The Released Methind

Nick Beuageard

# Released Method

*This is the Released Method, a model for managing the development of large scale software projects with the goal of delivering solutions quickly, on budget and with high quality*

## Table of Contents

### [Introduction](introduction.md) - Introduces the Method and its background

### [Pillars](pillars.md) - The key areas of the method

### [Systems](systems.md) - The key systems used

### [Why?](why.md) - The business case for the model

### [Team](team-model.md) - The Team model

## About the Method

The Released Method was developed over 20 years of leading software development teams by Nick Beaugeard. Nick is an experienced software architect and startup founder and has a background in running and operating software development teams:

* Released Microsoft SMS Installer with ISU (2000)
* Released Bellerephon Desktop Deployment Service (2003)
* Released Dimension Data Dynamic Desktop Deployment (2005)
  + Won [3 Microsoft Partner Awards](https://www.computerworld.com.au/article/162101/dimension_data_awarded_three_microsoft_global_partner_awards_2006/)
* Released Dimension Data SLAM (2007)
* Released Community Engine (2008)
* Released HubOne Modern Practice (2010)
  + Won ARN Software Developer of the Year Award
  + Won AFR Top 100 Companies to Watch
  + Won RedHerring 2011 Award
* Released HubOne Engine (2012)
* Released WorkflowMax PowerShell Library (2013)
* Released Scanned Document Manager (2014)
* Released H&R Block Training Portal (2015)
* Released Invicta Special Forces Management Tools (2016)
* Released CA Kairos (2017)
  + Won ARN Software Developer of the Year Award
* Released CCH iFirm Document Manager (2018)
  + Runner Up - ARN Software Developer of the Year Award
* Released Unblocked Platform (2019)

## Business Case

The point of using a structured and robust methodology, tested in many product cycles is to ensure that lessons learned in creating many millions of lines of tested code are applied rather than your project having to learn them again.

This results in a project which: \* Delivers its Outcomes \* Reduces Costs \* Increases Quality \* Reduces Risk \* Increases Team Cadence \* Increases Transparency \* Produces Commercial Quality Deliverables

## Measure Yourself

If you want to see how you compare, see the [Startup Checklist](startup-checklist.md).

## Be a part of it all

If you want to help grow the Released Method, make sure you read the [Code of Conduct](CODE_OF_CONDUCT.md) then head over to [Contributing Guidelines](CONTRIBUTING.md) to get started. [Home](README.md) | [Why](why.md) | [Pillars](pillars.md) | [Systems](systems.md) | [Team](team-model.md) # Contributor Covenant Code of Conduct

## Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

## Our Standards

Examples of behavior that contributes to creating a positive environment include:

* Using welcoming and inclusive language
* Being respectful of differing viewpoints and experiences
* Gracefully accepting constructive criticism
* Focusing on what is best for the community
* Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

* The use of sexualized language or imagery and unwelcome sexual attention or advances
* Trolling, insulting/derogatory comments, and personal or political attacks
* Public or private harassment
* Publishing others’ private information, such as a physical or electronic address, without explicit permission
* Other conduct which could reasonably be considered inappropriate in a professional setting

## Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

## Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

## Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at nick@releasedgroup.com. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project’s leadership.

## Attribution

This Code of Conduct is adapted from the [Contributor Covenant](https://www.contributor-covenant.org), version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq

[Home](README.md) | [Why](why.md) | [Pillars](pillars.md) | [Systems](systems.md) | [Team](team-model.md) # Contributing Welcome to the Released Method. We welcome contributions.

The documentation is all written in markdown. If you’d like to have additional pages, edits to pages and more considered, please fork this repository and when your edits are complete, submit a pull request for review.

We welcome submissions from corporations with products and services for this market, but please be aware we reserve the right to edit and modify blatant advertising. The purpose of this repository is to assist startup participants make better decisions and write better software.

