By: Tyler Goodburn From Entelect

Expected to grow/learn, not expected to know everything at first. Ask questions, be a team player. You only get the help you **ask** for.

In industry, systems can be very old/outdated (5-15+ years). This can be difficult if you aren’t used to old tech.  
  
**Why its old?**Would need 20 people + 2 years to update something that works, but the update wont change anything/make more money.

Hacking not like movies, but more scary: Session hijacking (Linus)

Good trait: Be open to learning from anyone, even juniors.

If you don’t ask others for help, stress/pressure can build up too much. Takes long time to recover from.

**Large** vs **Small** teams

Small: You know everyone personally, can give help more easily. Less is done.

Larger: You get to point where you can not talk to people in a month, or see everyone each day. Must make sure to communicate.

**Time management** is a big thing. Know when you can relax, and must work hard.

**Expectation management:** You have a month to do something, and there’s an expectation you will be done in that time. If something happens, eg Laptop stolen, you must speak to that expected person if can reduce work, extensions ect.

But you have to do this in advance, not on the day.

Ideal:  
Testing 40%  
Planning 40%  
Coding 20%  
  
But in reality normally coding is 40%.

**Pair + Mob** programming

Someone else may know what you are struggling with well, and vise versa. (Pair programming, 2 people) Very good for learning.

Mob: 3-5 people.   
1 person just typing, not thinking.   
1 person is in control saying use x variable, do registration page.  
1+ person is the observer, listening, asking questions, catching errors.  
Benefit: 5 people all learning at same time.  
Disadvantage: You have 4 people not writing code. In reality where deadlines exist, finishing a project without 5 way code split is difficult.

Everyone is different, don’t compare yourselves.

**Recent trends (NB)**

PWAs: Progressive web app. Instead of downloading app, develop website and you just have a link “app”. Takes up a lot less space. Don’t have to make an app + website.

Monoliths and micro services:   
-monoliths = 60+ projects in it. Any changes to one project, means many things have to be turned off and on as they rely on each other,  
-Micro services: each project are deployed by itself. May have to be running 6+ things simultaneously is the downside.

Scaling: Some apps need to have millions, some don’t. Must look at where in 5 years how much the user base will be.

Containerisation: Good for scalability. Wrap something to new platform eg Windows.

UX is NB: As this is what user interacts with. There is now a UX job where you just look at trends of what looks good, what is good User interaction ect.

**Don’t** make an app where a back button logs you out.

**Continuing Trends:**

Moving to cloud: Many things are moving to it

Continuous deployments and delivery: The quicker you can get your work out, the quicker you can deliver value. Amazon does 130k a year. They do microservices. Stuff is never taken down. Many smaller packages. Called **blue green deployments**

Cyber Security and Ransomware: Always very important and must be careful. Backups are ultimate thing must have.

Automate all things: if you can automate something, do so. But only if you wont spend more time automating than just doing it manually.  
-Stuff that auto documents stuff is useful

AI: You must see it as a tool, something you can use to better yourself/shorten a task. Don’t use it as a crutch though.

Usual day: Checking emails, 2 hrs dev work, stand up (everyone talks about did yesterday, what busy with, what blocking you) [must be quick, 15min in total], PR reviews, deploy to test environment (QA), testing + verifying, deployment.

Retros: What went well, what went bad, what can be improved. This can be done after a spring.

**Misconceptions of what its like to be a software engineer**

You are not coding 100%, alone in a room. You are actually talking to people. When given requirements for a project, you have to know why those are needed/wanted. Communication is very important.

**Team Ecosystem:**

You  
Team  
Product owners: Fight for their requirements to be met.  
Customers: end users  
UX: sometimes have a dedicated UX person  
BA: Business analysts. Plan 6-12+ months ahead for requirements  
Team Lead: Shield you from product owners. Must guide team in correct direction.  
Testers/QA: Not for you to just throw errors at that you don’t want to fix.

**Different hats:** You wont always have the same role. You can be the dev, tester ect. Don’t lock yourself in a set role.

**Skills needed to be a software engineer**

Critical thinking

Problem solving

Understanding of fundamentals

Good communication skills

Teamwork

Attention to details

**Skills needed to be a good software engineer**

Knowledge of good coding principles and why they are used: Being able to explain why they’re good

Critical thinking

Delivering quality consistently: You will be questioned less if you consistently give good work.

Time management: The deadline doesn’t move. Make compensations in advance (move more people to project team)

Ability to upskill: You wont know everything, but you must be able to learn

Troubleshooting

Expectation Management

Ability to handle stress and pressure

Ability to communicate and interact with people

Being a team plater

Food work ethic

Have a positive attitude

Commitment to learning

Ability to ask for help

Balancing business tech: Balance time on updates that don’t add value, but help with newer things/new system.

Want to know more?

Entelet

Glassdoor

Offerzone

Podcast: software daily

Other: C# Digest, what’s going on in industry  
Troy hunt: covers software breaches

Entelect has a 6 week bootcamp for newcomers to make sure up to scruff. Must be willing to learn.

Backups: Finance, every transaction must be immediately backed up. Cooking recipes, can backup irregularly.