Things I wish I had known before I started my first development job :)

By Sujan Kapadia

Take a deep breath!

CHARIOT SOLUTIONS

Starting a new job, joining a new project, or entering a boot camp can be nerve wracking. If you're changing careers, even more so. There are a ton of things to digest, but I'm here to tell you it's okay to breathe. Learning is a journey. What follows are a few things I wish someone had told me as an engineer, before I started my first job.

Disclaimer: This presentation contains no code :)

Don’t be afraid to ask questions!

- Ask Five Whys? - Get to the root cause.
- Two books I highly recommend:
  - Super Thinking: The Big Book of Mental Models by Gabriel Weinberg and Lauren McCann
  - Thinking, Fast and Slow by Daniel Kahneman

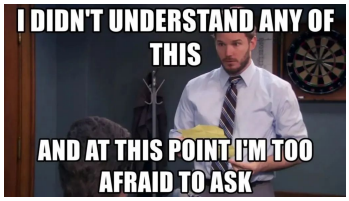
CHARIOT SOLUTIONS

Number one mistake I see even from experienced engineers: not asking questions, even basic ones, like "what does this mean?", "why is this important?", "why is it done this way?"

- If you can’t restate the problem you’re working on, it can be hard to find a solution.
- Good teammates speak up. If you're spinning wheels too long, it’s okay reach out for help.
- Ask Five Whys? Get to the root cause.
  - https://en.wikipedia.org/wiki/Five_whys
  - It's okay to question why. Early in my career, I was afraid to do this. I noticed the best engineers around me frequently asked why and questioned whether the approach made sense. They ended up excelling and were highly productive.
- A trick that works for me: Write the question out in a draft email or IM. The anxiety of clicking Send gets my mind to start thinking of solutions or places to find them.
- Try to answer the question yourself first - your question may change, or you may gather supporting information to help.
- There's a balance between DIY / do your own research AND asking questions.
  - Collect the questions and answers - it can help with onboarding new

- teammates, You may become a source of information for others.
- You're not slowing down your team. If anything, it's the opposite.
- Super Thinking: The Big Book of Mental Models by Gabriel Weinberg and Lauren McCann , Thinking, Fast and Slow by Daniel Kahneman

There's a lot of tribal knowledge in our industry



- You feel like everything you learned doesn't apply!
- Cornell Note Taking Method and The Best Way To Take Notes
- Software development is gathering, managing, and sharing knowledge.
- **IT TAKES TIME, AND THAT'S OKAY!**

CHARIOT SOLUTIONS

- You attend your first stand up on a new team and it sounds like a foreign language.
- You may feel like everything you learned doesn't apply, you have to relearn everything, or everyone around you "sounds" smarter.
    - This generally happens at the beginning of every project.
    - It'll feel disorienting until you get a handle on a specific piece.
    - Reach out and ask for help. I wish more companies assigned "buddies" early on. If you don't get one, ask for one.
- Start keeping a list of terms and questions, then share with the team. Ask them to edit and contribute.
    - A lot of places don't have comprehensive or up to date documentation.
    - You'll be leading by example, and will likely be recognized for it.
- Take notes, study, treat it like a class. Y'all are good at that😉
    - Cornell Note Taking Method
        - https://lsc.cornell.edu/how-to-study/taking-notes/cornell-note-taking-system/
        - https://medium.goodnotes.com/study-with-ease-the-best-way-to-take-notes-2749a3e8297b
- IT TAKES TIME, AND THAT'S OKAY.
    - A lot of software development is gathering, managing, and sharing knowledge.

Ask for a roadmap - where does your feature fit in?