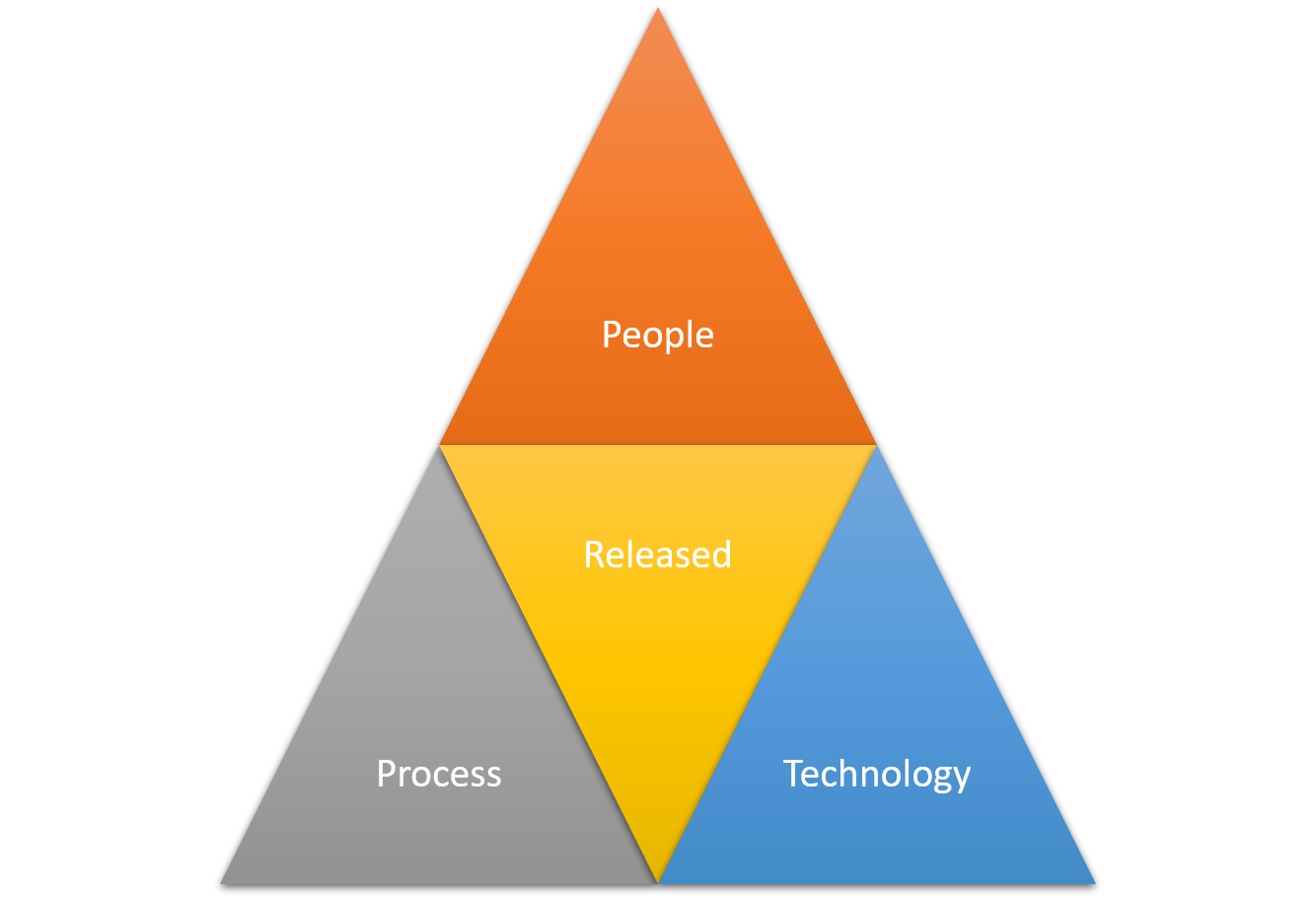
We aim to reduce the waste on incomplete or poor solutions and help springboard the fourth industrial revolution through free discussion and open help.

Finally thanks for contributing. We understand the time and effort needed to contribute and appreciate your dedication.

[Home](README.md) | [Why](why.md) | [Pillars](pillars.md) | [Systems](systems.md) | [Team](team-model.md) # Introduction Welcome to the Released Method. This is a method for the delivery and management of large scale software systems. Birthed as agile methodologies were first introduced, the Released Method aims to integrate all of the aspects of a software product team to ensure that what is delivered is extremely high quality and meets the needs and expectations of the customer.

Whilst the Released Method is an effective mechanism to deliver high-quality software, it also relies on people. The team needs to have specific skills based on what activities they are to perform, and also need to have effective communications skills.

Finally comes a reliance on technology. Whilst the original author has his specific favorite technology stack for development it is feasible to implement the released method using many combinations of technology. See the technology reference architecture for more information on the technology requirements.



PPP Triangle

The diagram above shows the **People, Process, Technology** triangle. Optimal projects are seen where the investment in all three is balanced. One of the core goals of the Released Method is to balance the three investments as follows:

|  |  |
| --- | --- |
| Area | Investment |
| People | By using an effective team model, and ensuring that all critical functions are well-staffed, the people segment is managed. |
| Process | By implementing all the pillars of the Released Method, the process segment is managed |
| Technology | By integrating solutions for all the systems requirements of the Released Method, the technology segment is managed |

## History

The initial genesis of the Released Method came from the author’s time at Microsoft at the turn of the Millenium. That was at the stage where the software industry realized that the “Waterfall Method” assumed to be espoused by Dr. Winston W Royce, was by accident more likely to be an agile methodology and the industry struggled to correct 40 years of incorrect process teaching and invented sometimes crazy ideas such as Xtreme Programming, Scrum and others.

Whilst some of the practices in those methodologies are “silly” or “strange”, their approach is in the main correct. The Released Method can, therefore, be thought of as an implementation of a serious agile methodology for the 2020s. [Home](README.md) | [Why](why.md) | [Pillars](pillars.md) | [Systems](systems.md)

# Why?

There are a number of software development methodologies available today, from Waterfall through to Scrum and RUP. In fact there is a nice discussion of all of them [here](https://www.weblineindia.com/blog/top-15-software-development-methodologies-with-advantages-and-disadvantages/), however many of them attempt to be generic and cover every conceptual approach. The Released Method is focussed on one goal, that of delivering a working and successful product to market.

The Released Method aims to deliver a commercially capable release, in the shortest amount of time and with the highest quality to meet the lean approach of the modern startup. It is assumed that the team have total executive control over the project and the only external influences are customers and the market.

It is not designed to work within large enterprise as they have a significant amount of internal stakeholders and product owners and it is not expected that this approach works well in that scenario.

It is designed, however to work well in a skunk works style environment and is tailored to delivering innovative or ground breaking software never seen before.

Key Features of the Released Method are:

* Quality Software
* Testing
* Automated Release
* Team Management
* Agility
* Reduced Cost

# Business Case

Embarking on a new software development project is a voyage into the unknown. Unlike constructing a bridge, it is not possible to predict how long a software development project will take as there are a significant number of unknown items and items that will be refined along the way.

Agile development also makes it difficult to accurately predict the cost of a project as the process of developing software in an agile fashion is fluid and can change.

Rather than use command-and-control techniques to control such a project, entrepreneurs need a way to predict outcomes and costs. The Released Method assures a level of quality deliverable and breaks the delivery into sprints, short bounded work units with agreed budget, time and outcome.

