Metrics

Measuring Agile Maturity and Mindset Adoption

Contents

Agile success can be measured by decreased cycle time, a decrease in generated defects, no increase of technical debt, and increased efficiency of resource use.

However, there are many ways to measure agile: agile adoption, agile maturity, and agile mindset: all important to identify where help is needed.

This document present some options for adoption maturity and mindset.

Net Promoter Score Measures for Agile Maturity

Anonymously ask: "Would you recommend xxx to your peers?" where xxx is each topic below. Rated as follows:

10-9 Yes! 8-7 Neutral 1-6 No way!

Calculate the NPS % per team for each topic:

((# of 9's and 10's) – (# of 1's through 6's))/Total number of responses

Agile Maturity Topics

Sprint Burndown Usage

Release Burndown Usage

Sprint Planning

Velocity

Information Radiator

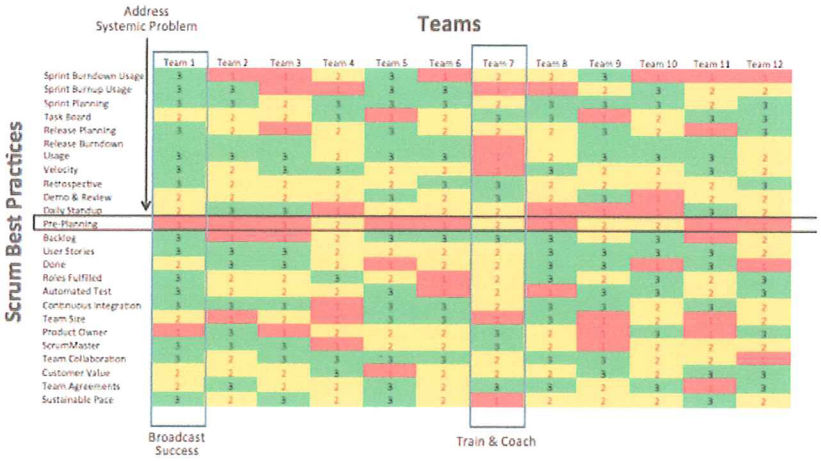
Retrospective

Release Planning

Daily Standup

Pre-Planning	Team Size
Backlogs	Product Owner
User Stories	Scrum Master
Done, Done, Done	Team Collaboration
Roles Fulfilled	Customer Value
Automated Testing	Team Agreements
Continuous Integration	Sustainable Pace

Quantitative Scrum Best Practices – Heat Map



Measuring Agile Principles

- Customer Satisfaction: Do they have one and if so, is it moving up the scale NPS and customer actively involvement during iterations.
- Embrace and welcome change: The amount of new stories and/or priorities changed during development cycle. Development process: number of process changes implemented as a result of the reflections.
- Deliver working software frequently: Increase in number of successful test cases run. Technical debt is decreasing or small at the end of each iteration and thumbs up from testers. Low numbers of open design issues.
- Business people and developers work together daily: Is development team getting a response in 30 minutes. Business active in story prioritization for each iteration.

- Build projects around motivated individuals. Support and trust them: NPS score of management system by dev.
- Face-to-face communication is best: Percentage of co-located teams.
- Working software is measure of progress: Length of the iterations - short is better. Decrease of time to value: shorter release cycles.
- Sustainable development with a consistent pace: relatively consistently velocity with room for exception handling. Amount of overtime.
- Attention to technical excellence: Adoption rate of TDD, Continuous integration, skill sets as evaluated by team members of each other.
- Simplicity is essential: Usefulness of Epics from POs measured by customer feedback per iteration.
- Best designs and requirements emerge from self-organized teams: Team owns the solution measured by: Joint design sessions and full participation of team in those sessions.
- Team reflects regularly on how to improve: reflections held at the end of every iteration and the number of actions implemented.

