Agile:

* General Principles
  + Face to face communication emphasized
  + Prioritize people over process/tools.
  + Allows you to adapt between iterations.
  + Write stories to describe customers’ needs
  + Self-organising teams – trust
  + Always improve – hence feedback wall
  + Don’t just think that story cards at the end all, constantly ask your Product Owners what they really want.
  + Push back if your PO is a jerk and wants unrealistic goals.
* Core Practices
  + Story Card
    - Placeholder to remember what you are talking about (physical reminder of work)
    - User focussed
    - Narrative e.g. As a <user> I want <goal> so that <value>
    - Acceptance criteria e.g. given that <condition> when <trigger> then outcome.
  + Estimation
    - Size of story relative to others, NOT MEASURED IN TIME but in complexity, hardness.
    - Not a promise, but best guess
    - Way of thinking about how hard things are
    - In Fibonacci scale or T shirt size scale
  + The Wall
    - Shows work to be done
      * Backlog
      * Analysis
      * Ready for Development
      * In Development
      * Test
      * Sign Off
      * Done
    - Can have less steps than this
* The Stand up
  + Short standing meeting (15 min) to talk about where people are at, and if they need help.
* The Show Case
  + Demonstrate to Stakeholders
  + No surprises, stakeholders always know what developers are working on.
* The Retrospective
  + Loop back to see how team could improve
  + What went well?
  + What didn’t go well?
  + What confused us?
  + How can we keep the good stuff and get rid of the bad?

Security:

* Is to maintain acceptable risk
* Risk = impact x likelihood
* Impact = how much it affects
* Likelihood = probability
* Residual risk = risk – controls
* Thread Modelling
  + List all possible threats
  + Build an attack tree <https://en.wikipedia.org/wiki/Attack_tree>
* Everyone on the agile team needs to be responsible for implementing secure systems, rather than leave it to a team.

Test Driven Development

* Write tests first, before writing your code.
* Which brings specifications to front of mind, rather than realising that you missed the mark and have to redo a whole month’s of code.
* Look this up because even though it seems like it adds a lot of time, it actually saves a lot more. Pretty much standard in industry, MUST KNOW!
* Look up SOLID Object Oriented Principles.
* TDD Mantra
  + Red failing test
  + Green minimum amount of code to pass
  + Refactor – clean up and repeat
* Test Pyramid
  + Top and Least: End 2 End
  + Middle and Moderate: Integration
  + Bottom and Most Written: Unit
* Also provides automated live documentation through test results.

U.X. Design

* Overall experience person has interacting with product
* Process of improving usability and interaction
* UI user interface
  + Facilitates and communicates user experience
  + User aesthetic
  + Interacting tools humans use for product, so e.g. Product may be cereal and milk, but UI is spoon, and UX is the actual eating of the cereal.
* UX process
  + Understand requirements
  + Define problem
  + Ideate
  + Prototype
  + Test
* Works as member in agile team.
* Books to read for devs to understand what fundamentals they work off:
  + Don’t make me think
  + Project guide to UX
  + The Design to Everyday Things

Continuous Delivery Pipeline

* A way to pass deployment and do tests in a server in a few commands, rather than set up an environment every time e.g. Phoenix.
* Make sure your deployment testing is on as close to production environment (OS) as possible.
* If you don’t know this as a dev manager, you will get fired.
* Good practice is to use a single code repository, and then git rebase very frequently.
* Good CD or CI is Circle CS or google for other online CS.
* You can add a lot of features on these servers as you want, but remember to at least play around with making a hello world on a CI/CD before going for a job interview.

