Introduction

By definition, this agile guide is a work in progress. We welcome community participation.

Prior to 2001, a majority of software development products failed to actually be useful to real people or were built over budget with missed deadlines. Early-stage technology coupled with an intuitive but ineffective development approach combined to make the actual delivery of successful, working software extremely difficult. During the 1980s in particular, the growing demands and expectations of the marketplace swamped the ability of the software development industry to build software as it was needed.

In addition, the widely-held belief that development teams could predict customer needs far in advance--sometimes many years in advance--turned out to be incorrect. So even when working software was delivered, it often did not meet the expectations of customers.

Analysis and experimentation conducted during the 1990's suggested that the so-called waterfall development process was largely responsible for this climate of failure. After ten years of work, the Agile Manifesto was published in 2001. The Manifesto ushered in the age of Agile software development by outlining a framework for a different approach to the problem. The (then) new Agile approach featured outreach to potential users of software, decomposition of large software projects into much smaller projects that were much less difficult

and risky, and empowerment of development teams to respond to evolving requirements.

Open Source and Contributing

This project is open source, and we welcome contributions from the public.

License

This project is in the public domain within the United States, and copyright and related rights in the work worldwide are waived through CC0 1.0 Universal public domain dedication.

All contributions to any repositories of this project will be released under the CC0 dedication. By submitting a pull request, you are agreeing to comply with this waiver of copyright interest.

Modern software product development

There are three basic approaches to software development: Waterfall, Agile, and Chaos.

Chaos is when you don't really have a plan, and you don't really learn anything, but everybody's super busy all the time. Chaos is often characterized (or justified, if you want to be more critical) as "firefighting", and it is suggested that there is no time to plan or learn because there is just so much to do.

Waterfall is when you lay out The Plan in full before you start: every detail, every feature, every meeting, every benchmark, and then you're wrong on most of them but forge ahead anyway, secure in the knowledge that following The Plan gives air cover in the likely event that the thing you're building doesn't work the way you expected it to when you made The Plan.

Agile is different. Agile has an end result in mind, and maybe a timeframe, but it only gets into the details as needed, which allows for change in response to new information.

Think of the three as a road trip from Washington DC to San Francisco:

With Chaos, you just start moving, and hope you wind up in San Francisco eventually. Maybe you go west, maybe not, maybe on foot, maybe by train, whatever, just keep moving!

Waterfall would have you carefully and exactingly mapping out your trip before you leave. Gas stops, meals, sleep, mileage targets, route. This would be fine if

there were never traffic delays, weather issues, car problems, or interesting detours and stops along the way. You can change nothing, and if something goes wrong, the whole trip fails.

With Agile, you decide you want to drive from DC to San Francisco and you want it to take five days, and you get in the car and go. You can flex around construction, bypass snow, and even choose to end up in San Diego instead because you heard San Francisco was cold and expensive.

(Some folks say there is a fourth option, “Agilefall”, but that’s really just Waterfall with new names.)

Agile is a thing already

Agile, as we know it, means any development framework or methodology that adheres to all or most of the values and principles listed in the Agile Manifesto, which was published in 2001. The Manifesto is the end result of several streams of exploration and experimentation that continued throughout the 90s (1990s, not 1890s). These streams resulted in the invention of eXtreme Programming, Scrum, Feature Driven Development, Crystal Clear, and other development paradigms. The genius of the Manifesto is that it captured in a very high level way the common philosophical foundations of these new approaches and dubbed those values and principles Agile.

In the 16 years since publication of the Manifesto, Scrum has become the Agile framework of choice around the world, and also the software development framework of choice around the world. The engineering practices associated with successful, high performance Agile software development have coalesced into the Devops movement, and in the modern world it seems incomplete to implement Agile without DevOps, and vice versa. Additionally, user-centered design approaches have learned to take advantage of the feedback cycles built in to Scrum. Dozens of books and hundreds or thousands of blogs, articles, papers, and videos have been created on the subject of Agile, Scrum, DevOps and associated topics.

The plethora of material available on the subject of Agile software development leads one to wonder why so many organizations feel the need to re-invent Agile by creating their own homegrown dogma. They choose various subsets of Agile

to preach, and sometimes to follow. We will default to using the discoveries and methods and knowledge of Agile in the existing worldwide community, only inventing our own when necessary.

Having said that, it's important to note that it is assumed that basic Agile dogma such as the Scrum Process Framework requires supporting process and culture that arise from the local environment in order to work well. You could say that Agile is necessary but not sufficient in terms of creating the best product development approach for you.

