
DO AGILE RIGHT
All AGILE, all the time.
AT THE END … BUT WHY AGILE ???

BASICS ELEMENTS OF AGILE DEVELOPMENT

- Roadmaps
- Requirements
- Product Backlogs
- Stand Up Meeting
- Retrospectives



Roadmaps

- Allow the organisation flexibility to plan as the business evolves
- Encourage appropriate near-term prioritizations, given the business' future direction
- Let product teams anticipate upcoming work that may influence decisions they make today



Common pitfalls

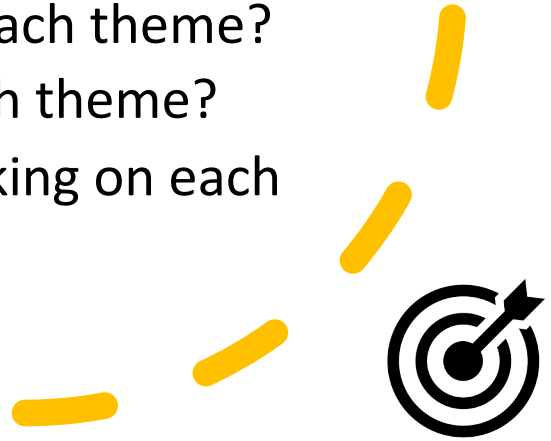
- Future planning is completely ignored — we're shooting' from the hip!
- The "rest of the business" is kept in the dark as to what the team is up to.
- The roadmap is continuously updated (or never updated).
- Detailed requirements weigh down the roadmap.



Do roadmaps right

Although there are many ways to do roadmaps right, most successful teams stick to three basic principles to avoid doing it wrong:

- Keep a single, canonical copy of the roadmap and make sure it's easy for all to access.
- Focus the roadmap on high-level themes, rather than burdening it with detailed requirements.
- Answer these questions in your roadmap:
    1. What are the relative priorities of each theme?
    2. When do we intend to work on each theme?
    3. How many team members are working on each theme?



Requirements

- Minimise the pain of making course corrections as feedback rolls in
- Let product owners spend more time on customer interviews and competitive analysis, less time editing documents
- Get devs into the code as soon as possible

Common pitfalls

- The entire project is spaced out in great detail before any engineering work begins
- Review and sign-off from all teams is required before work even starts
- Team members don't know when requirements are updated
- Requirements are never updated in the first place (because everyone signed off, remember?)

Do requirements right

Although there are many ways to do agile requirements right, most successful teams stick to these principles to avoid doing it wrong:

- Go high level at the start of the project, then add detail for features as the development team gets ready to tackle each one.
- Minimize requirements documentation — save the details for the user stories and tasks in your issue tracker (that's where the people implementing them will be working from anyway!).
- Whatever requirements documentation you must have, keep it in a wiki or shared repository so team members are notified of changes.
- A willingness to let the requirements evolve as feedback from user testing, stakeholders, and engineering (yes, even engineering) rolls in is critical for succeeding with agile.

Product Backlog

- Radiate upcoming priorities to the immediate team and stakeholders
- Provide a foundation for iteration planning
- Reflect a combination of customer value, business merit, and engineering effort

Common pitfalls

- The backlog is prioritised at the start of the project, and not adjusted as feedback rolls in from devs and stakeholders
- Items on the backlog are limited to those that are customer facing
- The backlog is kept as a document that is stored locally or shared infrequently, ensuring no one gets updates

Do product backlog right

Although there are many ways to do agile backlogs right, most successful teams stick to these principles to avoid doing them wrong:

- Make sure the backlog is easily visible to all contributors and stakeholders on the project.
- Sort items on the backlog in order of priority, with the top-ranking items at the top, so understanding what's most important is intuitive for others.
- Add items and re-shuffle rankings as new information comes in. A living, dynamic backlog is an effective backlog!
- Include all the team's work, even work that the customer will never see, so you can plan iterations effectively. If it'll require time from a team member, it goes on the backlog.

StandUp meeting

- Understand the work going on immediately around you
- Communicate any blockers to keep things flowing
- Get in, get out, and get back to work

Common pitfalls

- Discussions about how to approach a technical problem drift into the weeds.
- People outside of the immediate team use the standup as a forum for announcements and updates.
- Stand-ups take more than 10 to 15 minutes.

