your Agile Journey

Agile Techniques

Read more to learn about…

- User Stories
- Acceptance Test Drive Development (ATDD)
- Test Driven Development (TDD)
- Behavior Drive Development (BDD)
- Continuous Integration
- Continuous Development
- Continuous Testing
- Pair Programming
- Automated Testing
- Agile Games
- Burn Downs
- Retrospectives
- Kanban
- SAFe
- Jira/Confluence

User Stories

Definition from Agile Alliance

- In consultation with the customer or product owner, the team divides up the work to be done into functional increments called “user stories.”
- Each user story is expected to yield, once implemented, a contribution to the value of the overall product, irrespective of the order of implementation; these and other assumptions as to the nature of user stories are captured by the INVEST formula.
- To make these assumptions tangible, user stories are reified into a physical form: an index card or sticky note, on which a brief descriptive sentence is written to serve as a reminder of its value. This emphasizes the “atomic” nature of user stories and encourages direct physical manipulation: for instance, decisions about scheduling are made by physically moving around these “story cards.”

Acceptance Test Driven Development (ATDD)

Definition from Agile Alliance

- Analogous to test-driven development, Acceptance Test Driven Development (ATDD) involves team members with different perspectives (customer, development, testing) collaborating to write acceptance tests in advance of implementing the corresponding functionality. The collaborative discussions that occur to generate the acceptance test is often referred to as the three amigos, representing the three perspectives of customer (what problem are we trying to solve?), development (how might we solve this problem?), and testing (what about…).
- These acceptance tests represent the user's point of view and act as a form of requirements to describe how the system will function, as well as serve as a way of verifying that the system functions as intended. In some cases the team automates the acceptance tests.

Test Driven Development (TDD)

Definition from Agile Alliance

- “Test-driven development” refers to a style of programming in which three activities are tightly interwoven: coding, testing (in the form of writing unit tests) and design (in the form of refactoring).

- It can be succinctly described by the following set of rules:
  - write a “single” unit test describing an aspect of the program
  - run the test, which should fail because the program lacks that feature
  - write “just enough” code, the simplest possible, to make the test pass
  - “refactor” the code until it conforms to the simplicity criteria
  - repeat, “accumulating” unit tests over time

Test Driven Development (TDD)

Expected Benefits from Agile Alliance

- many teams report significant reductions in defect rates, at the cost of a moderate increase in initial development effort
- the same teams tend to report that these overheads are more than offset by a reduction in effort in projects' final phases
- although empirical research has so far failed to confirm this, veteran practitioners report that TDD leads to improved design qualities in the code, and more generally a higher degree of "internal" or technical quality, for instance improving the metrics of cohesion and coupling

Test Driven Development (TDD)

Common Pitfalls from Agile Alliance

- Typical individual mistakes include:
  - forgetting to run tests frequently
  - writing too many tests at once
  - writing tests that are too large or coarse-grained
  - writing overly trivial tests, for instance omitting assertions
  - writing tests for trivial code, for instance accessors

- Typical team pitfalls include:
  - partial adoption – only a few developers on the team use TDD
  - poor maintenance of the test suite – most commonly leading to a test suite with a prohibitively long running time
  - abandoned test suite (i.e. seldom or never run) – sometimes as a result of poor maintenance, sometimes as a result of team turnover

Behavior Driven Development (BDD)

Definition from Agile Alliance

- Behavior Driven Development (BDD) is a synthesis and refinement of practices stemming from Test Driven Development (TDD) and Acceptance Test Driven Development (ATDD). BDD augments TDD and ATDD with the following tactics:
- Apply the “Five Why’s” principle to each proposed user story, so that its purpose is clearly related to business outcomes
- thinking “from the outside in”, in other words implement only those behaviors which contribute most directly to these business outcomes, so as to minimize waste
- describe behaviors in a single notation which is directly accessible to domain experts, testers and developers, so as to improve communication
- apply these techniques all the way down to the lowest levels of abstraction of the software, paying particular attention to the distribution of behavior, so that evolution remains cheap

Behavior Driven Development (BDD)

Expected Benefits, from Agile Alliance

Teams already using TDD or ATDD may want to consider BDD for several reasons:

- BDD offers more precise guidance on organizing the conversation between developers, testers and domain experts
- notations originating in the BDD approach, in particular the given-when-then canvas, are closer to everyday language and have a shallower learning curve compared to those of tools such as Fit/FitNesse
- tools targeting a BDD approach generally afford the automatic generation of technical and end user documentation from BDD “specifications”

