Chapter 1

Why Agile and Lean Approaches Work
Agile and Lean Approaches

Why such approaches exist!
“Software, we have a problem”

It was thought you “could hand a software team a big requirements document and the project would magically deliver exactly what their requirements stated

… architects thought they could work on the architecture and had it off to a team to implement as designed, as if by magic.

Developers thought they could hand off code to testers and it would all “just work”

“Nothing could be farther from the truth”
Realities of Software Projects

• There are no one person projects
• At the very least… there is a developer and a user
• Usually… software requires a developer, a tester and a user
• … and the developer and the tester need to collaborate
• We can not predict everything about a project
• Software Development is not a “defined process”
  … it is (and must be) a “empirical process”
  … meaning assessment and learning is needed as development proceeds
“Any time people developing software learn as they develop, a serial approach to delivering value does not work that well.

That’s because a serial approach only encourages one delivery, at the end.

When teams deliver more often, they have an opportunity to learn more often.”
Defined Approach - not Waterfall

“It is typical to adopt the defined approach... only when the underlying mechanisms by which a process operates are reasonably well understood. When the process is too complicated for the defined approach, the empirical approach is the appropriate choice.”

Jeff Sutherland | Jun 16, 2002
The result of a **Defined Process** that should not have been!
Empirical Process Control

In **Scrum**, decisions are made based on **observation** and **experimentation** rather than on detailed upfront planning.

**Empirical Process Control** relies on the three main ideas…

1. **Transparency**
2. **Inspection**
3. **Adaptation**
1. Transparency

Transparency allows all facets of any process to be observed by anyone.

This promotes an easy and transparent flow of information throughout the organization and creates an open work culture. Transparency, in **Scrum**, is depicted through:

- Artifacts
- Project Vision Statement
- Prioritized Product Backlog
- Release Planning Schedule
- Meetings
- Sprint Review Meetings
- Daily Standup Meetings
- Information Radiators
- Burndown Chart
- Scrum Board
2. Inspection

Inspection in **Scrum** is depicted through:

- **Use of a common Scrumboard** and other “information radiators”
- **Collection of feedback** from the customer and other stakeholders … Creating a Prioritized Product Backlog, and a Conduct Release Planning processes.
- **Inspection and approval** of the Deliverables by the **Product Owner** and the customer in the **Demonstrate** and **Validate** Sprint process.
3. Adaptation

Adaptation happens as the **Scrum Core Team** and **Stakeholders** learn through **transparency** and **inspection** and then adapt by making improvements in the work they are doing.

Adaptation in **Scrum** is depicted through:

- Standup Meetings
- Constant Risk Identification
- Change Requests
- Scrum Guidance Body
- Retrospect Sprint Meeting
- Retrospect Project Meeting
Agile as a Cultural Change

Not just a different “life cycle” methodology
Agile uses an iterative and incremental approach.
An Agile approach creates a culture of working with transparency, driven by value, and collaboration across the organization…

… enabling a team to deliver often!
LEANING TEAMS

The New New Product Development Game

Hirotaka Takeuchi, Ikujiro Nonaka

From the January 1986 Issue
Agile is a cultural change!
A culture of working with transparency, driven by value, collaborating across the organization.
What makes it work…

• The team collaborates to deliver
• The team limits its work in progress (WIP) … either via iterations or with specified WIP limits
• The team delivers releasable product often … reviewing its work product and process on a regular basis … at the end of each iteration
However… there are “necessary conditions”

- The team is cross-functional with all the needed capabilities and skills to release working product
- There is a “Product Owner” who provides a ranked “Backlog…”
- The team finishes valuable (“go live”) chunks of work often (at the end of each sprint)
- The team can release working software often
- The team reflects on its work process at the end of each sprint
12 Principles of Agile Software Development

1. Attain customer satisfaction through continuous delivery of software
2. Don’t be afraid to make changes
3. Deliver working software, with a preference to the shorter timescale
4. Developers and management must work together
5. Build projects around motivated individuals
6. Face-to-face interactions are the most efficient & effective modes of communication
7. Working software is the primary measure of progress
8. Agile processes promote sustainable development
9. Continuous attention to technical excellence and good design enhances agility
10. Simplicity is essential
11. The best architectures, requirements, and designs emerge from self-organizing teams
12. Inspect & Adapt
Agile development requires a sustainable pace & continuous attention to technical excellence

“When you build small increments and ask for feedback often, you can deal necessary change. The change might be either in the product or in the team’s process. Agile teams fine-tune their work and the product when they reflect and adjust at regular intervals.”
Toyota president Gary Convis

The Toyota Way has two pillars that support it:
- Continuous Improvement and Respect for People

Continuous improvement, often called kaizen, defines Toyota’s basic approach to doing business.

“Challenge everything”

More important than the actual improvements that individuals contribute, the true value of continuous improvement is in creating an atmosphere of continuous learning and an environment that not only accepts, but actually embraces change.

