
Learning
Agile
Scrum, XP, Lean, Kanban

Agile Promises

- deliver software on time
- high quality software
- well constructed & highly maintainable software.
- makes users happy
- helps devs work normal hours.

Strong focus on changing your team's mindset.

Agile is a set of Methodologies.
helps your team think more effectively,
work more efficiently,
make better decisions.
Also a mindset. opens up planning, design & process improvement to the entire team.
Difference between successful teams & others is this mindset.

Daily stand up is meant to make everyone feel collective ownership of work and responsible for the plan. Team makes decisions, not some managers.
Don't give status, provide input for collective decision making.
Good dev might have opinions about the project as a whole.

Understanding Agile Values

4 Core Values:

- Individuals & Interactions over process & tools.
- Working Software over comprehensive documentation
- Customer Collaboration over contract negotiation
- Responding to change over following a plan.

Teams that use waterfall processes successfully usually have common characteristics:

- Good communication
- Good Practices
- Great requirements gathering

When waterfall projects are run effectively, it's because the teams take to heart many of the same values, principles and practices that agile follow.

"Agile" shops that follow practices without really having the mindset & principles, fall into the same problems as waterfall.

Waterfall:- write down all requirements & strictly follow the plan.
- focus on documentation - difficult to respond to change.
- teams that make it work follow effective software practices & principles

When agile projects fail, it's often because of cultural & philosophical differences between waterfall & agile methodologies.
- lack of exp. w/ using agile methods
- company philosophies at odds w/ agile values
- external pressure to follow waterfall practices

Teams often haven't really changed their ways from old waterfall. Just adding agile practices isn't enough to break them out of problems that cause conflict & avoidable changes. They're just become the most efficient waterfall team they can be.

Split perspectives leads to different team roles focusing on areas they're already good at, and miscommunication of each person's priorities creeps in. Same old problems.

- Better Communication helps a team manage change better.
- Planning as a team is more important than over-documenting a plan.
- Software projects are unpredictable.
- Many teams try going agile by adopting great agile practices that improve on what they already do well.
- Adopting individual practices is the most common way to adopt agile today, but it's not the most effective path to agile.

Agile manifesto helps teams see the purpose behind each practice.

"A great tool can sometimes help you do the wrong thing faster."

Team needs to agree on how to implement effective practices that help them act in a more agile fashion.

Stand ups, retrospectives, user stories, can all be great tools, but only if the whole team embraces these as opportunities to better communicate (interactions), rather than a tick box exercise. which can quickly become "What's your status?", "Did we get it all done?", "Stakeholder said X".

Team should focus on building software that adds value, documentation is merely a means towards that end.

Different Roles in a team "do" agile by implementing the agile practices relevant to their area, like velocity & burndown charts, CI/CD & pair programming.

Easily done, you'll find entire books of interest to your area helping you become more agile, while maintaining nothing of the other aspects of agile.

An Agile Methodology is a collection of practices combined with ideas, advice, and a body of knowledge & wealth of experience among practitioners.
Comes w/ different roles & responsibilities for everyone, and recommends certain practices

---

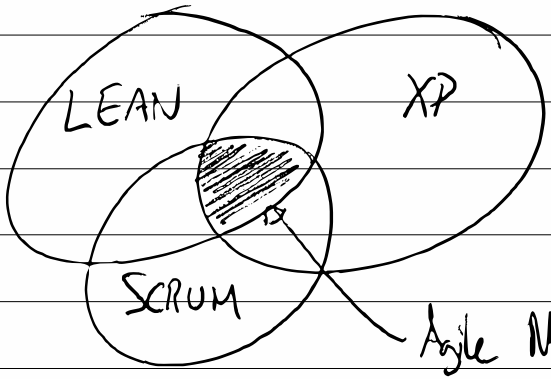
Scrum:	✓	X
Product Backlog:	list of requirements for software, prioritised for value	to-do list.
Sprints:	Goal of delivering functionality to sponsors.	Enough stuff to do for 2 weeks
Daily Standup:	Communicating about the project, risks/changes/opportunities	What I did yesterday
Scrum Master:	leader/coach/shepherd	Your manager or tech lead checking up on you.

These practices can be adopted in a way that doesn't truely reflect values & principles of agile.
"Scrum-like", but not scrum.

XP prescribes specific development practices, aimed at improving collaboration with users, planning, developing & testing. XP helps the team build simple, flexible software designs, easy to maintain & extend.