Agile is something you are, not something you do

If you take nothing else from all these words, take this. Agile is not a checklist, or a methodology, or a series of rituals. Agile is a way of thinking and a way of attacking problems. Embrace mistakes, learn, and keep trying. Mess up and learn again and again and again. Cut your losses. Fail forward fast. It's okay. You won't get fired. You're learning.

That is Agile.

18F Agile based project approach

The Agile Manifesto is realized at 18F in the combined practices of iterative software development, product management, user-centered design, and DevOps.

We start with a product vision and strategy, informed by users and the overall mission of 18F or one of our partner agencies. We do this so that the work always stays connected to an overarching goal that everyone understands and is excited about.

We also work to ensure that the infrastructure and process is there to enable continuous delivery of software to real users (DevOps), and that a clear agile delivery process is set up. Teams are free to tailor their agile process to suit their own situations. Many of them employ the SAFe methodology, while other teams use more traditional Scrum or Kanban methods.

We conduct discovery research before we build anything. Depending on the complexity of your problem space, this can take up to 2 to 3 months. As opposed to ‘requirements gathering’, this process involves actually visiting with users and prototyping to test out multiple concepts quickly before investing a lot of money in building something.

When we build, we aim to release early and often to real users in real life situations. Ultimately, the government’s investment should be measured in working software, not phase documents or milestones. Only working systems are of value to real constituents. Based on that premise, teams are only able to

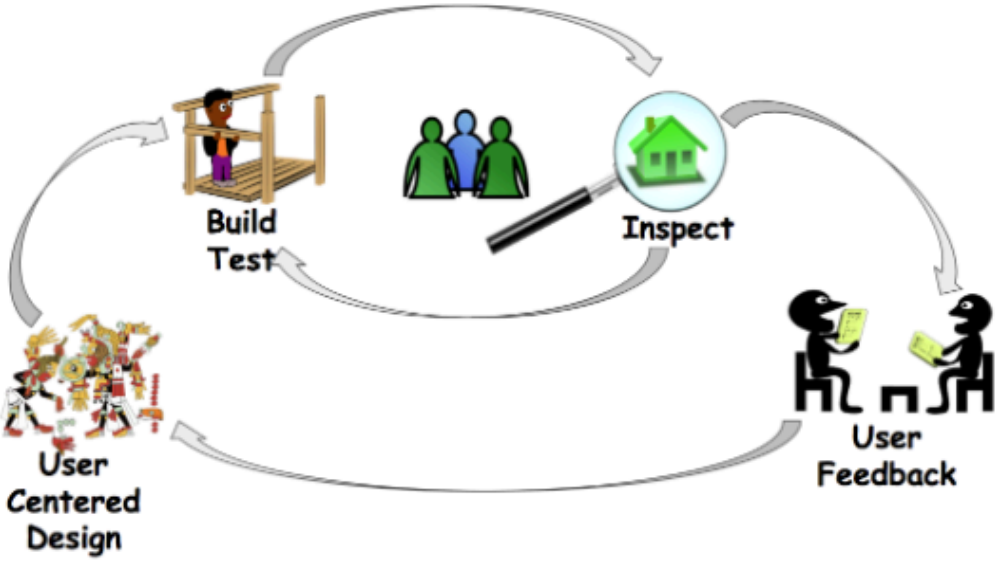
measure the success of a waterfall project after most costs have been incurred, because that is the first time working software is delivered. This is a huge risk. Agile projects allow the government to measure success at more regular intervals and if necessary make course corrections. The ability to measure and adapt reduces the overall risk to the project.

Not all of our projects are greenfield; sometimes we work on legacy systems. Incremental development is important in these situtations as well. Incremental development can allow end users to start seeing benefits even before the legacy system can be fully replaced. If a workflow is worth preserving, then portions of the old systems must be kept up until the new system is fully operational, no matter what approach is used. In a waterfall approach, the end user must wait many months or years to use the new system and see any benefit — and that's assuming there are no problems with the rollout. With an agile approach, the end user could start seeing benefits in a shorter timeframe. Depending on the architecture of the legacy system, parts of it may even be able to be decommissioned and turned off before the full new system is in place to save money.

Having well-researched hypotheses beforehand allows us to be deliberate about what we build and why. Research continues throughout the agile process that we can test our hypotheses and pivot when needed. Since our work is centered around user research and user feedback, it's important to develop participant recruiting strategies, and a research and feedback cadence that will work within the constraints of your agile process, and to ensure that the team is staffed appropriately.

