simplilearn
your pace, your place

Successful Scrum Adoption

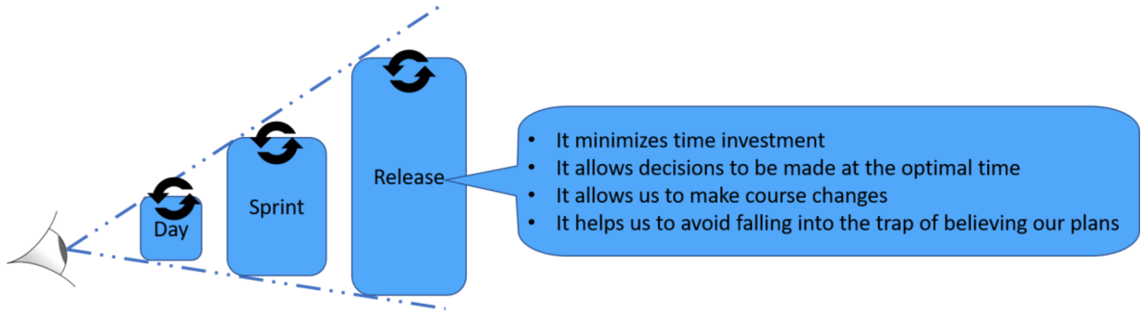
Guidance for Teams – Planning

- Many people may be fooled by the Agile manifesto statement "**Responding to Change over Following a Plan"** thinking that there is no planning in Agile.
- Planning is a fundamental aspect in Scrum.
    - Product Owners must arrive at the cost of developing a feature
    - Teams must be able to estimate how long it would take them to develop those features
- In Scrum, we progressively refine plans rather than starting with a fully developed plan.

Progressively refine Plans

- An early plan such as Release Plan created in a Scrum project captures the essence of what will be delivered but leaves the specifics to next level plans such as Sprint.
- Sprint Plans add details required for that Sprint.
- Day plans add more minute details, helping the team to plan their work better.

Advantages of progressively refining plans


Day
Sprint
Release
It minimizes time investment
It allows decisions to be made at the optimal time
It allows us to make course changes
It helps us to avoid falling into the trap of believing our plans

It minimizes time investment

- Planning is useful, but it can be time consuming.
- Teams must invest their time in estimating and planning.
- If we want to create a dedicated full project plan at the beginning of the project, we need a lot of time investment from the members involved in planning.
- Instead, if we progressively refine our plans, less upfront investment of time is required, and members involved in planning will be able to validate their assumptions and arrive at better plans.

It allows decisions to be made at the optimal time

- To make a fully detailed project plan, teams might have to take many decisions upfront.

- Unfortunately, most of these decisions might be taken using limited information. This will lead to errors as the project progresses.
- Scrum teams work on day-to-day planning. This helps teams with required knowledge to take correct decisions at the right time.

It allows us to make course changes

- Changes are inevitable in any project.
- By progressively refining a plan, we have the flexibility to alter project course as more is learned and more information is available.

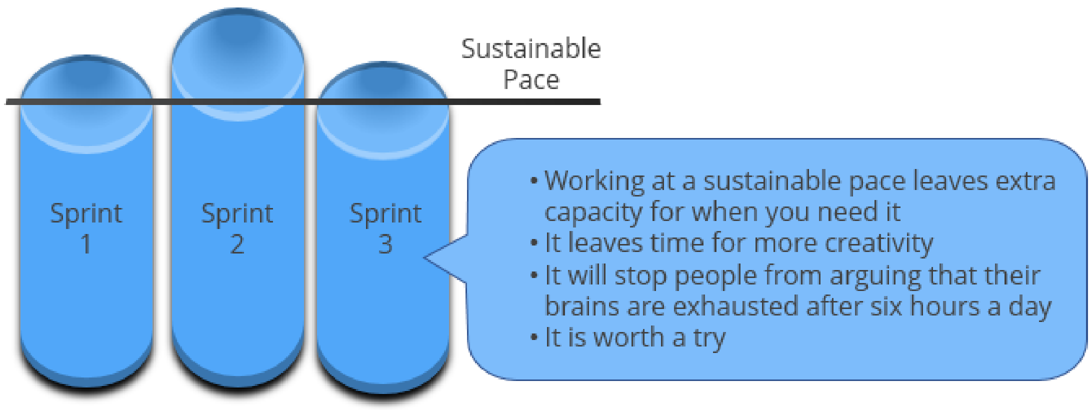
It helps us avoid falling into the trap of believing our plans