Adopting an entire methodology helps whole team avoid everyone just using practices specific to their role.
Need to change the way they think about their job.

Lean / Kanban are not methodologies, but mindsets.



A team that only focuses on individual practices can lose sight of the larger goal of better communication & responding to change.

Scrum teams can spend a whole 8 hour day planning for the next 30 days. They're all engaged in the process, care about the outcomes, get a feeling that this planning will allow the rest of the iteration to go well.

The Agile Principles

1. Highest priority is satisfy the customer w/ early & continuous delivery.
2. Welcome changing requirements, even late on. Helps competitive advantage.
3. Deliver working software frequently, weeks. Shorter = better.
4. Most effective communication is face-to-face.
5. Business people & devs must work together Daily.
6. Build projects around motivated individuals. Give them the environment & support needed, & trust them to get the jobs done.
7. Working Software is the primary measure of progress.
8. Agile processes promote sustainable development. Should be able to maintain constant pace indefinitely
9. Continuous attention to technical excellence & good design enhances agility
10. Simplicity - maximising work not done - is essential.
11. The best architectures, requirements & designs emerge from self-organising teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes & adjusts its behaviour accordingly.

#2: Welcoming Changes even late on.

- Nobody gets in trouble when there's a change.
- Everyone's in it together, & we're all culpable.
- Don't sit on the change until it's too late.
- Stop thinking of changes as mistakes.
- Learn from the changes.

#3: Deliver Frequently.

Project manager no longer a task manager & commander, but job is to keep everyone aware of big picture.
Timeboxing is the point of sprints.

Working software over comprehensive documentation.
The right amount of documentation is just enough to build the software. Might mean more or less, depending on how well the team communicates, or their build practices.
Unless it's required, the team doesn't make the documentation.
Managing changes in excessive documentation is a pain. Time sink.

Building documentation can become a goal unto itself.

Build Projects around motivated individuals.

Projects work best when everyone in the company understands what makes the software valuable to the company.

Projects can breakdown when people don't see the value in the software, or aren't rewarded for building it well.
Common "motives" against agile:

- giving programmers poor perf. reviews when code reviews routinely turn up bugs, and rewarding them for clean code reviews.
- rewarding testers for number of bugs they report. This encourages nitpicking & discourages them from partnering w/ the programmers, because it sets up an antagonistic relationship.
- basing BA perf. reviews on amount of documentation they produce, rather than the amount of knowledge they've shared w/ team.

Everyone's perf should be based on what the team delivers. Each person should not be discouraged from looking beyond the strict confines of their role.

Comprehensive documentation & traceability matrices are particularly insidious - encourages a "cover your ass" attitude.

Overly comprehensive Documentation increases risk of ambiguity, misunderstanding, and miscommunication between team members.

Everyone on an agile team feels responsible for the project and feels accountable for it's success.

7: Working Software is the primary measure of success.

If a manager promised to deliver something but didn't it might be embarrassing, but it should be impossible to hide, because the measure for progress is working software. Status reports are bullshit - show me the software running & delivering value.

8: Agile promotes sustainable development.

Only promise to deliver what they can actually build.

#9: Continuous Attention to Technical Excellence and Good Design
Enhances agility.
Fastest to deliver, because it's easiest to change.

#10: Simplicity is essential
dependancies increase the likelihood of one change cascading out to another part of the system. Domino effect.
All about maximising the amount of work not done.

#11: Best Architectures, Requirements, & Designs come from Self-organising teams.
Overly complex designs can come from too much up-front planning. Self-organising team doesn't have a specific requirements or design phase. Team works on the project, breaks it down to user stories, starts working on the ones that deliver most value to the business.
Everyone shares responsibility for the architecture.
Architects & designers no longer work in isolation.
Agile Architects use incremental design.

#12: Regularly Reflect on how things are going. Tune things to become more effective.

Need to be comfortable being brutally honest.

This is one of the most neglected principles. Teams don't set aside the time to do it.

Ch 9.

Scrum & Self-Organising Teams

3 roles: Scrum Master, Product Owner, team member.

- PO works w/ rest of team to maintain & prioritise a product backlog of features & requirements that need to be built.
- built in timeboxed iterations called sprints. Start of sprint, team does sprint planning to determine which features they'll build. This is the sprint backlog, team works on it during sprint.
- Every day, team holds short face-to-face meeting (daily scrum) to update each other on progress made, discuss roadblocks.

What have I done? What will I do? Roadblocks in my way?