A common pitfall is expecting agile to be a silver bullet to all that ails software development, and to expect agile to eliminate all project risks. We've seen teams try agile and, when the practices fail to eliminate all the challenges, write off agile entirely, claiming "agile doesn't work." In reality, agile does not eliminate risk completely; it provides techniques to manage risk more effectively than traditional waterfall processes. Unlike waterfall, agile accepts that the unknowns will lead to change. Agile treats change as an integral part of the process, as opposed to exceptions that need to be resolved via change control mechanisms. This enables agile teams to manage risk by allowing change to drive course corrections. However, some agile adoptions focus on the ceremonies as opposed to the intent. Incomplete, incorrect, or uninformed adoption of agile techniques can lead to an ineffective process we call “agilefall.” 18F has written a blog post about how to avoid this pitfall. Effective agile adoption will enable an organization to be nimble and respond effectively to the inevitable change that arises during software development.

Agile Development Integrates with User-Centered Design



Agile fundamentals

- Standups: the team meets daily for short meetings which are typically held standing up, face-to-face to encourage brief sessions. This is not a status meeting. This meeting is for people to ask quick questions that will allow them to get information or remove blockers. Long answers and discussions should have follow-up in smaller groups after the standup meeting.
- Retrospectives: on a regular basis, at end of each sprint, weekly or bi-weekly, a retrospective allows for the team to reflect and adjust practices. Any team member can voice a problem or propose a solution
- Sprints are the heartbeat of the agile process. Small units of work are delivered in short bursts, typically with 1 or 2 week cycles. We aim for visible progress for people from the target audience that is delivered and validated at the end of each cycle, allowing the team to move iteratively toward the goal, with regular opportunity for course correction.
- Sprint planning: include the whole team in reviewing stories, breaking down use cases into small user-facing stories
- Sprint Review: A ceremony at the end of each sprint when completed stories are demonstrated to team, stakeholders, and users.
- Backlog Refinement: An ongoing team activity of collaboratively updating the Product Backlog via reprioritization, adding/deleting/rewriting stories, splitting, and estimating. This practice ensures that the backlog is always actionable. (Note: This practice was formerly known as “grooming”, which term is out of favor.)

- Transparency: The idea that information should be shared freely within and between all Agile teams, projects, and stakeholders.
- Iteration (Inspect and Adapt): A problem-solving and development approach that solves large problems by decomposing them into smaller, discrete problems which are each easier to solve, then solving the smaller problems. As each smaller problem is solved, new information is uncovered (Inspect) and decisions can be made and re-made based on the new information (Adapt).
- Potentially Shippable Product Increment: In Agile software development, a fully tested and usable version of a product that is produced at the end of each sprint.
- Self-organizing Teams with Empowered Product Owners: Agile teams are composed of peers who share ownership of the team's work and decision-making processes. Self-organizing teams in Scrum are given ownership of their work process, their commitments, and their approach to meeting their commitments. Product Owner is the role in Scrum that represents the business and customer directly within the development team. The Product Owner must be empowered to make product decisions in response to feedback from stakeholders and customers. Taken together, a Scrum team has complete control over how it does its work and what work it does.

Agile lexicon

The goal of this Lexicon is to provide 18F and our partners a common language for discussion of Agile processes. The definitions are as simple, flexible, and inclusive as we could make them.

High Level Concepts

- Agile - pertaining to the Agile Manifesto and/or Agile Principles (http://agilemanifesto.org). Agile is a set of values and principles that describe a way of working that promotes continuous learning and user-focused value delivery.

Agile Disciplines

- Lean UX - a set of design principles which includes early customer validation, collaborative design, defining key success metrics (for more info see: Lean UX manifesto).
- Scrum - a framework for Agile product development centered around self-organizing team, customer focus, and responding to change.
- XP (Extreme Programming) - an agile software development approach that emphasizes business results first, using pair programming and test-driven development, delivering value continuously
- Kanban - a process management and diagnostic tool that can be used with both Agile and non-Agile environments. A Kanban board can be a

helpful tool for introducing a few Agile behaviors to a team less familiar with the methodology.