- A well-documented plan may fool us into believing that everything is taken care of and well thought of.
- Progressively refining a plan reinforces the idea that even the best plans need change.

Sustainable Pace

- Scrum teams work at a sustainable and even pace. However, every now and then team members may stretch a little to work on a few crucial items near the deadline or fix a critical defect that is preventing something from going live.

Advantages of working at a sustainable pace


Sustainable Pace
Sprint 1
Sprint 2
Sprint 3
• Working at a sustainable pace leaves extra capacity for when you need it
• It leaves time for more creativity
• It will stop people from arguing that their brains are exhausted after six hours a day
• It is worth a try

Working at a sustainable pace leaves extra capacity for when you need it

- When a team is constantly running at an all-out pace, that is, all their time is fully utilized, it will not have extra reserve of energy for the time when extra effort is truly necessary.
- For example, let us assume a typical work day of 8 hours and a typical work week of 40 hours. If all team members are occupied fully for a week and

work for 40 hours, how can they put in extra effort? Overtime is not a recommended option here.

It leaves time for more creativity

- Teams working at a sustainable pace will have the necessary mental energy to come up with creative ideas that may dramatically shorten the schedule or vastly improve the product.

It will stop people from arguing that their brains are exhausted after six hours a day

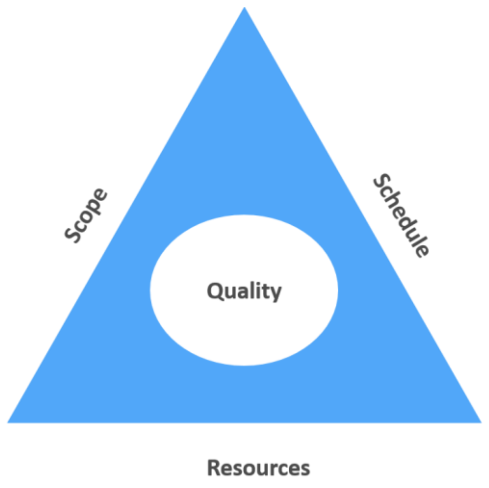
- Developers focusing continuously for six hours a day feel exhausted. They may not feel energetic and motivated to continue to work.
- With a sustainable pace, team members will get adjusted to a routine schedule that leaves them some time to explore a few options to refresh themselves, such as reading books, working on a pet project, etc.

It's worth a try

- Early in the project, teams will have less pressure.
- Collecting the velocity during this phase and then using it toward the end of the project where the team might have to stretch a few extra hours reinforces that the extra effort required is only for a few weeks.

Handling Scope Changes

Iron Triangle in traditional Project Management



- In traditional Project Management, the Iron Triangle shows the interdependencies between scope, resources (cost), and schedule.
- To meet customer expectations, project Managers would mostly offer two of these as fixed and one as flexible, for example, cost and scope as fixed but schedule as flexible.
- Quality at the center of the triangle indicates that it is fixed and untouchable.

- Scope will be fixed in most cases, allowing almost no changes to scope during the project. This defeats the purpose of Agile Manifesto principle “***We welcome changing requirements even late in development.***”

Alternate approaches

- Let us explore the demerits of fixed schedule and discuss options to work with flexible Quality, Scope, Resources, and Schedule.

Cut Quality

- When project is running behind the schedule, we may skip a little testing and leave a few bugs (not critical ones) unfixed.
- Deciding what to cut and what not to cut in quality is hard.

