Development - Development Playbook

01/02/2024 6:47 pm EST

Overview

This document describes the set of strategies and tactics used to develop and delivery software at Bold. The intended audience is new Development team members.

The Bold Development team is passionate about great software and a satisfying work environment that is free of drama. To this end, we have developed an efficient process that both maximizes customer satisfaction and reduces team member frustration. Our key to success is based on a shared appreciation for high quality, both perceived (i.e., overall perception of the product) and actual (i.e., the product meets the requirements). Striving for high quality is not only good for business, but it also reduces rework which improves overall efficiency.

Key Principles

Our software development strategy is guided by several key principles:

1. Make all work visible - Providing access to information enables team members to make the best possible decisions. We strive for complete transparency and require that all work items necessary to complete a task appear in our work item tracking system (DevOps). When you have complete knowledge of what your coworkers are working on, you can understand how their efforts impact your own work. When your team lead can see all the steps required for you to complete a task, s/he can adequately communicate any schedule changes to other stakeholders in the organization.

2. A bug is a test case that fails - When a bug makes its way past our QA process and into production, it is often because we didn't test something. To ensure that each bug never pops up again, we create written test cases for every bug, and add these test cases to an ever-growing arsenal of regression tests. As a general rule, every code change has a set of associated test cases, and we do not mark the change as "done" until the tests pass. Additionally, we record the results every time we execute a test case to document the validation of the code change (i.e., "Make all work visible"). As a result, we know exactly what was and was not tested each time we release new code.

3. Find bugs early - Studies have shown that the cost of a bug increases the longer it takes to detect it. This is common sense: if a QA engineer finds a bug before it is released, s/he can work with the developer who introduced the bug to fix the issue. The "cost" of the bug is the effort for two team members. Alternatively, if a bug makes its way into production and is found by a customer, the "cost" is much higher: the tech support technician who talked to the customer, the manger who handled the escalation, the QA engineer who reproduced the bug, the developer who fixed it, etc. Because of this, we use a variety of tactics to discover bugs as quickly as possible.

4. We cannot depend on individual achievement - One strategy for success is to staff a project with superheroes who are perfect in every way: they understand the entire code base, never make mistakes, and never let anything fall through the cracks. This works for small teams; however, this approach does not scale as the team grows. While we only hire talented team members, our code base is large and it takes time to learn everything - everybody makes mistakes. To compensate, we use tactics such as code reviews and testing, to check each other's work.

We are always striving to improve and evolve our process to best fit our company culture, industry, and team in order to achieve the most efficient operation for developing our products. To this end we have a split process for projects and maintenance workflows. Each one is tailored to fit the type of work and demands on the team as well as protect our team from chaos by flowing a guiding process.

Agile Scrum

> **scrum** (plural *scrums*)
>
> 1. A tightly-packed and disorderly crowd of people.
> 2. (Canada) Specifically used in the Canadian media to describe a tightly-packed group of reporters surrounding a member of a member of the Canadian House of Commons while in the Parliament Buildings.
> 3. (rugby) In rugby union or rugby league, all the forwards joined together in an organized way. Also known as a *scrummage*.
> 4. In Agile software development, a daily meeting in which each developer describes what they have been doing, what they plan to do next, and any impediments to progress.
>
> -- Wiktionary

We use the Agile Scrum framework to plan our project work and deliver new feature releases to production. Agile Scrum is a popular framework in which the team plans and delivers a completely working version of the product on a fixed periodic timeframe called a sprint or an increment. In our implementation, each sprint is three weeks in which we plan, code, test, and demo.

Key Concepts

> "The aim of the game is very simple - use the ball to score more points than the other team."
> -- BBC Sport. "The Basics of Rugby Union"

Here are some key concepts around the way we use Agile Scrum:

- Backlog - We maintain a backlog, or list, of all future changes and work tasks. The Product Owner is responsible for keeping the highest priority items at the top of this list. When the team plans the next sprint, they pull three week's worth of work items from the top of the prioritized product backlog and moves it into the sprint backlog. The sprint backlog is a piece of the product backlog that has been committed for the next sprint. A good backlog should align with DEEP principles:
    - Detailed - Backlog items should have enough detail so that everyone on the team can understand the intent.