*It should be noted, that a sprint is an intention, and as the project matures these get increasingly predictable*.

Given it is not possible to accurately predict the cost and duration of all but the most simple software projects, the Released Method allows delivery within a budget, ensuring that should cost be fixed, the methodology is agile enough to adapt cost and time to suffice.

In addition, using such a tried and tested methodology, users can be assured of a “decent” result, within the various constraints.

[Home](README.md) | [Why](why.md) | [Pillars](pillars.md) | [Systems](systems.md)

# Pillars

There are six pillars that make up the released process. These are:

* [Management Reporting](management-reporting.md)
* [Software Planning and Development](software-planning.md)
* [Testing and Quality Assurance](testing-qa.md)
* [Release Management](release-management.md)
* [Documentation](documentation.md)
* [Selling](sales-kit.md) [Home](README.md) | [Why](why.md) | [**Pillars**](pillars.md) | [Systems](systems.md) | [Team](team-model.md) # Management Reporting Management Reporting is critical to inform management, project investors and the overall executive how the project is performing. It consists of the following key outcomes.

## Outcomes

* Burndown
* Test and Regression Rates
* Development Cadence
* Estimated Dates
* Cost and Duration

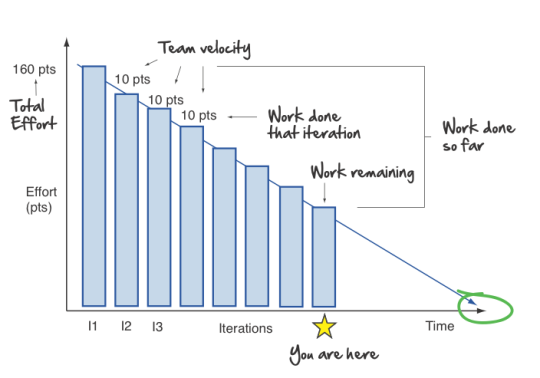
## How

Management reporting is normally delivered through a combination of reports, graphs, and charts.

Each of the outcomes and their required reports is shown below:

## Burndown

The burndown is a chart that shows how quickly you and your team are burning through user stories. It shows the total effort against the amount of work we deliver each iteration.



Burndown

We can see the total effort on the left and our team’s velocity on the right. This graph also gives us: \* Work done each Iteration \* Work remaining \* Work done so far \* When we can expect to be done

An example Burndown spreadsheet is available [here](http://www.agilenutshell.com/assets/Sample-Burndown.xls)

## Test and Regression Rates

[Home](README.md) | [Why](why.md) | [**Pillars**](pillars.md) | [Systems](systems.md) | [Team](team-model.md) # Software Planning and Development

The Software Planning and Development is run according to a cyclical life cycle model as follows:

* Envision
* Prototype
* Architect
* Validate

The Released Method adheres to the principles of the [Agile Manifesto](https://agilemanifesto.org/). It does this by running this model across a series of **sprints**. Sprints are defined as a time boxed, discrete work output that fulfils the Agile Manifesto by:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Released Method sprints always deliver valuable working software for the customer to work with to allow them to understand what is possible and adapt their requirements based on their new understanding.

Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.

The Released Group Specification, automatically documented by the tools at [Project Documentor](https://github.com/nickbeau/project-documentor) allows us to host specification items as GitHub issues and change them as necessary.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Released Group Sprints take a week of full time work, from everyone, but can occur over 1 to four weeks elapsed and always deliver a release of working software.

Business people and developers must work together daily throughout the project.

The Released Method Team model is a single team, regardless of discipline.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The Released Method team model is a single team, highly motivated to deliver results. The job of the process is to get out of the way and enjoy the results.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity–the art of maximizing the amount of work not done–is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

These phases are defined below

## Envision

In this phase the project is initially defined, it consists of three major processes:

* Brainstorm
* Hypothesis
* Wild Ideas

For most, the process of envisioning software is a boring process that involves lots of documentation. The Released Method involves asking ourselves, what do we want to solve, and ignores the how do we solve it. The team should be allowed to hypothesise anything. History, opinions and other external influences should not be allowed to interject here. This is why commonly, long term development teams employ a short term agile team to launch them into a new trajectory wherein they take over.

The output of Envisioning is a set of github issues labelled as follows:

* **FunctionalReq** - A functional requirement is something needed to work for the end user. It should describe how a user achieves an outcome by using the software, and needs to be better than the alternatives.
* **NonFunctionalReq** - A Non functional requirement is something like scalability, security, performance, design which does not affect the function but is as important.
* **Pri1** - A Priority One issue must be done immediately for the product to have any value
* **Pri2** - A Priority two issue is very important for commercial success
* **Pri3** - A Priority three issue is a “nice to have”
* **SpecRequired** - It’s something we want but we haven’t fully documented it yet

## Prototype

In this phase, initial prototypes are developed to see if the product is feasible. There are three major processes:

* Build
* Verify
* Fail Fast

We work to construct a working version of the different components. Initially, where experimentations is required, these can be rapidly developed prototypes. Should the concept of feature be feasible, the team should note the results and move to the next one. At all times, these prototypes should be available to the customer.

## Architect

Once out of the prototype phase, the build of the product components starts for real. Developers follow the processed below:

* Develop
* Document
* Refine

To develop, we take a prototype and add the items necessary for commercial code, including security, logging, error checking and bounds checking. We normally write unit tests here to ensure the feature or item works robustly. Finally we document what a 3rd party needs to know about the feature or item.

Unit tests force us to refine the component for security speed and performance.

