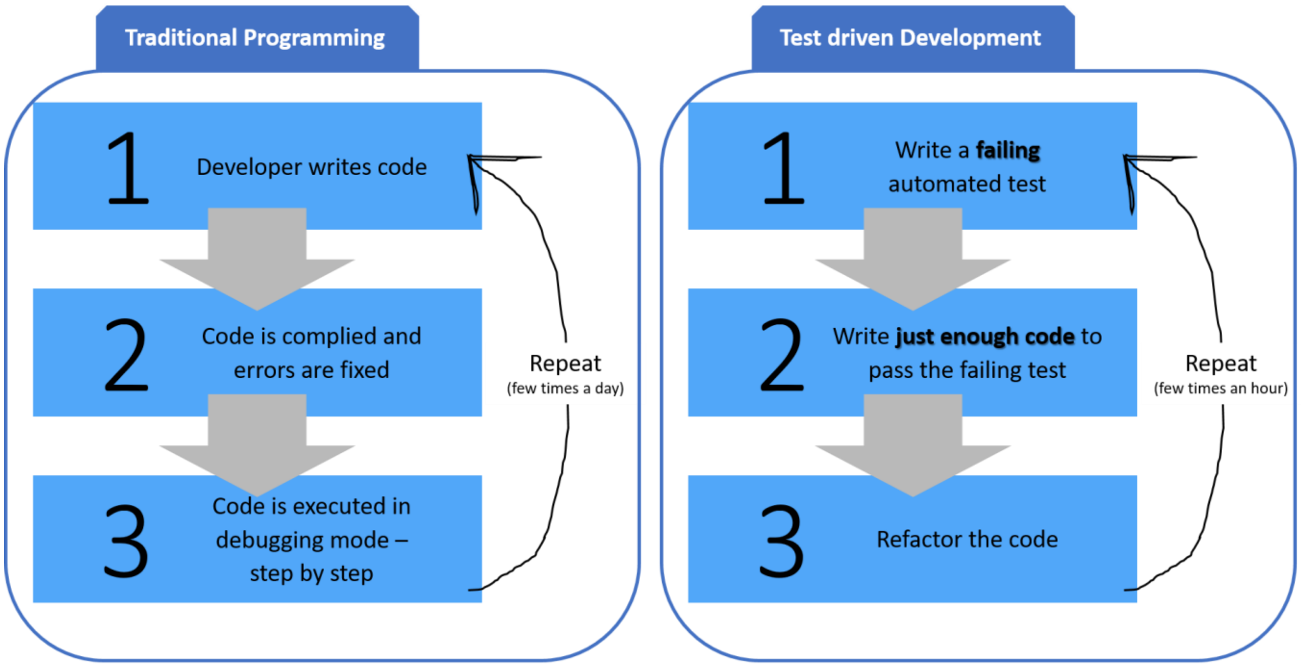
simplilearn
your pace, your place

Successful Scrum Adoption

Guidance for Individuals – Technical Practices

- Scrum not only introduces new roles and innovative ways of working, such as User Stories, Backlogs, Information Radiators, etc., but also introduces a few technical practices.
- Usage of technical practices enables Scrum teams to be more productive and helps them reach their goals easily.
- Let us explore a few technical practices.

Test-Driven Development


Traditional Programming
1 Developer writes code
2 Code is complied and errors are fixed
3 Code is executed in debugging mode – step by step
Repeat (few times a day)
Test driven Development
1 Write a failing automated test
2 Write just enough code to pass the failing test
3 Refactor the code
Repeat (few times an hour)

- In traditional programming, programmers write some part of code to deliver a specific functionality, compile it to check for syntax errors, and then do a code walkthrough step-by-step on the debugger.
- Test-Driven development has a different approach. First, the programmer writes an automated test. When executed, this automated test will fail because nothing is coded. Now, the programmer knows why this test has failed. So, the programmer writes just enough code this time so that the test will pass. After this, the programmer does code cleanup. Programmer repeats this every few minutes.
- The biggest advantage of Test-Driven Development is that no untested code goes into production.

- Test-Driven Development is more likely a design practice rather than a programming practice because the tests written will guide the design and development.

