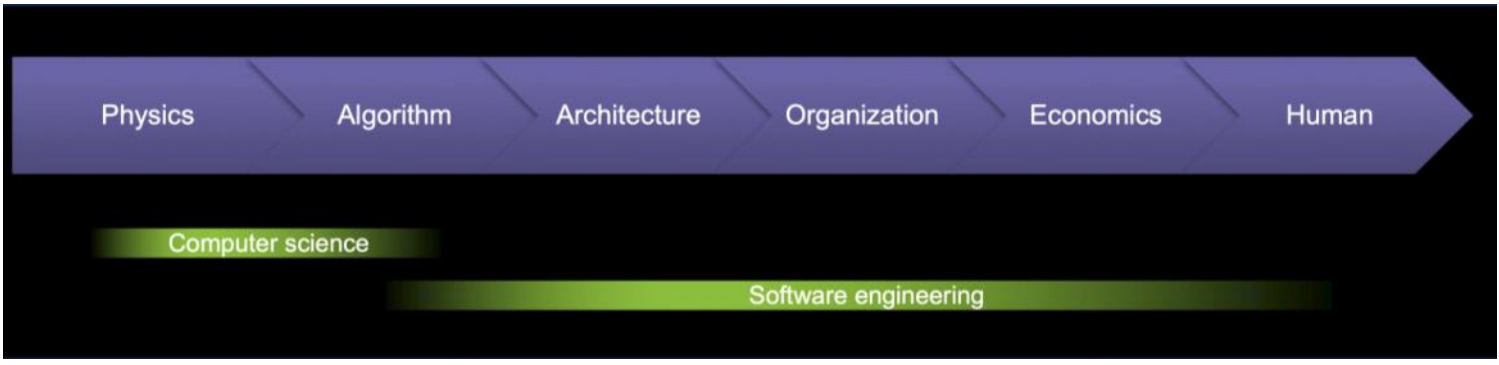
Lecture 1 - What is SE?

Monday, September 26, 2022 4:47 PM

People good a programming, but poor code still creating loss in economy



Need to practice SE before getting good at it
- Coaches integral to success of team in sports and SE

What is SE? Multiplayer multiversion programming

Want to create something that has:
- Predictable
- Repeatable
- Reliable
- High quality
- Cost effective

Programmer: writes code
SW: solves problems
SE > Programming
- Toolchains change, process does not

Rockstar Dev: fallacy that a single developer can be 10x a regular one, who knows everything

Most important thing about SE: Communication
- People quality > experience

Lecture 2 - Self Responsibility

Wednesday, September 28, 2022 5:44 PM

Be conservative in what you do, be liberal in what you accept from others

CS Ladder:
- Low level: math, binary
  - More subjective
- High level: users, graders
  - More predictable

Skill plateau: after reaching "acceptable" performance, more practice doesn't lead to improvement
- Do something small many times
- Avoid trying to deliver a perfect project

Training the Brain:
- Not about memorizing things, understanding is more important
- Need to maintain good physical and mental health to train
  - Sleep
  - Energy
  - Focus
  - Health
  - Attitude
- Don't focus too much on good gear, best gear only works if you can use it well
  - But: given the importance of typing, must have god typing skills, good keyboard
- No such thing as maximum productivity
  - Work at your rhythm
- Multitasking very inefficient
  - Yak shaving (doing work to get ready to do work) != productivity

Avoiding the pull of shiny new tools
- Simple and complex solutions can solve the same problem
- Promotions awarded to complex solutions
- Sometimes reach for new tools, but often rely on old tools for stability
- No need to switch tools often, try to master before evaluating
- New tools more dangerous than old tools
  - Why? More satisfaction from using newer tools, feel like you did something "cool"

Debugging: not just a technical process
- Reasoning, stepping back > hammering out code
- Avoid going straight to stackoverflow

Lecture 3 - Group Dynamics

Monday, October 3, 2022 5:00 PM

Knowing all the things mentality
- Brain has limited amount of capacity
- Not helpful for confidence, leads to imposter syndrome

