really like about this is it helps identify problems and bugs very quickly.

So, one thing that I've seen happen quite a bit in projects that didn't use Scrum or this kind of process is that if they have some piece of data they need, maybe it's communicating with an outside server, if they go a few days without asking for access to that server, then the entire two weeks could go by, and then at the very end, everyone gets in trouble because they didn't finish the feature because they were waiting for permission to access that server or access that third-party API.

With Scrum, in these daily stand-ups, what happens is on day one, whenever that developer needs access to that data, at the very end of their little meeting, they say, "Okay. This is what I did yesterday. This is what I'm doing today. By the way, I can't keep working unless I get access to that data." So, right away, the project manager can go out and get them that access, and it doesn't slow down the rest of the project.

So, now that we've talked about a few key characteristics of Scrum, now let's talk about Kanban. Now, Kanban is a project development process. It was popularized by a number of Japanese and Asian companies that were trying to make their manufacturing processes more efficient, and so the software development people took those concepts and applied them to being able to manage a software application process.

In Kanban, you don't have two-week sprints. Instead, you take each individual feature, and you assign it to a developer, and you have them take it through stages, so they will have what is called a backlog, which means that it's a feature that needs to be built, but it has not been started yet. That will get assigned to a developer, they will take a card, they will move it into a work in progress. It's also called W.I.P for short. It takes it to the work in progress stage.

From there, after it's completed, the developer will say, "It is completed. It's now ready for QA testing, for Quality Assurance testing." After the QA team member has gone through the entire feature and they ensure that it's working properly, then they will move it to completed, and so that is the standard Kanban process.

Through the years, I've utilized both of these approaches on various projects. I don't really think that one is better than the other. I've had success and failures using both. It really comes down to, at the end of the day, how well they're implemented, how closely the developers are following those processes, and once again, how clear the communication is between all of the team members.