Any team member should be able to look at the item and understand what is required.
    - Estimated - The team should beable to estimate the backlog items appropriately based on the level of detail given.
    - Emergent - The backlog should evolve to reflect the most current market or business needs (add new items and refine or delete existing backlog items on an ongoing basis).
    - Prioritized - Backlog items are ranked in order of importance with the top items being the ones that will be implemented next.
- Effort - During planning sessions, the team will use relative estimation to size new backlog items. Using the point scale in the diagram below, we take into consideration the volume, complexity (including how long it will take someone to perform the task), current knowledge, and uncertainty of an item.

mall	Medium	Large	XXL
1 point	3 points	8 points	100 points
1-2 days Dev and unit testing	3-5 days Dev and unit testing	6-10 days Dev and unit testing	Too large! Needs to be broken down into smaller chunks.
"Judge me by my size, do you?" - Yoda	"Pit ti tu tipt ti ti tuuuuuuuuuu ti pip tu tuuuut pi piii pi pi." - R2D2	"WWWWWWWGGGGHHHRRRRW." - chewbacca	"You've never heard of the *Millennium Falcon*?" - Millenium Falcon

- Capacity - Prior to the start of each sprint, we estimate the "capacity" of the team using the same point scale as effort for the upcoming sprint. Capacity is the availability of the team taking into consideration vacation, holidays, and other commitments. This gives us the number of points the team expects to complete in the sprint. We multiply the capacity by 90% to create a buffer for unplanned events (e.g., sick days). The capacity tool in DevOps tracks the team's work days and time off to help make sure the team isn't overcommitted in a sprint. For the first capacity estimate, we will figure that each full-time Agile team member has a capacity of 12 points per three-week sprint. Subtract 1 point per day off for each team member then add up the total number of days for the entire team. To account for incorrect capacity or point estimates, we multiply the team's capacity by 60%.
    - Example for a team of 5, 1 vacation day each:

      Estimate Capacity = 5 * 11 pts = 55 pts/sprint

      Final estimated capacity = 55 * 60% = 33
    - Capacity Allocation: Capacity allocation refers to how much of the total effort is allotted to each type of activity. The following shows how we will allocate effort in each sprint:
        - 60% Feature Development (New and Improvements)
        - 20% Bugs
        - 10% Paid Enhancements
        - 10% Buffer
- Velocity - After running through a few sprints, we can use the DevOps forecast tool to better estimate the team's velocity. Velocity is based on the average points completed in previous sprints. We use the velocity measurement to predict how many points worth of work the team can complete each sprint. Over time, our velocity estimates

will become more and more accurate.

- Increment (or Release) - The calendar year will be broken up into 4 increments (a timebox spanning multiple sprints), each lasting one quarter. We will deliver a major release at the end of each quarter and a minor release on demand. A major release consists of completed features/epics and bug fixes while a minor release will only consist of bug fixes.
- Sprint - Each sprint duration is 15 weekdays. A sprint will never be extended due to events such as holidays; days off are counted in the sprint duration. During the planning session, we pull enough items from the top of the prioritized backlog so that the total number of points matches the expected capacity in the sprint. The team commits to delivering a fully tested release candidate with all the items in the sprint by the end of three weeks. It is recommended that sprints begin and end in the middle of the week since most holidays/vacation days are at the beginning or end of the week; therefore, our sprints are scheduled to begin on Wednesdays and end on Tuesdays.
- Sprint Goals - Sprint goals are high-level business goals the team reviews and commits to each sprint. These goals that align with overall mission. These goals reflect what we intend to accomplish in the sprint.
- Acceptance Criteria - Clearly define the conditions, specific to the item, that need to be met for the item to considered complete. This provides more context for the team when working on the item. Acceptance criteria can be gathered from the customer to understand what they need (e.g., understanding the customer's formatting preferences). Below are some examples of criteria formats:
    - Test that <criteria>.
    - Demonstrate that <this happens>.
    - Verify that web <a role> does <some action> they get <this result>.
    - Given <a context> when <this event occurs> then <this happens>.
- Requirements Gathering - Before we begin development of new features, we ensure that all stakeholders agree on the criteria for success for the code change. To communicate the criteria with the team, we document the specific requirements along with each feature (e.g., who will use the product and how will they use it. These requirements are specific enough to prevent confusion during development and verification of the new feature. We also use test cases as a means to document the requirements. This ensures that the developer and the tester are in sync with regards to desired behavior.
- Innovation and Planning (IP) Iteration - We will close out each Increment with a one-week Innovation and Planning Sprint. During this week, we will not plan any new backlog items. Instead, we use this time to demo, plan, and continually improve. This time may also be used to clean up any backlog items or bugs from the previous sprint. This time acts as a short break to help the team avoid burnout.

