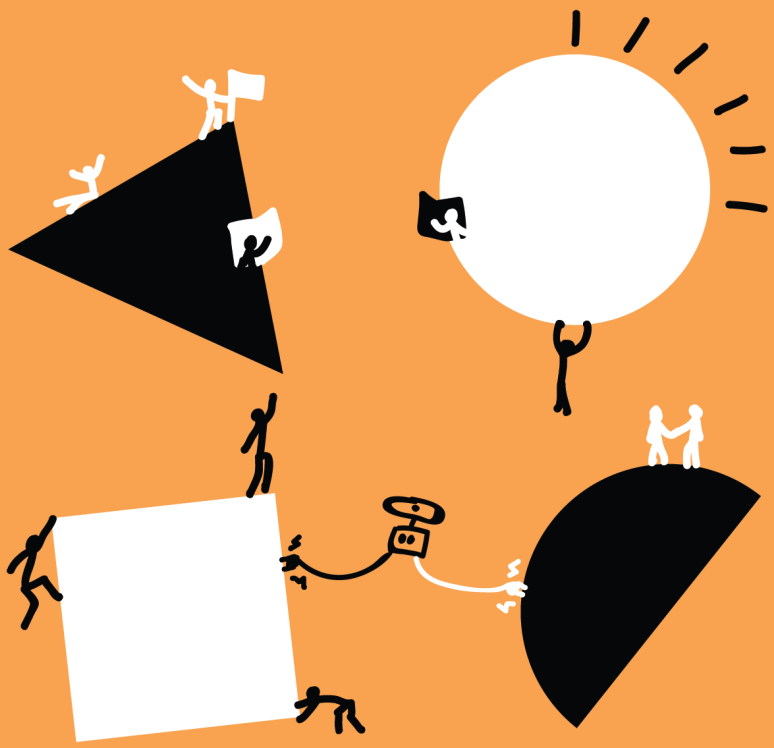
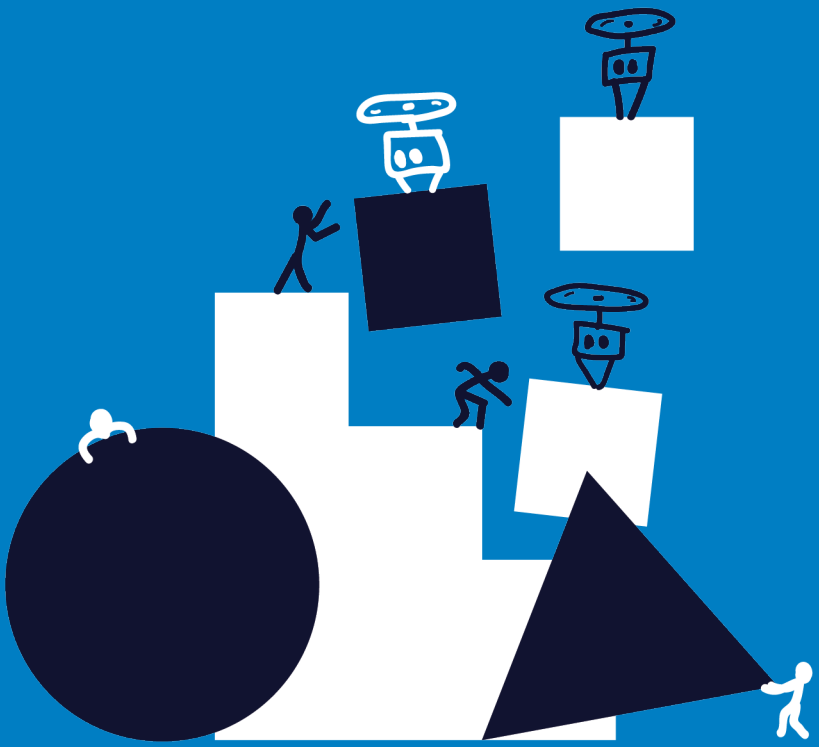


The Way of Working.

A practical guide on how to deliver products

CODEBOTS



Bots that code.

The Continuous Modernisation Playbook

Learn more about our Codebots platform in the book by
Eban Escott & Indi Tansey.

Head to **www.codebots.com** for more information.

CODEBOTS

Version

28.June

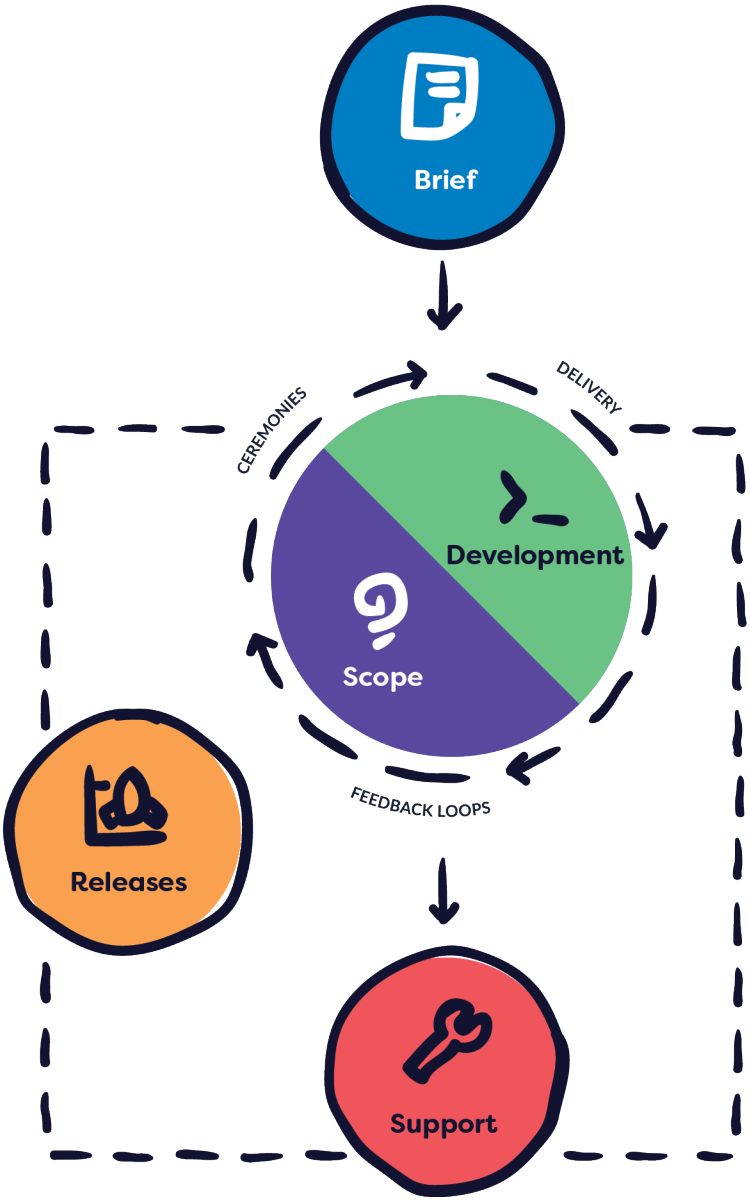
2019

Contents.

Way of Working.

This Way of Working is a compilation of the process, practices, tools and principles which we recommend a team follows throughout the life of a product. The contents of this book come from many years of experience of building software, the result of both successes and failures, and our learnings from both. This book is our recommendation on how a software project team can plan, build and release a product, ensuring customer and team satisfaction along the way.

Within the context of this Way of Working, when we say “customer” we refer to the person or persons that engage a project team for the intention of developing a piece of software. This entity could be internal or external but they have a vested interest in the success of the project and a say in its direction.


Brief
CEREMONIES
DELIVERY
Development
Scope
FEEDBACK LOOPS
Releases
Support

Broadly speaking, a project is broken into five phases: Brief, Scope, Development, Releases, and Support. Normally, the project will go through the first four stages with the same project team, before being handed over to support engineers.

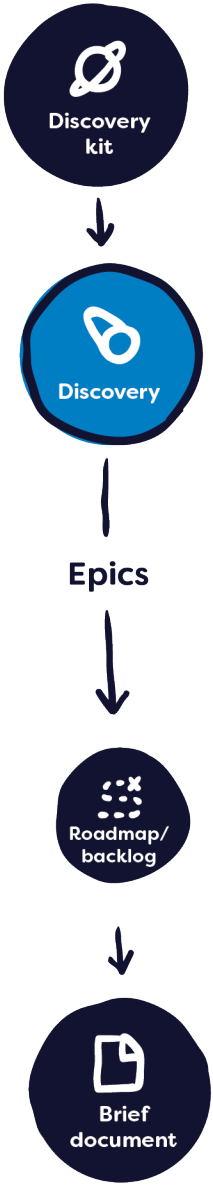
Depending on the size of a project and the velocity required, we have found that a small team consisting of a Web Engineer and a Testing Engineer pairing (Senior & Junior) along with a UX Designer work efficiently together as a project team. This project team is overseen by a Team Lead (Scrum Master) who can manage multiple projects and teams depending on the size of the projects. If greater velocity is required for a project, then additional developer pairings can be engaged for increased productivity.

Our company mission is to discover new boundaries. Through experimentation and the scientific method, we are constantly evolving this Way of Working in order to increase the probability of creating successful projects.

Brief.

The goal of the brief stage is to understand the problem (or opportunity) and set the criteria for its success.

In the brief stage, the problem is unpacked and an initial roadmap is presented. The roadmap indicates a broad sense of how the project vision can be reached.


Discovery kit
Discovery
Epics
Roadmap/ backlog
Brief document

Brief details.

Prerequisites

The account manager:

- Talks to customers about project fit.
- Arranges to have a brief meeting with the customers.

Commitment

- The account manager is the main facilitator of this stage, though they may call in other team members to participate.
- The length of time required varies, though this process normally happens over the course of a single week.

Outcomes

The account manager is responsible for:

- Creating and printing the brief document.
- Presenting the brief document to the partner.
- Proposing the initial pathway.