Behavior Driven Development (BDD)

Common Pitfalls, from Agile Alliance

Teams already using TDD or ATDD may want to consider BDD for several reasons:

- Although Dan North, who first formulated the BDD approach, claims that it was designed to address recurring issues in the teaching of TDD, it is clear that BDD requires familiarity with a greater range of concepts than TDD does, and it seems difficult to recommend a novice programmer should first learn BDD without prior exposure to TDD concepts
- The use of BDD requires no particular tools or programming languages, and is primarily a conceptual approach; to make it a purely technical practice or one that hinges on specific tooling would be to miss the point altogether

Continuous Integration

Definition from Agile Alliance

Teams practicing continuous integration seek two objectives:

- minimize the duration and effort required by each integration episode
- be able to deliver a product version suitable for release at any moment
- In practice, this dual objective requires an integration procedure which is reproducible at the very least, and largely automated. This is achieved through version control tools, team policies and conventions, and tools specifically designed to help achieve continuous integration.
- Continuous integration aims to lessen the pain of integration by increasing its frequency. Therefore, “any” effort related to producing intermediate releases, and which the team experiences as particularly burdensome, is a candidate for inclusion in the team’s continuous integration process. (This is the reasoning that leads teams to continuous deployment.)

Continuous Deployment

Definition from Agile Alliance

- Continuous deployment can be thought of as an extension of continuous integration, aiming at minimizing lead time, the time elapsed between development writing one new line of code and this new code being used by live users, in production.
- To achieve continuous deployment, the team relies on infrastructure that automates and instruments the various steps leading up to deployment, so that after each integration successfully meeting these release criteria, the live application is updated with new code.
- Instrumentation is needed to ensure that any suggestion of lowered quality results in aborting the deployment process, or rolling back the new features, and triggers human intervention.

Continuous Testing

Continuous testing is the process of executing automated tests as part of the software delivery pipeline to obtain immediate feedback on the business risks associated with a software release candidate. Continuous testing was originally proposed as a way of reducing waiting time for feedback to developers by introducing development environment-triggered tests as well as more traditional developer/tester-triggered tests.

Pair Programming

Definition from Agile Alliance

Pair programming consists of two programmers sharing a single workstation (one screen, keyboard and mouse among the pair). The programmer at the keyboard is usually called the “driver”, the other, also actively involved in the programming task but focusing more on overall direction is the “navigator”; it is expected that the programmers swap roles every few minutes or so.

Pair Programming

Common Pitfalls from Agile Alliance

- both programmers must be actively engaging with the task throughout a paired session, otherwise no benefit can be expected
- a simplistic but often raised objection is that pairing “doubles costs”; that is a misconception based on equating programming with typing – however, one should be aware that this is the worst-case outcome of poorly applied pairing
- at least the driver, and possibly both programmers, are expected to keep up a running commentary; pair programming is also “programming out loud” – if the driver is silent, the navigator should intervene
- pair programming cannot be fruitfully forced upon people, especially if relationship issues, including the most mundane (such as personal hygiene), are getting in the way; solve these first!

Automated Testing

In software testing, test automation is the use of software separate from the software being tested to control the execution of tests and the comparison of actual outcomes with predicted outcomes. Test automation can automate some repetitive but necessary tasks in a formalized testing process already in place, or perform additional testing that would be difficult to do manually. Test automation is critical for continuous delivery and continuous testing.

Agile Games – Buy a Feature

Teaches feature prioritization

example from AgileCoffee.Com

- Best played in groups of 3-8. Takes 10-15 minutes.
- Each player receives two items: (1) a handout with a menu of features and their prices (2) a sum of play money. (Features can be anything: items to have on a vacation, goals of a two-day training, benefits of a high-functioning team, etc.) The play money should contain a variety of denominations. The sum total of all players' money should be less than the total of all feature prices – this introduces scarcity and forces the team to make trade-offs because it's not possible to purchase all items on the list.
- Players take turns using their individual sums of money to but the features they deem most valuable. Once players have spent most of their funds (either they don't have enough individually to make another purchase or they don't value what's left on the menu to buy anything else), the group will pool the remaining funds and discuss what to buy from the remaining items.