Additional Consideration:

- Some of the measures can really apply to more than one of the principles. e.g. a measure for embracing change is continuous iteration re-planning including both development and business - which also covers "Business and Development work together daily"
- Delivering Software Frequently is really closely tied to iteration length and low technical debt. Low numbers of open defects and design issues only mean something if the testing is good. In addition to successful test-case growth and test thumbs up I like code coverage to validate the testing.
- Motivated individuals: NPS of the management system is good to measure the developers' view of the support they get - but you can extend it by NPS of the managers. I suspect a Morale measure might be helpful in assessing individual motivation.
- The amount of refactoring is quite a good measure of technical excellence - but tricky to interpret. Both too little and too much are bad. Low numbers of field problems and field defects are a good measure of technical excellence but it's a very lagging indicator.
- Self-organizing teams may need another metric for the self-organization. Essentially - do team members own their own work allocation?

Agile Mindset

Do team members agree or disagree on the following:

- My team is motivated and delivering high value, high quality software on a regular cadence
- My Team is learning and getting better at delivering high value, high quality software on a regular cadence
- I would recommend this team to a friend or colleague.
- The team is prudent with setting expectations and manages itself to deliver on those expectations.
- The team expects risk and uncertainty and the team manages that uncertainty well.
- The team has a means of prioritization and works on the highest priority items first whenever possible.
- The team has access to the business and domain experts as needed to efficiently clarify what is needed.
- There is good visibility of progress so that the team and stakeholders understand where they stand towards the next major release.
- Dependencies between my team, other development teams and other groups in the enterprise are managed efficiently and effectively.
- The team feels empowered to decide how to solve their challenges of delivering.
- All members of the team are contributing and working to make the team better
- The team has the tools necessary to develop, test, deploy and to manage the process to deliver high value software.
- The team's technical practices (continuous integration, unit testing, coding practices, test automation, etc.) are solid.
- The team is keeping quality at a high level by ensuring the defect and testing backlog is not growing and that code technical debt is not increasing.

Agile Red Flags Agile Adoption Red Flags

Scrum Process

- Concept of "We don't plan" at the beginning of iterations
- Teams don't pick or commit to what they do in a sprint.
- Story's 'size' is too big to be completed in one iteration
- Blocked issues don't get raised or resolved leading to waiting
- Same impediment keep coming up again and again
- No definition of done, done, done or teams not committed to
- The demos repeatedly show no real valuable progress.
- No retrospectives. "We don't need to do retrospectives because we know what is wrong." Leading to no actions and no improvements
- Teams are date driven
- The fun ends
- Teams not understanding why they are doing the work

Quantitative

- They can't produce a release burn up chart
- Do they monitor the release burn up chart?
- Defect backlog is growing.
- Sprint burn down chart stalled
- Technical debt growing instead of diminishing.
- Shipping products stops
- No updating of information radiator.
- Support costs are not decreasing ($)
- Automation - build/release/integration times now vs. prior to agile
- Customer satisfaction decreases
- No early customer feedback that is supportive (NPS)

Organization and Culture

- "We have OUR agile.
- Teams 'burning out' e.g. how much overtime etc. takes place
- No collaboration
- Development still rules the roost - QA, ID not fully engaged in every iteration.
- Handoffs are in document form instead of people actually talking to each other
- Concept of "You can't do that. It's not agile."
- Missing support for the team to help sort out the bigger organizational blocks and finding ways to help the teams.
- Teams attempt the agile mechanics and say "we are agile!"
- Ownership of Agile adoption/transformation in a central place as opposed to the teams and leadership structure.
- If the organization is not continuously learning and improving its not agile.
- Team is afraid of speaking up in retro and pretending everything is fine

Personnel Behaviors

- Thrashing and finger pointing
- "You're not following the rules."
- Questioning ROI
- Managers find work for teams
- Manager controlling release and iteration planning
- Manager doesn't support scrum master to remove impediments
- Middle manager speaks out of both sides of their mouth (to executive management: "we are doing agile and team is agile". To teams "just do what I say")
- Leadership comparing velocity between teams
- PO can't define business value
- PO lies or is 'missing in action'
- PO does not participate in prioritizing backlogs
- PO keeps saying "I don't understand why this is taking so long"
- PO pushing for late additions to be included in the shipment - overloading the later iterations or extending the schedule rather than not shipping the lower value stories and holding the delivery date.
- Team does the agile mechanics and says "we are agile!"
- PO & Business judge the demos as showing no real progress