- The “big picture” - it’s motivating and inspiring
- User Stories!
  - https://en.wikipedia.org/wiki/User_story
  - https://www.mountaingoatsoftware.com/agile/user-stories

CHARIOT SOLUTIONS

- **Ask for a roadmap / bigger picture - where does your feature fit in?**
  - It helps you understand the requirements more deeply
  - When something changes, or you have an idea for a new feature, you can question whether it makes sense or not.
  - It's motivating and inspiring
  - I try to always follow the "user story" format for features: As a <user/customer/etc>, I want <some goal>, so that <some reason>
  - It helps me think about what a feature should really do, how it fits within the product, and whether it makes sense in the big picture
  - This is a very simple, yet powerful structure. If you leave any part out, it's incomplete and dangling...
  - You will quickly gain a better understanding of the users, the product, the market and stand out if you think about features this way.
  - https://en.wikipedia.org/wiki/User_story
  - https://www.mountaingoatsoftware.com/agile/user-stories

Make a learning plan: Be honest about the things you don't know.

- How to create a personal learning plan

CHARIOT SOLUTIONS

- **Make a learning plan: Be honest about the things you don't know.**
  - Keep this in plain sight.
  - Prioritize by what your team + lead/manager feel are useful to get the current job done + your career goals.
  - Review it with your manager (annually, biannually, quarterly)
  - https://www.mindtools.com/pages/article/personal-learning-plan.htm

Learning never stops!

The best engineers spend time outside of work learning a domain or new technology.

Kent Beck's "Background Work"

CHARIOT SOLUTIONS

- **Learning never stops: The best engineers do spend time outside of work learning a domain or new technology.**
  - Pick one area and focus on it - learn it deeply
  - Kent Beck, one of the original founders of the agile movement, recently posted an article on engineer - https://medium.com/@kentbeck_7670/background-work-dce930c0675a
    - **"**"Background Work" is the work you do over and above what is strictly required to complete a task. It may be done for learning, to satisfy curiosity, or just because you forgot to stop working when you were done. I've noticed that the most accomplished programmers make a habit of background work."
    - You want to peek under the hood to understand how something truly works. That will help you make better technical decisions, and be able to apply similar techniques to different situations.

Learn how to build software locally

- Except for an initial download of packages and getting the latest code, you shouldn't need a persistent internet connection.
- **You can work from the beach!**
- A great local development environment is not a nice to have, but a must have

CHARIOT SOLUTIONS

- **Learn how to build the software locally**
  - So what does this mean?
  - It means you can build and test most of your components locally. Except for an initial download of packages and getting the latest code, you shouldn't need an internet connection.
  - **You can work from the beach :)**
  - A local development environment allows you to iterate faster between testing and coding. Everything you depend on can be quickly set up and tore down on your machine.
  - Unintended side effect: You learn an insane amount of what your application depends on - databases, services, test data, and configuration
  - If you depend on an external service that is hard, expensive, or time-consuming to use, create a mock version.
    - There are plenty of tools out there to easily stand up mock REST (HTTP) servers with canned requests and responses
  - In fact, you may have already learned how to bootstrap your own application and dependencies at the bootcamp. I think this is where bootcamps surpass what is taught in a traditional college.
  - https://levelup.gitconnected.com/a-great-local-development-environment-is-not-a-nice-to-have-but-a-must-have-ed678ba4c8ed

Logging is your number one friend

- Log files are your “flight data recorder”
- At its core, you want to log (record) important data that will help you identify a problem.
- Logs should be structured - treat them as data.
- This captures some good principles - The 10 Commandments of Logging

CHARIOT SOLUTIONS

- **Logging is your number one friend. Learn how to produce and read log files - they are your "flight data recorder" - but you should always open it and keep it open!**
  - At its core, you want to log (record) important data that will help you identify a problem. That data includes timestamps, actions, events, inputs and outputs (appropriately masked or omitted if they are sensitive data)...
  - Log files help answer who, what, where, when, and why?
  - A log file can reveal a lot about what happened and help you understand where a problem may lie
  - Many times, you don't have access to the production environment or it's expensive to access it, so log files are the only recourse.
  - Logs should be structured - treat them as data. For example, a JSON object. This allows you to search, query, and filter against them, build dashboards to help monitor,
  - Modern software consists of many moving parts. There's just no way to understand everything or keep it in your head.
  - This captures some good principles - https://www.sentinelone.com/blog/the-10-commandments-of-logging/
  - If there is little to no logging, START adding it right away. Your software will quickly outgrow your ability to keep it all in your head.