Agile Games – Human Know

Teaches self-organization

example from AgileCoffee.Com

- Best played in groups of 6-20. Takes about 2-3 minutes per round, including instructions.
- Start with all players standing to form a large circle, facing inside the circle. Everyone moves in close (shoulder-to-shoulder) and reaches their left hand into the mix, grabbing the left hand of another player (not their immediate neighbor). Next, reach in the right hands and grab a new person's right hand (again, not the neighbor next to you).
- While keeping all hands connected, the group then proceeds to twist and turn out of the scrambled knot they've formed. After a minute or more, the group should once again return to a large circle of people clasping hands.
- Remember: Safety first. If someone is getting squeezed or about to trip, it's better to release a hand or two tham to end up with broken bones or sprained thingies.
- It's fun if two or more groups of equal number compete to see who can "untie" themselves first.

Agile Games – Multitasking

Shows how multi-tasking reduces effectiveness

example from AgileCoffee.Com

- Played individually at a table or whiteboard. Playing two rounds takes less than 5 minutes.
- Each player needs one sheet of paper (or on a whiteboard) and something to write with. It's best if they each have their own stopwatch (phone, duh), but the facilitator can monitor time for the group if necessary.
- The paper will have three blank columns to be filled in, and the task is the same for each of two rounds. Column one will contain the letters A thru J; column two lists the digits 1 thru 10; column three will have roman numerals I thru X. All finished columns should contain their ten elements in their proper order.
- The difference between the two rounds comes in how the columns get filled out. in round one, players must write one letter, followed by one digit, followed by one roman numeral, then repeat. In round two, players should write all letters first, before moving on to the digits, and filling in the roman numerals last.
- Each of the two rounds is timed, and we see that the multitasking of round one consumes a greater time than the focused task completion of round two.

Agile Games – Pair-Origami

Illustrates importance of face-to-face communication

example from AgileCoffee.Com

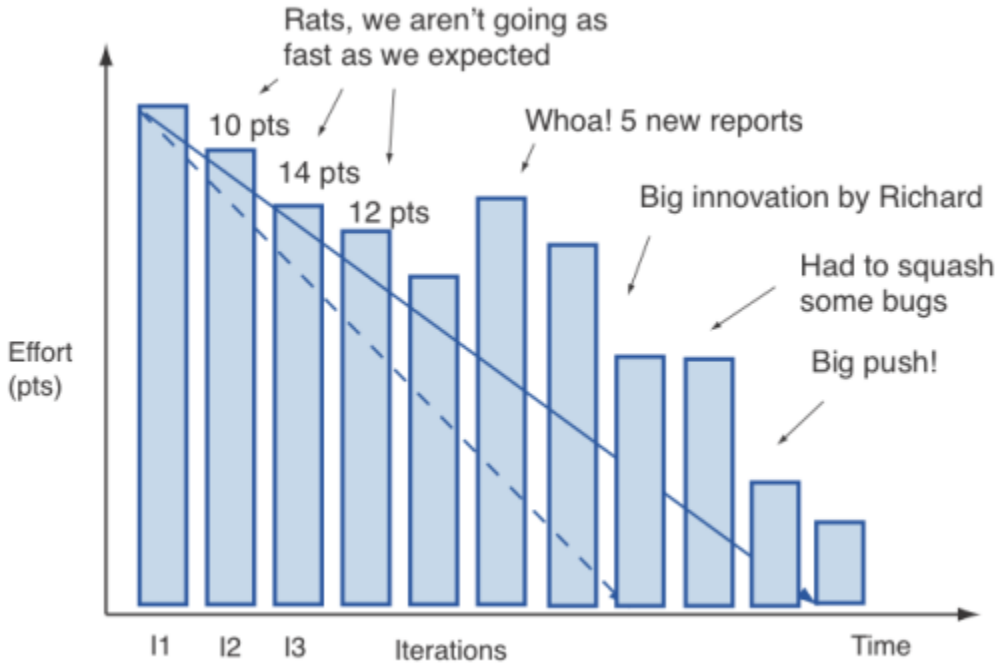
- Need total of six or more players. Takes about 10-15 minutes.
- Divide the players into three groups (A, B & C). Players for pairs in each group. Each pair consists of a folder and a PO (product owner) or manager. All folders get a single sheet of 8.5×11 paper, and each PO/manager receives an instruction page (download link).
- Players in group A sit side-by-side. Only the folder player may fold the origami, but both partners may see the instruction sheet.
- Players in group B sit face-to-face. Only the folder player may fold the origami. The PO/manager may give feedback but may not show the instructions to the folder.
- Players in group C sit back-to-back. Only the folder player may fold the origami. The PO/manager may read/explain the instructions to the folder but may not see the origami as it's being folded.
- When the timer starts, all pairs get to work. When a pair completes their origami, their time gets recorded. The facilitator may (mercifully) call time when it's apparent that pairs in group C are about to explode.