## Validate

Once the product component is ready, it goes through the following major processes:

* Validate
* Optimize
* Enhance

We validate the component against the written and agreed specification. If it meets it, we discuss if it is what the client wants or needs. We make sure its what everyone wants, then we run through the code, reviewing and improving where we can and raise issues for future refactoring. The code gets released and the burndown now contains new items to perform.

## The Sprint

A spring contains all the items above, for one or more features. Sprints are initially articulated based on a defined specification, but as the project continues can be used differently based on the changing customer requirements.

[Home](README.md) | [Why](why.md) | [**Pillars**](pillars.md) | [Systems](systems.md) | [Team](team-model.md) # Testing and Quality Assurance Testing and Quality Assurance follows a cyclical model as follows:

## Testing

* **Unit Tests** are automated components of electronic testing of the software to ensure functions operate as desired
* **Code Analysis** is the process of analysing code for defects at a line by line level using automated tools.
* **Automated Tests** commonly automate the user interface and test the UI does what is required.
* **Manual Tests** are manual tests performed by humans to ensure the application works as desired.

The person (or manager of a system) which creates a bug is called the **originator**

## Bug

The result of testing is a **bug**. A **bug** is a record of something being wrong with the software. Bugs should include the following detail:

|  |  |
| --- | --- |
| Item | Description |
| Title | One line, simple description of the Bug |
| Id | Preferably an integer, representing the bug number |
| Description | a detailed description of the bug, describing the issue, how to **reproduce it** and the impact |
| Severity | The impact of this bug to the software or customer |
| Priority | The order in which the fix should be developed |
| Comments | An ongoing commentary of this issue |
| Labels | Additional Labels depending on workflow |
| Originator | The individual who created the bug report |
| Owner | The individual who currently owns this bug report |

People are commonly very poor at creating bug records that can be resolved easily by a developer. In order for a bug to be a “good bug” it needs to have a **repro**

## Repro

Repro or reproduction is step-by-step instructions a developer can take to reproduce or recreate the behavior classified as a bug.

The Repro should be described in discrete steps, showing a developer how to execute enough to reproduce the errant behavior. It should then describe why the observed behavior is errant.

## Severity

Severity is the impact this bug has on the product or customer experience as a whole, severities are as follows:

1. Cannot use the product. This bug breaks the product or feature
2. Difficult to use the product. This bug makes it unintuitive or difficult to use the product and requires a work around.
3. Annoying, but does not stop a workflow.

*the severity is commonly chosen by the bug originator*

## Priority

The Priority is how important the development team consider this bug to be and in which order it will be addressed. Development staff are encouraged to resolve priority 1 items first, followed by 2 and so on.

Priority is set in Triage.

## Triage

Triage is the process of analysing requests to the project team and managing them in a highly efficient manner. In short, anyone can raise a request or issue for the team and it is the triage team which uses the triage process to manage and process the request.

During a short meeting the team agrees one of the following actions on a request.

* **Investigate** - the request will be investigated further
* **Approve** - the request will be done by the team. An approved request is always assigned to a **resolver**.
* **Reject** - the request will be denied. This is normally done if the issue cannot be reproduced (called no repro) or it is designed to work that way and is not a bug (by design).
* **Defer** - the request will be deferred. Triage have the power to defer (punt) a bug report to a future version of the software, especially if that report has low priority and/or severity.

Once the bug has been assigned, a **resolver** will attempt to 1resolve the bug. Once they believe they have resolved the bug, they makr the bug report as **resolved** and re-assign it to the **originator**

The **resolver** or **originator** both have the ability to revert the bug for discussion at triage. Commonly this is performed by unassigning the bug, however it can be done by adding a label called Triage.

Once the bug has been resolved, the **originator** is responsible for ensuring the bug has been fixed. They do this by:

* **Validate** the originator ensures that the bug is no longer present
* **Regression Test** - it is best practice for the test team or originator to develop a regression test to ensure this issue never occurs again.

Once ths bug is resolved, it is marked as closed. [Home](README.md) | [Why](why.md) | [**Pillars**](pillars.md) | [Systems](systems.md) | [Team](team-model.md) # Release Management The Release Management pillar consists of the following:

* Quality Bar
* Automated Release Process
* Staging and Production Platforms
* Release Authorization
* Version Control

## Quality Bar

It’s really difficult to decide when a release or product is “done”. In fact it’s pretty easy to keep going making things incrementally bettwe without ever really achieving a release.

Most successful startup companies reward the release or “ship” with an award, because for a company, releasing software is the most important task a development team can perform.

However, with the pressure on making the software “perfect” coupled with the pressure of releasing the software, leaders need a way to understand when the software is good enough to ship.

This is the role of the Quality bar. Once software has met the quality bar, it is ready to release.

I espouse a certain quality bar, but you are, of course free to implement whichever quality bar you wish.

There are certain things we can measure in software development and items we cannot. Items we can measure are:

* Number of outstanding tasks
* Number of outstanding Bugs
* How many tasks have been tested
* How many tasks have been documented.

If we were to take a single **user story** or **feature**, this will have a number of tasks. A task should go through the following states:

* New [Unassigned]
* New [Assigned]
* In Development
* In Test
* In Documentation
* Closed

The phases are as follows:

### New

A New task has been considered, and *requested* but is not currently approved or in active development. Once a task has been approved, it is assigned to a developer. The developer moves it between New [Assigned] and In Development until the task is considered “code complete”

## In Test

Once a task is code complete, it is moved to the test team who *attempt to test as much of the results as they can*. Commonly testing will have target metrics such as code coverage or features coverage. See [Testing & QA](testing-qa.md) for more information.

Once all bugs have been raised (not they have been solved), the task is considered tested and moves to the In Documentation phase, even while bugs are being fixed.