Team Roles

> "With great power comes great responsibility"
> -- Uncle Ben Parker, Spider-Man

The key roles we will be using are the following:

- Scrum Master - The Scrum Master is responsible for removing impediments to help keep the team on track. S/he facilitates meetings, helps the team focus on executing the tasks, and helps them continually improve their work. The Scrum Master gives final approval before each release is deployed.
- Product Owner (PO) - The Product Owner has primary responsibility for aligning the sprints to the overall vision and leading the maintenance of the team backlog including reviewing, prioritizing, and creating new backlog items as needed. S/he assists in incorporating the voice of the customer and clearly defining requirements for all backlog items. The PO accepts backlog items as done based on the defined acceptance criteria and definition of done.
- Agile Team - Agile Team members, consisting of Developers, Software Engineers, and QA Engineers, are responsible for defining, building, testing, and deploying features and components. The team is also responsible for planning, sizing, and managing their work and overall creating value. The team is self-managing and will update their progress in DevOps as they work through an item.

Release (or Increment) Lifecycle

We will deploy a major or minor release to production at the end of each quarter (a quarter is equal to one increment). the diagram below shows a calendar of activities when deploying a release:

- Deploy Major Release to Production - At the end of the fourth sprint in each increment (aligns with the end of the quarter), we will deploy a major release consisting of completed features/epics and bug fixes. Before deployment, the QA team runs a set of smoke tests to verify that the new release is operating properly.

Sprint Lifecycle

> "If you won't run, you can't win."
> -- Sybil Gordon (perf. Alice Krige). Chariots of Fire (1981; Dir. Hugh Hudson)

Every three weeks, we plan, code (execute), and test (check) a new version of the product. We also fine-tune (retrospective) our process to continually improve our sprints. Each sprint will be named according to the increment and sprint it is in (e.g., a sprint in Increment 3, Sprint 2 will be titled "2019 Q3 Sprint 2"). The following calendar shows the sprint lifecycle:

Monday	Tuesday	Wednesday	Thursday	Friday
		Sprint Meeting ***1st day of sprint***	Daily Standup **Backlog Planning** **(Scrum Master & Product Owner)**	Daily Standup
Daily Standup	Daily Standup	Daily Standup	Daily Standup **Backlog Planning** **(Scrum Master & Product Owner)**	Daily Standup
Daily Standup	Daily Standup	Daily Standup	Daily Standup **Backlog Planning** **(Scrum Master & Product Owner)**	Daily Standup **Full Length Demos**

Monday	Tuesday	Wednesday	Thursday	Friday
Daily Standup	Daily Standup ***Last day of sprint***			

Planning, coding, and testing a new version involves the following activities in addition to the ceremony meetings in the next section:

- Product Owner updates prioritized backlog - As noted in the description of the backlog above (see Key Concepts), the Product Owner reviews the backlog before the backlog refinement meeting and ensures that the items at the top of the backlog are in the correct order, with the highest priority items at the top.

- Scrum Master estimates team capacity based on historical velocity - The Scrum Master calculates the average velocity in points per sprint for the last few sprints, then adjusts this velocity calculation based on the number of team members that will work on the upcoming sprint. For example, if five team members completed an average of 100 points in the last few sprints, and only four team members will contribute to the upcoming sprint, then the team capacity for the upcoming sprint is 80 points. This also needs to be done before the backlog refinement meeting.

- Sprint Meeting Schedule (3 hours):
    - 1 hour - Team Review & Demo
    - 30 minutes - Retrospective
    - 1 hour 30 minutes - Sprint Planning

Ceremony Meetings

> "If you had to identify, in one word, the reason why the human race has not achieved, and never will achieve, its full potential, that word would be 'meetings'."
> -- Dave Barry

While the general impression of meetings is, to quote Star Trek's captain James T. Kirk, "an event where minutes are taken and hours wasted," productive, effective meetings can actually make what we do more efficient.