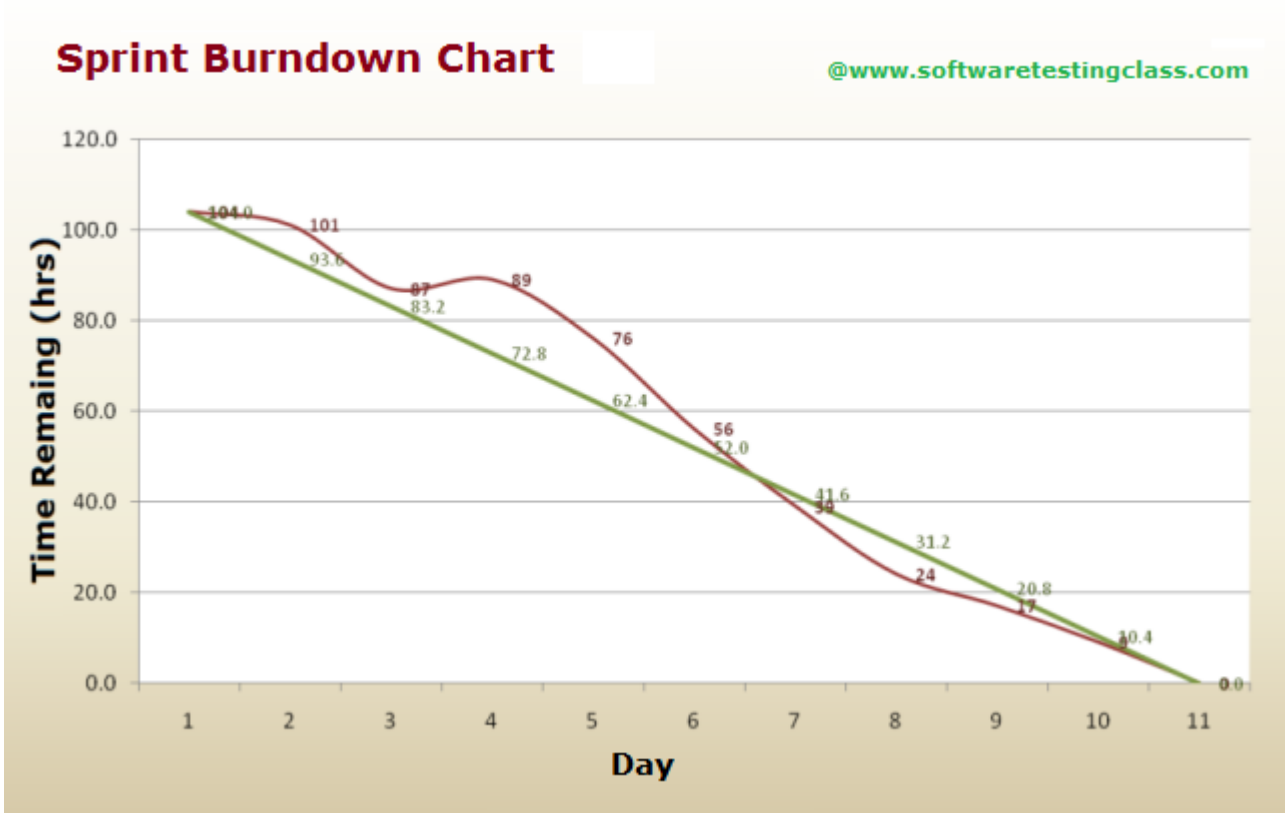
Burn Downs

Definition from Agile Alliance

The team displays, somewhere on a wall of the project room, a large graph relating the quantity of work remaining (on the vertical axis) and the time elapsed since the start of the project (on the horizontal, showing future as well as past). This constitutes an “information radiator“, provided it is updated regularly. Two variants exist, depending on whether the amount graphed is for the work remaining in the iteration (“sprint burndown”) or more commonly the entire project (“product burndown”).