Do standup meeting right

Although there are many ways to do standup meetings right, most successful teams stick to these principles:

- Empower the scrum master (and all team members, for that matter) to politely interrupt a discussion that's gone awry, and bring the standup back to its central focus. A non-verbal signal or chime can work well for this, too.
- Follow a consistent pattern for updates such as "Since last time, I... My plan for today is... I have/don't have any blockers right now." This helps keep stand-ups moving along, and to the point.
- Bring in remote team members using a conference call or video chat so everyone is included.

Retrospective meeting

- Continuously improve on processes and technical practices
- Provide a reliable forum for raising concerns
- Build team trust & cohesion



Common pitfalls

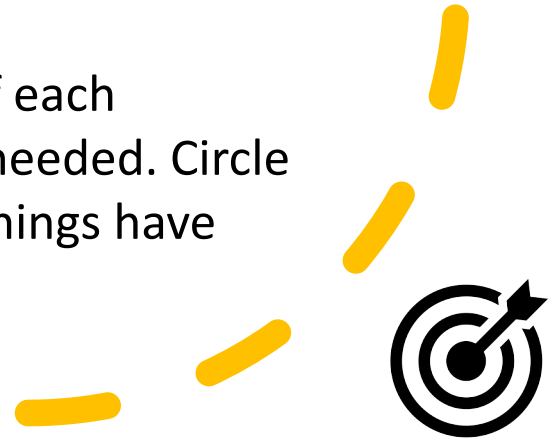
- Lots of finger pointing happens when things go awry
- Scrum master does all the talking (perhaps with a slide deck thrown in)
- Retrospectives are held inconsistently and infrequently
- No follow-ups are identified or committed to



Do retrospective meeting right

Although there are many ways to do agile retrospectives right, most successful teams stick to these principles:

- Hold retrospectives regularly. The last day of an iteration is ideal for scrum teams. Kanban and lean teams do well with release or project retrospectives.
- Use exercises like The 5 Whys to autopsy things that aren't going well. Understanding root causes = opportunities for remedying them. Blaming = death sentence for team morale.
- Let the individual contributors on the team do most of the talking when it comes to raising issues and exploring solutions. The scrum master or team lead is primarily there to facilitate and provide structure.
- Agree on "stuff we're gonna do" at the end of each retrospective and assign action items where needed. Circle back to them in a future session to see how things have progressed!



Scrum

- Team Structure
- Meetings
- Estimation Techniques
- Tracking Progress



Team Structures

- Keep everyone working on the project in close contact and in sync
- Work on a feature's different layers (graphics, back end, code quality) in parallel
- Create a close-knit team with a high level of trust and morale

Common pitfalls

- Team membership changes from sprint to sprint based on what work is undertaken.
- Team membership is based on discipline or role, rather than product (ie, there's a dev team, and a QA team, and an ops team, and a design team).
- The functional manager is filling the role of scrum master and/or product owner.

Do team structure right

Although there are many ways to do team structure right, most successful teams stick to these principles:

- Choose the team's membership carefully and keep it stable, even when project work ventures outside their usual area of expertise.
- Include a cross-section of disciplines on the team: developers, testers, graphic designers, DBAs, sys admins and any other roles needed to complete the project.
- Fill the roles of scrum master and product owner with people not doing the day-to-day project work. A dedicated scrum master and product owner for each team is ideal.
- Avoid using the team's functional manager as their scrum master. An agile manager's primary job is building and growing their teams — not overseeing the team's day-to-day work.

Meetings

- Keep stakeholders updated on progress
- Create feedback loops for the team
- Set the short-term direction for the team's work

Common pitfalls

- Meetings are scattered throughout the day, with only 30-60 minute gaps for project work.
- Even those who aren't critical for the decision(s) at hand are required to be present for informational purposes.
- Ceremonies are scheduled as needed, or with an inconsistent cadence.
- Meetings are too long and boring

Do scrum meeting right

Although there are many ways to do scrum ceremonies right, most successful teams stick to these principles:

- Cluster meetings together when several occur on the same day (such as the demo and retrospective on the last day of the iteration).
- Invite stakeholders or members of other teams to meetings when absolutely necessary, but always keep the list of required attendees as short as possible.
- Hold scrum ceremonies on a regular cadence.
- Ask yourself whether a desk drive-by or quick huddle around the coffee maker might work faster before calling ad-hoc meetings.