Team dynamics
- Groups evolve and should be self-aware to become high functioning
- Communication is critical
- Team size
  - Too many: overwhelmed by volume
  - Too few: too much work per person
  - Organize under hierarchies to organize management
- Team Composition
  - Specialist trend: each person has unique roles that they excel at
  - Generalist trend: each person can do a little of everything (typically bad)
  - Avoid dominance hierarchy: coding at top with QA and docs at the bottom
- Organization
  - Pathological: Reward driven, dog eat dog system
  - Bureaucracy: Rules driven, fairness / no specialized treatment
  - Generative: Too much autonomy can become runaway train
  - Artifacts can values can indicate what organizational method a group prefers
  - Conway's law:
    - Small Distributed Teams -> Modular Service Architectures
    - Large Collocated Teams -> Monolithic Architecture
- 5 Things for Great Teams
  - Psychological safety (paramount) : need to be comfortable taking risks
    - Model (behavior)
    - Allow for failure
    - Avoid blame
    - Empathize
    - Avoid cliques
  - Dependability
  - Structure and clarity
  - Meaning of work
  - Impact of work

Communication
- Signal degradation is the problem in communication
- Learned activity
- Avoid mob programming (bunching up)
- Breaks down when one person breaks down the network

Lecture 4 - Group Responsibility

Wednesday, October 5, 2022 5:08 PM

Psychological Safety
- Integrity : ethics with candor and without retaliation
- Innovation : fearless collaborative creativity, shared success
- Inclusion : authentic membership and respect

What can go wrong:
- Group think : fear of ridicule
- Project risk and quality reduction : similar to group think
- Poor retention : equation of $ does not eliminate requirements for satisfaction

Communication
- Each person's prefers different communication methods
- Provide objections and advice when appropriate and when delivered correctly
- Effective teams will have diverse conversation patterns
- **Don’t make it personal**
- Practice makes better
- Empathize, especially when things go wrong

Dependability

Structure and Clarity
- Alignment over autonomy : pick a direction to all go in
- No correct way to do structure

Meaning and Impact of Work
- Everyone's work contributes to the whole project, NASA janitor helped get US to the moon

Geniuses and Hiding
- Not helpful to bring up how another company does things
- But also don't hide and avoid detection
- Bus factor: number of people who can leave before project is doomed

Managers and Coaches
- Additive: Does work and manage
- Multiplicative: Does less work but allows other people to accomplish more
- Subtractive: Makes things worse

Lecture 5 - Recap of Self and Group Responsibility

Friday, October 7, 2022 5:01 PM

Goofus and Gallant

- Goofus (the one not to be)
  - Tries to code as quickly as possible
  - Only thinks about what the boss wants
  - Think doesn’t need to consult other people
  - Always uses the hottest tool no matter what
  - Own personal hackathon
  - 10x rockstar and works alone
  - Doing many things at once
  - No time to write docs
  - Ships code as soon as it runs
- Gallant (the one to try to be)
  - Does some research before diving in
  - Thinks about what the user needs as well
  - Talks to end users, team
  - Pick the best tool for the job and team
  - Structured and practice following
  - Part of the team and shares with the team
  - Focuses on one thing
  - Writes docs as he does
  - Ships code after testing it well

Lecture 5 - Problems and Projects

Friday, October 7, 2022 5:10 PM

Problems:
- Simple: puzzles, only one solution, **very constrained**
- Complicated: problems, may be many solutions, **some constraints**
- Complex: mess, requirements are not clear, **few constraints**

Project: CRUD App
- Create
- Read
- Update
- Delete

App is complicated but not complex
- Complicated things can be solved with processes and patterns
- Complex things have unknown things, processes and patterns are not as useful
- Minimize unnecessary solution complexity

Local first software
- Resides on user's device
- But also collaborates with others
- Changes the traditional relationship with the cloud
- Ideals:
    - No waiting for data
    - Work is not trapped on one device
    - Network is optional
    - Seamless collaboration with colleagues
    - Long now: will work even after support stops
    - Security and privacy built in by default
    - Users retain ultimate ownership of content
- Self-contained software
    - Support maximum capability while offline
    - Sync and store architecture

Project Domain: Personal information management

Lecture 6 - Design Engineering

Monday, October 10, 2022 5:02 PM

Project Domain: Personal information management
- Posts
  - Tweets, blog posts, etc
- Pictures
- BROAD DOMAIN
- Conduct research and narrow a design

Technical: Core technologies
- Raw Web Platform
  - HTML
  - CSS
  - JavaScript
- Does not NEED to be a website, can be desktop app

Architecture: CRUD
- Local first, remote second
- Can add 3rd party destinations later
  - Should create abstraction layer to future proof

Avoid feature explosion. Start simple then increase complexity.
- Plan ahead
- Start early, don’t wait for things to come