- Start with a minimum amount, and add more as needed.
- Think about your audience when writing log messages. Who will be reading it? Developers? DevOps / SRE teams? First level support? Is it being used to build monitoring dashboards?

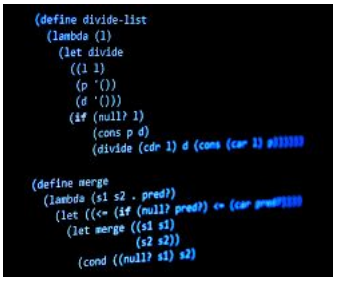
Write unit tests and open up a debugger

- ***Hyper local*** development environment
- You become familiar with both the code ***and data***
- Effective Unit Testing and Test Driven by Lasse Koskela
- Open a debugger before tapping on your coworker's shoulder!

CHARIOT SOLUTIONS

- **Write a unit test and open up a debugger**
  - This is very similar to building a local development environment, except it's "hyper local" - you're building a mini environment for a specific feature, that is cheap/fast to set up, run, and tear down. **One click from your IDE.**
  - Just as before, you quickly learn what dependencies, data, and configuration are actually needed
  - You become familiar with both the code *and data* - I can't stress this enough.
  - It's a LOT easier to try out changes when you have a safety harness
  - Effective Unit Testing and Test Driven by Lasse Koskela
  - Just as important as logging is knowing how to use a debugger for your language / platform.
    - A debugger allows you to step execute a thread of code, one line at a time. You can inspect the call stack, local variables, and heap (memory) data structures and objects
    - You become intimately familiar with how particular sections of code behave
    - You can run a unit test via a debugger as well. I've done this a number of times to learn new code bases.

Initially focus on functions and data



- Instead of classes, interfaces, and services
- You can iterate faster - “hack” away!
- Focus on function, rather than form.
  - The form is necessary, because you are almost always writing code others will use, but don't let that derail you.
- Write throw away code for the sake of learning.
- Take a look at functional programming.

CHARIOT SOLUTIONS

- **Initially focus on functions and data instead of classes, interfaces, and services**
  - Write an empty unit test and try to prototype the basic logic, mocking any external dependencies.
  - If you're working with a dynamic language or one that has a REPL, just start hacking away! :)
  - Focus on what needs to happen to the data (function), and how it needs to be used (data format).
  - **Focus on function rather than form.** (The form is necessary because you are almost always writing code others will use, but don't let that derail you)
  - You can take this code and then refactor into a class, service, and interface once you have a better understanding of what needs to happen, and who needs what.
  - It's easy to fall into the trap of spending countless hours thinking about class hierarchies, method names, and design patterns. This can turn into long debates with other developers and architects and you end up producing documentation rather than code.
  - If you want to take this to the extreme, look into pure functional programming. It's no silver bullet, but it helps you look at problems from a different angle

        - https://en.wikipedia.org/wiki/Functional_programming
        - https://www.infoworld.com/article/3613715/what-is-functional-programming-a-practical-guide.html
    - In general, you are trying to learn more about the problem first, then come up with a solution. **Writing code for the sake of learning is absolutely okay, and what great engineers do.** This code can be then "thrown away", which brings me to my next point:

Don’t be afraid to DELETE code

- Sunk Cost Fallacy
- Delete Old Code
- Pros of Deleting Unused Code

CHARIOT SOLUTIONS

- **Don't be afraid to delete code**
  - https://www.reddit.com/r/ProgrammerHumor/comments/lo3ku7/that_part_of_code_that_im_scared_to_delete/
  - It's really hard to delete or rewrite code you may have put a significant investment in
  - Sometimes you need to start from scratch, or the code is no longer needed because a requirement has changed.
  - Keeping it around means maintenance, potential misuse or copy/paste, security issues, and misleading new developers who may think it's being used.
  - You likely learned a lot the first time you wrote it, so you may have a much better implementation now.
  - In a broader context, this is an example of the Sunk Cost Fallacy - you'll encounter this many times as an engineer
  - https://builtin.com/software-engineering-perspectives/delete-old-dead-code-braintree
  - https://stackoverflow.com/questions/15699995/could-someone-explain-the-pros-of-deleting-or-keeping-unused-code
  - Sometimes what you work on does not get used right away, or the original problem changes. While this can be disheartening, I still learned a lot, and it motivated me to get better at asking more

- questions upfront before taking on a task.

Don't get caught up in the "full stack" hype

- Focus on an area you are passionate about (or an area you feel the team is currently lacking), and spend time getting good at it.
- FOMO is a real thing
  - Early on, I worried that I had to dive deeply in every single tech that I heard about or my team worked on.
  - How to FOMO

CHARIOT SOLUTIONS

- **Don't get caught up or afraid of the "full stack" hype**
  - You see this in a lot of job titles and descriptions.
  - It is possible to become an expert in different parts of the stack over your career.
  - It is extremely difficult to remain an expert in every part of the stack over your career, and even more so simultaneously.
  - Early on, I worried that I had to dive deeply in every single tech that I heard about or my team worked on.
  - MAJOR FOMO
  - The stack has grown over to time also include:
    - DevOps / Cloud Infrastructure
    - Data Pipelines / Data Engineering
  - Focus on an area you are passionate about (or an area you feel the team is currently lacking), and spend time getting good at it.
  - How to FOMO - nice write up from the perspective of a developer

Learn how to effectively summarize

- We live in an age of information overload.
- Summarizing requires a deeper understanding of the problem, and how it impacts the audience you are writing for.
- It's a skill that takes time to develop, but at the end of the day, it makes you a better engineer!
- How to write a problem statement

CHARIOT SOLUTIONS

- **Learn how to effectively summarize**
  - A significant part of an engineer's job involves communicating status, requirements, issues, etc. to teams and management
  - Details are critical, but beginning with a summary helps someone determine whether it impacts them, what decision needs to be made, and whether they need to read further
  - **We live in an age of information overload.**
  - Earlier in my career, enough of my emails went unread or the point I was making was buried deep in text, which no one got to :)
  - **Summarizing requires a deeper understanding of the problem, and how it impacts the audience you are writing for.**
  - **What is the problem, who does it impact, and what is the next step?**
  - In some cases as you are writing the summary, you find a solution, or determine the problem is not actually a problem.
  - It is time consuming, and sometimes it's frustrating. It's a skill that takes time to develop, but at the end of the day, it makes you a better engineer!
  - https://www.indeed.com/career-advice/career-development/how-to-write-a-problem-statement

Criticize ideas, not people!

- Healthy teams take time to listen to each other, but also have a way to evaluate which ideas to pursue, and which ones to "backlog".
- How to give criticism without sounding like a jerk

CHARIOT SOLUTIONS

- **Criticize ideas, not people**
  - We get paid to solve problems, and are constantly generating new ideas in this quest.
  - Not every idea makes sense or can be applied to the current problem.
  - BUT you want to be in and encourage an environment where ideas can be freely discussed and exchanged.
  - Healthy teams take time to listen to each other, but also have a way to evaluate which ideas to pursue, and which ones to "backlog".
  - When you are intensely focused on a problem, you can become emotionally attached. That is part of being passionate about what you do, but doesn't make it okay to criticize others.
  - https://www.inc.com/jayson-demers/the-7-golden-rules-of-how-to-give-criticism-without-sounding-like-a-jerk.html
-