Estimation techniques

- **Provide estimates that are accurate enough that they can be factored into prioritisation and planning**
- **Identify places where requirements aren't clear enough to gauge the work**
- **Learn from prior sprints and build estimation as a skill**



Common pitfalls

- **Most issues are "spiked" before the team can/will estimate it.**
- **Estimating stories and bugs takes at least 10 minutes apiece.**
- **The functional or product manager does all the backlog estimation without input from the team.**
- **Stakeholders pressure or criticise the team when items take longer than estimated.**



Do estimation right

Although there are many ways to do estimation right, most successful teams stick to these principles:

- **Invite input from all disciplines involved (development, QA, design, ops, etc) when estimating work.**
- **Set a time limit for how long items are discussed before calling for an estimate. Talking an item to death can actually lead to confusion and a less accurate estimate.**
- **De-scope items whose estimates sound "too big" rather than badgering the team into more palatable answer.**
- **Remember these are estimates, not blood baths. Work will take longer than expected sometimes. It's the third most unavoidable thing in life after death and taxes, so stop stressing.**



Tracking progress

- Track progress towards releases in a way that's accessible to all
- Normalise on a comfortable, yet challenging, sprint length and capacity
- Build stakeholder confidence in the team's ability to deliver

Common pitfalls

- Sprints drag on until the original committed work is finished.
- Product owners constantly change the team's sprint priorities during the sprint.
- Estimates are never discussed during retrospectives.

Tracking progress right

Although there are many ways to track progress right, most successful teams stick to these principles:

- Ensure sprints are always the same length; try two weeks. Longer sprints are more likely to veer off track.
- Abort the sprint when things go so far astray that the team cannot recover. (It happens to even the best of teams!) Take some time to understand what went wrong and start over.
- Use instruments like a burn-down chart or task board to radiate successes and struggles.
- Resist the temptation to compare teams by points or items completed — everyone calibrates velocity a bit differently.

Kanban

- Team Structure
- Flow and pull
- Work in progress



Team structure

- Keep everyone focused on their incoming queue
- Allow ownership of items to be fluid until the work actually starts
- Limit the impact to the team when priorities on the backlog change

Common pitfalls

- Issues on the backlog are assigned to team members in advance.
- Temporary team members are pulled in to fill in skill gaps for a project or issue.
- Work piles up on one or two team members who are experts in a particular area.

Do team structure right

Although there are many ways to do team structure right, most successful teams stick to these principles to avoid doing it wrong:

- Choose the team's membership carefully and keep it stable, even when project work ventures outside their usual area of expertise.
- Select the team based on the type of work in the backlog. You'll want some redundancy in the team so you can scale if any type of work starts becoming critical path.
- Focus on optimising the work of the individual as well as the team by looking for trouble spots that impede flow.

Flow & Pull

- Accept new work only when there is capacity for it
- Identify and correct bottlenecks
- Establish a clear separation of duties between developers and product owners

Common pitfalls

- Work in progress is so limited that only small items can be accommodated and larger projects are ignored.
- Team members are consumed by a project for days or weeks because they aren't broken into smaller pieces.
- The test team gets blamed for all bottlenecks in the process.
- Work is pushed into the active queue rather than pulled from the backlog.

Do flow & pull right

Although there are many ways to maximise flow, most successful teams stick to these principles:

- Choose the workflow for the team carefully. Ensure that each state has a distinct function in the development process.
- Empower the team to pull the most important items from the backlog or an upstream state; healthy teams want to get work done.
- Develop a trusting relationship between the team and the product owner. As the team delivers the the highest priority items, the product owner has confidence in ordering the backlog.

Work in progress

- Size work items in small, manageable chunks
- Distinguish stages in your workflow by discipline (e.g. development vs. quality review)
- Emphasise sustainable and scalable practices

Common pitfalls

- WIP limits are raised as needed so the team doesn't hit them anymore ("debt ceiling," anyone?).
- Everyone has a large "background task" on their plate to mask times when they'd otherwise be idle.
- Team members sit idle waiting for more work to pull in, rather than swarming on bottlenecks.
- Throwing more person-hours at persistent bottlenecks is preferred over improvements in engineering practices or team processes.