Refactoring

- Refactoring refers to changing the structure but not behavior. For example, let us consider a web development project. The user's first and last names are prompted by labels and text boxes. The user enters first and last names. The process of capturing first and last names can be used in multiple pages. Instead of writing the same code in each page, this specific code fragment can be moved to a function and called in every page where required. This makes code modifications and additions easy to handle, as only the function is modified, and not each page where it is called.
- When a software development project is underway, multiple programmers write code as per their understanding and thought processes. While writing code, they might use statements that might be irrelevant. For example, it is quite common in programming to write the values of variables into a log file for tracking of code. This code is of no use to the business. Refactoring helps to eliminate such code, which might not be required in the first place.
- By constantly refactoring and fixing small problems before they become big, code maintenance becomes easier.
- Refactoring is compared with the ***Boy Scout Rule:*** Leave the campground cleaner than you found it.

Collective Ownership

- Code and automated tests are owned by ***all*** developers.
- It encourages each team member to feel responsible for all parts of the program. This allows a team member work on any code written by anyone.
- Team members can seek help of other members who might have initially written that code.
- The following responsibilities are shared by all team members under Collective Ownership:
    - No developer becomes specialized so he/she contributes in one area only
    - No area is so intricate that it is understood and worked upon only by one member

- Collective Ownership encourages developers to learn the new parts of the system.

Continuous Integration

- Continuous Integration refers to integrating new or changed code into an application as soon as it is written and testing the complete application to ensure that nothing is broken due to the introduction of this new code.
- In traditional programming, code will be checked in and checked out at a fixed interval such as end of day. In Scrum, this happens every few minutes.
- Continuous Integration requires tools such as Version Control systems, automated testing, etc.
- Continuous Integration eliminates big integration issues that usually come around toward the end of the project and it also helps the team to get near real-time feedback.

Pair Programming

- Pair programming refers to two developers working together to write code. When one programmer is writing code, the other programmer will look for improvements, testing scenarios, etc. Then they switch roles, and this continues.
- Pair programming can be a bit difficult to implement.
- Pair programming can be done either full-time in a day or part-time at regular intervals in a day.
- Pair programming also improves quality and facilitates knowledge transfer.

Emergent Design on Agile Projects that is done intentionally

- In a traditional project, a lot of time is spent on upfront design. In Scrum projects, design, like everything else, is done incrementally.
- In a Scrum project, design is both emergent and intentional.

Anticipation versus Adaptation

- With upfront design and analysis, we try to ***anticipate*** the user's needs.
- Anticipation can't be perfect, so there will be mistakes, which leads to rework.
- In Scrum, coding and testing a User Story is started without much upfront design. This will enable ***adaptation*** to user needs or requirements.
- For example:

    - In medicine, a life support system can **anticipate** user needs and might need an upfront design.
    - In a global application, each region, such as North America, Europe, and Asia Pacific, might have its own requirements. This calls for an **adaptive** approach.
- In the following table, let us look at both the approaches.

Anticipation	Adaptation
Early Planning	Real-time planning
Big design upfront	Emergent design
Testing toward end	Integrated (or automated) testing
Signed handoffs	Collaborative decisions
Early, complete, and exhaustively documented requirements	Just-in-time and just-enough requirements

- Creating a balance between both of these is the main objective of emergent design on Scrum projects.

Shifting to work with no upfront and emergent design

- There are a few challenges when moving from upfront design to emergent design:
- **Planning is harder**
    - With the absence of an upfront design, planning, design, and estimation become hard.
    - On the flipside, in Scrum projects, the work that needs to be estimated will be smaller, so individual features can be estimated easily, quickly, and more accurately.
- **Hard to partition work among teams or individuals**
    - With upfront design, we know what is to be built and what skills are required. This makes task allocation easier.
- **It is uncomfortable not to have a completed design**
    - Upfront design may create a comfortable feeling as teams may feel that upfront design is solid. Mistakes can emerge later.
    - Teams will have a mindset that later changes to design can be minor and easily done.
- **Rework is inevitable**
    - Without upfront design, there could be situations where emergent design must be redone. It is like taking two steps forward and being pulled one step backward.

    - Refactoring, Automated Tests, and Test Driven Development can help to reduce the rework.
- It is proven in Software engineering that defects are more expensive to fix than development. By using Refactoring, Automated Tests, and Test Driven Development, Scrum Teams can build clean code with lesser defects.

Emergent Design guidance from Product Owner

- Product Owner owns the Product Backlog and prioritizes the requirements. Therefore, emergent design relies heavily on Product Owner inputs.
- Product Owner must listen to the team on design and technical aspects.
- Product Owner may select a few Product Backlog items that present opportunities for team members to include design elements.
- For example, during the initial stages of a web development project, Product Owner, in consultation with the team, may select a few items that will help the team members to build user interfaces for both desktops and mobiles. This will help the team to work on user stories that will help the design to emerge as the project progresses.
- The key success for emergent design is Product Owner and Technical Personnel working together.