Meetings

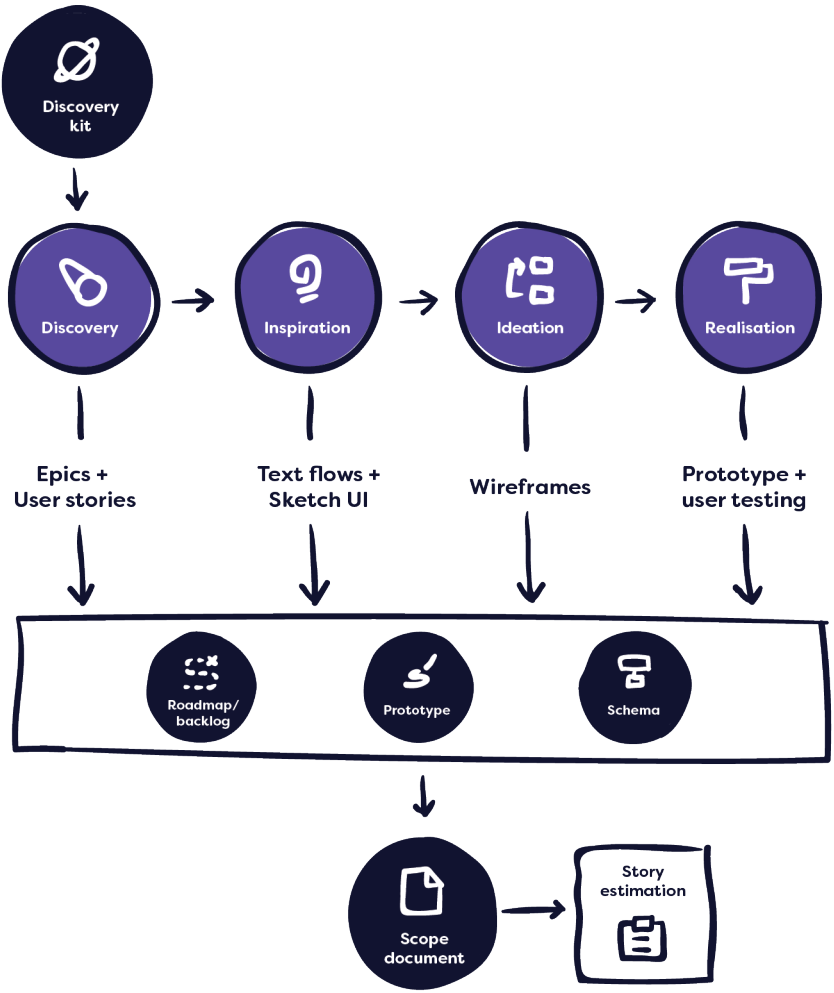
- Initial phone call (approx. 15 minutes): The account manager establishes what the problem is and arranges a time for the discovery workshop. Discussion around what will be done during the workshop.
- Discovery workshop (approx. 2 hours): Used to discover enough about the problem to get a broad overview of what may be required for the solution. The team should gather enough information to be able to propose an initial pathway.
- Presentation meeting – to present our realisation of the project in the form of a printed brief document.

Tools

Our Discovery Kit

Scope.

In the scope stage, the problem or opportunity is explored further and a solution is proposed. We have found it is best to move in sequence through the discovery, inspiration, ideation, and realisation stages. The project plan follows a divergent-convergent approach to problem solving. The plan diverges at the beginning during the discovery phase to ensure as many solutions are considered as possible, before then converging at the realisation stage with a solution that feels provides the most value.


Discovery kit
Discovery
Inspiration
Ideation
Realisation
Epics + User stories
Text flows + Sketch UI
Wireframes
Prototype + user testing
Roadmap/ backlog
Prototype
Schema
Scope document
Story estimation

Scope details.

Prerequisites

- There is a signed an agreement about the agreed minimum and maximum length of time for this stage.
- The project has completed the brief stage.
- The account manager presented all available knowledge and artefacts to the scoping team in a pre-scope meeting.
- The scoping team members have read the brief document.

Commitment

- The recommended project team should consist of a least one UX designer, testing engineer, and web engineer who are led by a team lead, accompanied by an account manager.
- Typically, this stage can last anywhere between 2 and 5 weeks, though there is a preference for shorter and smaller scopes to encourage iterative development.

Outcomes

The team delivers:

- A scope document
- A roadmap
- A user stories backlog
- A schema
- Scope estimations
- A protoype of the solution
- Any other artefacts produced from discovery kit activities

The account manager delivers:

- A scope presentation for the partner

Meetings

One or two scope meetings per week, normally going for approximately 2 hours.

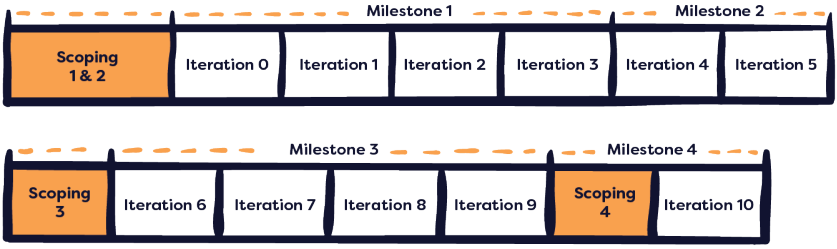
Tools

- Our Discovery Kit
- Our estimation tool

Scope frequency.

The initial scope for a project generally prepares the project team for the first one to two milestones of development, rather than an entire roadmap. This leaves the project open to adaptation and flexibility, rather than being locked onto one path. Once these milestones have been built, most projects will pause development, and re-enter scoping phases to discover the next milestones of work. The scoping timeline diagram shows different project timeline examples.

Through past experience, we have found that it is important that small batches of work are scoped during this time, so that a project team can quickly adapt in response to user feedback or changes in business processes. This will prevent redundant features from being scoped or code that requires refactoring.

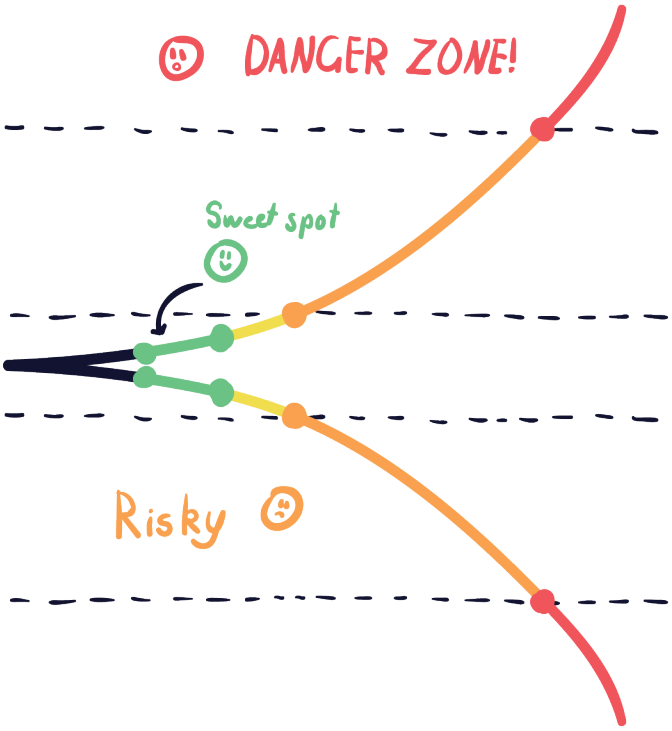

Milestone 1
Milestone 2
Scoping 1 & 2
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Milestone 3
Milestone 4
Scoping 3
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Scoping 4
Iteration 10

Cone of uncertainty.

The Cone of Uncertainty is a depiction of the level of uncertainty around the length of time required to complete a project based on the amount of knowledge at the beginning of a project, and the risk of the project time blowing out due to this uncertainty.

Before the beginning of a project, there is a large amount uncertainty. As no work has been completed yet, the project team has only past experience to base their estimations on. In software development, no two projects are the same, meaning that even with years of experience it is impossible to provide an accurate estimation. When combined with the fact that product requirements are likely to change over time, it makes it difficult to provide an accurate estimation to the product manager in the beginning.

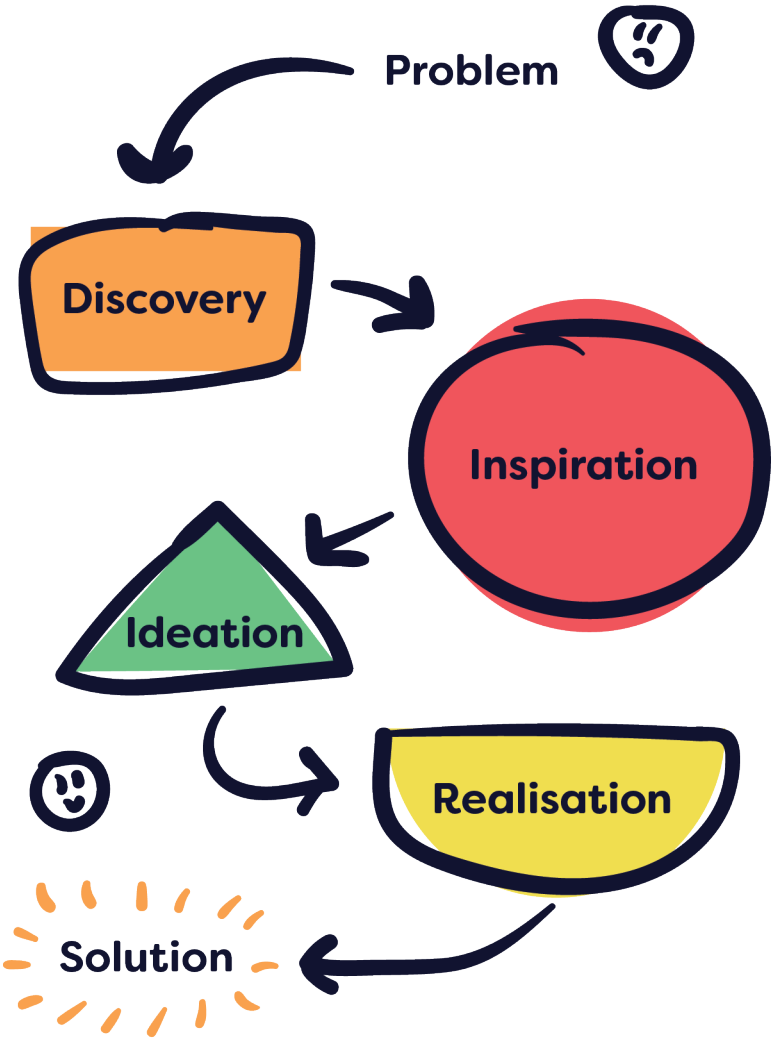
For this reason, we recommend that the team focuses only on a small amount of work at any time. This gives the team a chance to become more familiar with the product and its requirements, and once it comes time to estimate again, they can be that much more confident about how long the future work will take.


DANGER ZONE!
Sweet spot
Risky

Discovery kit.

The due diligence of any product expert is to deliver a product that has value. The tool to use to help reach this goal is our Discovery Kit. The Discovery Kit contains a range of activities that can be conducted throughout the scoping process to help clearly identify the problem an application needs to solve and how to best achieve this for the applications users.

While some customers may arrive with a reasonable level of clarity around the problem they are trying to solve, these tools help unpack the problem in a way that everyone in the project team can understand. This then enables you to use your expertise to propose the best solution.


Problem
Discovery
Inspiration
Ideation
Realisation
Solution

Capturing requirements.

Since customers can express the requirements of their software in many different ways, there are many methods for capturing requirements. Whether it happens verbally during a meeting or via diagrams on a whiteboard, the aim of these strategies remains the same: to achieve a shared understanding of what needs to be built.

Capturing these requirements accurately is important for decreasing the risk of flawed iteration estimates (though the risk cannot be eliminated completely). A strategy for capturing these requirements is to understand the user intent through epics and user stories, which are then captured and maintained in a requirements backlog.

Alongside the stories and epics, capturing acceptance criteria and assumptions against each one to ensures that the whole team has a thorough understanding of what is required. This reduces the likelihood of miscommunication or misunderstandings affecting the estimations. Risks are also captured so they can be considered when estimating.

