Development Workflows

How a list of goals get worked on

- Past: Waterfall
- Recent: Agile

Waterfall model

Each step done in turn before the next starts

- If a hole is found in a previous step, back up
- Common key: All requirements in advance
- Fantastic for "known" tasks
- Great for hard requirements:
    - features
    - resources (incl people)
    - deadlines
- Generally terrible for most software
    - slow
    - inaccurate

Waterfall aims for predictability

- If you know everything to be done
- and how long that will take
- You have:
    - A schedule
    - A list of needed resources (incl. people)

No one likes:

- "it will be done when it is done and will cost whatever it costs"

Waterfall process does not just produce code

- Plans and descriptions
    - Task Dependencies
- Documentation
- So many Gantt charts
    - Identify "critical path"

Waterfall Criticisms

Coding is still as much art as science

- Our estimates are awful
- Coders are not fungible

Waterfall "resets" when requirements change

- Software requirements change all the time
    - "Didn't know we needed to know"

Waterfall+Software

- Tends to break schedules + budgets
- Create products clients aren't happy with

Agile development

https://agilemanifesto.org/

Common ideas:

- Focus on product
- Favor team over process
- Produce working code very frequently

Agile Ideals

- Driven by team members
- Regular feedback with "client"
    - Outside client, PM, manager, etc
- Product able to release frequently
    - Not that it does, but it CAN
- Team build realistic estimates
    - Often based on consensus
    - For the next batch of features
- Each task is small and quick to add
    - Fully tested & "done"
- Done when it is done

Agile issues

- Companies still want reliable
    - schedules
    - budgets
- Often assumes coders are fungible
- Often assumes team is uniform
- Coders may not be given sole focus
    - multitasking has overhead costs
- Devs often hate meetings

Agile in Practice

- Scrum-like
- Kanban

Scrum

Scrum is a particular approach of proj management

- Exactly followed OR borrowed from/altered

Notable features:

- **sprints** (1 week, 2 week, 1 month)
  - Starts by defining **tasks** due for the sprint
  - Focused work period
  - Daily "scrums" (or **standups**)
  - Ends with a **sprint review** (incl **demo**)
- **Standup**
  - Very quick(?) meetings to maintain progress

Kanban

Inspired by Toyota efforts, but different

- No sprints, continuous
- Track tasks through stages
    - Never stay more than 1-2 days in a stage

Generally similar to Scrum (or vice-versa)

- Standups
- Tasks
    - Define and estimate
    - Tracking
    - Fully tested and done

Tasks or Stories

Tasks (or "**User Stories**" in formal Scrum)

- Small enough to be done in 1-2 days
    - If bigger, break it down
- Team creates or has input on estimate
- Should produce noticeable change
    - In practice: just track time
- Should not be used to track time
    - In practice: used to track time
- May be in **backlog** (not for current work)
    - Backlog may have estimated/not estimated

Defining Tasks

- Might come from "client"
- Might come from PM
- Might come from team
- May lack, but needs before work starts:
    - Full definition
    - Estimate
- Often includes a "definition of done"
    - "Done when..."
    - Should be confirmable by someone else
- Tasks are done/not done
    - 80% done is...not done

Estimating Tasks

Story Points vs "hours"

- Story Points tries to disconnect from clock time
    - Companies often reconnect these
- May be in weird sequences (fibonacci)
    - To highlight tasks that are too big
        - Estimate is too much guess
        - Should be broken down before work
- "**Planning Poker**"
    - Team voting and discussing estimate
    - Hard when team has different specialties

Velocity - Dealing with bad estimates

Doesn't care about (ideally):

- Actual hours vs estimate
- If one dev is numerically ahead

The sum of estimates for tasks/completed

- Divided by actual time (or sprints)

Gives actual time to complete remaining work

- Even if estimates are terrible
- As long as consistently terrible

Standup Meetings

- Theory: **Daily**
    - Practice: 1-5/week
- Theory: **Quick** (15 minutes)
    - Practice: 15-120 minutes

Should NOT be about justifying work

- About identifying + resolving "**blockers**"
- Don't forget extra tasks you may have in a sprint
    - Want to avoid surprises at end of sprint

Code Review

Code must be reviewed and approved

- Before merge

Everyone has code reviews

- Not based on seniority

Code Review Purpose

Review is not for technical accuracy

- About team understanding
- ...and overall quality
- confirm API, tests, tangled logic
    - names

Even new devs have contributions to make

- Can you follow with minimal context?
- You aren't ignorant!
- Keep codebase approachable
    - Easy for a codebase to become arcane
        - Quickly dies as soon as team changes

Learning from a Code Review

Code Reviews are chances to learn

- Understand the code base + concepts
    - Every codebase will have built in concepts
    - This context is key task to joining team
- Learn best practices

After school, no one is teaching you new tricks!

PR Etiquette

Pending Reviews block tasks!

- So do yours quickly
    - I suggest setting a time block each day
    - Easy to delay in favor of your other deadlines
        - But that leaves team with a problem
- Allow for delay with your own work
    - Don't push PR last minute

Don't make your Review an attack

- Call out good things!
    - Small choice is often worthy of praise
        - "I love this variable name"
        - "I stan the intuitive API!"
- Feedback is often better than "LGTM"
- Learn to be non-critical in critique
    - Don't say "this is bad"
    - Do say "would it be better to...?"
    - Do say "is there a way to...?"
- Be clear about expectations for Approval

Reviews of you are not attacks (hopefully)

Don't consider a Review a pass/fail

- much like your review notes in class
    - not every comment is a loss of points
- Instructing you on codebase conventions
    - you may have had no way to know!
- Instructing you on best practices
    - Chance to learn!
- May be bad advice/needlessly critical
    - Did they test your review skill before hire?

Agile In Practice

If you fall behind, *must* change one of:

- Resources (people)
- Features
- Deadline

Why identifying blockage is so important

Companies say "Agile", not Agile, not Waterfall

Not Agile:

- Set deadline
- Mostly "set" features
    - "nice to haves"

Not Waterfall:

- No per-day schedule
- Requirements adapt
    - Within features

Common issues

- Too many meetings
  - = Too many **low-value meetings**
  - Interrupted work/broken **flow**
- Sprints/Tasks as time-tracking
  - Different purposes
- Coders are not fungible
  - Different areas (mobile, frontend, backend)
  - Different skills (service calls, arch, UI)
- Not done tasks