Such an environment can only be created where there is respect for people—hence the second pillar of the Toyota Way.
The **Toyota Way** has two main pillars

**Continuous Improvement** and **Respect for People**

Respect is necessary to work with people. By “people” we mean employees, supply partners, and customers. …

We don’t mean just the end customer… on the assembly line the person at the next workstation is also your customer.

That leads to teamwork.

If you adopt that principle, you’ll also keep analyzing what you do in order to see if you’re doing things perfectly, so you’re not troubling your customer.

That nurtures your ability to identify problems, and if you closely observe things, it will lead to kaizen (that is, **Continuous Improvement**)

The root of the **Toyota Way** is to be dissatisfied with the status quo; you have to ask constantly, “Why are we doing this?”
The process...
The Two Pillars of Lean

“… respect for people”
“… commitment to continuous improvement”

“Lean thinking – using the lean principles – helps agile teams use agile approaches to create better products and deliver more value.”

The Lean Principles applied to Software Development:

1. Eliminate waste
2. Amplify learning
3. Decide as late as possible
4. Deliver as fast as possible
5. Empower the team
6. Build integrity in
7. See the whole
Iteration-Based Agile

“… the Product Owner and the team manage the work in progress by estimating the number of stories (and other work) the team can commit to in a timebox … team estimates provide feedback to the Product Owner on the scope of the required work … the Product Owner has choices, either to add more stories or not.”

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Requirements</th>
<th>Requirements</th>
<th>Requirements</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td>Analysis</td>
<td>Analysis</td>
<td>Analysis</td>
<td>Analysis</td>
</tr>
<tr>
<td><strong>Design</strong></td>
<td><strong>Design</strong></td>
<td><strong>Design</strong></td>
<td><strong>Design</strong></td>
<td>Repeat as needed</td>
</tr>
<tr>
<td>Build</td>
<td>Build</td>
<td>Build</td>
<td>Build</td>
<td>Build</td>
</tr>
<tr>
<td>Test</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
</tr>
<tr>
<td>Deploy</td>
<td>Deploy</td>
<td>Deploy</td>
<td>Deploy</td>
<td>Deploy</td>
</tr>
</tbody>
</table>

Each timebox is the same size. Each timebox results in running tested features.
Flow-Based Agile

“… the team maps the flow of value *through* the team.
… the team sets the limit to the WIP for each column and monitors how long features take on average.
… the team and the Product Owner manage work based on these limits.”

<table>
<thead>
<tr>
<th>Feature</th>
<th>Feature</th>
<th>Feature</th>
<th>Feature</th>
<th>Feature</th>
<th>Feature</th>
<th>Feature</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clarify Req'ts</td>
<td>Clarify Req'ts</td>
<td>Clarify Req'ts</td>
<td>Clarify Req'ts</td>
<td>Clarify Req'ts</td>
<td>Repeat as needed</td>
<td>Design</td>
<td>Design</td>
</tr>
<tr>
<td>Analysis</td>
<td>Analysis</td>
<td>Analysis</td>
<td>Analysis</td>
<td>Build</td>
<td>Analysis</td>
<td>Analysis</td>
<td>Analysis</td>
</tr>
<tr>
<td><strong>Design</strong></td>
<td><strong>Design</strong></td>
<td><strong>Design</strong></td>
<td><strong>Design</strong></td>
<td>Build</td>
<td><strong>Design</strong></td>
<td><strong>Design</strong></td>
<td><strong>Design</strong></td>
</tr>
<tr>
<td>Build</td>
<td>Build</td>
<td>Build</td>
<td>Build</td>
<td>Test</td>
<td>Build</td>
<td>Build</td>
<td>Build</td>
</tr>
<tr>
<td>Test</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
<td>Release</td>
<td>Test</td>
<td>Test</td>
<td>Test</td>
</tr>
<tr>
<td>Deploy</td>
<td>Deploy</td>
<td>Deploy</td>
<td>Deploy</td>
<td>Deploy</td>
<td>Deploy</td>
<td>Deploy</td>
<td>Deploy</td>
</tr>
</tbody>
</table>

In flow, the team limits the number of features active at any time with WIP limits for each team activity.

There is no timeboxing build into flow.
Cadence versus Iteration

**Iteration**… has the team planning on the same day at the beginning of each iteration…
… with the team holding a retrospective on the same day as the end of the iteration.”

**Cadence**… not relying on a fixed length timebox. Teams might “deliver” several times during the week… not always on the same day.
… planning work may not be on the same day each week.
… But retrospectives once a week on the same day each week.
Integrating Agile and Lean Principles

Use of collaborative cross-functional team structure so that the entire team works on the same feature(s)

Emphasis on seeing the work as it proceeds... to get feedback on the work and the process

The teams do not start more than they can complete in a short period of time (a sprint or iteration)

Delivering working product often and as fast as possible

To

See progress

Increase customer collaboration

Receive feedback
Agile methods...