Epics

An epic is a high-level theme or feature that is used to categorise related stories. An epic contains the following information:

- Summary
- User personas involved
- Acceptance criteria for each persona

Stories

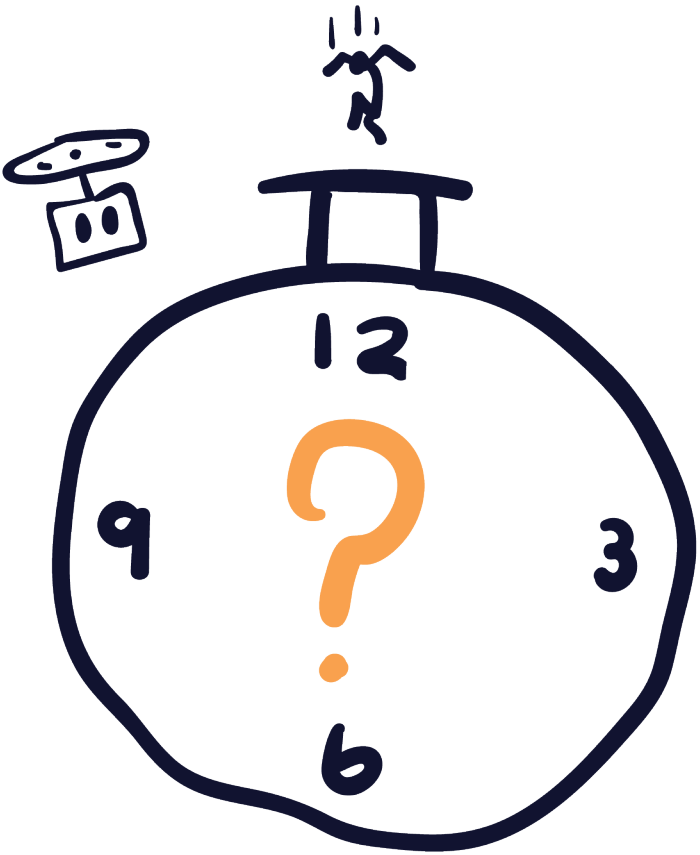
A story is a fine-grained requirement, written from a user's perspective. Each story belongs to only one epic. A story is the classification of tasks which are delivered during development, so they must be small enough to be completed within one iteration (i.e. you must be able to build it in less than 2 weeks). A story contains the following information:

- User story: *As a* **[user persona]** *I want to* **[do something]**, *so that* **[reason for task]**.
- Background of story
- Acceptance criteria
- Implementation notes

Scope estimations.

Estimations are just that - estimations. For any person that has dealt with software, they know anything unexpected could happen to throw off a project's timeline. The only time you can ever predict an exact delivery time is after it has been delivered.

The estimations which are provided to customers offer a plan for the project, taking into account the difference between an ideal time estimate and a realistic estimate. Using a scientific approach, we have formulated a recommended estimation method which takes into account past project experiences, alongside the project's complexity and risk.


12
9
3
6

Time estimations.

Once scoping is complete, the team estimates the time units they believe are required for a typical team member to complete each story. Our time estimation values follow a precise, Fibonacci-like sequence. This is a standard practice in agile estimation processes (like estimation poker) to ensure that developers don't get too caught up in a getting a specific time, and instead focus on the general size of a task.

1 hour
2 hours
4 hours
1 day (8h)
2 days (16h)
4 days (32h)
1.5 weeks (64h)
3.5 weeks (128h)


128
64
8
4
1
2
16
32

Risk multiplier.

The more complex and unfamiliar a story is, the higher the risk that the development time will be underestimated. To ensure that the risk is appropriately considered in estimations, two multipliers are included: unfamiliarity and complexity. Both multipliers are quantified with a number rating between 1 and 5.

Unfamiliarity

The unfamiliarity multiplier takes into consideration the unknowns of the project. When estimating each task, team members should consider the following: experience with similar tasks, available resources, and whether it uses new or existing technologies. For example:

- If the developers have worked with the technology before and are quite confident, then the risk score would be low (probably one).
- If the developers haven't worked with the technology, but someone else in the company has, then the risk score would be moderate
- If no-one in the company has ever worked with the technology before, then the rating would be quite high (probably five).

Complexity

The complexity multiplier quantifies how intricate and challenging the task may be. When estimating, the team should consider how many other parts of the product are affected by this task, how complicated the development will be, and how likely this requirement is to change (i.e. how confident the customer is about this task, and whether new information may come to light and change it). For example:

- If the story uses a feature which has been developed in the past and it won't require any changes, then the complexity would low (probably one).
- If the story can use an old feature, but it would need modifications to meet the requirements, then the complexity score would be moderate.
- If the story requires completely custom development and extensive testing, then complexity score would be quite high (probably five).

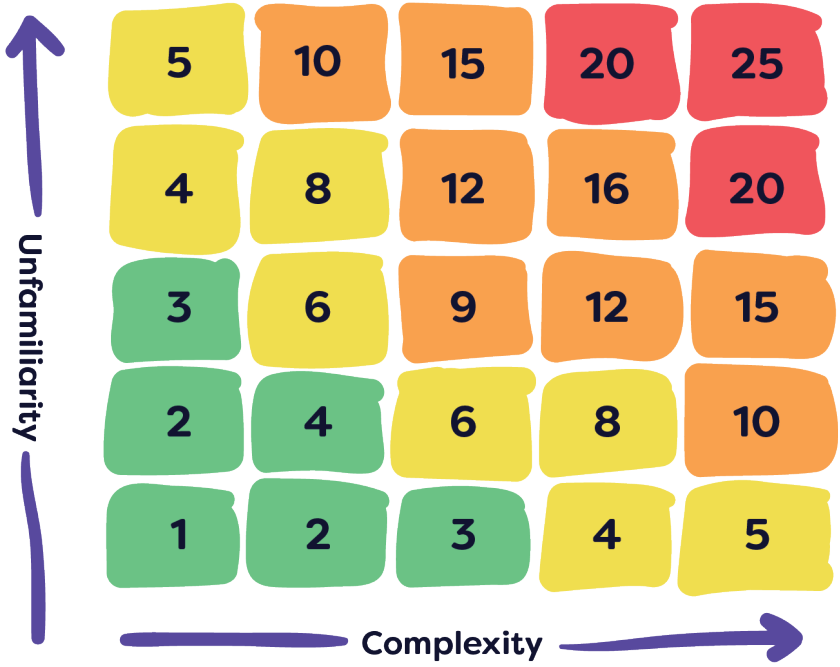
Once the two factors have been considered, they are multiplied together to get a risk score out of 25. Depending upon this score, it may be either low, medium or high risk. As a rule, a story cannot enter into a development iteration unless it has a score of eight or lower.

The risk score is then passed through a formula to create a multiplier which is applied to the time estimate.

[Time with risk = estimated time * (1 + ((unfamiliarity * complexity) /25))]

Once there is an estimation that includes risk, there are three other factors that need to be considered before the estimation process is complete.

**These modifier amounts are a recommendation based on our experience*

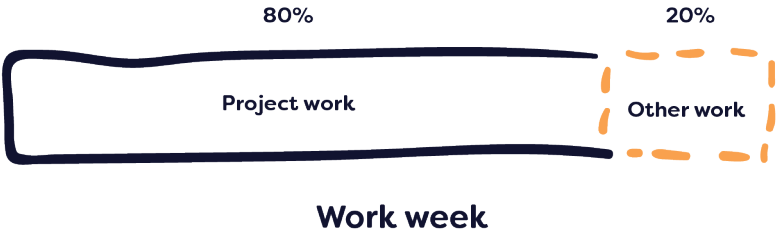

5
10
15
20
25
4
8
12
16
20
3
6
9
12
15
2
4
6
8
10
1
2
3
4
5
Unfamiliarity
Complexity

Allocation factor.

It is a fact of any workplace that employees are not able to dedicate 100% of their time to their primary role at work. Activities such as meetings, assisting colleagues and alternate side projects take up their time and should be recognised when estimations are being considered. This is done by applying what we call an allocation factor, which assumes that any project team is able to work on active development for 80%* of their time.

Once there is an estimation including risk, it is then multiplied by 1.25* to calculate an estimation which also includes allocation factor.

**These modifier amounts are a recommendation based on our experience*


80%
20%
Project work
Other work
Work week

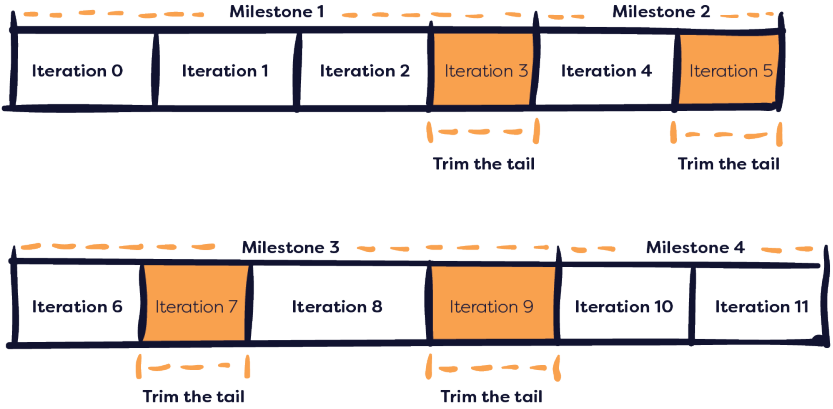
Trim the tail allowance. (Iteration N)

The trim the tail allowance is used to add a chunk of time into the project to account for any bugs or minor improvements which may pop up. Experience has shown us that every project requires periods of cleaning up time where the team needs time to pause on active development, and instead focus on the work they have already done. Whether it is fixing minor bugs which have been put off, addressing technical debt, or implementing minor improvements which have come from user feedback, it's important that they get some time to clean up.

The trim the tail factor calculates a period of time which can be used throughout the project's timeline. This creates a bank of time which can be drawn from at any point in the project's timeline. When the team lead and the customer agree it is required, the standard development progress can be paused, and the team can enter into an iteration N. The length of this iteration can vary depending upon the amount of work which is required. This period of time is then taken from the bank and the running tally is updated.

Currently, the trim the tail is calculated as a 1.25* multiplier on the time estimate (including risk and allocation factor).

**These modifier amounts are a recommendation based on our experience*


Milestone 1
Milestone 2
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Trim the tail
Trim the tail
Milestone 3
Milestone 4
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
Iteration 11
Trim the tail
Trim the tail

Tech spike allowance.

Similar to the trim the tail allowance, the tech spike allowance creates a bank of time which can be drawn upon when the project team needs it. The tech spike allocation is used as a chance for the team to take time to research or test concepts before they estimate properly on something. Typically, it is a project team member that indicates the need for it, though all tasks with a risk rating higher than eight must be de-risked before estimating, which is almost always reduced through the completion of a tech spike.

A tech spike can be included in the development process in two ways: the issue can be moved into a later iteration, and a tech spike can be put into the next iteration in its place (this option doesn't interrupt the development flow). If the issue being estimated has a high priority and cannot be moved into another iteration, then the other option is that a tech spike for the issue can slot in before the next iteration as a chance to research before estimating.

Currently, the tech spike allocation is calculated using a 10%* modifier on an estimation including risk, allocation and trim the tail factors.

**These modifier amounts are a recommendation based on our experience*

Case 2

Swap ticket out for tech spike and then complete ticket next iteration.