- Scrum Master keeps things rolling by working w/ team to get past roadblocks & issues they've asked for help with.
- End of sprint, working software is demo'd to PO & stakeholders in sprint review. Team holds a retrospective to figure out lessons learned, so they can improve the way they run their sprints, and build software in future.

Need to do more than just follow basic scrum pattern, need to be self-organising. Team must understand "collective commitment".

Sprint Planning — 8 hours / 30 day sprint.

PO brings prioritized backlog for product. items stakeholders and users have bought into.

Part I: 4 hours.

Team picks stuff that will be delivered, based on value, estimations & capacity.

Team agrees to demo this working in 4 weeks time.

Part II:

teams figure out the individual tasks they'll do to implement those items. At the end of the sprint planning, items selected became the sprint backlog.

Daily Scrum

Scrum Master, PO, members must attend.

Interested stakeholders may attend, but must remain silent.

15 min top. Each answer 3Qs. No discussion, follow up after.

Each sprint timeboxed. Team can get outside help, but outside cannot tell team how to build their software. If overcommitted must tell PO immediately if sprint in danger. PO works w/ stakeholders to reset expectations.

Run out of work? Bring in more.

End of sprint, hold demo / sprint review meeting w/ stakeholders/users. Only demo items completely finished. Only present functional working software, nothing intermediate, like diagrams / schemas.

Post Sprint retrospective.
What went well? + non functional items to
How can we improve? backlog.

Keeping the team committed

- Never push members away from planning.
- Don't allow Scrum Master to make the plan, nor keep to the plan, the plan is made by the team.
- This is encouraging the team not to plan, as someone else will do it - leaves the team feeling less responsible, and less committed to the commitments they'll make.

Individual members don't commit to individual tasks assigned to them, everyone as a whole commits to the sprint.

Scrum Values:

Each person is committed to the project's goals.
Team members respect each other.
Everyone is focused on the work.
The team values openness
Team members have the courage to stand up for the project.

Basic pattern for scrum outlines team roles & meeting patterns.
But to really "get" Scrum, the team members need to go beyond just implementing practices - they need to understand & put into practice self-organisation & collective commitment
Teams are delivering valuable software & feel committed to the product outcomes.
Teams need to internalise scrum values to be effective.
Teams self-assign their tasks, when they're done with the current one. Done during current daily stand-up.

Much easier for team to spot problem tasks than project manager.

How to hold an effective daily scrum

Act like you're committed to the sprint goal.
Make sure everyone is feeling accountable and committed, have we collectively been meeting previous commitments?

Take detailed status meetings offline.
Identify problems - don't solve them. Only 1 min discussion.

Take turns going first. No specific person is "keeper" of the schedule, nobody more important than somebody else. Keep everyone listening by mixing it up, and hearing from everyone.

Don't treat it like a ritual
Everyone needs to be present & engaged.
Rituals fade over time because people treat them as perfunctory.

Everyone participates PO also answer 3 Qs.
Testers, BA, PO, anyone else. All equitably committed.
PO highlights what's especially important to users/company

Don't treat it like a status meeting

Keep 2 purposes at forefront of everyone's mind.
- keep everyone on the team informed
- keep management informed.

Make sure this isn't just one way communication, and not always project manager asking questions.

Inspect Every task

Don't look for roadblocks in what you're currently doing, look ahead.

Change the plan if it needs to be changed

Part of visibility - inspection - adaptation cycle, makes team self-organising.

Breaking a project into deliverable phases & incremental development, but Scrum is about understanding the value that the software delivers, and changing course if there's a way to deliver more value. Iterative Development.

Product Owner makes or breaks the sprint.

- Understands what company needs most, brings knowledge back to team.
- Understands what software features the team can potentially deliver.
- Figure out which features are more valuable to the company
- Work out w/ team which features are easier/harder to build,
- Use knowledge of value/difficulty/uncertainty/complexity to help team choose the right features to build in each sprint.
- Bring that knowledge back to the rest of the company, to help them prepare for next release of the software.

Owns the Product Backlog
Helps team decide what goes into sprint backlog.
PO must have a lot of authority, and use it.
Needs a good sense of what's valuable to the company

Important for team & PO to agree on what definition of "Done" is.
"Done" means accepted by the PO & delivered to the rest of the company. Without this definition, so much confusion arises. With a clear definition, everyone easily agrees on current state.