Burn Charts

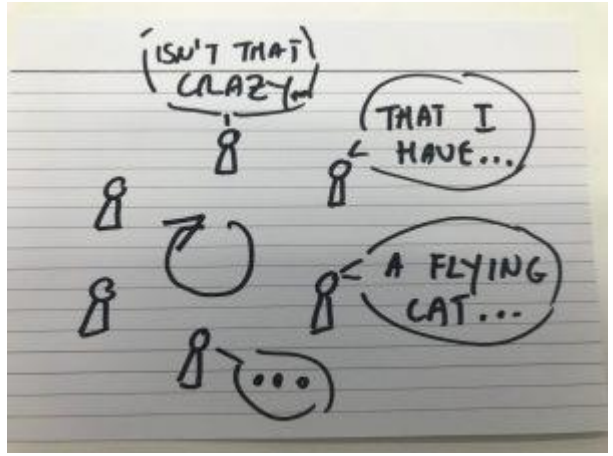

Rats, we aren't going as fast as we expected
10 pts
14 pts
12 pts
Whoa! 5 new reports
Big innovation by Richard
Had to squash some bugs
Big push!
Effort (pts)
I1
I2
I3
Iterations
Time


Sprint Burndown Chart
@www.softwaretestingclass.com
Time Remaing (hrs)
120.0
100.0
80.0
60.0
40.0
20.0
0.0
104.0
101
93.6
87
83.2
89
72.8
76
62.4
56
52.0
41.6
31.2
24
20.8
17
10.4
0.0
1
2
3
4
5
6
7
8
9
10
11
Day


Story Points Remaining
40
35
30
25
20
15
10
5
0
Day 1
Day 2
Day 3
Day 4
Day 5
Day 6
Day 7
Day 8
Day 9
Day 10
Story Points Remaining

Retrospectives – Isn't that crazy

Idea from FunRetrospectives.com



The Isn't that crazy energizer is amazing for getting people talking and collaboratively creating a story (usually a funny one). It fosters engagement and everyone participation while being very easy to deliver as it is done verbally.

Step by step

Instruct the participants to form a circle

Identify the order in which the communication will flow (e.g. clockwise).

One person starts by saying "isn't that crazy?"

The next person has to continue the story by adding 3 words

Then the next and so forth until the story ends.

It can go like this:

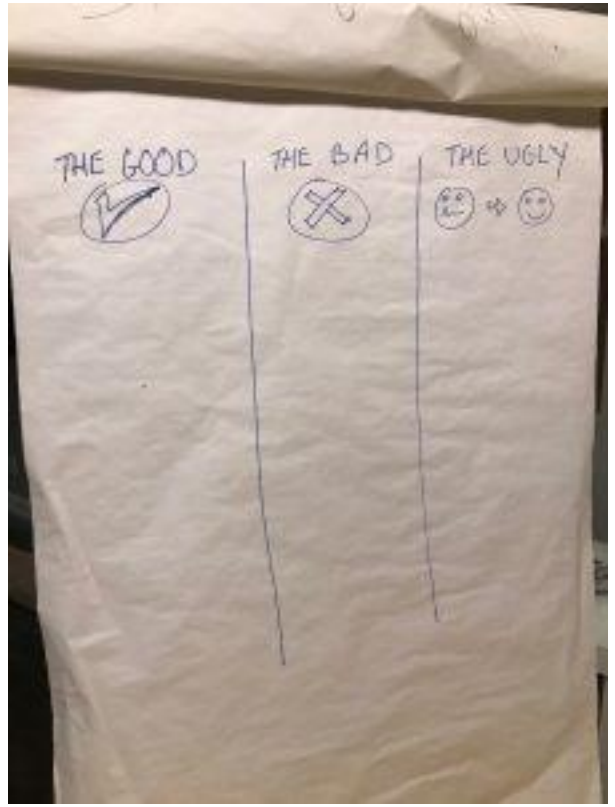
"isn't that crazy?"
"That birds fly"
"But I know"
"a flying cat"
"That has superpowers"
"And laser eyes"
"Freezes mobile apps"
"but not ours"
"Because of kryptonite"
…

This is a really fun and simple activity. Another variation is to start with ˜Once upon a time" and ask each person to add four words.

Retrospectives – The Good, The Bad & the Ugly

Idea from FunRetrospectives.com

This activity is based on the 1966 Westerm movie The Good, the Bad and the Ugly, starring Clint Eastwood. It is (at least it was) a common jargon back on the days. But it still giver a good hint that we should focus on the good, get rid of the bad and turn the ugly into beauty.