Add resources

- To meet the planned schedule, most managers think of adding resources.
- However, adding manpower doesn’t bring the project back to schedule.
- Adding resources may become risky as knowledge transition is required to bring the additional developers to speed.

Extend the Schedule

- From a development team perspective, extending the schedule is music to ears. From a business perspective, it could be suicidal.
- Changes to deadline may not be feasible always.

Adjust Scope

- Keeping scope flexible means dropping a few requirements.
- By keeping scope flexible and Product Owner prioritizing it, teams will deliver the most important and valuable features to Product Owner and may move the lowest priority items to future Sprints.

Conclusion on alternate approaches

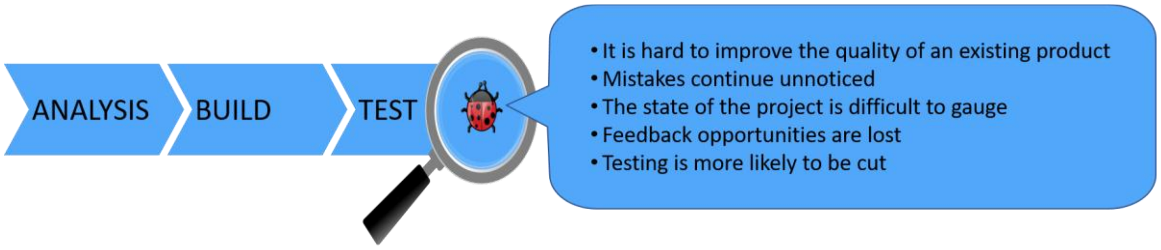
- Reducing quality is not a good option.
- Adding more people makes the project unpredictable.
- Extending the deadline is not always feasible.
- So, the best alternate approach is to keep scope flexible, supported by proper prioritization by Product Owner.

Guidance for Teams – Quality

Integrate Testing into Process

- In traditional waterfall method, quality is evaluated, that is, testing is done toward the end of the project. Here, we test the quality ***after*** a product is developed.
- Scrum teams make testing a part of development cycle rather than doing it after the coding is done. Here we build quality into the process and product ***as it is being developed.***

Why testing at end doesn't work


ANALYSIS
BUILD
TEST
• It is hard to improve the quality of an existing product
• Mistakes continue unnoticed
• The state of the project is difficult to gauge
• Feedback opportunities are lost
• Testing is more likely to be cut

It is hard to improve the quality of an existing product

- By the time testing commences toward the end of the project, the product might have already been built.
- Toward the end of the project, it is difficult and time consuming to improve the product.

Mistakes continue unnoticed

- We will not be able to know whether a product works until it is tested.
- If you have not realized the mistake, you may continue doing it repeatedly.
- If tested along the way, that is, throughout the development rather than at end, we can avoid unpleasant surprises at the end.

The state of the project is difficult to gauge

- We may realize that the project was off the track only toward the end of the project, when we test and defects emerge.
- Instead, if we do periodic testing throughout the development, we can easily know how well the product development is progressing.

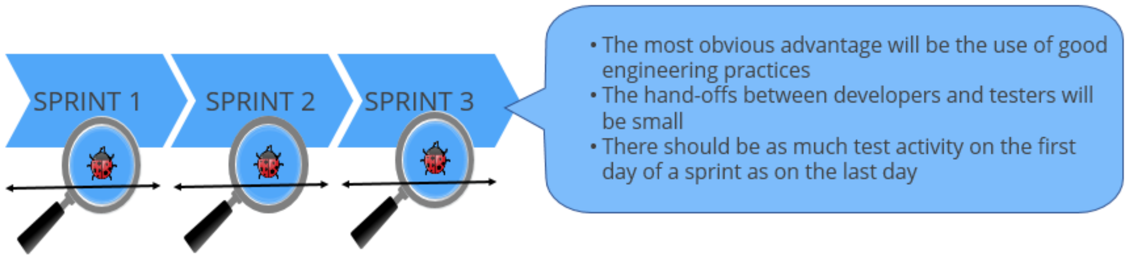
Feedback opportunities are lost

- There would be too little time to react to customer feedback if testing is carried out toward end of the project.
- On the other hand, in Scrum projects, each Sprint provides an opportunity for the customer to test the product increment and provide feedback.