Team doesn't get credit of the item isn't "Done".
Item goes back into the product backlog.
EVERYONE is responsible for not getting that task "Done".
Valuable because it makes sure the team never gives the users the impression that they delivered value that wasn't actually delivered, ie. "Done".
Better to be conservative about what commitments were delivered.
During sprint review, team has to look the users in the eye and say what they've delivered, and what they have not - keeps the team feeling a sense of collective commitment.
Even though users & team talk & collaborate, it's the PO who actually accepts the work on behalf of the company.

Rarely in case of poor planning or unforeseen circumstances, PO can halt the sprint & reset everything. This breaks trust with the users & company, last result.

Elevating Goals motivates Everyone on the team.

Paying people isn't enough to get people to really "care".
You can buy someone's time & effort, but ingenuity can only be given willingly.

The leader's job, really, is to try to frame the activity in such a way that people can understand the value of it. They need to feel that their work is important.

On an agile team, software "value" is some measure of making your user's lives better.
Someone using your software & saying thank you. is much more rewarding than a company executive saying "this helped the company earn more money."
Devs are highly motivated by pride of workmanship.

#1 Agile Principle:
Our highest priority is to satisfy the customer through early & continuous delivery of VALUABLE SOFTWARE.

most effective motivation for teams is making them believe they are providing value.

How to Plan & Run an Effective Scrum Sprint.

Start with the backlog - which means starting w/ the users.

Be Realistic about what you can deliver.

Change the plan if it needs to change.

Get everyone talking about VALUE

VALUE

Throwing everyone into the same daily scrum, just because they're on the same team, although not on the same project.

BAD.

Planning at the last responsible moment encourages everyone to accept the constraints as they arise, and not set arbitrary constraints. Team stays open to change.

User stories: putting yourself in the user's shoes, envisioning them using the software for a reason. So you don't build something they don't want / won't use.

"As a <type of user>,
I want to <specific action>,
so that <achieve end goal>."

Explains WHO, WHAT, WHY.

Team has to understand why we use user stories. Otherwise the concept seems stupid.

Take user stories, define definition of done.
Break down into tasks, estimate size.

Story Points & Velocity Fuck this book's ideas.

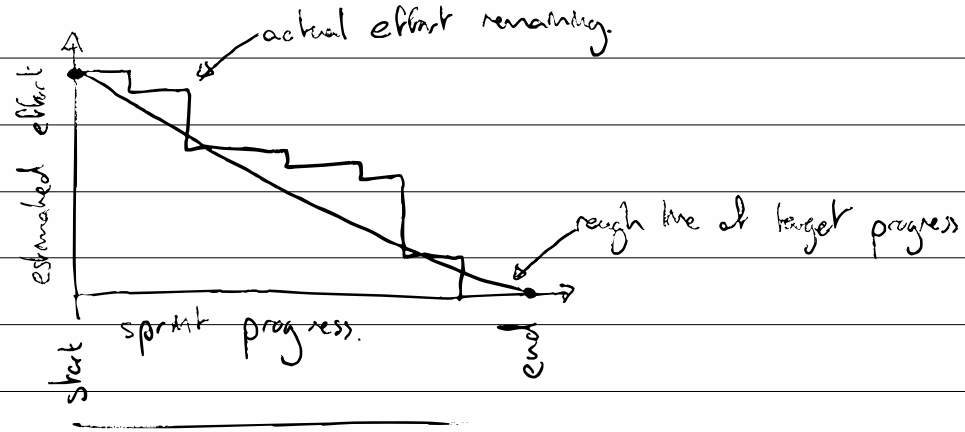
Compare story to past stories

Numeric story points are a fucking terrible idea.

Too much temptation for comparison, across sprints, across teams.

Would be great to measure using "development hours", NOT "employee hours". Only works in an environment of trust, where team doesn't feel as though they're being micro managed. Can then have some idea of inefficiency. Need to make it clear that this isn't a punishable metric.

Burndown Chart



If you don't get self-organising teams & collective commitment, you don't get scrum.

Is the team Ready for Scrum?

Ready for commitment? Ok with:

- trusting team to decide what gets delivered
- no "single writingable neck"
- listening to comments & feedback
- whole team willing to take responsibility for successes & failures? Does everyone on the team feel the same.

Ready for respect? Ok with:

- trusting team with doing the right thing
- give the team enough time to do work, not demanding overtime.
- trust team to choose tasks right for them & project, rather than relying on strict roles & RACI matrix.