## In Documentation

Once a task has been tested, it is released to the User Education team to write documentation on how to use the individual feature within the task. Once the documentation is written, it too goes through a similar test process. Once the documentation is written (complete) and all bugs have been raised (tested), the task is marked as closed - even if there are still outstanding bugs.

## Closed

The task is complete. Even if there are bugs to resolve, the task is finished and we are just fixing bugs and issues.

## Measure 1

One Measure of a quality bar is: **All tasks slated for this release are closed** - That means there is no more development to do in this release and every item has been tested and documented.

## Bugs

I wont’ go into why we call problems with software or documentation bugs, but a bug is something wrong with the software or documentation. Bugs are identified and move through a formal process to be resolved.

## Measure 2

One measure of a quality bar is **There are no open bugs for this release** - It should be noted that this doesn’t mean there are no bugs, quite the contrary, it means there are no bugs considered important enough to block this release.

## The art of punting

Punting is the process of moving a work item from this release to a future release (commonly the next). It is used by triage to determine if an items **priority** or **severity** warrants fixing.

## Measure 3

Severity and Priority items are considered in the quality bar, for example, any bug under severity one, priority one will be able to be punted to the next release.

## Our Quality Bar

Following on from this one defines the released method quality bar as follows:

1. All tasks are closed
2. There are no open Sev 1, Pri 1 Bugs
3. The Triage Team agrees to release

As Triage is a team of peers, it is common practice to include triage in a release authorisation process, ensuring that everyone agrees to release or not.

## Release Authorisation

Release Authorisation is the process whereby Triage agrees to release (or not release) this version of the software or solution. Commonly, any individual can block the release, without repercussion, their issues will be reviewed and docuemnted by triage and resolved to their satisfaction.

Following that, release can occur. [Home](README.md) | [Why](why.md) | [**Pillars**](pillars.md) | [Systems](systems.md) | [Team](team-model.md) # Documentation Many items in a project need to be documented and documentation is key to a successful project. Documentation items are developed according to the phases of the project. They serve different purposes but together deliver the key public requirements of any software company.

## 1. Envisioning

Envisioning is defined as *the process of simplifying complex goals into a single statement or statements that can be learned and repeated by the entire team* - It’s important to have a goal, and product teams are no different. People rise to lofty ideals and goals and the early stages of a project is where you envision what will be with your software and solution. Examples of good envisioning statements are:

xxx product will help anyone move across borders by being a secure, verifiable COVID-19 Vaccine passport

yyy product will help accountants save money and delight their clients by streamlining the creation, storage and retrieval of documents

Poor envisioning statements are overly large, wordy or difficult to get behind. In both the statements above, the products’ vision statements include their benefit to people, and this is again the key to a good statement.

## 2. Architectural

While the parable of building a house on sand rings true in physical architecture, the same is true in software design. Every product needs a strong and robust architecture and set of guiding principles.

## 3. Developer

A Development team need to understand how to develop to this product, how you manage source control repros, how you perform and approve pull requests and how automated builds work.

## 4. SDK

A Software Development Kit (SDK) should be developed to assist external developers use your software.

* Implementation
* Platforms and Release Management
* Documentation Quality Control [Home](README.md) | [Why](why.md) | [Pillars](pillars.md) | [Systems](systems.md)

# Selling

When you are going to market, you need a number of key deliverables to support your sales process.

These are detailed below:

* ☐ [Sales Presentations](saleskit/sales-presentations.md) - Powerpoint (or equivalent) presentations to be given to sales staff and videoed.
  + ☐ 1 Slide
  + ☐ 3 Slide
  + ☐ 15 Slide
  + ☐ 30 Slide
* ☐ [Proposal Template](saleskit/proposal-template.md) - Document to allow sales staff to propose the solution to prospects.
* ☐ [Brochure](saleskit/brochure.md) - Customer facing document describing the solution.
* ☐ [DataSheet](saleskit/datasheet.md) - Technical document detailing the features and benefits of the solution.
* ☐ [Sales Battlecard](saleskit/sales-battlecard.md) - This is a quick reference guide for sales and marketing people to position your solution in a discussion.
* ☐ [Competitive Analysis](saleskit/competitive-analysis.md) - This document compares your product to the competition.
* ☐ [Pricing Matrix](saleskit/pricing-matrix.md) - This document details the pricing options for the solution.
* ☐ [Web Site](saleskit/web-site.md) - This is the website (microsite) for the product.

[Home](..\README.md) | [Why](why.md) | [Pillars](pillars.md) | [Systems](systems.md)

# Sales Presentations

Sales Presentations are PowerPoint decks which describe the solution and can be used as follows:

## 1 Slide Presentation

Normally inserted in another deck, the 1 Slide presentation is a high-level overview of the solution and its key problem it solves.

## 3 Slide Pitch

This is a simple 3 slide pitch giving an overview of the solution, discussing key benefits and outlining pricing

## 15 Slide Pitch

This is a short 30-minute presentation on the solution.

## 30 Slide Pitch

This is a 45 + Questions = 1hr deep session on the solution. [Home](..\README.md) | [Why](why.md) | [Pillars](pillars.md) | [Systems](systems.md)

# Proposal Template

Customer Ready Document

The proposal template is a detailed proposal document for a client. It will commonly discuss the solution, implementation and pricing. Microsoft have proposal templates [here](https://templates.office.com/en-au/services-proposal-business-blue-design-tm02911896).

This helps a larger client decide on your solution, understand how it will be implemented into their environment and how they will pay for it.