Attached Examples

- Agile Maturity and Adoption
- Agile Mindset

Response type Team summary
Team Name: Wildcats

Dates: 6/6/2014 10/15/2014

Weight	Focus	Question	Last lowest reply across team	Current lowest reply across team	Trend	Priority to address (1 = highest)	Team Plan to address or help needed (top 3 priortity)	Notes/guidance on answering the question and suggested evidence
2	**Delivery**	**My team is motivated and delivering high value, high quality software on a regular cadence**	Strong Agree	Agree	↘	1	Want to make our release management processes slicker so we can release more often	Guidance is we are releasing finished working software to STAGEor PROD within 2 months that can be used for demo/overall feedback.
2	**Improvement**	**My Team is learning and getting better at delivering high value, high quality software on a regular cadence**	Agree	Neutral	↓			Generally the team is effectively retrospecting and putting improvements in place that increase velocity and/or quality.
2	**Team**	**I would recommend this team to a friend or colleague.**	Agree	Strong Agree	↗			This is a good indicator of team morale, autonomy and effectiveness of communication .
1	Expectation Management	The team is prudent with setting expectations and manages itself to deliver on those expectations.	Strong Disagree	Agree	↑			Are you ableto plan effectively, track progress and deliver on commitments , for instance in an iteration and mid-range planning cycle. Iteration and Release healthcheck metrics in e.g. Rally will help you answer this ine.
1	Risk Management	The team expects risk and uncertainty and the team manages that uncertainty well.	Disagree	Strong Disagree	↘			This incluses the management of delevlopement archictural risk and any external risks/issues outside your immediate control.
1	Prioritization	The team has a means of prioritization and works on the highest priority items first whenever possible.	Neutral	Neutral	→			The team has a clearly defining rank for work items just in time for mid range planning, iterations and/or Kanban
1	Domain Experts	The team has access to the business and domain experts as needed to efficiently clarify what is needed.	Agree	Neutral	↘			For instance can get prompt and clear replies to questions relating to acceptance criteria, gets completed stories reviewed on time and gets regular customer feedback.
1	Tracking Progress	There is good visibility of progress so that the team and stakeholders understand where they stand towards the next major release.	Strong Disagree	Neutral	↗			This could be a form of burnup chart based on a assumed velocity. For the mid range plan and longer the degree or accuracy is understood. Generally conversations are happening around the information to steer prirotisation and the scope/time/resource balance.
1	Dependency Management	Dependencies between my team, other development teams and other groups in the enterprise are managed efficiently and effectively.	Agree	Neutral	↘			This could be because the team is working alongside other teams as part of a program, and they need to coordinate planning, release and tesitng etc. Might also be dependent on shared services work such as entitlements, infrastructure etc. The team makeup enables them to be as self sufficient as possible.
1	Empowerment	The team feels empowered to decide how to solve their challenges of delivering.	Agree	Neutral	↘			The team understands the near terms goals and general roadmap for the product but is given time to solve problems and come up with solutions, sometimes neding to experiment, fail and ty something new.
1	Team	All members of the team are contributing and working to make the team better	Agree	Neutral	↘			The team communicates well, there is open and honest conversation and a culture of helping each other succeeed.
1	Tools	The team has the tools necessary to develop, test, deploy and to manage the process to deliver high value software.	Agree	Neutral	↘			This could include code management, testing and requirements management tools.
1	Technical Practices	The team's technical practices (continuous integration, unit testing, coding practices, test autromation, etc.) are solid.	Agree	Agree	→			The team have agreed these practices and are working to a clear definition fo 'done' for stories/defects, iterations, mid-range planning and release, aiming to bring quality forward in the process as much as they can.
1	Quality	The team is keeping quality at a high level by ensuring the defect and testing backlog is not growing and that code technical debt is not increasing.	Agree	Neutral	↘	2	Want to increase auto test coverage as still finding issues late in final regression testing	There is visibility of defects levels, but also the technical debt backlog is being prioritised along with new development. The testing processes are keeping track with development and test coverage is ensuring there are no major underlying quality issues that could surface later on.
		Overall rating	70	38	-32			