As a team, we will participate in five ceremony meetings that facilitate communication across the team and enable the development of our agile practices. Refer to Team Roles for a description of the team roles. The calendar below illustrates when the ceremony meetings take place with respect to the sprint.

Backlog Refinement

> "Truth, like gold, is to be obtained not by its growth, but by washing away from it all that is not gold."
> -- Leo Tolstoy

Before each sprint begins, the team reviews the items in the backlog and estimates/re-estimates the effort for the items near the top of the backlog using points. The process for estimating effort is described in Key Concepts. We also break down items and add requirements as needed (e.g., description, acceptance criteria, etc.). Although there is a dedicated

backlog refinement session, this is an ongoing effort, meaning the team can refine throughout the sprint.

Attendees:

- Product Owner
- Scrum Master

Agenda:

- Review and update backlog item definition and develop acceptance criteria.
- Work with the teams to establish technical feasibility and scope estimates.
- Analyze ways to split backlog items into smaller chunks of incremental value.
- Determine the enables spawned by new features and capabilities and establish their capacity allocation.

Sprint Planning

> "It does not do to leave a live dragon out of your calculations, if you live near him."
> -- J. R. R. Tolkien

This is a relatively simple exercise, since the important work was already completed in the Backlog Refinement meeting. To plan the sprint, the Scrum Master pulls PBIs from the backlog one-by-one until the size of the sprint (in points) matches the estimated capacity of the sprint (in points). The processes for estimating capacity and velocity are described in Key Concepts. The Scrum Master leaves a 10% pad to account for any error in estimation. For example, if the team capacity for the upcoming sprint is 80 points, the Scrum Master pulls in just enough PBIs to add up to 72 points. The team commits to completing all the items in the sprint by the end of three weeks. The commitment is not just to write the code; instead, it is to prepare to deliver a fully tested release to production. This meeting is time-boxed to 4 hours.

Attendees:

- Product Owner
- Scrum Master
- Agile Team
- Other Stakeholders, as needed (e.g., SMEs)

Agenda:

- The team determines the velocity available for the sprint.
- The Product Owner presents the sprint goals; the team discusses and agrees on the goals.
- The team discusses each backlog item in ranked (sequenced or prioritized) order. For each backlog item, the team:
    - Sizes/resizes each PBI in points and splits them if necessary.
    - Elaborates acceptance criteria through conversation.
    - Based on size and value/time/risk, the PO may re-rank backlog items.
    - Optional: In ranked order, the team breaks backlog items into tasks, estimated in hours, and takes responsibility. (This is generally done by the assigned team member after the meeting)
    - Planning stops once the team runs out of hours.

    - The team and PO negotiate and finalize the selected backlog items
    - Everyone commits to the sprint goals

Daily Stand-up

> "When eating an elephant take one bite at a time."
> -- Creighton Abrams

Daily Stand-ups encourage constant communication between team members and occur a the same time and place each day. It is imperative that team members arrive on time. They are time-boxed to 15 minutes and cover the brief agenda items below (remember not to go into detail. That is reserved for a later time). Topics requiring further discussion will be noted for a "Meet After" meeting that includes only the involved individuals.

Attendees:

- Product Owner
- Scrum Master
- Agile Team

Agenda:

- What did you do yesterday?
- What will you do today?
- What is blocking progress?
- Are you on track?
- As needed: Write down topics for a "Meet After" and invite the involved individuals to a followup meeting.

Sprint Review and Demo

> "The trouble with programmers is that you can never tell what a programmer is doing until it's too late."
> -- Seymour Cray

The review and demo meetings bring closure to the sprint and allows the team to demonstrate new functionality. Team members demo work throughout the spring as soon as it is completed to ensure problems are addressed quickly. It should be time-boxed to 1-2 hours.

Attendees:

- Product Owner
- Scrum Master
- Agile Team
- Other Stakeholders (e.g., Management)

Agenda:

- Review the business context of the sprint and sprint goals.

- Demonstrate only working, tested systems of completed backlog items or features. Gather feedback.
- Discuss any stories that were not completed and understand why (this may help planning for the next iteration).
- Identify new current risks and impediments that may have emerged from the sprint of the demo.