Software Activities and Ordering
- Software has life cycle: created, maintained, dead
- Factory thinking: build and stamp out many copies, production engineering
- Design thinking: design and create unique solutions, design engineering
- Bottom up thinking: solve the low level problems before thinking about the top level ideas
- Top down thinking: create the top level ideas before solving the low level problems
- Linear approach: perfect one idea at a time
- Iterative approach: try a few ideas in one iteration, keep improving them over time
- Balance between cost, scope, schedule, quality

Lecture 7 - DDD, Pitch, Tensions and Tradeoffs

Wednesday, October 12, 2022 5:00 PM

Domain Driven Design
- Must understand domain to design project for specific domain

Pitch requirements
- First principles: what does this app accomplish?
- Research: look at other projects, etc
- User thinking: What are the users? What requirements do they have?
- Feature thinking: what features does the project have?
  - Flow charts: show how the app works
  - UML Diagram
  - Event modeling
  - Class charts
- Systems architecture: how does the user, app, system, cloud interact?
- Wireframes: sketch how the app will look
- Organizational structures: github, team, etc
- Exploration: create small tests to verify the feasibility of sub components

Tensions and Tradeoffs:
- Tradeoff between people, cost, features
- Always will be tradeoffs

Avoid appeals to popularity: "everyone is using ___ so it must be the best"

Lecture 7 - Process Models

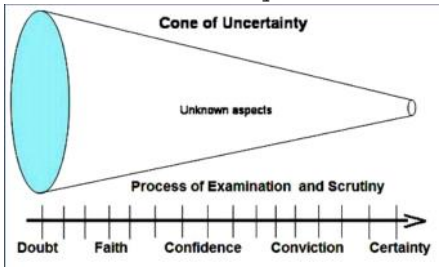
Wednesday, October 12, 2022 5:29 PM

Deadlines:
- Never can finish a project exactly on the anticipated date
- Set project complexity so that you can finish early and use the extra time if needed
- Time pressure: poor code when under pressure

Scope:
- Dietzler's Law
  - 80% of what the user wants is fast and easy
  - Next 10% is possible but difficult
  - Last 10% is *impossible*
- Common people risks:
  - Weak personnel
  - Heroics
  - Negative personalities
  - Wishful thinking
  - Politics
  - Inappropriate work space
  - Lack of buy-in, patrons, etc.

Quality:
- When deciding, avoid focusing on only the outcome. Must consider the value of outcome against risks.

Cone of uncertainty:



Process types and methodology: want to not be miserable
- Waterfall: Set of steps in linear fashion. Plan then execute then deliver.
- Incremental design: implement one item at a time, without need for an overall goal.
- Agile: Break large problem into small tasks, try each one at a time

Midterm 1 Notes

Thursday, October 13, 2022 7:13 PM

Why practice SE?
- Costs of poor software
- Need to focus on problems and users rather than technology

What is SE?
- Multi-Person construction of Multi-Version programs
- SE > programming
- Consists of some technical problems, but mainly social ones

How to best do SE?
- Start at the problem, then work to a solution
- Avoid getting caught in tech details
- Understand SE is a people problem, tools are not the most important factor
- Start with yourself
  - Train the brain
  - Focus on good health, sleep, energy -> focus and good emotional state
  - Gear is important, but is not the most important
  - No need to "grind" or 996
  - Understand your work rhythms
  - Understand your work load limit
  - Confidence is not about how much you know\
  - Embrace failure

What to do/ What not to do?
- Don't try to be a 10x/rockstar programmer: a single rockstar can't carry a team to success
- Don't become a -10x programmer: don’t do more harm than good
- Avoid focusing on tools, use the right tool for the task
- Avoid yak shaving: doing work to get ready to do work

On groups
- Communication: becomes difficult as more team members
- Composition: more diversity is better, aim for specialists rather than generalists
- Organization: different organizational methods have their pros and cons
- 5 Things for great teams:
  - Psychological safety (paramount) : need to be comfortable taking risks
    - Model (behavior)
    - Allow for failure
    - Avoid blame
    - Empathize
    - Avoid cliques
  - Dependability
  - Structure and clarity
  - Meaning of work
  - Impact of work

Problems:
- Simple: puzzles, only one solution, **very constrained**
- Complicated: problems, may be many solutions, **some constraints**
- Complex: mess, requirements are not clear, **few constraints**

Process models: want to not be miserable
- Waterfall: Set of steps in linear fashion. Plan then execute then deliver.
- Incremental design: implement one item at a time, without need for an overall goal.
- Agile: Break large problem into small tasks, try each one at a time