Agile Development Survey

Demographics and Base Data

Which Division are you in?	
If you use iterations how long are they in weeks? (Assume a month = 4 weeks)	

Practice	Description		To what degree do you and/or your team use this practice? Scale: 0 = Not at all 10 = In every applicable case	Would you recommend this practice to other similar teams? Scale: 0 = Never 10 = In every applicable case
			Your Use	**Your Recommendation**
Business Involvement	Business Individuals are actively involved in your Release Planning, Iteration Planning and Demos			
Epic Stories	Your release goals are defined as Themes and Epic User Stories			
User Stories	You have user stories (or similar artefacts) to communicate your requirements within your team.			
Story Point Estimating	You use story point estimating to size the User Stories to be delivered within the release.			
Self-Direction	The team itself (not management or even a team lead) have the authority and latitude to decide how to distribute work within the team.			
Iterations	You use iterations with fixed end dates and adjust content if needed to hold the date.			
	Whole Team	The whole team involved in planning and executing each iteration (Business, Dev, Test, UX, ID, Service, ...).		
	Prioritized Backlog	You select the content of each iteration, at the start of the iteration, from a continually reprioritized set of requirements?		
	Estimating	You re-plan your work for each iteration based on your previous "Velocity" (how much work got done in previous iterations)?		
	Time Boxed	The end of the iteration is never moved to accommodate additional work.		
	Information Radiator	You use an information radiator to manage progress and work items within the iteration.		
	Progress Tracking	You continually track progress within the iteration by using Burn Down charts or equivalent.		
	Test Early	Each iteration is fully tested. Unit Test, FVT, Systems Test for the new capabilities included.		
	Done done done	The team jointly defines and commits to the definition of Done,Done,Done to be used in this iteration		
	Stability / High Quality	At the end of each iteration you have high-quality stable code with low technical debt? (No Sev 1 or 2 defects remain, and all must-fix severity 3 defects have been fixed.)		
	Stakeholder Feedback	You periodically gather feedback from stakeholders (Executives, customers, users, insiders, and/or partners) during or at the end of each iteration.		
	Reflections	You continually improve your process by holding periodic 'lessons learned' sessions throughout your release?		
Daily Scrum	You conduct daily short meetings to discuss high-priority items such as what your did, what you are going to do, and to identify obstacles?			
	Timing	Your complete the daily scrum within 15 minutes		
Continuous Integration	You build continuously. Several builds a day (triggered by checking in parts), would earn a high score. A weekly build (which perhaps does not always pass a smoke test) would earn a low score.			
Unit Test	Developers frequently run automated tests.			
TDD	Developers use Test Driven Development. (Write Test Cases before Code)			
Early Defect Removal	**Static Analysis**	You use code review and analysis tools to detect bugs during development.		
	Reviews	You hold informal reviews to remove defects as early as possible in the development cycle.		
	Inspections	You hold formal inspections to remove defects as early as possible in the development cycle.		
	Reviewer / Committer	You use a reviewer / committer check-in process to remove defects as early as possible in the development cycle.		
	Pair Programming	Your developers work in pairs during coding, creating, reviewing and fixing defects in the code during data entry.		
Release Tracking	You track and project overall release progress using story point velocity			
Sustainable Pace	Your teams work at a pace that can be continued over the long term.			
Early / Beta Programs	You make your iterations available to key customers during the development cycle to get their feedback and suggestions			
Customer Based Testing	You use Residencies, Reverse Residencies or Transplant Testing to validate your solutions against customer requirements.			