Even with off-the-shelf software, it is important to create a proposal document as it assists you in understanding the adoption of your technology from a customer perspective and ensures you think about and implement the key elements which are:

## Solution Description

Describe the solution in terms the customer uses when choosing the solution.

## Implementation Plan

Describe how this technology will be implemented in the customer environment.

## Migration Plan

If you are moving from legacy technology, describe how this process will occur.

## Payment Plan

Describe how the customer will purchase the software and when they start paying.

## Change Management Plan

Describe how staff will adopt the new technology bd be trained to be effective and proficient in its use.

## Risk Management Plan

Describe any risks the customer may have implementing the technology and how you think they can be mitigated. [Home](..\README.md) | [Why](why.md) | [Pillars](pillars.md) | [Systems](systems.md)

# Brochure

Customer Facing Document

The brochure is a print-ready customer-ready graphical document detailing the core aspects of the product or solution. It should discuss the following:

## Customer Need

Discuss the customer need that this solution solves, for example

* The XYZ CRM helps organizations keep track of their customers, relationships between customers, and the history of the business done between the organization and its customer.
* The ABC Utility helps IT support engineers quickly understand where disk space is being used, saving time in identifying issues with capacity.

## The Solution

Care should be taken to outline the solution in a way prospective customers can understand. Ensure you include some screen shots showing the product solving its biggest pain points.

## Benefits

Outline what benefits your customer will get from using the solution, what they can expect to achieve and in a way attempt to start creating a business case.

## Next Steps

Commonly called a call to action this helps the customer understand the next steps to take when they are interested.

Microsoft Include a [Sample Brochure Template](https://templates.office.com/en-au/software-brochure-tm00591095) with Office 365. [Home](..\README.md) | [Why](why.md) | [Pillars](pillars.md) | [Systems](systems.md)

# Datasheet

This is a customer facing document.

The datasheet describes in detail the features, functions, pre-requisites and requirements of the solution.

Microsoft have a [DataSheet Template](https://templates.office.com/en-au/technology-business-datasheet-two-sided-tm16402914) for use with Microsoft Office 365.

The datasheet is primarily aimed at a technical decision maker audience and contains the technology specifics required for technology staff to assist in the decision for purchase of the software.

The datasheet should include:

## Features and Benefits A list of features and benefits, sometimes comparing this and other editions or previous versions.

## Requirements The technology requirements to deliver the solution, for example a Microsoft Exchange Add-on may require Office 365.

## Technical Support The datasheet should contain information on the support options available for the customer.

## Availability Specifically for SaaS applications, the datasheet needs to contain information on availability guarantees and Service Level Agreements.

[Home](..\README.md) | [Why](why.md) | [Pillars](pillars.md) | [Systems](systems.md)

# Sales Battlecard

Internal Document

The Sales Battlecard is used by sales and marketing staff to quickly understand how to simply articulate a solution to customers and prospects. it should include:

* **Simple pricing matrix**.
* **Conversation Starters** - How to start a conversation about the topic
* **Common Objections** - common customer objections and how to handle them
* **Top Benefits** - Top benefits of the solution
* Very **high-level overview** of the solution.

[Home](README.md) | [Why](why.md) | [Pillars](pillars.md) | [Systems](systems.md) | [Team](team-model.md)

# Startup Checklist

The purpose of this checklist is to provide a list of items to verify if you want to ensure that your startup is operating with the best possible chance of success.

*It should be noted that, although this is a checklist that covers development items, completing this checklist does not ensure that your startup will be successful*

## Team

For technology startups, the team model is incredibly important. Make sure you have filled the following roles:

* ☐ Architect
* ☐ Developer(s)
* ☐ Tester(s)
* ☐ Build and Release Automation Designer
* ☐ User Experience Developer(s)
* ☐ Designer(s)

## Process

Ensure the following processes are documented and are being followed:

* ☐ Daily Triage Process
* ☐ Issue Management Process
* ☐ Task life-cycle process
* ☐ Automated Build Process
* ☐ Automated Release Process
* ☐ Peer Review Process
* ☐ Quality Bar Process
* ☐ Static Code Analysis Process
* ☐ Code Coverage Process
* ☐ Documentation Process
* ☐ Sprint Planning and Execution

## Technology

Ensure you have the following technology systems in place:

* ☐ Issue Management System
* ☐ Task and Work item Planning System
* ☐ Software Version Control
* ☐ Development Analytics System
* ☐ Artifact Management System
* ☐ Automated Build System
* ☐ Automated Test System
* ☐ Automated Release System
* ☐ Documentation System

## Minimum Viable Team

Ensure you have the following team members:

* ☐ **Hustler** - This is the business person who will not only manage the business, but manages sales, marketing, social presence and mode.
* ☐ **Hacker** - This is the technology leader with a good deal of knowledge of building large, scalable commercial quality systems.
* ☐ **Hipster** - This is a user experience expert, responsible for design, user flows, user education and more.

## Documentation

Successful projects are quite obvious through their use of documentation. A well documented system and business is far more likely to be successful.

* ☐ Code Commenting
* ☐ API Documentation
* ☐ Design and Purpose Documentation
* ☐ Risk and Issues Documentation
* ☐ Sprint and Project Planning Documentation
* ☐ End User Documentation
* ☐ SDK Documentation [Home](README.md) | [Why](why.md) | [Pillars](pillars.md) | [Systems](systems.md) | [Team](team-model.md)

# Corporate Setup

The following checklist details the items that need to be performed when establishing your corporate presence. It assumes you are establishing a single company (LLC or PTY) and that is will trade and hold all intellectual property. It is recommended that you *seek advice* prior to commencing with this checklist as all circumstances differ and this can only offer advice which is general in nature.