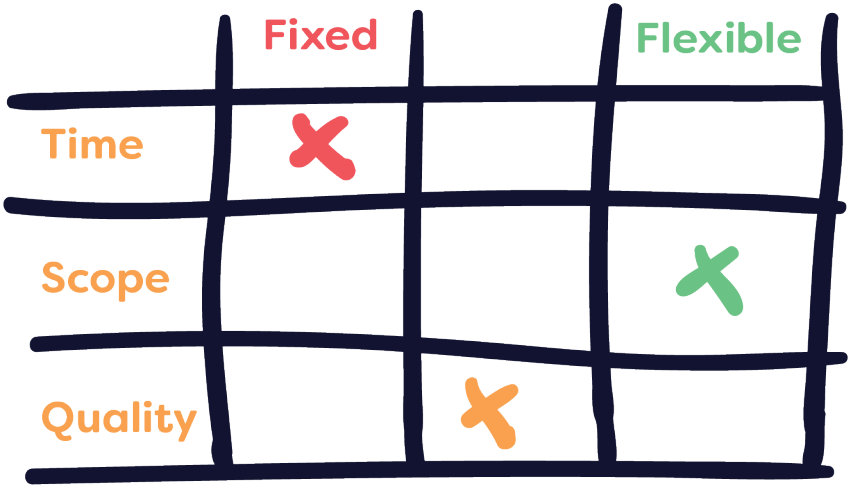
Case 1

Tech spike issue before starting iteration.

Development approach.

Trade-off sliders are used as a way of handling and communicating expectations between the stakeholders within a project. They display how the choice made around flexible and fixed options can affect the other options available. This can help inform the decision around what development approach will be used for the project. By selecting one of the values for three options, it shows what is available for the other two and lets the stakeholders decide where their priorities lie.

Note: No two crosses can be in the same column.



Flexible time
+
Fixed scope

time = estimation ± cone of uncertainty

In the case of a fixed scope (where every requirement is a must-have), all stories are delivered. However, since we are trying to estimate things which are far off into the future, it is harder to get an accurate estimation.

Fixed time
+
Flexible scope

Time is fixed

In the case of fixed time (and therefore fixed budget), the team strives to deliver the tasks which hold the most value, whether they were in original requirements or not. This process is the most open to being adaptable.

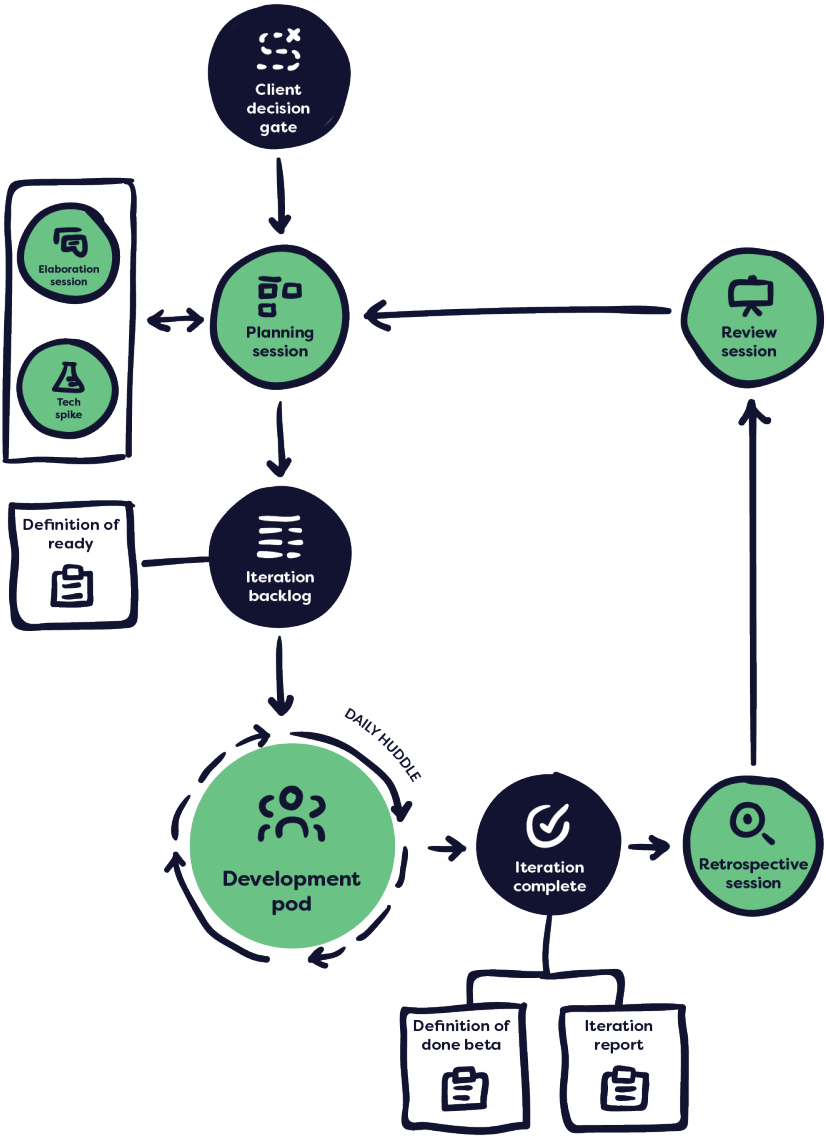
Fixed time
+
Fixed scope

Scoping time is unknown

This is an older developmental approach. The scope continues until all requirements are listed and estimated. Due to the large range of possible risks, this estimation becomes signifi-cantly more difficult and therefore results in a much longer timeline than the others.

Development.

The development stage is the part of the cycle where the product is actually built and tested. The project team delivers early and often, striving to maximise learning along the way. This stage is comprised of iterations which repeat until development is finished. Every iteration consists of a series of meetings and checkpoints to ensure both customer and team satisfaction, providing everyone with a chance to make suggestions and learnings.


Client decision gate
Elaboration session
Tech spike
Planning session
Review session
Definition of ready
Iteration backlog
DAILY HUDDLE
Development pod
Iteration complete
Retrospective session
Definition of done beta
Iteration report

Development details.

Prerequisites

Before going into the development stage, a project must have:

- Completed the scoping stage.
- A backlog of stories in the first milestone.
- A roadmap for the project.
- The details required for hosting the product:
  - DNS information.
  - Email hosting details.
  - If the product is an app, relevant app store account details.

Commitment

- A project team consisting of at least one UX designer, web engineer and testing engineer, led by a team lead.
- An account manager to assist with managing the partnership and invoicing.

Outcomes

- A series of meeting notes and iteration reports, produced for each iteration.
- A completed Transition to Support document for the support team.
- A product which has passed all acceptance criteria and has been released to production.

Meetings

User acceptance testing session:

- Conducted after each iteration has been completed.
- A chance for the customer to review the work which was done and make sure it meets the requirements
- Typically done between the testing engineer and the customer

Almost all the meetings in this stage come from the iteration cycle, outlined in the next section. (Please see the following table)

Tools

User Testing Interviews

- Conducted after an iteration or milestone has been completed.
- Used to validate that what was built provides value to the user, and users can easily understand how to use the product.

Iteration details.

Prerequisites

Before starting an iteration, the following things must be done:

- All issues in the iteration plan must meet the Definition of Ready **(see page 50)**.
- A planning session has been completed and the meeting notes sent to the customer
- An elaboration must be completed internally to ensure all team members are confident of the tasks.
- An estimation has been calculated and communicated to the customer.
- A final iteration plan has been proposed and accepted by the customer.

Commitment

- At least one development team which is available for the whole iteration, led by a team lead.
- An account manager to assist with managing the partnership.

Outcomes

- An iteration report, detailing how the iteration progressed and what was delivered.
- Planning/Review meeting notes
- Retrospective actions
- The product is released to beta for UATs

Meetings

The following meetings and ceremonies appear in the development diagram **(see page 43)**.

Planning Session

- Before each iteration, the team meets with the customer to define the iteration goal and choose priorities for the issues.
- The team also works with the customer to define acceptance criteria for each task.
- Normally, this meeting is also merged with the review meeting for the previous iteration so that there aren't two meetings in one day.

Elaboration Session

- This is an internal meeting within the project team. Typically, the account managers do not attend as the talk often gets quite technical, and they many influence the estimations.
- The purpose of this meeting is for the team to get more technical about each issue and ensure that all members understand both the requirements and the solution which will be implemented.
- At the end of the meeting, the team produces estimations against each issue.
- All issues included in the plan must satisfy the Definition of Ready **(see page 50).**
- Depending on the risk ratings of the issues, the team may need to use a tech spike in order to reduce the risk **(see tech spikes on page 39).**

- Once the meeting has finished, the squad lead and account manager contact the customer to tell them the estimations and projected timeframe.
- At this point, the customer will either be:
    - Happy to proceed, and the iteration will start with the estimated end date, or
    - Further decisions will need to be made around the iteration plan (i.e. larger tickets may be removed to shorten the projected timeline.) before it can be kicked out.

Daily Huddle

- Every day the team members have what we call a daily huddle, where they discuss:
    - What they completed yesterday.
    - What they plan to complete today.
    - If they have any blockers inhibiting their work, and
    - if there are any risks that need to be called out.

Retrospective Sessions

- At the end of each iteration, the development team conducts a retrospective session. The retrospective is used as a chance to have a reflection on how the iteration went.
- The team identifies actions for improvement going forward. These actions could be:
    - Experiments or improvements which could be made part of company processes.
    - New features which should be called out

- New tech spikes or tasks to include in the backlog

Review Session

- Normally, this is conducted at the same time as a planning session.
- The team presents the completed work from the iteration to the customer (UATs get done in a separate meeting).
- The team talks about what went well and what impeded development progress.
- This is the chance to review how the project is progressing and whether we are on track to deliver the milestone on time.

Tools

Definition of Ready

- The Definition of Ready ensures that we are ready to commit to the iteration.
- Each issue must meet all criteria before it can be started on.
- **See page 50** for more information about the definition of ready.

Definition of Done

- The Definition of Done ensures that the issues in the iteration are ready to be released to beta.
- **See page 54** for more information about the definition of done.

Tech Spike

- The development team may need to make use of the tech spike allowance to de-risk issues.
- **See page 39** for details on tech spikes

Definition of ready.

The Definition of Ready helps ensure a project team is as prepared as possible to commence an iteration. It should be ensured that each user story adheres to the following checklist:

- [ ] **Is an appropriate size, aiming for less than half the iteration length.**
- [ ] **Has a risk rating of eight or lower.**
- [ ] **Is written as a proper user story**
- [ ] **The knowledge base has been reviewed for existing and previous solutions, ideas, or insights which may assist in the development of this issue.**
- [ ] **If required, has a preliminary UI/UX plan agreed to by the customer. This could be a UI design, prototype, sketch, or anything which can guide the project team on what it will look like.**

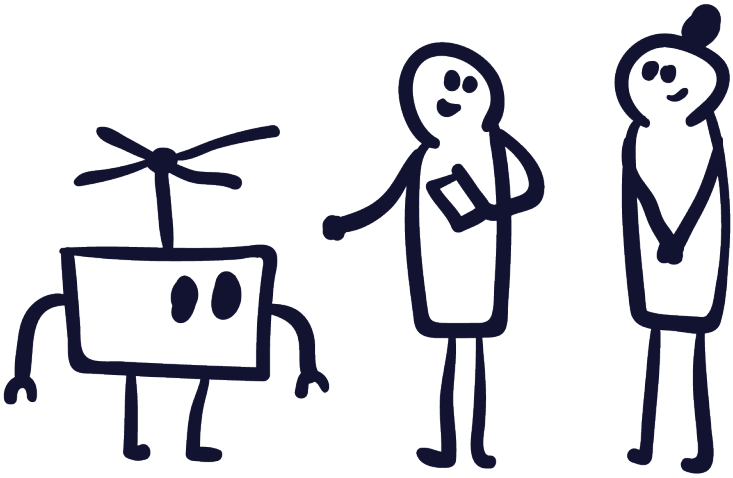
Acceptance criteria.