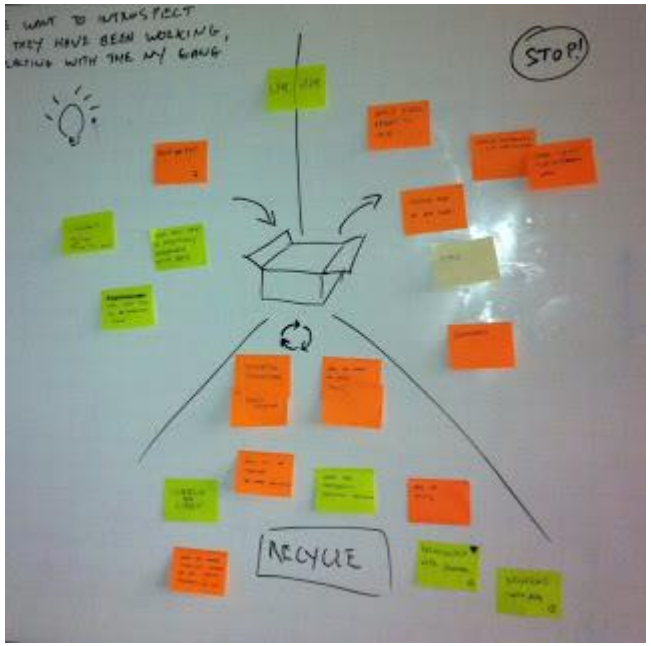


Step by step

1. Start by playing the ***The Good, the Bad and the Ugly (1966)*** movie trailer
2. Split the canvas into three areas:
    - **The Good** – things that went well, and we should repeat, do more of it
    - **The Bad** – things that should have never happened, and we must get rid of it
    - **The Ugly** – things that did not go so well, and we should look for improvements, turning it into it beautiful
3. Ask the participants to add notes to each of the three areas
4. Conversations and action items

Retrospectives – Open the Box

Idea from FunRetrospectives.com



This activity foster innovation and challenges the current activities performed by the team.

Running the activity:

1. Start by reading the following quote
"The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking." — Albert Einstein
2. Bring the participants attention to the box metaphor (If possible, bring a box with you).
3. Split the whiteboard or canvas in 3 areas. Draw the open box in the center.
4. Explain each of the areas:

Please add your notes to the areas accordingly:
Which activities should be removed from it?
Which activities should be added?
What to recycle?

Handling Bugs and Urgent On-Demand Tasks

- Not all of these issues can wait for the next Sprint. Some of them can be filed in the Backlog and wait for their turn, but some have to be handled as soon as they arise, potentially even “in real time” as per clients’ demands.
- Each week, the Batman of the group abstains from partaking in any Scrum tasks, and instead focuses all their attention on the stuff that just crops up like bug fixes, urgent requests, and general support. It’s the stuff of superheroes!
- **Allow Time for Bug Fixes Along the Way**
- Inevitably, as the project moves forward, bugs will appear out of the woodwork—perhaps not so many in the early stages of development, but certainly as the project is nearing a production release. Therefore, it’s a good idea to increase the time allocated for fixes as development progresses.
- It’s advisable to consider dividing your Sprint into two separate components:
    1. Work outside of the normal product backlog or the Bug Backlog
    2. Product Backlog Work
- It’s essential to allocate time and resources accordingly to bug backlog work (i.e., 20%). Then calculate this capacity of work per Sprint in hours.

-https://dzone.com/articles/6-challenges-in-applying-scrum-and-how-to-overcome

Kanban

Definition from Agile Alliance

The Kanban Method is a means to design, manage and improve flow for knowledge work and allows teams to start where they are to drive evolutionary change.

A Kanban Board is a visual workflow tool consisting of multiple columns. Each column represents a different stage in the workflow process.

SAFe

Definition from ScaledAgileFramework

The Scaled Agile Framework, or SAFe, methodology is an agile framework for development teams built on three pillars: Team, Program, and Portfolio. SAFe is designed to give a team flexibility and to help manage some of the challenges larger organizations have when practicing agile.

Jira/Confluence

Jira is an Agile Project Management tool that is used to help plan and coordinate the work of Agile teams. Some people take the Agile Manifesto value of "Individuals and interactions over processes and tools" to mean that tools like Jira are not appropriate in Agile and will get in the way of being Agile.

Thank you