Midterm 1 Recap

Monday, October 17, 2022 5:07 PM

- Defining SE
  - SE >>> Coding
  - Multiplayer multiversion programming
- Developer Outwards
  - Improve quality of developer outweighs process, tech , tool
  - Improvement from mindset and realistic time/practice
  - 10x developer takes time and effort
- Process models
  - Top down: Big Design Up Front
  - Bottom up: No Design Up Front
  - Contextual use
- Engineering Pragmatism
  - Tradeoffs and Iron Triangle
  - Facing cone of uncertainty
  - Tools first or solutions first?
  - Solve the problems we face now, not the problems we might face
  - Balancing risks and thinking in bet
  - What we learn may change and finding unchanging truths underneath is true aim

Lecture 8 - User Centered Development

Monday, October 17, 2022 5:01 PM

Deciding what to build?
- You?
- Users? Who?
- Both?

The premise
- Understand your users and their needs

UCD: User centered design
- Emphasis on the user during the constructive process

- Figure out what the Users want:
  - You != Your users
  - Users != your designers
  - But, Users can't always know what they need
    - More page views, time on site, etc = better | aka line go up
  - Must be employed with extreme caution
  - Sampling
    - Persona generation
      - Beware of personas becoming stereotypes
    - user stories
      - Agile concept
      - As a ___ I want to ___ in order to ___
    - customer journeys
      - Try to understand your software lives in your user's world and is not their whole world
    - Observation:
      - direct observation
      - Indirect observation via analytics
    - Interviews

- Document decisions made in Architectural Decision Records
  - Illities: level 0 decisions
    - Utilities: does the system do what the user wants
    - Availability: is there access to the system
    - Performance: access within acceptable time
    - Accessibility: able to use the functions
    - Usability: able to use the system successfully
    - Satisfaction: enjoyment of using the functions
  - User mental model != Your mental model
    - System model can be hidden from the user's mental model

Lecture 9 - Agile Methodologies

Wednesday, October 19, 2022 4:56 PM

Why Agile?
- Want SE to be flexible and react to changes, to be nimble and quick
- Fix resources and time but be flexible on scope
- Came from poor state of affairs in dev affairs
  - Consequence of dot com crash
- Adoption driven partially by pros, but also by social proof, FOMO

What is Agile?
- Agile != speed; Agile is to move properly not necessarily quickly
- Perform iterations of waterfall method many times
- Mindset:
  - Flexibility
  - Pragmatic
  - Openness
- Values:
  - Individuals and interaction over process and tools
  - Working software over comprehensive documentation
    - But also make sure to document well they are compliments
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan
- Principles:
  - Speed, user focus, communication, self-organization, good tech and design, keeping it simple and dealing with change
- Practices:
  - Majority people use Scrum
- Tools:
  - Github Issues/Projects, Jira, Trello
    - Most tools will do the same things anyways, some more overkill than others
  - Burn down chart, Kanban board
    - Track the state of things
- Process and Ceremonies:
  - Daily standup : make sure everyone knows what is happening
  - Sprint planning : pick an items to work on for that time period
  - Sprint Review : show what was accomplished, take stock of where the project is
  - Retrospective : reflect on how the last time period went
  - Andon cord : signal to stop everything and figure out what's going wrong

What to do?
- User stories:
  - As a ___ user I want ___ to get ___
  - != tasks
- Tasks
  - Not too small or large
  - Need to evaluate the size of each task and distribute accordingly
  - Learn from previous sprints
- Product backlog
  - Cumulative list of deliverables
  - Don't make too many, order by priority
  - What **must** we do
  - What **should** we do
  - What **could** we do
  - What **won't** we do
- Sprints
  - Iterations of the process, 1-2 week size

Pros/Cons?
- Use of all techniques are not always employed

Lecture 10 - Problem and Solution

Friday, October 21, 2022 4:51 PM

Problem -> Research -> Problem Definition -> Narrowing Down -> Solution

Alignment diagrams: Who/why will a user use the app? How can it be implemented?