Retrospective

> "Hindsight is always 20/20."
> -- Billy Wilder

The retrospective is a brainstorming session intended to reflect on the sprint just completed and discuss ways to improve the upcoming sprint. Focus on things the team can address rather than how individual team members can improve. It is helpful if you only choose one or two things for improving the next sprint, this way the team is not overwhelmed. Progress on these items should be demonstrated in the next sprint review. This meeting is time-boxed to 1 hour.

Attendees:

- Product Owner
- Scrum Master
- Agile Team

Agenda:

- What worked well in the sprint?
- What didn't work well in the sprint?
- What do we need to change to do better?

KanBan

> One of the main benefits of kanban is to establish an upper limit to work in process inventory to avoid overcapacity. Other systems with similar effect exist, for example CONWIP. A systematic study of various configurations of kanban systems, of which CONWIP is an important special case, can be found in Tayur (1993), among other papers.
>
> A goal of the kanban system is to limit the buildup of excess inventory at any point in production. Limits on the number of items waiting at supply points are established and then reduced as inefficiencies are identified and removed. Whenever a limit is exceeded, this points to an inefficiency that should be addressed
>
> -- Wikipedia

The primary benefits achieved through the use of KanBan is high visibility, planning flexibility, focused work completion, and reduced team member frustration. Through this process the work being actively done in development will be clearly visible to the entire company removing any doubt on what we are focused on. Work not in the active stages of the KanBan board should have no expectation that it will be completed in the near term. What development is committed to is focusing on the work that has been deemed most important to our company and customers and driving it to completion through an open and transparent process.

Key Concepts

1. All work must be done on a KanBan board. No side projects.
2. Limits must be followed for work in progress (WIP) through out the KanBan states
3. Time spend in each state should be monitored
4. Work on each item must be completed before moving on to a new item

High Visibility

KanBan allows the use visual metrics to track work. Work is represented by cards on a board available to everyone. Also this process allows metrics to be gathered by tracking how long it takes for an item to traverse the board. The team's goal is to reduce the amount of time to move across the board as much as possible. This takes coordination between team members and alertness to items stalling.

Planning Flexibility

Because KanBan does not follow fixed development iteration cycles, the ability to adjust the priority of the backlog items is open on a continuous basis. This gives us the freedom to be extremelyflexible to critical issues or pressing customer needs. Additionally, if the work to fix a given issue is larger than first expected due to unknown complexity, the team is not overextended by having promised X amount of fixes in a time-bound iteration.

Focused Work Completion

> Multitasking kills efficiency. The more work items in flight at any given time, the more context switching, which hinders their path to completion. That's why a key tenet of kanban is to limit the amount of work in progress (WIP). Work-in-progress limits highlight bottlenecks and backups in the team's process due to lack of focus, people, or skill sets.
>
> -- Atlassian Agile Coach

With many team members involved in the complete development process, it is very easy to lose focus on an item after it leaves your personal role. This becomes a prime driver of stalled work. Our use of KanBan promotes complete ownership by the developer through the full cycle until the item is closed. This means that they can't move to the next item just because the one they finished is in Code Review. They need to follow up on that item with the team lead to shorten the time and be able to quickly address any issues found in that stage. The same holds for the QA stage. They can do some preliminary study of a new work item or spend the down time doing additional documentation on that work item while waiting for the other stages to progress. This ownership by the developer helps driver reduced time in each stage increasing our efficiency.

Work in Progress (WIP) limits should be established for each stage and monitored. If we exceed the limit in any given stage, there is

something creating a bottleneck and we must identify and address the cause. The limit should be established per team based on the number of team members. Ideally there is only one item in progress per developer but some flexibility is ok within reason.

KanBan Release Lifecycle

We will deploy a **patch** release to production as needed to support rapid maintenance of the products. These should contain bug fixes in support of our company and customers.

- **Deploy On-Demand Release to Production** - Releases containing **only bug fixes (patch)** will be deployed in two fashions throughout the increment. The client-server architecture products will release after each increment. This is typically 3-4 release cycles. Hosted server architectures will deploy fixes after each week of development. Before deployment, the QA team runs a set of smoke tests to verify that the new release is operating properly.

Standups

KanBan Standup meetings are much like Agile in that they should be held every morning by the entire team. The team lead will run the meeting and review each item on the board. Any blocking points should be brought up by the developer and discussed. Additionally, it is good to have a representative from other departments attend to increase awareness and inform the team of any newly prioritized work in the backlog. (Backlog grooming is done by the support team in our process) The standup should conclude by confirming the items being actively worked are the proper ones to drive the company forward at that moment.