You will be asked to provide estimates :(

- This is not easy to do unless you or the team has worked on something similar before.
- Build the smallest possible thing that works first.
- Start with T-shirt sizing - it's effective for backlog planning
- Delivering smaller features in a steady release cadence builds trust and confidence more so than generating estimates.
- I am a big fan of the eXtreme Programming (XP) agile approach
- How to handle difficult clients / customers / stakeholders

CHARIOT SOLUTIONS

- **You will be asked to provide estimates**
  - This is not easy to do unless you or the team has worked on something similar before.
  - You may be using a language, library, or framework you don't have experience in.
  - A single requirement can end up being hundreds or thousands of lines of code.
  - You may be dependent on other people or teams, and unforeseen issues will occur.
  - In every case, first build the smallest possible thing that works first. As you learn more, you will gain a better idea of where you are, and can update the overall estimates.
  - I am a big fan of the eXtreme Programming (XP) agile approach to building software - it focus on small releases, developing in iterations, integrating early and often, building the smallest thing possible, test driven development, and frequent communication with stakeholders.
  - Delivering smaller features in a steady release cadence builds trust and confidence more so than generating estimates.
  - One approach is using t-shirt sizes for effort and complexity.
  - https://zimtik.com/en/posts/how-to-handle-difficult-clients - succinct descriptions on de-escalation and scope/timeline negotiation

- https://www.redagile.com/post/tshirt-sizing
- Solicit estimates from each of the developers on your team. Ask them to provide a low, medium, and high estimate.
    i. Then get together to discuss these and agree on a range for each.
    ii. The low number is if things end up being simpler than expected. The high number is if it's more complicated.
    iii. At the end, add a "buffer" that takes known and unknown events into account - vacation time, sick time

-

We tend to blow things out of proportion

- It may feel like the sky is falling and you end up panicking.
- This will happen at least once on each project, so plan for it.
- **BUT *when is the last time it mattered a year later?***
- Instead, capture lessons learned when a problem arises and openly discuss risks.
- **Test, test, test!**
- **Log, log, log!**
- Do not ignore technical debt.

CHARIOT SOLUTIONS

The recent AWS us-east-1 outage wreaked havoc on many businesses of all sizes. Roomba vacuums even stopped working!

Technical debt - When you prioritize speed over quality code/design/architecture, you end up accumulating this debt. You're choosing the easy road now, instead of a better approach, that would take longer.

There are different types of technical debt.

- One is related to how the code is organized. You may end up writing a feature within an existing function, class or module to save time. However, the longer term and better solution may be to put this feature into its own service - this could involve moving more code, data, configuration and deployment. That may take a lot of time and can't be done within the current deadline.
- Another is upgrading dependencies (e.g. libraries, frameworks, utilities) - a newer version may be out that has performance improvements, security fixes, and more features. The longer you wait to update, the harder it may become if it impacts large parts of the codebase.
- Another is what algorithm is chosen - you may choose a simple to implement one at the expense of performance (time, memory, disk space) in order to ship a feature fast. However, this algorithm may not scale up to more data.
- Keep track of technical debt, bring it up at standups and planning sessions when discussing feature implementations. Make it visible and known. Ideally, prioritizing and addressing it can be part of each iteration.

- Try to provide a business case for each technical debt / refactoring item. If there is not a driving business need, it may not be as important as you think, and it will be harder to persuade management of.

Sleep is truly underrated!

- Sleep is literally a superpower - watch this TED talk!
- Everyone has different patterns, but waking up rested and rejuvenated makes me a better engineer (and person)
- A key component to productivity and creativity.
- There will be late nights, but don't make it a habit.
- **It's the closest thing to a reboot button for humans :)**

CHARIOT SOLUTIONS

Sometimes it's hard to step away from the keyboard when you're stuck on a problem or creating something new, put down a good book, or fight the urge to hit next episode on Netflix

Questions?

CHARIOT SOLUTIONS