Ready for focus? ok w/

- never asking someone to do work that's not part of the sprint.
- not asking someone on the team to take on work that not everyone has agreed to.
- Putting what's most valuable to the company at top of the pile
- Not being able to demand that the team works on tasks in a specific order.

Ready for Openness? Ok w/

- Devs thinking about business value, and the work of everyone else on the team in these terms.
- Everyone discussing progress towards project's goals?
- Everyone thinking about users? about project planning?

Ready for Courage? Ok w/

- Not being able to blame lack of planning on project manager?
- Not being able to blame poor requirements that don't hit the mark on your Product Owner or Senior managers?
- Taking time to really understand your users?
- Building something that isn't perfect, because your users needed something "good enough"?

If these are all yes, Scrum is a go!
Otherwise, there's work to do on cultural values.

Sprint Demo must always show something.
Even infrastructure & DB changes. (Non-functional)

Successful scrum teams have members who talk about "WE achieved". "OUR blockers", instead of me & my tasks.

Sprint Planning, Daily Stand Ups are not enough for high performing teams. It's communication & collaboration.

Instead of person by person, they go through the sprint backlog highest to lowest priority. End when daily scrum timebox expires.
Keeps everyone focused on delivering value.
Daily scrum is a formal inspection meeting where the goal is improve the quality of the team's planning & communication.

Important to keep the daily scrum short, concise, focused on goals & communication.
Format is loose though, can be
- person by person, 3 Qs
- browse sprint backlog
- one person present current state of sprint (different person each time), everyone else challenge assumptions & ask questions.

Don't let individuals "own" tasks / stories.

XP and Embracing Change

Primary Practices

- Programming Practices
- Integration Practices
- Planning Practices
- Team Practices

---

(1) Test First Programming - higher quality in software.

(2) Pair Programming - more eyes on code, catch bugs before they're written

One person types, the other observe.

Reduce fatigue, can swap who's typing.

Less likely to get distracted!

Constant discussion & brainstorming. More innovative.

Hold each other accountable w/ good practices.

(3) 10 minute build - entire codebase + run unit tests.

longer than 10 mins, team is less likely to run it.

Quick answer to "Is our code working so far?

④ Continuous Integration

Lets many people work on single set of source files simultaneously. Can be difficult actually integrating changes to codebase though, if multiple changes are in-flight.

Useful to have a physical "build-token", like a rubber duck, to indicate which pair currently holds authority for this.

Planning Practices are similar to Scrum.

- weekly cycle
- planning meeting ⑤
- stories
- estimation

Also quarterly planning to look at big picture, themes. Discuss internal & external problems they're experiencing. Long standing bugs, repairs. Like a retrospective, but with a wider scope.

⑥ Slack - adding minor stories to the iteration, only to be done at the end. Can easily cut this slack if there are problems. Still only deliver "Done" software that can be demo'd to stakeholders.

Team practices

⑦ Sit together - have easy access to your colleagues

(8)

Informative Workspace

Work environment set up to automatically communicate important info to anyone working on the project. - Task board
"Information radiators". - Burndown Chart
- Blocked Tasks

Embrace change.

When change comes along, either the business has found a new opportunity or changed course, which is great for the company to follow the better direction. Or the user didn't know what they wanted initially. They aren't changing their mind on purpose, the just now have a fuller idea of what the software should do.

XP Values:

- Communication: everyone's aware of everyone else's work
- Simplicity: devs focus on working most simple & direct solutions possible
- Feedback: Constant tests & feedback loops keep quality of the product in check
- Courage: everyone makes best choices for the project, even if discarding old work
- Respect: Every member is important & valuable to the team.

If values aren't internalised by everyone, you get responses like "it won't work for our team" without any concrete reasoning behind it. No argument against, just a feeling.

Are you OK with XP?

* Comfortable throwing out old code & ideas when you learn that it doesn't work? Is your boss?
* If boss wants a tighter deadline, does the team truly believe that writing unit tests is the fastest way to meet it? Even though that means writing more code?
* If a junior dev takes on a task, is the rest of the team ok to give them the authority to finish it? Even if someone thinks they can do it faster? Is it OK for the developer to fail & learn from it?

XP principles

Humanity - software is built by people. there's a balance between the needs of the people and the needs of the project.

Economics - somebody is always paying for the software, keep the budget in mind.

Mutual Benefit - search for practices that benefit the individual, the team, and the customer together.

Improvement - Be introspective, realise how you can improve.

Diversity - Lots of opinions & perspectives work together to come up w/ better solutions.