KanBan Board Adjustments

With the highly flexible structure provided by KanBan, the team can support rapid adjustment of work load at any point. That said, these changes must be governed by a process to ensure consistency and clarity to the company as a whole. This prevents overloading the team and making promises that we can't keep. The rules for this are as follows:

- Work in the Code Review and QA states are not available for adjustment
- If a new item needs to be addressed immediately, an item from the active state must be removed
- The current item is moved back to the New state and can be put a the top of the prioritized backlog
- The new item can be assigned and work begun

Microsoft DevOps

> "Coming together is a beginning, staying together is progress, and working together is success."
> -- Henry Ford

DevOps is an online tool used for team collaboration, requirements, testing, and release management. To make all work visible, we use DevOps to track where each feature is in the development lifecycle and communicate progress between team members. This forces us to limit he conversation thread to a single topic and ensured that everyone can view all information as opposed to using email for communication.

Contents

Work Item Types

> "It's not a bug; it's an undocumented feature!"
> -- Anonymous

We use the following work item types in our collaboration tools:

- **Epic** - We create an epic or collection of features, to represent a broad functionality we want our product to deliver. An epic is incremental and is typically delivered over multiple increments. Each epic is accompanies by a spec that describes the requirements for each feature it includes. When all the features, backlog items, and tasks are complete, the epic is considered done.

- **Feature** - We break down an epic into a set of manageable features to gain complete visibility of the work required to implement it. A feature, or collection of backlog items, represents a deliverable that fulfills a stakeholder need. Features are the main functional characteristics of the product. They typically fit into on increment.
  - **NOTE**: Each sprint will have a feature titled "Unexpected Items" to account for new items that come up during a spring and need to be done in that sprint.
- **Product Backlog Item (PBI)**[1] - We create Product Backlog Items to represent requirements the team intends to create. These backlog items should fit into a single sprint and are typically estimated in days. If a PBI is too big to fit into one sprint, we will split the item into multiple, smaller items. This allows for a faster and more reliable implementation. PBIs include the following fields:
  - *Description*: Provide enough detail so others can understand how to complete the item. Include who the feature is for, what users want to accomplish, and why.
  - *Acceptance Criteria*: See Key Concepts.
  - *Effort*: Estimate the amount of work required to complete teh PBI in points (see Key Concepts).
  - *Related Work*: Link any related work by linking it to another PBI. Be sure to specify when a PBI has a dependency (predecessor or successor).

[1] In DevOps, Product Backlog Items are called User Stories.

- **Task** - Tasks act as the developer's "To-Do List." A Product Backlog Item can be further divided into the actions required to deliver that item. These tasks are typically estimated in hours. Each task should take less than one day to complete. Tasks include the following fields:

    - *Description*: Provide enough detail so others can understand how to complete the task.
    - *Original Estimate*: The original time estimate for this task. Enter how much time you think this task will take to complete.
    - *Remaining Work*: Estimate the amount of work required to complete the task in hours (these hours calculate team capacity). Update this field every day for tasks that have been updated.
    - *Completed*: When the task is complete, enter the amount of time you spent on this task. The more accurate, the better.
    - *Related Work*: Link any related work by linking it to another task. Be sure to specify when a task has a dependency (predecessor or successor).

- **Bug** - A bug is a defect in the software. All bugs are accompanied by a test case that describes how to reproduce the issue. In this way, a bug is a test case that fails. Bugs include the following fields:
    - *Repro Steps*: Clearly describe the steps required to reproduce the bug so others on the team can consistently reproduce it. Use the template below when filling out this field:
        - Steps to reproduce the bug.
        - Expected behavior -- What should happen once the bug is fixed?
        - Actual behavior -- What actually happened after the bug was fixed (update as information becomes available)?
    - *System Info*: Information about the software and system configuration that is relevant to the test.
    - *Acceptance Criteria*: See Key Concepts/
    - *Priority*: See Priority and Severity Rankings.
    - *Severity*: See Priority and Severity Rankings.