Acceptance criteria are written against every individual ticket in a project. They may be initially defined during the scoping stage, but all tickets must have them before they go into active development. The acceptance criteria live up to their name, as they are used as the criteria which must be met in order for a ticket to be considered complete.

At the start of each iteration, during the planning session, the team lead assists the customer to write a list of criteria against each ticket. Depending on the size and complexity of each ticket, the list may be small or large. What is important is that the customer is directly involved with the writing of this criteria, as this is what determines whether a ticket is done or not. If they wish to change the ticket once the criteria are agreed upon, then it must be submitted as a change request (rather than the ticket being considered incomplete).

Occasionally, a need will arise where the criteria may need to get changed. For example, the project team may discover that a certain point may be incredibly complex and will blow out the estimated time. In these cases, the customer should be contacted and informed of this issue, and it is up to them around how they wish to proceed. They could either chose to remove the point altogether, alter it slightly so it is less complex, or leave it in and accept the longer ticket length. If either of the first two options is chosen, then it may be entered as a ticket of its own in the project backlog so it can be addressed later.

During an iteration, the team should consult with this list before and after they have finished developing and use it as a list of the requirements that they should build to. Anyone who tests the ticket will also make use of the list. A ticket cannot be considered done until all items have been checked off.



This is exactly what I needed!

Definition of done.

The Definition of Done checklist ensures that everything that is developed is delivered to a high standard and meets the pre-defined criteria. For all stories to be considered done while in an iteration, they must comply with the following criteria. The checklist is as follows:

This story has automated tests which are passing.

The implementation has been code reviewed.

All acceptance criteria have been satisfied.

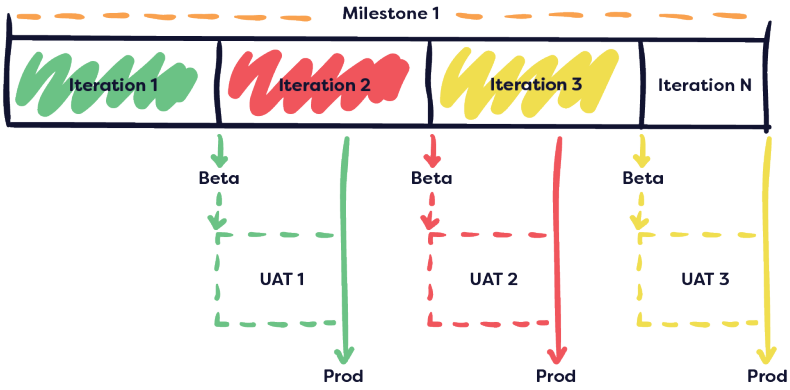
The code has been merged into develop in preparation for a beta release.

Conducting UATs.

Once an iteration has been completed, it requires User Acceptance Tests (UATs) to be conducted by the customer against all the issues which were completed (and deployed onto beta) during the iteration. User Acceptance Testing involves the product manager reviewing each ticket which was completed and confirming that it meets the acceptance criteria which they defined at the beginning of the iteration.

This can be done in two ways: the first involves the product manager joining a member of the project team in a walkthrough session in person. This allows a member of the team to see any issues which occur as they occur, which can be very valuable when testing. This is the preferred method, as it ensures that the tests are completed quickly and completely, and the team can be sure that any issues which occur are truly issues, and not just user error.

The alternative is for the customer to do this process in their own time and submit their findings remotely. This has advantages in that if they struggle to make into the office or aren't available during work hours, it can still be completed in a timely manner.


Milestone 1
Iteration 1
Iteration 2
Iteration 3
Iteration N
Beta
Beta
Beta
UAT 1
UAT 2
UAT 3
Prod
Prod
Prod

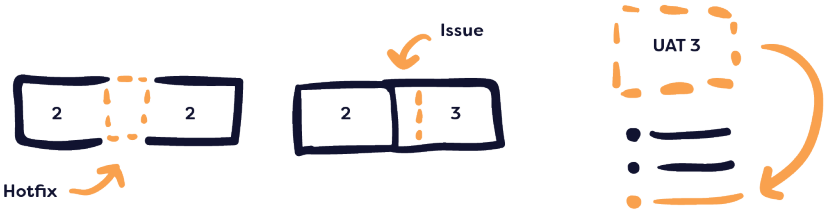
UAT kickback.

The results of a UAT session are recorded against each issue, and if all pass, then a release to production can be prepared (see Releases section). However, it is not uncommon for a customer to provide feedback which needs to be addressed. Depending on the urgency of the change, it can be handled in three ways:

- **Urgent Resolution** - This is when the customer requires an urgent fix which must be resolved in the current beta version (i.e. before it reaches the production environment). In this situation the project team will pause the current iteration and apply the fix. It should be noted that this will create a delay for the end of the next iteration.

- **Next Iteration Resolution** -This is when the customer is willing to wait for the fix to be in the next version of the beta environment. In this situation the defect or minor change request will be added to the next iteration's backlog.

- **Backlog Resolution** - This is when an issue has been deemed to be a low priority. In this situation the UAT feedback will be added to the product backlog and be brought up during a future planning session for.



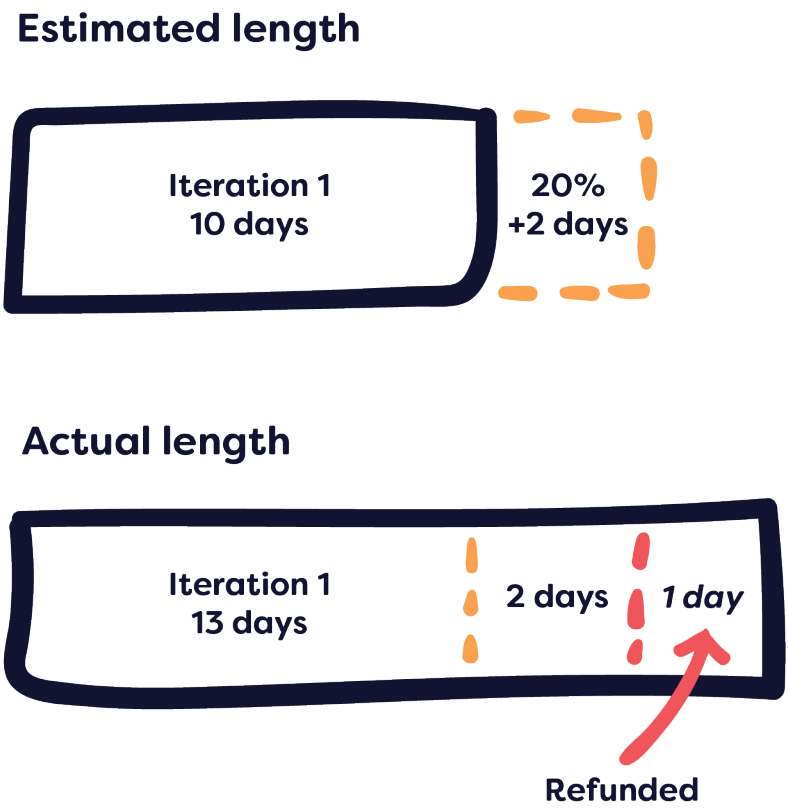
Urgent resolution **Next iteration resolution** **Backlog resolution**

Iteration guarantee.

Although there may not be certainty around the amount of effort required to complete a future milestone, there should have a good deal of certainty around the effort required for the next iteration. This provides an opportunity to give the customer some piece of mind and also provides a selling point.

Holding planning and elaboration sessions, and possibly completing further analysis through tech spikes, should make the team reasonably confident that they can hit their estimated deadline for the next iteration. As a result of this, a Guarantee can potentially be placed on the iteration around the stated estimation.

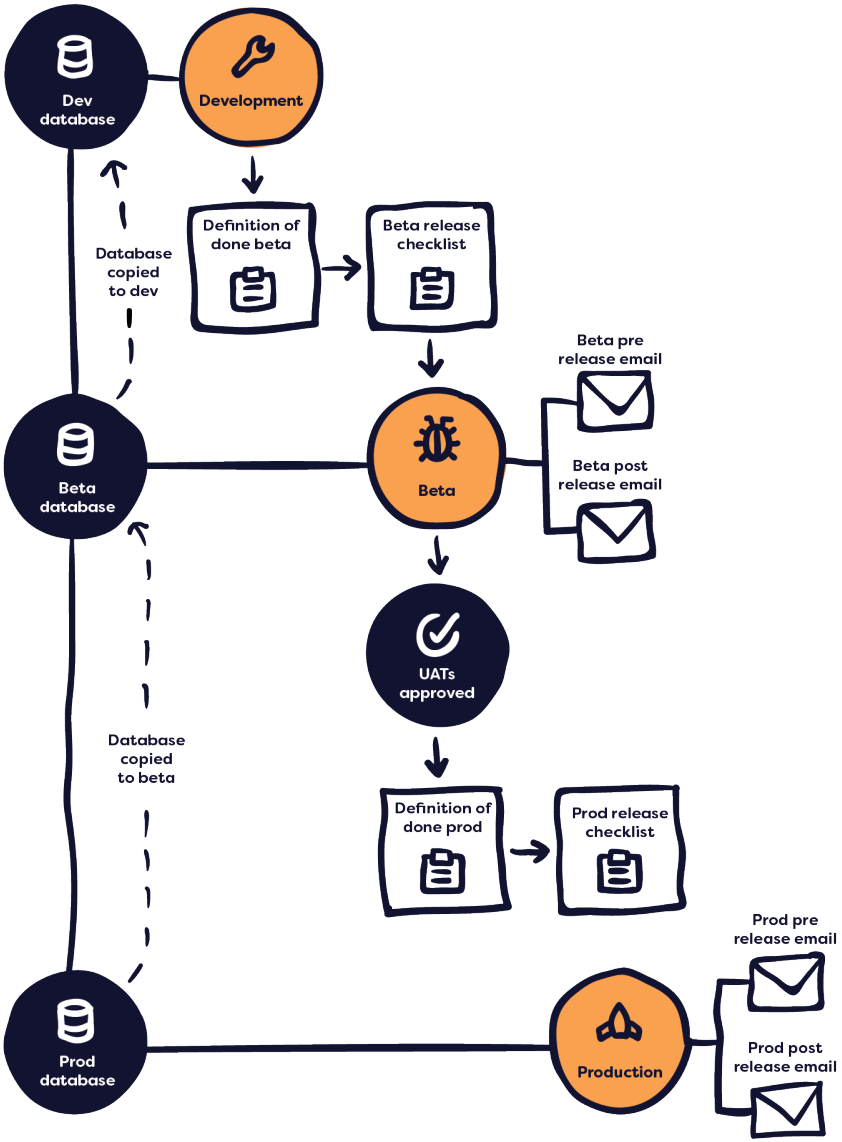
The Iteration Guarantee states that if it is estimated an iteration is going to take x days to complete, should the Iteration take x + 20% days, the cost of the overrun is not passed on to the customer. For example, if the project team estimates the next iteration will take ten days and it ends up taking thirteen, the customer will not be charged for the one day which overran. This time owed will be noted and can refunded from the final invoice.