Flow - constant delivery means a continuous flow of development work, not distinct phases.

Opportunity - Each problem is a chance to learn something new about software development

Redundancy - Can seem wasteful at first blush, but avoids big quality problems.

Failure - you can learn a lot from failing, it's OK to try things that don't work.

Quality - you can't deliver faster by accepting a lower quality product.

Accepted Responsibility - If someone is responsible for something, they need to have the authority to get it done.

Baby Steps - Small steps in the right direction when adopting practices.

Principles of the methodology help you understand why implementations fail, and why the practices are important. Much more important that my team understands the values, agrees with the principles. Practices will fall into place, and can be correctly optimised, if the principles are there. Nobody would do a practice without a principle, and "somebody else does it" is a worthless principle.

11 Corollary Practices of XP

Real Customer Involvement
Incremental Deployment
Team Continuity
Shrinking Teams
Root Cause Analysis
Shared Code
Code & Tests
Single Codebase
Daily Deployment
*Negotiated Scope Contract

consulting companies. Fix time, negotiate scope. Get feedback about usage

Incremental deployment: deploy small peices of the system individually, rather than one big shot.

Keep effective teams together.
As teams become more efficient, don't give them more work, take one person away & move them to a team to help spread the values & practices of XP.

Ch 7.

XP, Simplicity & Incremental Design

Goal of XP is to build software that can be extended and changed easily

YAGNI - "Always implement things when you actually need them, not when you foresee needing them".

Easy to fall into the "framework trap" this way.

Being too clever, trying to solve problems that don't exist yet.

Only build a library or framework, if your goal was to write a library or framework.

Not as a consequence of the work you're doing.

Code smells increase the complexity of a project.

Incremental Design - making code design decisions at the last responsible moment.
Avoids solving bigger, tangential problems you're close to.

Energised work: establishing an environment where every team member is given enough time & freedom to do the job.
Keeps them in a mental space where they can develop & use good habits that lead to better, more naturally changeable code.

Coding is mostly a mental exercise. (Think how long it takes to create the code, vs. re-typing it out).
15 - 45 min to get into a state of "flow".
Un-energised work makes it impossible to get into flow state.

Only work 40 hours a week. Productivity goes down after.

Whole team - must trust each other & make decisions together.

Don't aim for 100% test coverage - the goal of a software team isn't to write tests, it's to create working software, and tests are a tool to aid that. Focus on tests more when you have issues appearing w/ quality.

---

* Talk about YAGNI.
  don't allow team to fall into framework trap focus on making decisions at the last responsible moment.

* Recognise when team member is uncomfortable w/ an XP practice (TDD or pairing), and help them get past their discomfort.

* Don't let over overly excited team member feel like an XP bully or zealot. Help them be patient.