## Checklist

* ☐ Choose a company name
* ☐ Register your new company
* ☐ Open Bank Accounts
* ☐ Register for Tax
* ☐ Purchase and implement cloud accounting software
* ☐ Obtain a Domain Name
* ☐ Create a website with information and details on your new company
* ☐ Create Email Addresses and acquire a cloud commercial email platform
* ☐ Create a GitHub Repo
* ☐ Create required cloud accounts
* ☐ Decide Office Location
* ☐ Create basic Sales Kit
* ☐ Create Product Roadmap
* ☐ Start Build
* ☐ Start Selling [Home](README.md) | [Why](why.md) | [Pillars](pillars.md) | [Systems](systems.md) | [**Team**](team-model.md)

# Team Model

The Team model consists of a number of roles and functions. However there is a minimum viable team required for any startup. In our experience, any team with less than the below roles is unlikely to succeed. *Thanks to Angela Bee Chan, from* [*Hackathons Australia*](https://www.hackathonsaustralia.com/) *for the inspiration for this one*.

At a high level The ‘hacker’ is someone who can code, the ‘hustler’ brings the concept together, whilst the ‘hipster’ is the designer.

# Minimum Viable Team

## Hustler

Commonly the CEO or “business person” in a startup, the Hustler is responsible for the overall idea, the marketing and business strategy, sales, growth and how the business spends and generates money.

## Hacker

The Hacker needs to have the skills to build the product that the startup will sell. Whilst having a single hacker is superb for prototype, you will need a Tester to get into production.

## Hipster

The Hipster has the design skills in the team. They should be comfortable doing everything from designing user interface, plotting user experience to building websites and cute emails. They should be able to generate graphics, logos and more.

# Release Team

The Minimum viable team will only get you to prototype (which is a way to seek funding). If you want to actually release a product to market, you will need a few more roles filled.

## Accountant

Good accountants are worth their weight in gold. You will need accounts, cards, books, ledgers, invoicing, billing systems and more.

## Lawyer

You will need terms and conditions, NDAs, Privacy Policies, contracts and many more documents.

## Tester

The software your build will have bugs. It will always have bugs. The role of the tester is to make it have less bugs. There is no such thing as no bugs. Once you are cool with that, hire a tester who can also code and you’ll have automated test and releases all day long.

# Scale Team

Once you’re in market, you will need to scale. The following roles are your scale roles:

## Support Team

These folks speak to customers with problems and help them.

## Developers

These folks write more and more code.

## Desginers

These folks make more and more pretty stuff

## Testers

Still trying to get rid of bugs

## Admin

You know you’ve made it when you need Accounts people, sales people and HR People. If you need these before profitability you are probably doing something wrong. [Home](README.md) | [Why](why.md) | [Pillars](pillars.md) | [Systems](systems.md) | [Team](team-model.md) # Useful Resources

This section details useful resources, tools and technologies. [Home](README.md) | [Why](why.md) | [Pillars](pillars.md) | [Systems](systems.md) | [Team](team-model.md) # Glossary

## Burndown

The burndown is a chart that shows how quickly you and your team are burning through your customer’s user stories. It shows the total effort against the amount of work we deliver each iteration

## User Story

## Iteration

[Home](README.md) | [Why](why.md) | [Pillars](pillars.md) | [Systems](systems.md) | [Team](team-model.md) # CRMMe.io

## Table of Contents

* [CRMMe.io](#crmmeio)
  + [Table of Contents](#table-of-contents)
  + [Introduction](#introduction)
  + [Background](#background)
  + [Original Idea](#original-idea)
  + [Concept Development](#concept-development)
  + [Following the Pillars](#following-the-pillars)
  + [The development Process](#the-development-process)
    - [Decision One: Platforms](#decision-one-platforms)
    - [Setting Up the Environment](#setting-up-the-environment)
  + [Authentication](#authentication)
  + [Accessing the Office Graph](#accessing-the-office-graph)
  + [Release Process](#release-process)
  + [Progress](#progress)
  + [Sales & Marketing](#sales--marketing)
    - [Video](#video)
    - [WebSite](#website)
    - [Marketing Tools Budget](#marketing-tools-budget)

## Introduction

This document details a case study on the project CRMM.io. Built using the Released Method, CRMme.io is a real world example of the Released Method and how it can work in real life.

## Background

CRMme was envisioned on the 5 June 2020 at the Hornsby Railway Hotel north of Sydney. During Lunch between [Nick Beaugeard](https://www.linkedin.com/in/nickbeaugeard/), [Luke Butler](https://www.linkedin.com/in/lubutler/) and [Bill Barden](https://www.linkedin.com/in/bill-barden-1b771314/), Bill was complaining about how he had thousands of contacts, but commonly they were locked up in email messages and calendar appointments and never really made it into his contacts list.

## Original Idea

The original idea of Bill’s can be summed up in the following sentence:

“Create a system, that will real emails and calendar messages and create a list of contacts and alow these to be exported to a spreadsheet”

## Concept Development

As lunch continued, the concept continued to evolve and after leaving, they settled on a similar solution, maybe using the Office graph.