Solve problems by understanding the users as real people
- People won't use your project unless it's really useful
- Consider context of users using the software
- Start with the customer experience and work backwards to the technology

Project Artifacts: Personas
- Fictional character representing what the user needs, how they will benefit
- Try not to create stereotypes

Project Artifacts: Journey maps
- Journey of how the user gets through the experience

Project Artifacts: User stories
- As a ___ I want to ___ so that ___
- Design for the purpose of addressing user stories

Tools
- Miro: Drawing and design tool, can create diagrams and flowcharts.
- Git/Github: Git is file tracker, Github shares git repos online
- IDE (VSCode): Features vs speed
- Code grading

Lecture 11 - Requirements and Planning

Monday, October 24, 2022 5:05 PM

Need to have consideration for the libraries and dependencies that we install

Planning: significantly cheaper than coding
- Plan from general to specific
- Requirements > code
- Want to code right away, but need to plan first

Requirements
1) Who - the actor
2) What - the action that the actor takes
3) When/Where - state of the system and actor's relation to it
4) Why - goal of the actor
5) How - means the action is done

Good Requirements
- Measurable and precise requirements
- Must be quantitative things that can be measures

Requirements to Specifications
- How formal?
  - Depends on the user, governments will be more formal
- Create flowcharts / state machines?
  - Could be done by flow charts
- UML
  - Formal way to describe specifications
  - Not widely used

1) Wireframing: Design user thinking, with some state logic
2) Storyboarding: showing how the user flows through the app

Key design principles:
- Less is more
- Users don't read
- When existing expectations are not enough need guidance

Avoid bricklaying, be an architect

Lecture 12 - Build Dev 1: CI/CD

Wednesday, October 26, 2022 5:02 PM

Attention to detail at the beginning more impactful than attention to detail at the end

Idea: create a software factory
- Create parts individually and verifying them and then deploying for testing or release
- Factory = tools (CI pipeline) and processes (checklists, human procedures)
- Manufacture = an instance of running code through the factory to make evaluation

Faster quality factory means figuring out problems faster and producing new attempts
- Make sure the order of steps will fail as early as possible
- Do shorter steps before longer steps
- Do important things first

Build automation
- Creating a process which creates the working application quickly

Continuous Integration
- Practice of creating many internal builds to test new features iteratively

Continuous Deployment
- Practice of continuous deployment of final releases
- Always have working deployable software

Technologies
- Build pipelines will depend on
    - Dependencies
    - Needs of the products

Example CI in various incremental steps:
1) Git push -> Deploy
2) Git push -> Unit tests -> Deploy
3) Git push -> Style enforcement -> Unit tests -> Deploy
4) Git push -> Style enforcement -> Unit tests -> Minimization -> Deploy

Steps:
- Start with simple HTML page to practice pushing, issues, merging, deploying
- Explore each element before work
    - Throw that code away when done
    - Avoid social proof (don't just take google at its word)
    - Do hands on work rather than assess "feels"
- Work on each component type independently
- Work on integration of pieces
- Make sure everything is documented so anyone can run it

Lecture 13 - Architecture 1

Friday, October 28, 2022 4:57 PM

Def: Fundamental organization of a system, component's relationships to each other and the environment

- Architecture s important things, whatever that is
- Stuff that's hard to change later
- Theory or design about how the system will be implemented
  - Breaking the app into pieces and how they relate to each other
  - Affects user related things like: Performance, availability, security, maintainability, extensibility

Lecture 14 - Good Coding 1

Monday, October 31, 2022 5:00 PM

What does good code look like?
- Readable
- Modular
- Simple
- Does what needs to be done
- Just enough dependencies

SSoT: Single Source of Truth
- Where to find the answer to all questions?
- Keep your plans on site
- Documentation

Simplicity
- Generally less is more
- Complex code may be more brittle
    - May not survive time
- Always be cleaning, fixing bugs, implementing TODOs, paying down technical debt
- Adapt and grow but guard against trendiness

Repos
- Max directory size: 10-20
- Clear directory naming
- Clear directory hierarchy
- Clear directory file grouping
- Prune branches when done
- Don't let issues pile up
- Actually evaluate pull requests

Lecture 15 - Good Coding 2

Wednesday, November 2, 2022 5:11 PM

HTML
- Use consistent style
- Validate the markup: HTML is very permissive with problematic code, need to validate markup to ensure correctness
- Aim for valid markup
- Use semantic markup: use <nav> over

Lecture 16 - Architecture 2

Friday, November 4, 2022 5:00 PM

How to address big decisions?

Ex: What kind of app?
- WebAPP?
  - Simgle page?
  - Multi page?
- PWA?
- ElectronJS?
- Cordova?
- Chrome Extension?