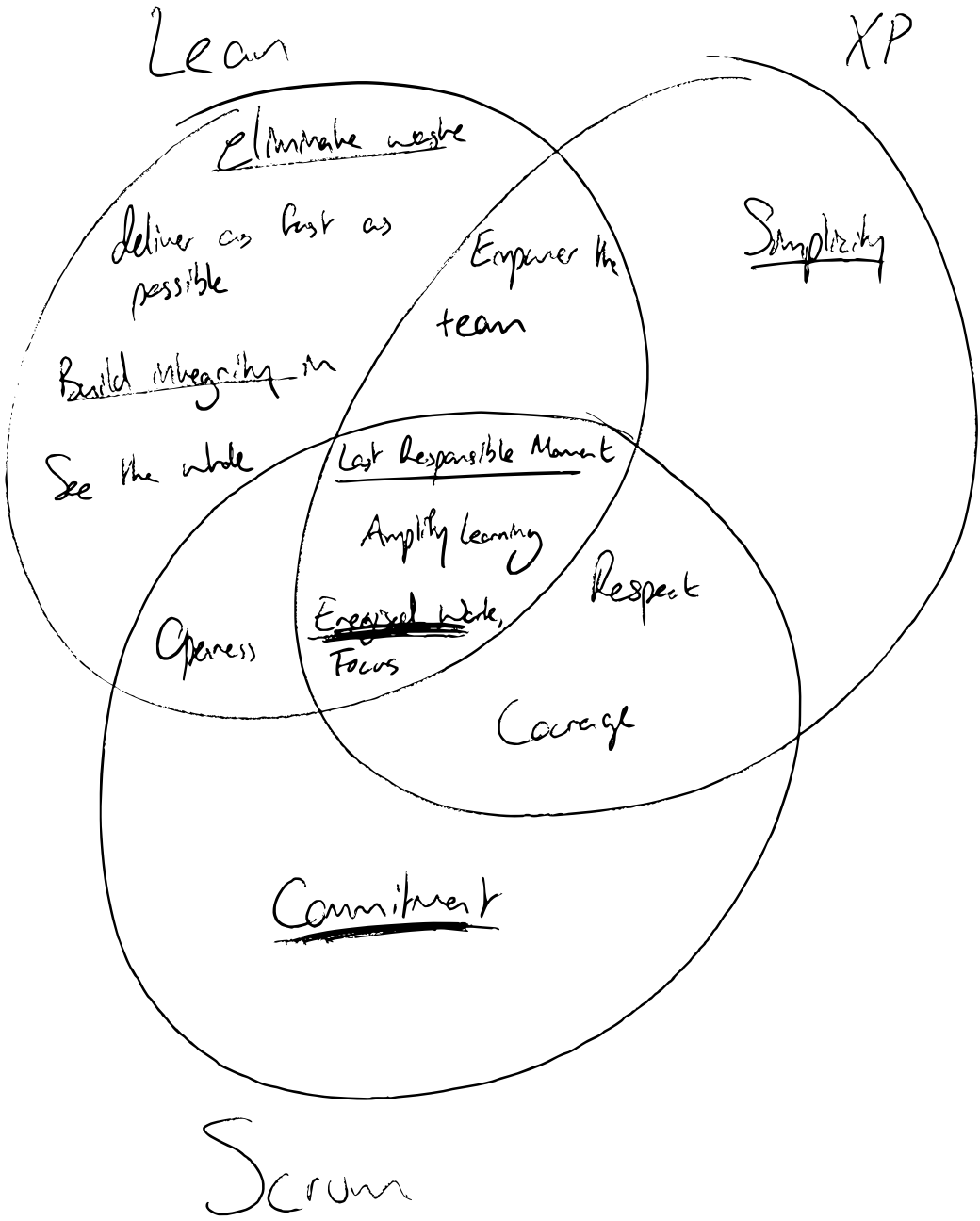
Ch 8.

Lean, Eliminating Waste, and Seeing the Whole.

Lean has no practices. Mindset of values & principles.
Thinking tools, lean thinking.

Lean Values

* Eliminate Waste.
  Find work you're doing that doesn't directly help create valuable software & remove it from the project.
* Amplify learning
  Use feedback from your project to improve how you build software.
* Decide as late as possible
  Make every important decision for your project when you have the most information about it - the last responsible moment.
* Deliver as fast as possible
  Understand cost of delay, minimize it using pull systems & queues.
* Empower the team
  Establish a focused & effective work environment, build a whole team of energised people.
* Build integrity in
  Build software that intuitively makes sense to the user, forms coherent whole.
* See the whole
  Understand the work on your project. Right kind of measurements to see clearly


Lean
XP
Eliminate waste
deliver as fast as possible
Empower the team
Simplicity
Build integrity in
See the whole
Last Responsible Moment
Amplify learning
Respect
Openness
Engaged work, Focus
Courage
Commitment
Scrum

Options Thinking - seeing the difference between something you're committed to, and something you have the right, but not obligation, to do.

Set-based Development.
Running a project in a way that gives you more options by having the team follow several paths simultaneously to build alternatives that they can compare.

In lean thinking, anti-patterns for running projects are waste. Waste is anything your team does that doesn't actively help them build better software.

Often hard to see wasteful activities as waste. Because they're almost always someone else's priority: PM, Contractor, Senior Manager.

Seven Wastes of Software Development.

* Partially Done Work.

Only deliver work that's "done done"; if it's not 100% complete, you haven't delivered value to your users.

Any activity that doesn't deliver value is waste.

* Extra process

time spent giving status updates for example, or creating estimates. Tracking & reporting time does not directly contribute to delivering software.

* Extra Features

Something nobody asked for.

* Task Switching

Scrum value of "Focus" helps see that switching between projects, or even between unrelated tasks on the same project, adds unexpected delay & effort: context switching costs cognitive overhead.

* Waiting

reviews, approvals, provisioning hardware, obtaining a licence. Waste.

* Motion

When not sitting together, team spends days / weeks walking to/from each other.

* Defects

Time spent fixing bugs later in SDLC that could have been found earlier by TDD & Quality measures. Waste.

ch. 9

Kanban, Flow, and Constantly improving.