Do WIP limits right

Although there are many ways to manage work in progress, most successful teams stick to these principles:

- Allow the product owner to determine the order of work, but empower the team to decompose items into 8-16 hour tasks so WIP limits highlight trouble early.
- Establish consensus on what the team's core skill sets are, and how they map to each stage of the development process.
- When a team member is idle, encourage them to help an upstream or downstream team member.
- Take time out to look at areas of chronic idleness or overload, and get feedback from the entire team.

Delivery

- Testing
- Technical debt



Testing

- Identify test cases as soon as possible after requirements are defined (ideally before coding begins)
- Test as soon as possible after the code has changed
- Make everyone on the team owners of qualities

Common pitfalls

- A "testing sprint" is scheduled before each release (or designated testing days at end of sprint).
- Developers use formal hand-off documents to tell QA how and when to test
- Items not tested during the sprint are marked as "done" (but have a testing task added to the next sprint).
- Testing is largely manual.

Do agile testing right

Although there are many ways to do testing right, most successful teams stick to these principles to avoid doing it wrong:

- Focus QA's efforts on understanding the intended functionality, the level of risk involved, and speccing out test cases to cover them that anyone can code to or execute.
- Automate, automate, automate! Developers already have the skills to write test code, and nobody ever regretted training their QA team to do it, too.
- Use code comments and renderers like javadoc or rdoc to represent test plans instead of spreadsheets. The code is the documentation.
- Run automated tests against every build, and build several times throughout the day — a practice known as continuous integration — so you discover new defects when they're easiest to fix (which is now).

Technical debt

- Reduce or eliminate technical debt over time
- Think strategically about what qualifies as true debt vs. structural improvements
- Treat the prevention of technical debt like the low-risk, high-return investment it is

Common pitfalls

- Debt pay-down and prevention is shifted to a "minor release" in favour of launching a feature sooner.
- Technical debt remediation is kept (hidden, even?) in a separate backlog.
- "Hardening sprints" devoted exclusively to technical debt are commonplace.

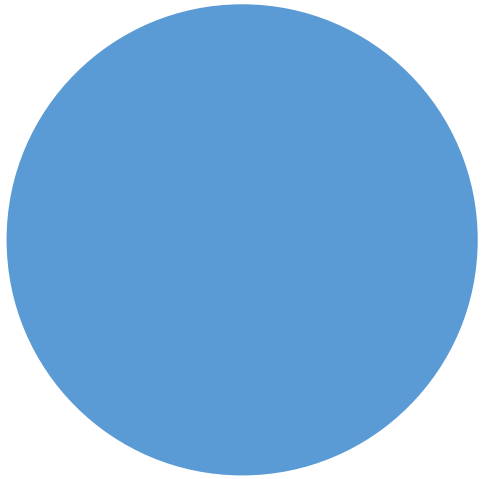
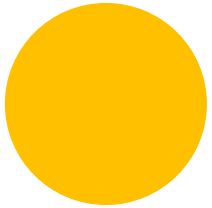
Do technical debt right

Although there are many ways to do technical debt right, most successful teams stick to these principles to avoid doing it wrong:

- Like any other debt, budget for technical debt in your release and iteration planning.
- Track technical debt in the product backlog. Addressing it is most certainly delivering value to customers, even if indirectly.
- Update the definition of "done" as needed to prevent debt in the first place.
- Don't obsess over whether to give velocity or story point "credit" to technical debt items — just fix the damn thing.

SCRUM		KANBAN
A Scrum Master, Product Owner and Team Members make up a Scrum Team.	**Team Roles**	No set roles are defined. Roles are not required to be cross functional.
Velocity	**Key metrics**	Lead time, cycle time, rest time, WIP
Idetified by team in retrospective	**Bottleneck**	Identify by manager thouw metrics analysis
Columns are labeled to reflect periods in the work flow from beginning through the team's definition of done.	**Work Boards**	Columns are likewise labeled to show work flow states, but also publish the max number of stories allowed in column at once.
Scrum processes place heavy emphasis on schedule with a prioritized list of story points. This iterative process enables accurate estimations of work flow and effective management of multiple projects.	**Scheduling/Cadence**	There are no required time boxes or iterations. While the Kanban method is iterative in nature, the continual improvement is expected to occur in an evolutionary fashion as work is continually completed.
Best for teams with stable priorities that may not change as much over time.	**Best Applications**	Best for projects with widely-varying priorities.

WATERFALL
specifications provided by the customer and blocked by contract



THANK YOU !!!