Estimated length
Iteration 1
10 days
20%
+2 days
Actual length
Iteration 1
13 days
2 days
1 day
Refunded

Releases.

The goal of the release stage is to deploy the product into production and ensure that the stakeholders are ready to go into a live environment.

We recommend three environments to work with: Development, Beta, and Production. Software development requires at least three stages of environments to ensure quality control. When releasing, it is vital to make sure everything is working as expected in the previous environment before progressing it onto the next one.


Dev database
Development
Definition of done beta
Beta release checklist
Database copied to dev
Beta pre release email
Beta database
Beta
Beta post release email
UATs approved
Database copied to beta
Definition of done prod
Prod release checklist
Prod pre release email
Prod database
Production
Prod post release email

Release details.

Prerequisites

- Access to the customer's cloud services.
- A completed definition of done checklist for everything being released.
- A completed beta or prod release checklist (for whichever environment is being released to, see following pages for more information).
- Verified database update scripts.
- Passing tests on the current environment

Commitment

- A web engineer is required to start the release process.
- The project team members smoke test to ensure quality once the release has been completed.
- An operations engineer is required to approve and manage the release process.

Outcomes

- Copies of the environment's database from before the release was completed.
- The application is now available on the appropriate environment.
- There are appropriate levels of security in the released product.
- A report of the passing tests against the newly released environment.

Meetings

- The project team lead notifies all stakeholders when a product is about to be released, and once the release has been completed.

Tools

- Definition of Done
- Beta Release checklist
- Production Release checklist
- Automated scripts to improve the release process

The environments.

Development (Local) environment.

The development environment is used by the project team to build the application. All team members have their own local version running on their computers, where they work on the tickets currently in the iteration.

Location
The project team's secure local machines.

Access
This environment is available only for the project team and isn't shown to the product manager.

Beta environment.

Once an iteration has been completed and all tests are passing, the product is released to beta for testing. The testing is completed both by the project team and the customer as part of UATs.

As this can be a volatile environment, it is not recommended that any sort of important information is kept on this environment as the database may need to be wiped at any moment.

Location
Hosted on the cloud using the customer's nominated service

Access
Typically, this is a closed environment and therefore not open to everyday users, though the customer may choose to invite some people for user testing. Anyone with the URL can get to it, though a login is normally required.

Production environment.

The production environment is the final stage where only the most thoroughly tested code ends up. Nothing can end up in this environment without first being reviewed in the other ones.

While most people associate the production environment with the product being live, this is not necessarily the case. Until the URL is actually shared with the public, the production site can act as a platform where the public-ready code is kept.

Location
Hosted on the cloud using the customer's nominated service

Access
Anyone with the URL (and login if required).

Releasing mobile applications.

Mobile applications are typically built and released in conjunction with an environment release. As required, the project team can also release Android debug or Apple TestFlight builds depending on the features or testing required.

Development build / local environment.

The development process is done in a browser simulating the screen size of a mobile device. As required, the project team can also do a debug build onto an actual device which points to the local environment.

Debug build / beta environment.

As part of the beta environment release process, the debug build is compiled which points to the beta environment. Typically, this compiled app is sent on the customer for review.

Depending on the development work done in an iteration, there can be a debug build released without the need for a beta environment release (i.e. only if the interface is changing without a database change).

Location
Installed manually on devices. We also recommend that it is saved somewhere the product manager can access it (i.e. a wiki).

Access
Anyone with access to the debug build file.

Production build / production environment.

As part of the production environment release process, the production build is compiled which points to the production environment and published to the applicable app store.

As with the beta environment, there can be a debug build released without the need for a production environment release, however this is uncommon.

For iOS app releases, they must go through TestFlight and Apple's approval process before they can be released to the iOS App Store.

Location
Stored on app store.

Access
Anyone.

Checklists.

As part of the development lifecycle, an application moves from the development environment, into beta, then eventually into production. In order to progress into the next environment, the application needs to meet certain criteria before it can be approved for release. These criteria are outlined in two checklists: The Beta Release Checklist and Production Release Checklist.

After both releases have been completed, they should be smoke tested by the project team to ensure that the release happened correctly.

Beta release checklist.

This checklist is to be completed when an iteration has finished development on the local environment and is ready to be released onto beta for testing. Before the release starts, the project team needs to complete the Beta Release Checklist and submit it to an Operations engineer for approval.
While most of the criteria involves routine security and quality checks, two important artefacts which are required are a Testing Report and a Traceability Matrix.
A Testing Report shows the current state of the tests run against the application, which are expected to all be passing before the release can continue. The tests which are written cover the iteration requirements, acceptance criteria and functionality.
A Traceability Matrix maps tests to the requirements of a project. From this artefact you can see the test coverage, and where tests need to be written in order to give peace of mind over the applications reliability.

Before beta moves onto production, the production database is first copied down onto beta so that smoke testing is using accurate information.

Production release checklist.

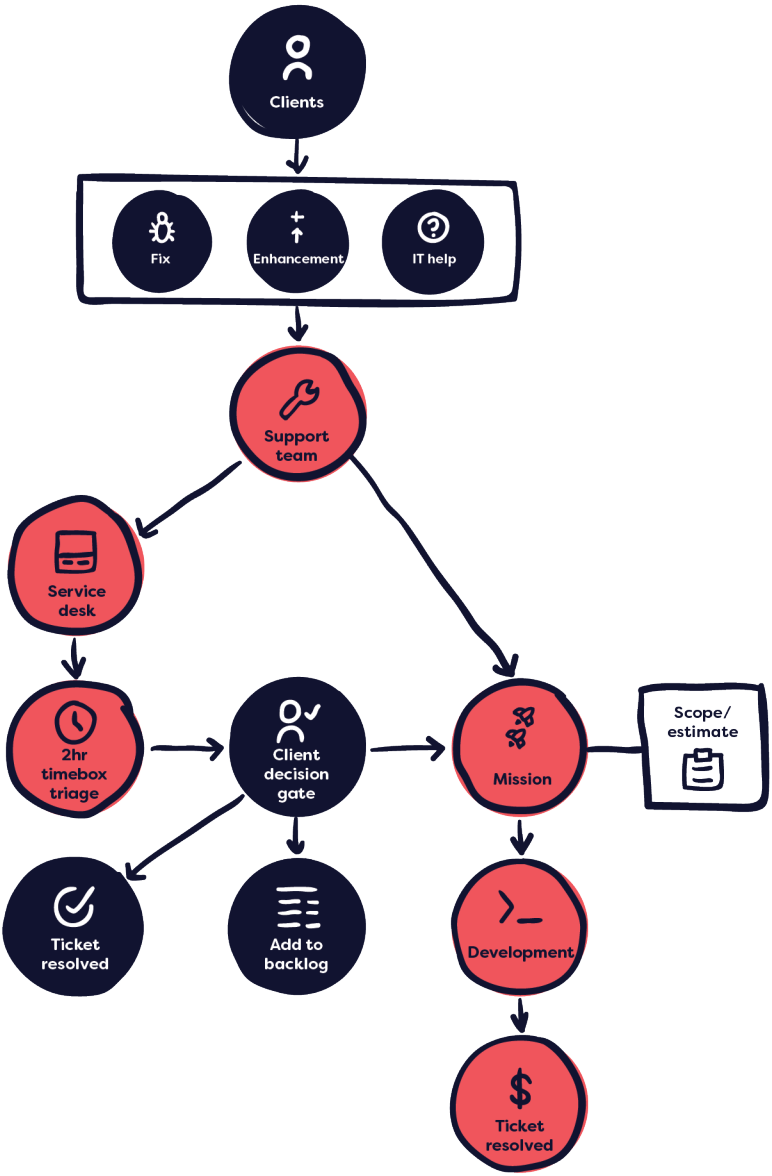
The Production Release Checklist is prepared when the product on beta is ready to be released to production. Before the application can be released into the production environment, the team also needs to make sure that the customer is happy with the current state of beta (through a UAT session). Once the checklist has been completed, then the proposed release and its checklist is submitted to an Operations engineer to begin the release.

Support.

The support stage is focused on helping the product transition from active development to daily operation. The support team helps to monitor the production environment and focuses on keeping the product running smoothly.

While the support stage is optional, planning and preparing for operational support in advance allows for a more detailed knowledge transfer to take place. For that reason, it is best to assume that all applications will go through this stage.

The support staff have a system to triage, replicate, diagnose and attempt to resolve any problems which may be encountered. The original project team can also be called upon if an escalation is required.


Clients
Fix
Enhancement
IT help
Support team
Service desk
2hr timebox triage
Client decision gate
Mission
Scope/ estimate
Ticket resolved
Add to backlog
Development
Ticket resolved

Support details.

Prerequisites

- The project is in the production environment.
- There is an active support agreement.
- The product has completed the first two stages of support.

Commitment

- The support agreement and issue severity dictate the amount of support time which is available to work on an issue.
- The original project team may need to be called upon if an issue is escalated.
- The project team must be available to complete the initial handover to the support team.

Outcomes

- A product which is maintained as long as the service agreement is active.

Meetings

2 Daily Huddles

- Twice a day: one in the morning, one after lunch.
- Used to prioritise any new tickets which have come in.
- New tickets and any other outstanding pieces of work are assigned to available team members.

Weekly Team Meeting

- A chance for the support team to share new knowledge
- Review tickets which have been completed recently or are making trouble
- Review customer satisfaction

Knowledge Handover Session

- Completed at the beginning of the support cycle
- Held with the original development team
- See the stages of support section **(page 72)** for more information

Tools

- Product backlog
- Service Desk portal
- Automated tool to set up local environments quickly

Tickets.

There are three types of requests which may be raised: bug fixes, feature enhancements or general inquiries. The requests which come in are typically allocated into a time box (which would have been pre-defined in a support agreement with the customer), and a support team member is assigned to work on them.

Severity

All requests which are submitted are given a severity level, which dictates how urgently the ticket is worked on. There are currently four levels of severity:

Blocker (Severity 1)

Produces an emergency in which the software is inoperable, produces incorrect results, or fails completely.

Critical (Severity 2)

Produces a detrimental situation in which performance (throughput or response) of the software degrades substantially under responsible loads, such that there is a severe impact on use. The software is usable but materially incomplete. One or more mainline functions or commands might not operate, or the user is otherwise significantly impacted.

Major (Severity 3)

Produces an inconvenient situation in which the software is usable but does not provide a function in the most convenient or expeditious manner, and the user suffers little or no significant impact.

Minor (Severity 4)

Produces a noticeable situation in which the software use is affected in some way but can be corrected by a documented change or by a future, regular release from a project team.

The stages of support.

When a project is transitioning from active development into support, it is important that the handover of information is completed successfully. This reduces the risk of support team members having insufficient knowledge to solve a problem and needing to call in the original project team to assist.