## Following the Pillars

As this was a Released project, we decided to follow the Released Method:

* [Management Reporting](management-reporting.md) - We decided to use [GitHub](https://www.github.com) to host the management reporting platform. It is a super rich platform and we have a number of tools which work with it.
  + First is [PowerBI for Github](https://app.powerbi.com/groups/me/getapps/services/pbi-contentpacks.pbiapps-github) provides a ton of useful dashboards and reports based on data in Github.
  + Second is our [Project Documentor](https://github.com/nickbeau/project-documentor) which reports the current status of the project and includes any additional data for management reporting.
* [Software Planning and Development](software-planning.md) - Key to software planning is the KanBan and we used [GitHub Projects](https://github.com/features/project-management/) with the released method extensions to manage the project.
* [Testing and Quality Assurance](testing-qa.md)
* [Release Management](release-management.md) - For Release Management we used [GitHub Actions](https://github.com/features/actions) to automatically build, test and release the software when we attempted to release it into production .
* [Documentation](documentation.md)- Documentation is written in markdown and stored in the documentation folder
* [Selling](sales-kit.md) - We decided on a distinctly freemium model with a simple website and video advert to start, with a facebook and email campaign to get our first beta testers.

## The development Process

The first 7 days of the the project were involved in getting a project up and running. We decided to use Microsoft’s Blazor framework with its authentication model to the Microsoft Graph.

### Decision One: Platforms

For the Minimum Viable Product we decided to only support Office 365.

### Setting Up the Environment

We decided to implement a new GitHub Organisation and looked for a domain name. We found CRMme.io and decided that was a super name and we would use that. We setup github.com/crmme and created two repositories;

* App would hold the application and its source code and issues
* crmme.github.io would host the application once it was built from the MAster Branch.

We created two Milestones under App \* Beta 1 \* Beta 2

We created a new project under App \* MVP Release

We started coding.

## Authentication

The very first hurdle was to create a client-side blazor webassembly application that would successfully authenticate with Office 365. Blazor is new, and the documentation is still growing. However the process documented at the [Microsoft Documentation](https://docs.microsoft.com/en-us/aspnet/core/security/blazor/webassembly/standalone-with-azure-active-directory?view=aspnetcore-3.1) seemed to work and after some testing, we had a Blazor Application that would authenitcate to Office 365.

## Accessing the Office Graph

Not so well documented was how to access the Office Graph. To do this we needed to add some code to our Blazor Pages:

At the top of the .razor file

@using Microsoft.AspNetCore.Components.WebAssembly.Authentication  
@inject IAccessTokenProvider TokenProvider  
@inject SignOutSessionStateManager SignOutManager  
@inject AuthenticationStateProvider AuthenticationStateProvider

And in the code

tokenResult = await TokenProvider.RequestAccessToken(  
 new AccessTokenRequestOptions  
 {  
 Scopes = new[] { "https://graph.microsoft.com/Calendar.Read"   
 }});  
  
  
if (tokenResult.TryGetToken(out var token))  
{  
 //DO WORK HERE  
}

Once we had that model worked out, we could begin working with the Microsoft Graph.

## Release Process

We wanted the release process to be as simple as possible, so the following .yaml file drives github actions to compile, build and test our software and then release it.

name: .NET Core  
  
on:  
 push:  
 branches: [ master ]  
 pull\_request:  
 branches: [ master ]  
  
jobs:  
 build:  
  
 runs-on: ubuntu-latest  
  
 steps:  
 - uses: actions/checkout@v2  
 - name: Setup .NET Core  
 uses: actions/setup-dotnet@v1  
 with:  
 dotnet-version: 3.1.300  
 - name: Install dependencies now  
 run: dotnet restore  
 - name: Build  
 run: dotnet build --configuration Release --no-restore  
 - name: Publish  
 run: dotnet publish -c Release  
 - name: Test  
 run: dotnet test --no-restore --verbosity normal  
 - name: Publish artifacts  
 uses: actions/upload-artifact@v2  
 with:  
 name: blazorapp  
 path: App/bin/Release/netstandard2.1/publish/wwwroot  
 deploy:  
 needs: [build]  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout  
 uses: actions/checkout@v2  
 with:  
 persist-credentials: false  
   
 - name : download artifacts  
 uses: actions/download-artifact@v1  
 with:  
 name: blazorapp  
 - name: Deploy  
 uses: JamesIves/github-pages-deploy-action@releases/v3  
 with:  
 GITHUB\_TOKEN: ${{ secrets.GITHUB\_TOKEN }}  
 BRANCH: master  
 ACCESS\_TOKEN: ${{ secrets.PERSONAL\_KEY }}  
 FOLDER: "blazorapp"  
 REPOSITORY\_NAME: crmme/crmme.github.io  
 TARGET\_FOLDER: /

## Progress

Now we have an application that can talk to the Office Graph, can be published and tested at will and is fully automated.

Next step is to invite people to join a beta program

## Sales & Marketing

Selling and Marketing a very early stage startup is very different to selling and marketing a well established product. We like to use stealth marketing techniques and not be too salesey. In fact we need customer feedback to make our product work, so we need to be able to put it all together.

### Video

Seems that today, you need video. We put one together in about an hour using:

* [Camtasia](https://www.techsmith.com/video-editor.html) - Always my go-to for video editing, Camtasia is quick and I know it really well. It’s also amazing for putting together product demos and presentations. I’ve used it for about 20 years.
* [Assets for Camtasia](https://library.techsmith.com/camtasia) - The Camtasia Asset library is an amazing and cost effective place to get video, audio and photo assets for Camtasia.
* [Speechelo](https://speechelo.com/?hop=myketodiit) - While the website seems quite scammy, the product works well.

You can judge for yourself:

### WebSite

We needed a website, quick and simple. [Wix](www.wix.com) came to the rescue. This platform has so many features, it can handle our Beta Program, mailins and more.

### Marketing Tools Budget

|  |  |
| --- | --- |
| Item | Cost |
| Camtasia | $420.98 |
| Assets for Camtasia | $352.43 |
| Speechelo Pro | $379.01 |
| Wix Website | $178.29 |
| Domain (GoDaddy) | $183.43 |
| **TOTAL** | **$1,514.14** |