<table>
<thead>
<tr>
<th>Named Approach</th>
<th>How the Approach Works</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scrum</td>
<td>Timebox-based project-management framework for delivering working product often</td>
</tr>
<tr>
<td>Extreme Programming</td>
<td>Collection of technical practices guided by these values: communication, simplicity, feedback, courage, and respect</td>
</tr>
<tr>
<td>DSDM (Dynamic Systems Development Method)</td>
<td>Timebox approach to delivering functionality. Facilitated workshops to determine the requirements and gain agreement on them.</td>
</tr>
<tr>
<td>Crystal</td>
<td>Focus on the people. Depending on the size of the project team and the product criticality, select the approach that fits the team, the business people, and customers.</td>
</tr>
<tr>
<td>Feature-Driven Development</td>
<td>Deliver functionality incrementally after creating a flow-fidelity) framework for the architecture or object modeling. Focus on building value for the customer.</td>
</tr>
<tr>
<td>Kanban</td>
<td>Visualize the flow of work, work by value, and manage the work in progress (WIP). Deliver incremental value as the team completes the value.</td>
</tr>
</tbody>
</table>

Table 1 - Some Agile Approaches
Scrum

The Agile Scrum Framework at a glance

Inputs from Customers, Team, Managers, Execs

Product Owner

Product Backlog

1. Prioritized list of what is required: features, bugs to fix...

2. 3
3. 4
4. 5
5. 6
6. 7
7. 8

Sprint Planning Meeting

Sprint Backlog

Team selects starting at top as much as it can commit to deliver by end of Sprint

Sprint Backlog

Task Breakout

Sprint Master

Burn Down/Up Chart

Daily Standup Meeting

24 Hour Sprint

1-4 Week Sprint

Sprint end date and team deliverable do not change

Finished Work

Sprint Review

Sprint Retrospective

Scrum
Not just any team is a good candidate… for Scrum

A team working on more than one project
A team that is distributed across more than 4 time zones
A team that is not cross functional
  Developers must collaborate with testers
A team that does not have skill or capabilities it needs
  example: no UX or DBA
A “team” where all work is done independently… without a culture of collaboration
Keeping iterations relatively short is essential

**Key point is the need for feedback**

… how much time before the team would know if what the team is doing is “wrong”?

“In my experience, it’s easy for a duration of more than 10 days to lead to the trap where you end-up with iterations that are Waterfalls…

… be careful of iterations more than 10 business days.”

Rothman
Keep iterations short!
Looping until you meet the release criteria for the project…

“Build something, release it to someone else to gain feedback… use that learning from feedback to decide what to do next.”
Create a Agile “Mindset”

… meaning the values, beliefs and principles that guide your actions in any situation

… and the valuing collaboration & feedback

… and the valuing in small steps & frequent checking of progress

<table>
<thead>
<tr>
<th>Fixed Mindset</th>
<th>Growth Mindset</th>
</tr>
</thead>
<tbody>
<tr>
<td>We are born with fixed skills or talents.</td>
<td>Skills arise from hard work. We can improve.</td>
</tr>
<tr>
<td>Avoid challenges. In the face of challenge, give up.</td>
<td>See challenges as an opportunity. Persist until we get it right.</td>
</tr>
<tr>
<td>Coast by: don't bother with effort</td>
<td>Effort is essential to mastery.</td>
</tr>
<tr>
<td>Get defensive with feedback.</td>
<td>Learn from feedback.</td>
</tr>
<tr>
<td>Blame others for setbacks. Get discouraged by setbacks.</td>
<td>Setbacks are something we use to try harder the next time.</td>
</tr>
<tr>
<td>Fell threatened by others' successes.</td>
<td>Find inspiration in others' success.</td>
</tr>
</tbody>
</table>

Table 2 - Fixed vs. Growth Mindset
“Avoid frameworks until you understand the full context to the work”

Advice

“… many frameworks are too prescriptive and do not create the agile mindset for a team”

all too often “… management thinks the framework is all the team needs.”

Rothman
Framework

Each programming language has at least one universal, reusable framework.

Libraries within each framework offer reusable bundles of that language (code for a drop-down menu)

Frameworks are more than just the code—frameworks are fully layered workflow environments.

They can include:
Libraries: shareable, reusable bits of low-level code in each language, e.g., Ruby on Rails’ “gems”

APIs, which facilitate access to the database back end

Scaffolding: a technique some MVC frameworks employ that strengthens how a database can be accessed. This means more powerful sites, with better leverage of the database.

AJAX: Some JavaScript frameworks are embedded into larger frameworks, incorporating the rapid technology of AJAX into a site’s functionality.

Caching, which cuts back on server workload

Security, via authentication and authorization frameworks

Compilers, or Just-in-Time compilers