Phase 1: Analysis and Transition

Goal: To complete a knowledge transfer from the project team, and ensure there is enough knowledge available for the support team

- Complete a knowledge transfer meeting with the project team
    - Identify the single individual resource in the support team who will be responsible for production support and escalations
- Hold a comprehensive audit of the existing infrastructure, application and processes
    - Look into the health and performance of the application across platforms
    - Review previous UAT results
- Complete an audit of the documentation, performance testing, and reports
- Create a gap and requirements analysis on the project backlog

Phase 2: Planning and Readiness

Goal: To complete any missing information identified in the previous step and fix any high-priority bugs currently in the product.

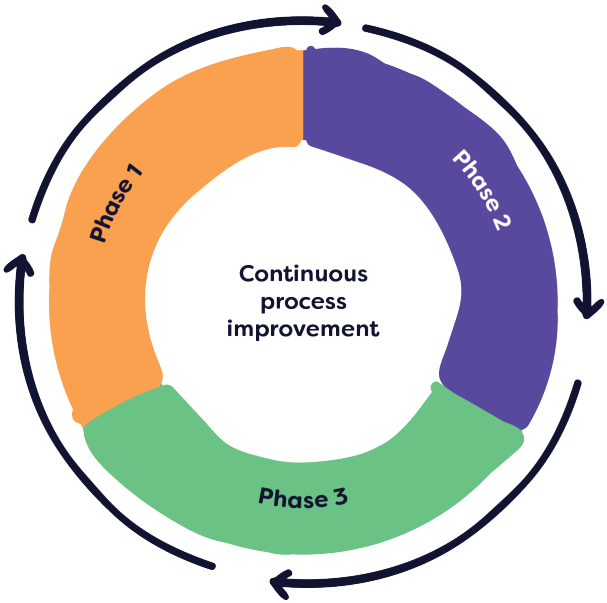
- Fixing immediate and important issues
    - Either through an additional iteration, ad-hoc development resources or short-term hire.
- Create a Missing Documentation report

- Write an Overall Performance Enhancement report
- Create a summary of the current backlog

Phase 3: Continuous Maintenance and Operational Support

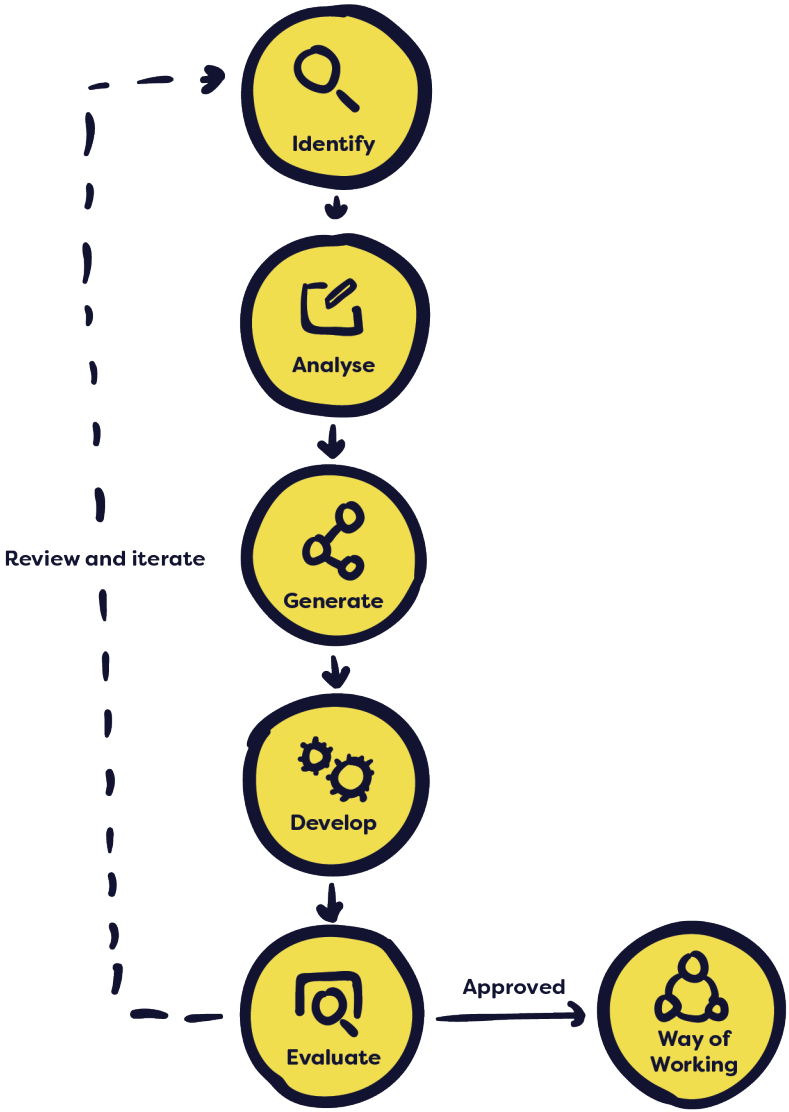
Goal: An ongoing phase to support the product.

- Advise the customer on how the product's performance could be enhanced or improved upon as its usage takes off.
- Provide proactive and preventative monitoring through regular usage and testing of the product
- Issue and bug tracking and troubleshooting
- Completing timely updates and requirements-based upgrades
- Maintain back-ups and provide recovery for the software
- Provide helpdesk or on-call support.



Experimenting.

The mantra behind this Way of Working is to discover new boundaries. It is encouraged that everyone proposes experiments in the name of constantly improving the methodology. If the results of an experiment are approved by the appropriate stakeholders, it becomes a part of this Way of Working. Any adopters of this Way of Working are encouraged to continue this process of experimentation to adapt the Way of Working to suit their unique needs and experiences.


Identify
Analyse
Review and iterate
Generate
Develop
Evaluate
Approved
Way of Working

1. Identify improvement potential

The first stage of an experiment is to identify the need to conduct one, and then clearly explain what the problem is. Useful questions which can assist in this process are:

- Where did the problem occur?
- When did the problem occur?
- What process did the problem involve?
- How is the problem measured?
- How much is the problem costing?

2. Analyse current methods

To find the best solution, the root cause of the problem must first be properly identified. The root cause can be found through analysing the historical data and speaking with people this problem has affected.

3. Generate ideas or improvements

The next step is to identify potential solutions. Brainstorming is an important part of this step, as it helps conceive of a range of solutions. For each idea, establish how it will help solve the problem and any potential pros and cons.

4. Develop an Implementation Plan

After a range of solutions have been identified, the most appropriate one for the problem can be chosen. The next step is to create a strategy to implement a solution. The following points should be answered in the Implementation Plan:

- Who is in charge of the experiment?
- Who is involved in the experiment?
- Duration of the experiment
- How we are implementing the experiment
- How to track or measure the experiment (in money, time, customer satisfaction, or another critical metric)

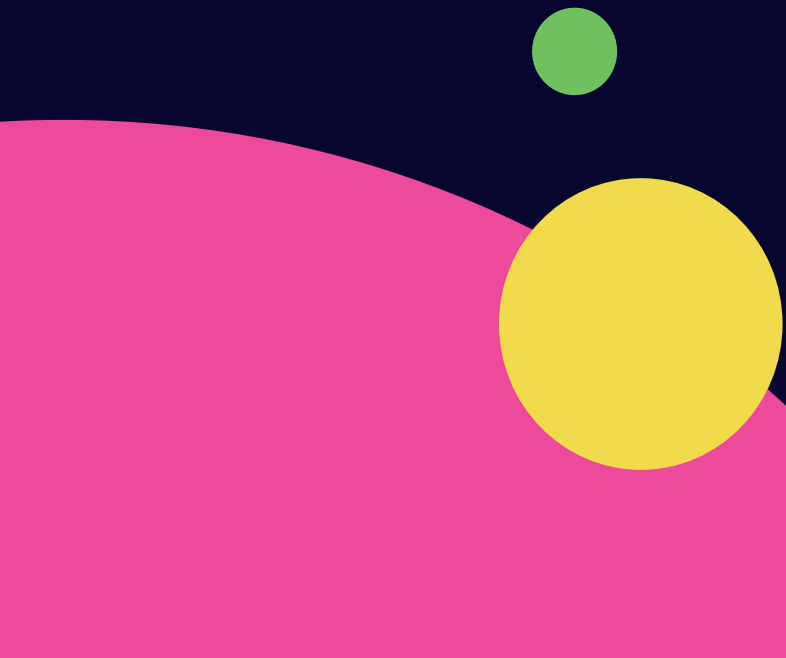
5. Evaluate the new method

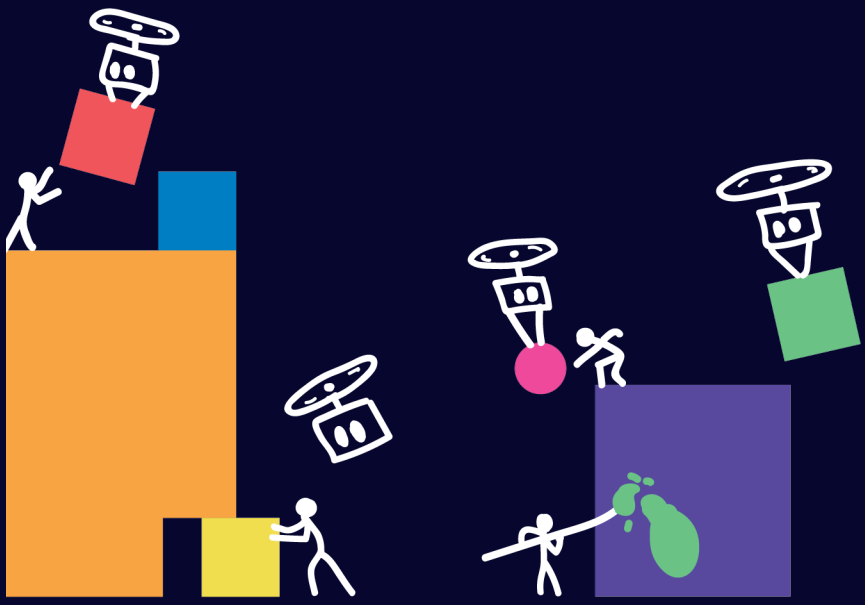
Once the Implementation Plan has ended, the findings of the experiment are discussed amongst the stakeholders. They will then determine whether the experiment achieved the measurable goals and if it could be considered a success. If the method failed, it should be identified why and a new strategy should be iterated over.

If the method worked, then the strategy should be proposed to management, who alongside the experiment stakeholders, will determine if the strategy is both beneficial and applicable across the whole company. If the method is approved, it becomes a part of the businesses Way of Working.

Evolving a codebot.

The dream of the Codebots platform is to create a product where humans and bots can work together to create awesome software. For certain customers there is the potential to create a new bot that writes a specific technology stack or framework.





Identifiy an opportunity.

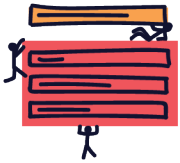
Part of what makes the codebots so efficient is the use of 'behaviours', which are patterns of functionality which are often repeated in different products (i.e. CRUD is in almost every product in some form or another). Codebots takes advantages of these repetitions and uses them as a chance to save development effort by creating reusable behaviours which can be applied to any part of a product.