Testing is more likely to be cut

- Because of deadline pressures, teams may cut testing toward end of the project.

Advantages of building in quality, that is, testing throughout development


SPRINT 1
SPRINT 2
SPRINT 3
• The most obvious advantage will be the use of good engineering practices
• The hand-offs between developers and testers will be small
• There should be as much test activity on the first day of a sprint as on the last day

- A team that has integrated testing into its day-to-day work will have these traits (advantages of building in quality into the process).

The most obvious advantage will be the use of good engineering practices

- A team focussed on building quality into the process will do whatever it takes to produce high-quality software.
- To achieve this, teams may practice technical practices such as Pair programming, refactoring, test driven development, etc.
- Code will be continuously integrated and tested.

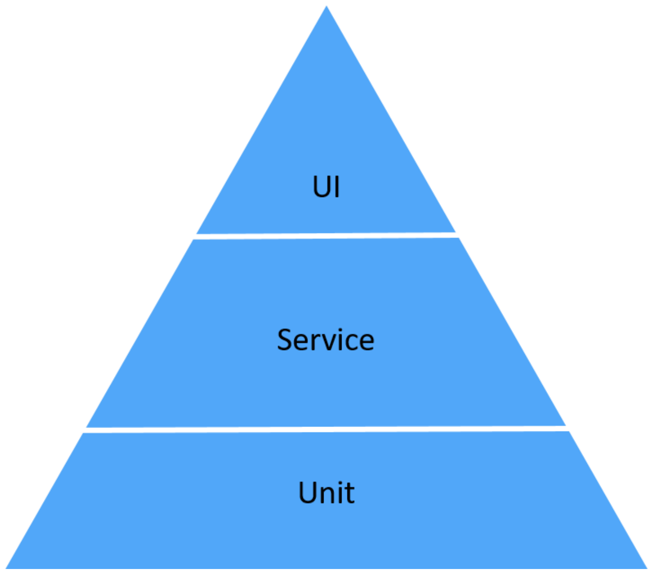
The hand-offs between developers and testers will be small

- Developer and tester will work collaboratively throughout the Sprint, that is, when developer is about to code, tester will start his/her test case development.
- This will eliminate hand-offs between the roles and will improve productivity.

There should be as much test activity on the first day of a Sprint as on the last day

- Each Sprint doesn't follow mini-waterfalls, that is, analysis, design, code, and test inside a Sprint.
- All activities with respect to analysis, design, code, and test are carried out on each User Story throughout the Sprint.
- This means that testers will be busy from day 1 of a Scrum project.

Test automation at different levels



- An effective test automation strategy calls for automated tests at three different levels, as shown in the image above.

Unit Testing

- Unit Testing is the foundation and largest type of test.
- Programmers will be more comfortable with unit tests as they are usually written in the same language used to develop the project.

Service Testing

- Service-level testing involves testing the service separately from its user interface.
- Test cases are fed into the system to test the service.

User Interface Testing

- This is at the top of the pyramid as teams tend to spend less time testing the User Interface.
- Testing User Interface can be time consuming and expensive, so it can be minimized.

Technical Debt

- Technical Debt refers to the increased cost of working on an application with “immature” or “not-quite-right” code.
- Examples:
    - Unexpected data (example – reserved characters) in the database that can causes the application to crash
    - Code that breaks when a programmer touches it

- Software version not upgraded when a new stable version is released by manufacturer

Handling Technical Debt

Stop the bleeding

- The first priority when dealing with technical debt is to stop the bleeding, that is, stop things from getting worse.
- Teams should target low-hanging fruits such as reducing automated testing effort by manually performing a few tasks.

Stay current

- After adding automation testing, servers, and tools, team may stop bleeding.
- Now, the team can focus on learning how to write and automate tests for any new features added during the Sprint. This will stop accumulation of new technical debt.

Catch Up

- Team can then focus on the remaining technical debt.