Kanban is a manufacturing term, adapted for software.
Effective way to bring lean thinking into your organization.

Scrum primarily focuses on project management:
what will be done, when it will be delivered, whether it meets the needs of the user/customer.
XP focuses on software development.
Values & practices built around creating an env conducive to development, and developing programmer habits that help them design & build code that's simple & easy to change.

Kanban is about helping the team improve the way that they build software. Gives practices to help stabilize & improve your system for building software.

First follow fundational principles:
- Start at what you do now
- Agree to pursue incremental, evolutionary change
- Initially, respect current roles, responsibilities & job titles

Then adopt core practices
- Visualise
- Limit WIP
- Manage Flow
- Make process policies explicit
- Implement feedback loops
- Improve Collaboratively, Evolve Experimentally.

Kanban Boards vs Task Boards.

Tasks boards have tasks on them
Kanban boards have work items.

A work item is a single, self contained unit of work that can be tracked through the entire system.
Typically larger than a minimum measurable feature, requirement, user story, or other individual scope item.

Tasks flow across a task board.
Work items are <u>not</u> tasks.
Tasks are what people do to move the work items through the system. Tasks are the cogs of the machine pushing the work item through to completion.

Kanban isn't a system for project management.
It is intended to improve & change the process in use on the project: this can & will affect how the project is managed.
Kanban is typically used to improve predictability of flow, which will affect planning & scheduling
Extensive use of kanban & its metrics is likely to have a significant knock on effect on the method of project management.

Kanban: don't copy someone else's board. it's a representation of your own team's process.
Should have columns for each step of your process.
Helps visualise column where work is getting stacked.
The concept of a "blocked" column will serve to hide where in your flow those items are getting stacked.

Limit Work in Progress.
Once unevenness in the workflow has been identified, we can use it to control the amount of work that flows through the whole system. By placing a strict limit on the amount of work that is allowed to pile up behind it.
Set a limit on number of work items that can be in a particular stage in the project's workflow.

Setting a WIP limit for a step in your workflow means limiting the number of features allowed to move into that step.

WIP limit changes what the team will choose to work on next.

Measure & Manage Flow

Flow of the system is the rate at which work items move through it. When the team finds an optimal pace for delivery combined w/ a comfortable amount of feedback, they've maximised the flow.

Cut down on unevenness & overburdening.

A Kanban team uses the "manage flow" practice. By measuring the flow & taking active steps to improve it for the team.

How do you know you're increasing flow when you add WIP limits? Cumulative Flow Diagrams help (CFD).

Work items accumulate in later stripes & bars.

Look for patterns that indicate a problem: unevenness, loop.

CFD lets you look at the way your entire process is performing over time, so you can find & fix the root cause of any long term problems.

Longer lead times seem to be associated w/ significantly poorer quality. 6 fold increase in lead time, led to 30% increase in initial defects. Longer avg. lead times result from greater amounts of **WIP**. Management leverage point for improving quality is to reduce the quantity of WIP.

Use a CFD to experiment w/ WIP limits & manage flow.

Managing flow w/ WIP limits naturally creates slack. Devs need slack / wiggle room in the schedule. Needed to make sure they have time to do a good job, otherwise, rushed developers cut corners to meet deadlines.

Kanban teams adopt a delivery cadence, rather than timeboxed blocks. They commit to a regular release, but don't specify what will be in that release.

With a WIP limit, when the limit is reached, new items can't be added by stakeholders, they have to remove something. They see this not as a fault of the team, but a limitation of the system.

It's all about maximising flow.
Measure it & observe w/ WIP board & Cumulative Flow Diagram.
Set WIP limits to stabilise the system.
Kanban is Japanese for "signal card".

Kanban is not for project management it is for process improvement.
Need to know what the process is currently, and measure.

Teams use kanban boards a lot, to track items & update progress, but very much without actually using kanban - they treat everything like tasks, and don't put WIP limits in place to identify their issues.
The idea of tasks should be separated from kanban story cards: These are used for 2 different things.

Coaching Agile.

Need principles.

* Industriousness - it's hard to gain these new skills, requires hard work.
* Enthusiasm - gotta put your heart & soul in, it will rub off!
* Fundamentals - "the finest system cannot overcome poor execution of the fundamentals." KNOW these fundamentals, base every decision & argument on them, embody them.
* Development of team spirit - teamwork & unselfishness must be encouraged at every opportunity. Need to nurture trust & togetherness.