Architecture Decision Record: Captures key choices and why

Models
- Model, View, Controller:
  - Model: How the application is presented
  - View: How the user interacts with the application
  - Controller: How the application functions below the hood
- Content, Structure, Presentation, Logic

Progressive Enhancement:
- Move from HTML to CSS to JS
- Degrades to standard site if JS or CSS is disabled

Graceful Degradation:
- Move from JS to CSS to HTML
- Prevents users from using app if JS is disabled

Microservice: break app into many smaller components

Monolith: keep application as one large component

Architecture Astronauts: trying to solve the template for many problems rather than the exact problem

Lecture 17 - Good Coding 3

Monday, November 7, 2022 5:01 PM

JavaScript
- Not just for web apps
- Can be run in any host environment including servers, desktop, mobile

Types:
- Primitives: int, bool, string, undefined, null
- Composite/Reference: Object, Array*, Function

Style:
- Keep it consistent
- Use comments when code needs explanation
- Use comments as annotations (TODO, HACK, XXX)

Lecture 18 - Testing and Quality

Wednesday, November 9, 2022 5:00 PM

What does quality mean?
- Works
  - Bug free?
- Efficient
  - Memory
  - CPU
  - Load time
  - Response time
  - RAIL: response, animation, idle, load
- Easy to use
- User friendly

How do we get quality?
- Cannot prevent users from destroying the software
- More dependencies means more complexities and more bugs
- Testing pyramid
  - Many small tests for each part
  - Less tests for larger parts
  - Top level testing may be human
  - Unit -> Service/API -> UI
- Use CI to create a quick way to run tests
  - Unit testing: write automated tests to check expected vs results
- Code Coverage
  - Trivial tests passing means little
- Code reviews: can't be too brief but not harsh, needs to be constructive
- Evaluate third party code, be careful using it
- Avoid General Browser Stats
  - Too many possible browsers and versions
- Load testing
  - Test how much traffic the app and servers can handle and determine fail points
- User acceptance and usability
  - You have to like your own app
  - Friends have to like your app
  - Unfriendliness should like your app from a user point of view
  - Public …

Things will fall apart
- Assume the worst, don’t hope for the best

Test Driven Development
- Create tests before implementation
- Tests should fail first and then be patched to pass

Behavior Driven Development
- Tends to have more English-like assertions
- May be more friendly to QA or business stake holders
- expect(…).toBe(…)

Lecture 19 -

Monday, November 14, 2022 5:03 PM

Midterm 2 Review

Wednesday, November 16, 2022 5:01 PM

- UCD: User centered design
  - Define: putting user at the center of design
  - Laws and tips:
    - You are not the user
    - Users can't be your designer
  - Techniques
    - Create personas, user stories
  - Artifacts
    - Personas
    - User stories: As a <blank> I want to <do blank> in order to <blank>
  - Illities: broad nonfunctional characteristics
    - Determines how users feel
- Agile
  - Values: Communication, Feedback, Simplicity, Courage
  - Daily Standup: Quick meeting of what you did and what you need to do
  - Sprint: time box to conduct work, can be 1 - 6 weeks
  - Sprint planning : start of sprint meeting to pick items to work on
  - Story points: generalize time estimates to categories (S, M, L)
  - Sprint review: end of sprint meeting to show and tell work
  - Retrospective: A retrospective meeting allows us to look back at our previous sprint and discuss the high level issues of Agile and what went right and wrong
- Development
  - Take something small and roll it uphill, don’t make big thing and fix
  - CI/CD pipeline: factory to stamp out software pieces
    - Many small steps quickly
    - Feel the hate before you automate: do step manually before deciding to automate
  - Avoid spreading: keep things together
  - Teams should code as one: code owned by everyone
  - Play styles
    - Mobbing: everyone working on the same code together, great at start or in emergencies
    - Pair: two people work together, code review as we go and teaching people as you go
    - Solo: working alone, need self-discipline to follow team rules
  - Tasks: break big tasks down until it’s the right size
  - Definition of Done: must define all aspects of task, done iff they are all addressed
  - Document as you go
- Requirements and Specifications
  - Planning is cheaper and faster than coding
  - Need to get requirements from different people
  - Visual representations: diagram down levels
  - Technical debt: buildup of not doing work the right way
    - Must document major decisions in ADRs
    - Prevent hindsight doubt
  - Specifications depends on the project
  - Use dependencies with caution, evaluate before use
- Testing and Quality
  - Test pieces -> test integration -> test users
  - TDD: test driven development, write tests then write the implementation
  - BDD: behavior driven development, write code to match behavior which better matches user expectations
  - Acceptance testing: do people use the app?