Quality Assurance

* Technically are not testers – as testers work in waterfall processes and are used to find defects.
* QA
  + Prevents defects
  + Spends most of time conversing and thinking
  + Writes acceptance criteria
* Uses a mindmap to check through everything
* Costs to fix error increases in nonlinear rate, so earlier to find defects and prevent, cheaper to fix.
* QA Tools
  + Equivalence Partition:
    - Grouping tests in similar tests
    - E.g. calculator
      * Functions
      * Numbers
    - Cuts testing down to make less work rather than testing every combination
  + Boundary Analysis
    - Take value of boundary, one below, one above, and boundary value. Just test this and everything else should be good for this boundary test.
  + Brian Table
    - Work out all possible combinations and outcomes
    - E.g.

|  |  |  |  |
| --- | --- | --- | --- |
| * + - Correct Username | * + - Correct Password | * + - Correct Token | * + - Outcome |
| * + - T | * + - T | * + - T | * + - Successful |
| * + - T | * + - T | * + - F | * + - Error message |

Refactoring Legacy Code

* How to make legacy code good?
* Start by putting tasks into separate functions so that you can view input and output.
* Refactoring does not change code functionality.
* Use refactoring tools of IDE whenever possible to lower rate of mistakes.
* Turn things to tests

Business Analyst

* Analysing how business functions
* Heaps of types of BA
* Works in conjunction with UX and PO

Soft Skills

* Someone yells at you, count to 10
* Someone becomes a jerk, ask how they are going
* You need to rebut, wait until you’re calm. Leave it for a while.
* Acknowledge, and accept before disagreeing if you can’t find middle ground. Be genuine about this.
* Need to remember issue – resolve to person as close as possible but as far away from desk as possible.
* Reread How To Win Friends and Influence People.
* Become good at receiving feedback (don’t react immediately, fix structural problems rather than apply band-aid solutions)
* Tech is a means to an end, not end itself.
* Find right person to talk to, break rules if necessary.
* Arrive to meeting early. Set up AV.
* Eisenhower Matrix.

Resume (Getting a Job)

* Cover letter
  + One page
  + Highlight skills/experience
  + Demonstrate you know company and applying to and speak to what they’re looking for.
  + Don’t oversell yourself (be honest, but be confident)
  + Address cover letter to actual person.
  + Be seen in industry.
* Resume
  + Design it, one page, make it quality design.
  + Honest, don’t pretend and don’t lie.
  + Web presence is helpful.
  + Awesome sauce other activities/interests that make you stand out.
  + No references
  + Accomplishment oriented
  + Relevant to job
  + Don’t need objective statement
  + Never have strengths or MS Office
  + Don’t have irrelevant job/experiences
  + Get multiple people to proofread resume.
* LinkedIn
  + Create headline
  + Experience – show in moderation. Choose most critical.
  + Recommendations
* Phone Screen
  + When resume looks interesting, company wants to find out more
  + Have 2 minute pitch about yourself prepared, and your goals.
  + Always have questions ready, not precanned googled questions. Questions like “career progression” or what you will be doing in the job are good questions.
  + Don’t ask questions you can google.
* Programming interviews
  + - White board
    - Online
    - Take home assignment
  + For:
    - Coding knowledge
    - Data structures and algorithms
    - Problem solving ability
  + Practice these for two weeks before test
* White Board interview
  + Take time to think about it
  + Ask clarifying questions, don’t be quiet, if you do need to be, tell them.
  + Think aloud.
  + Choose whatever language you know best.
  + Talk though brute force solution, and ask for permission to optimise it.
  + Big O time/space complexity, data structures and algorithms.
  + [www.careercup.com](http://www.careercup.com)

Functional Programming

* Single input -> single output
* A function should have only one reason to change
* Open/closed principle
* Lishov Substitution principle
* Interface Segregation
* Dependency Inversion
* Parametricity
* Hoogle and Hayoo search engine for Haskell

Learning:

Youtube Lectures very good for following on.

Not many employers hire for junior engineers.

Update your CV and LinkedIn with keywords to get through the machine filtering.

Technical Interview:

MAKE SURE ALL OF YOUR APPS WORK!

Keep asking questions to work fast.

Lone wolfing is not an option.

Behaviour Stuff.

Situational constraints predict behaviour better than individual-level traits.

-unfulfilled needs.