Roles

- Partner - The agency with whom we are working. 18F doesn't have "clients," we have partners. We deliver something together with agency partner.
- Client - When the government hires a vendor to perform services, we are a "client" to that services organization. We cannot delegate responsibility for success to the vendor. In keeping with the Agile principle of small, empowered teams, at least one person from the Client team must be an active part of the Agile team. (For an agency or organization working directly with 18F, see "Partner".)
- Customer - the purchaser of a (digital) product/service.
- User - the consumer of a product/service
- Product Owner (PO) - owns the product vision and incorporates stakeholder feedback to set priorities and manage the backlog
- Stakeholder - an interested party whose buy-in you want (or need) but is not the PO

How we talk about the work:

- Prototype - A functional representation of a feature or group of features not meant to be sold to customers, but rather to get feedback from potential users or customers.

- Minimum Viable Product (MVP) - The smallest usable thing you need to start validating the actual idea of the business or product. The very first increment of the build/measure/learn cycle (Zappos example in The Lean Startup.) It's there to help demonstrate the market exists for a particular idea. The term "MVP" means something different to virtually everyone, so be sure to get clarity from the person or persons using the term in conversations.
- Release - a usable increment of customer value delivered to users in the wild.
- User story - a description of functionality or value written from the perspective of the user. Each story should be made available to users as an incremental improvement; however, it often makes sense to collect a set of user stories that will be promoted or validated together as a release.
- Definition of done - A working agreement among the team detailing the standard for achieving a "Done" Product Backlog Item (i.e, user story). This applies to all user stories and includes meeting acceptance criteria.
- Acceptance criteria - A set of conditions detailing the functional goals of the user story. This is a user story-specific description of when that story is done. The acceptance criteria could be used to demo the user story or to write an automated test.

Concepts:

- Non-Functional Requirements (NFRs) - desired characteristics of a product that do not deliver direct user value, such as load handling, performance, browser compatibility, or responsive design.
- Test coverage - the portion of the code that is actually exercised when a certain set of tests are run. Usually expressed as a percentage. Often specified as a quality standard, as in, “you must have 90% code coverage to consider your tests adequate.”
- Unit tests - Automated tests that concentrate on verifying that the code works as written, and which act as safety flags in high-churn agile development environments with constant check-ins, changes, and multiple developers in the same code. They are best run automatically as part of every build.
- Integration tests - Also called Feature Tests, Acceptance Tests, and Regression Tests, these are tests that validate a complete, integrated software product against its business goals. They test the user-visible behavior of the software.
- Continuous Integration - Running automated tests on every code submission, within an integrated codebase.
- Continuous Deployment - Deploying software automatically *to production* if automated tests pass.
- User research: a range of techniques are used to understand the target audience and the problems that they have. Contextual inquiry, involving direct observation, is favored over interviews. Usability testing: Observation of people attempting to use a product, prototype or mockup. See Design Method: Usability Testing

- Prototype: A term that means something different to everyone. You want to come to common understanding about this word early on in any project; but it should be noted here that prototypes should not be referred to as a deliverable and are best described as a supporting activity that promotes learning.

Process words, how we work:

- Sprint - the unit of iteration in scrum: a short, fixed interval in which an agile team commits to and delivers a piece of customer value from the backlog.
- Sprint Backlog Item - A more granular item than one one would find in a product backlog. It could be a task, chore, bug, or small or finely sliced user story. These items are usually what you find in a trello or waffle board.
- Product Backlog - A prioritized list of user value to be produced.
- Scrum Master - the team member who is responsible for the team's rituals and process, who ensures the team can execute on its commitments by removing blockers and facilitating activities like the daily standup, sprint planning, and retrospectives.
- Velocity - Any measure of the value produced during each sprint. Average velocity can help guide time-boxed iterations. Velocity is most useful for planning and tracking purposes and does not measure team productivity or performance.

- Daily Standup (aka Daily Scrum) - A short, daily meeting during which the team self-organizes for the day. Never more than 15 minutes. Information exchanged can vary but always covers what each team member did yesterday, what they plan to do today, and what if anything is blocking their work.
- Retrospective - A meeting of the whole team held once per sprint, wherein the team reflects on what happened in the iteration and identifies actions for team improvement going forward. This is a critical part of the Agile continuous improvement, as contrasted with a post-mortem which happens only after launch, which limits the opportunity for applying lessons learned.
- Kickoff meeting - The meeting held with all stakeholders before work begins. The kickoff meeting helps align the team on the project goal, terminology, the day to day process expectations, and roles and responsibilities.
- Waterfall/SDLC - a sequential design process, used in digital development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of conception, initiation, analysis, design, construction, testing, production/implementation and maintenance. Primarily used in instances where requirements are not expected to change and user-designer interaction is not desired. (Source: https://en.wikipedia.org/wiki/Waterfall_model)