- **Test** - A test case is a set of steps required to validate that software (i.e., a PBI or a bug) meets the acceptance criteria. We close a PBI or a bug when all the associated test cases pass.
- **Test Execution**- These are the results from running a test. Since we may run tests multiple times, each test may have multiple test executions. In DevOps, each test shows a list of test executions for that test case.
- **Git/DevOps Naming Conventions for Branching** - Each feature should have a branch, as should each PBI and Bug. A Feature branch would be a branch of `master`, while the PBI or Bug branch would be a branch of its respective Feature branch. When work on a PBI or Bug is complete, the associated branch would be merged back into the feature branch (via a Pull Request), and when the Feature is complete, the branch would be merged back into `master`. Each of these pull requests would trigger a code review and a QA review. See the Git Workflow section on more about this.

Branches will have the following naming conventions:

`type/1234-short-description`

Where `type` is the Work Item type:

- Feature - `feature`
- PBI - `pbi`
- Bug - `bug`

`1234` is the Work Item Id, and `short-description` is a short description of the Work Item.

The most effective way to name your branches is by using lowercase " kebab case ". This means that while typing ever character within the name of the branch, you do not need to ever hit the `SHIFT` key. Do not use "snake case " or capitals, as both capital letters and the underscore character require pushing the `SHIFT` key.

Work Item Workflow

> A woodsman was once asked, "What would you do if you had just five minutes to chop down a tree?" He answered, "I would spend the first two and a half minutes sharpening my axe."
> -- C. R. Jaccard, "Objectives and Philosophy of Public Affairs Education" (1956)

Epics and **Features** move through the following states in the **production backlog** during the increment(s):

Board Column	Work Item State	Assignee[1]	Description
Planning	New	None	Approved for Development
Development	Active	Developer	Implementation started
QA	Resolved	QA	QA is evaluating feature
Documentation	Documentation	PM	Document for the full release process
Closed	Closed	Developer	All work is complete, item is accepted and ready to be released
N/A	Removed	None	Epic/feature is no longer being developed and is removed from backlog

[1] Assignee is responsible for updating his/her own progress in DevOps.

PBIs and **Bugs** move through the following states in the **backlog items board** during the sprint. The backlog items board represents the workflow of the sprints.

Board Column	Work Item State	Assignee[2]	Description
New	New	None	Approved for development and assigned to developer but has not yet started
Development	Active	Developer	Developer is working on item
Code Review	Active	Developer	Developer has completed work, awaiting code review/pull review
QA	Resolved	QA	QA is evaluating acceptance criteria of item
Documentation	Resolved	QA	The PR has been completed into Master, Release notes being written
Done	Closed	Developer	Item is ready to be integrated into feature
N/A	Removed	None	Item is not longer being developed and is removed from backlog

[2] Assignee is responsible for updating his/her own progress in DevOps.

Tasks move through the following states in the **sprint backlog** during the sprint. The backlog items board represents the workflow of tasks within the sprint.

Board Column	Work Item State	Assignee[3]	Description
New	New	Developer	Approved for development and assigned to developer
Active	Active	Developer	Developer started the work
Closed	Closed	Developer	All work is complete

[2] Assignee is responsible for updating his/her own progress in DevOps.

We plan sprints with the expectation that the team will move all items through the workflow to Done within three weeks. If any items are not Done at the end of the three-week sprint, we create a new issue that represents the remaining work for the item and modify the scope of the original item to reflect the work that was completed. This is a tedious exercise; therefore, we plan carefully to avoid this situation.

Priority and Severity Rankings

> "Leadership is the art of getting someone else to do something you want done because he wants to do it."
> -- Dwight D. Eisenhower

We prioritize each item in the backlog using The MoSCoW method (also called the PICK Chart, and remarkably similar to the Eisenhower Matrix)

1. **M**ust Have (**I**mplement): Things that must be delivered or the solution will provide no value.
2. **S**hould Have (**P**ossible), if possible: Things that are very important, but could be omitted if absolutely necessary.
3. **C**ould Have (**C**hallenge), if possible: Things that are less important, but would be "nice to have."
4. **W**on't Have (**K**ill), maybe in the future: Things that might be important in a future Program Increment, but not in this one.