The Codebots library of behaviours has been built out over many years as a result of the multitude of products which have been worked upon. During the brief and scoping stage of each product, the team were able to identify behaviours which could be used to complete certain requirements. They were also able to determine pieces of functionality which could potentially be turned into new behaviours.

The ability to identify a potential behaviour is about recognising patterns in both functionality and user experience. A behaviour may be something which the team developed in a previous application, or it may be something which is quite common and is most likely going to need to be developed again in the future.

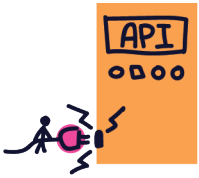
Here is sample list of some behaviours are:

(This is not a full list of what is available on Codebots, it is just an example)

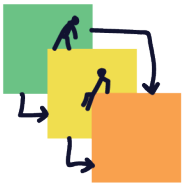


CRUD

Forms



API



Workflows

Dashboards

Timelines

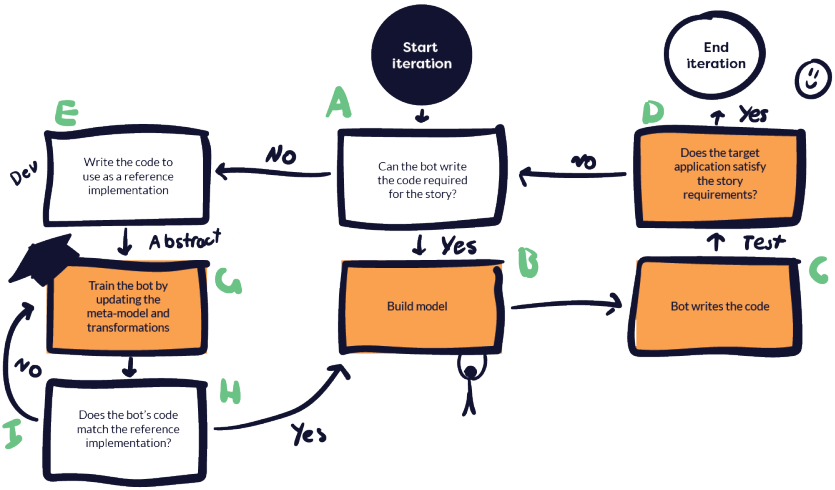
Starting the evolution.

Once an opportunity has been identified, the project team have an opportunity to train the bot in the new behaviour (or improve/extended an existing one).

The following diagram demonstrates how this idea can turn into a chance to improve the bots. It follows the initial development process of the concept and how the bot gets trained.

(In this flow chart, the developer would not write inside the protected regions of the code.)

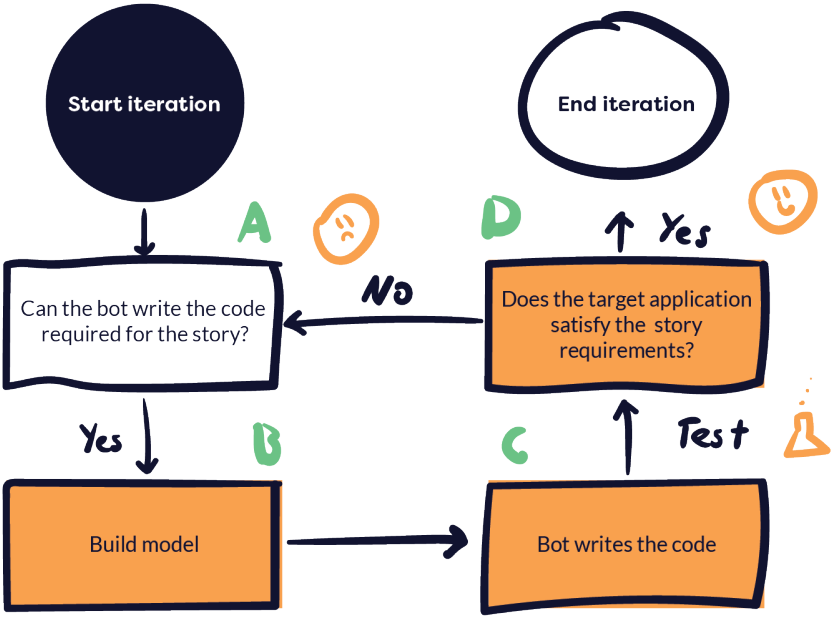
A. For each issue: “Can the bot write the code required to meet the requirements?”
B. If yes, the developer will build the model. (If not, go to **E**)
C. The codebot will write the code.
D. If the requirements are fulfilled, the issue is complete.
E. If the requirements can’t be met, the developer will create the reference implementation.
F. The developer will train the bot by updating the model and transformations.
G. The developer will compare the codebot written code to the reference implementation.
H. If they match, the developer will build the model (go to step **C**).
I. If they don’t match, the abstraction step needs to be performed again (go to step **G**).


Start iteration
End iteration
A
B
C
D
E
G
H
I
Dev
Write the code to use as a reference implementation
NO
Can the bot write the code required for the story?
NO
Does the target application satisfy the story requirements?
Yes
Abstract
Yes
Test
Train the bot by updating the meta-model and transformations
Build model
Bot writes the code
NO
Does the bot's code match the reference implementation?
Yes

Let the bots take over.

The following diagram demonstrates how the bot could potentially write a feature which meets the requirements without any customisation being required. At this stage, all of the code has already been abstracted into the bot's model. Developers only need to build the model for the bot to receive its instructions and write 100% of the target application.

A. For each issue, confirm that the codebot can write the code required to fulfil the requirements.
B. If yes, the developer will build the model. If no, then it will need to go back to diagram one.
C. The codebot will write the code
D. Then, if the requirements are fulfilled then the issue is complete. If not, development continues.

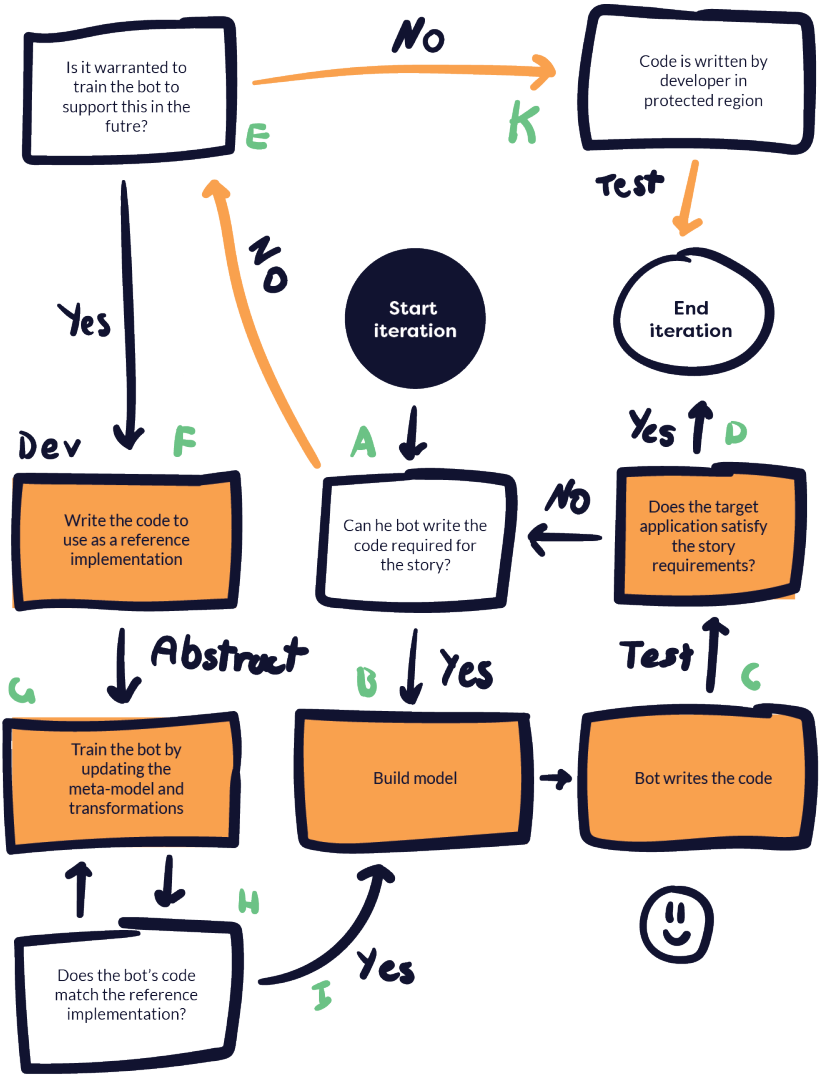

Start iteration
End iteration
A
D
Yes
No
Can the bot write the code required for the story?
Does the target application satisfy the story requirements?
Yes
B
C
Test
Build model
Bot writes the code

Business as usual.

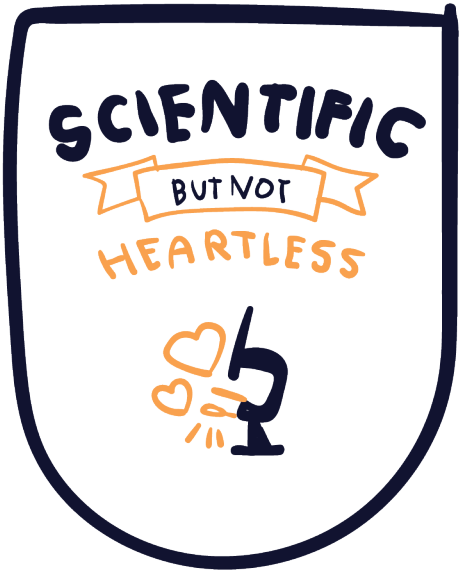
The following diagram is the stage where we want to operate. Developers work with the bots to create software, following the process for training the bot and ensuring the code written by the bot meets the requirements.

The ultimate goal for this process is to progressively increase the amount of project code produced by the bots.

A. For each issue: “Can the bot write the code required to meet the requirements?”
B. If yes, the developer will build the model.
C. The codebot will write the code.
D. If the requirements are fulfilled, then issue is complete.
E. If the answer to **A** no, ask: “Is it warranted to train the bot to support this code in the future?”
F. If yes, the developer will create the reference implementation. (Go to **K** for no)
G. The developer will train the bot by updating the model and transformations.
H. Compare the codebot’s code to the reference implementation.
I. If they match, the developer will build the model (go to step **B**).
J. If they don’t match, the abstraction step needs to be performed again (go to step **G**).
K. If the answer to **E** is no, the code will be written by the developer in a protected region. This would occur in the case of one-off development or customising the code that the codebot has written.


Is it warranted to train the bot to support this in the futre?
No
Code is written by developer in protected region
E
K
Test
Yes
No
Start iteration
End iteration
Dev
F
A
Yes
D
Write the code to use as a reference implementation
Can he bot write the code required for the story?
No
Does the target application satisfy the story requirements?
Abstract
G
B
Yes
Test
C
Train the bot by updating the meta-model and transformations
Build model
Bot writes the code
H
Does the bot's code match the reference implementation?
I
Yes


URGENCY
BUT NOT
RUSHED


SCIENTIFIC
BUT NOT
HEARTLESS


FUN
BUT NOT
UNPROFESSIONAL


INITIATIVE
BUT NOT
A LONE WOLF

Get in contact

Interested in learning more about Codebots or think it might be a good fit for your organisation?

(07) 3371 2003

sales@codebots.com

www.codebots.com