We use the severity field in DevOps to reflect the impact of each bug on the project, as follows:

1. Critical
    - Potential/Cause of taking the site down
    - Feature broken/no workaround for all or major subset of clients
    - Applications/lead capture broken for high-traffic workflows.
    - Immediate security concern
2. High
    - Feature broken for all or major subset of clients with workaround
    - Applications/lead capture broken for moderate traffic workflows.
    - High security concern
    - Usability issue affecting a large number of clients
3. Medium

- Feature broken for small percentage/single client
- Applications/lead capture broken for low traffic workflows
- Moderate security concern
- Usability issue affecting a moderate number of clients
- Minor bug reported by client

4. Low
    - Feature broken has easy workaround or is cosmetic
    - Application/lead capture flows minor issue
    - Minor security improvements/update
    - Usability issue affecting a small number of clients
    - Styling/UI/Cosmetic issue

Issues and Addressing Them

Issues always arise, and we want to have a consistent strategy on how to manage, track, and resolve them. We plan to use the DevOps item type of "Issue" to help with this. To do so effectively we need to identify a consistent way of using these.

Three Uses of Issues in DevOps

- **Under Defined Requirements** - This is where the Feature, User Story, or Bug does not contain enough information to begin working on this item. An Issue should be created and linked as a child to the item in question. A Tag of "Under-defined" should be added to the Issue.
- **Blocking Issues** - This is where work on a User Story or Bug is blocked pending resolution of the issue. This may be due to waiting on outside information or other developers code changes. An Issue should be created and linked as a child to the item in question. A Tag of "Blocking" should be added to the Issue.
- **QA Returns** - This is where a User Story or Bug that was sent to QA as complete was found to have either introduced a regression or did not fully meet the initial item intent. The use of Issues for this will provide a way to track and manage these through to resolution. An Issue should be created and linked as a child to the item in question. A Tag of "Inadequate" should be added to the Issue.

In all cases, the description of the Issue should clearly provide the reason for the Issue being created. This provides the context needed to find a solution.The issue should be assigned to the owner of the original item so they get notified of its creation. Clarification of the issue should be done in the Discussion section so there is a way to track comments and suggestions.

Managing Issues

Because solving issues is crucial to achieving our Sprint goals, we need to be constantly aware of them and address them quickly. The solution may be to create a new bug for later or even reshuffle the sprint. The following are methods that will make sure we monitor and address issues as they arise.

- **Issues Query** - A query should be created to show all open issues. This should be reviewed at Stand-up every day and discussed.
- **Dashboard** - A chart from the Issue Query should be added to the dashboard for quick reference
- **Under Defined Query** - This is used to track how well we are building our requirements. It will be monitored and tracked by the Product Managers. The resolution for this should be to understand what caused the confusion and note it in the discussion section. We could then perhaps identify a trend and solve it through the use of a template or some other means.

At the end of a sprint, there should be no open issues. This could be part of the retrospective but each sprint should start clean.

Enhancement Requests

> "Enhance. Stop."
> Rick Deckard (perf. Harrison Ford). Blade Runner (1982; dir. Ridley Scott)

1. **Initial Request**:

   a. Customer submits requests via web form (see Enhancement Request Form)

   b. If the customer submits the request via phone, email, or support ticket, the tam member receiving the request will direct the customer to fill out the web form.
2. **Review Enhancement**: Product Owner follows up with customer that we received their request and are reviewing it.
3. **Decision**: Make decision with team if we should move forward with enhancement request.

   a. **No** - **Archive Request**: Document reason for not moving forward and archive.

   b. **Yes** - **Scope Work**: Product Owner details requirements needed to carry out work (see Key Concepts).
4. **Plan Work**: Product Owner moves request into DevOps backlog for planning and assign to Increment and Sprint; Communicate development timeline to customer. (Also inform customer if we are not moving forward with request).
5. **Develop and Deploy Work**: Work has been developed, tested, and deployed into production.
6. **Enhancement Complete**: Inform customer of release date for their enhancement.

NOTE: Quotes will not be provided to customers until we have a clear understanding of the request. Similarly, requests need to be fully understood by the development team and approved by the CTO before they are added as contract addenda.

Aha!

Aha! is a DevOps plugin we use for high-level planning and strategy management. New features will initially be captured using the ideals portal in Aha!. At this time, we clearly define the requirements to help us think about how features fit with each other and incorporate each into the overall vision and roadmap. Features ready to be developed will then be moved into DevOps where they will be split into backlog items, sized, prioritized, and assigned to an upcoming sprint.