Blackburn Labs

Software & Mobile Applications Starter Kit

Rev 1.1.58

Contents

Welcome to the Starter Kit

The purpose of this document is to provide basic information on some of the key concepts around developing a modern application or software product.

Please keep in mind this is not intended to be a comprehensive explanation of these topics. There is a great deal of in-depth information, and this article doesn't cover all of it. However, the hope is to give someone new to these concepts enough information to get them started and to begin successfully interacting with professionals in these fields.

We hope you find this helpful,

The Blackburn Labs Team

Common Project Roles

Project Role Reference

Core Roles

User
This is any and all users of the software. When evaluating who your users are, don't forget to include both internal (like administrators or support staff) as well as external users (like customers).

Super-User
This is a subset of trusted and experienced users. These users should be prepared to make a significant time commitment to the project, and they should be ready to act as representatives of all the users. They will be involved in many steps of the project, especially the User Acceptance Test (UAT), which we will discuss later in this document. The success of a project often hinges on having knowledgeable and engaged super-users.

Programmer
These are the professionals who will be writing the actual code of your app. Having experienced and innovative programmers is critical to any software project.

Designer

There are many types of designers, such as 3D graphic artists, UI designers, and web designers. Unfortunately, many underestimate the importance of having the team's designers engaged with the programmers and with other members of the team on a daily basis. This can be a huge mistake, leading to software that not only looks amateurish but does not function or flow for the users as expected.

QA Engineer/Tester

These people test the software and report bugs and issues to the team. Sometimes these are dedicated professionals known as quality assurance engineers. Other projects might use other team members as their testers. This depends on the process the team decides to use. We will discuss this in more detail later in this document.

Scrum Master/Agile Coach

An Agile Coach or Scrum Master is a facilitator who helps teams adopt and improve Agile methodologies by guiding them through the processes, practices, and principles of Agile development. We will cover the Agile methodology in more detail later in this document.

Responsibility Assignment Matrix (RAM)

Who is responsible for, or contributes to, each phase or step of the project? Clearly defining roles and responsibilities early in a project leads to success. Try putting together a Responsibility Assignment Matrix (RAM) and getting it signed by all the stakeholders. This may save your project later.

Ancillary Roles

Project Sponsor
This is the person or organization who is funding the project. Therefore, it’s obviously essential to have them on board with any changes to the project and to keep them informed on progress.

Architect
Most organizations have architects who are responsible for ensuring the consistent use of technology and the adherence to best practices between the organization’s various applications. This person may or may not be a part of the project team.

UX Engineer
UX (user experience) engineers will work to insure the usability of the software. They achieve this through many different techniques. We will describe the UX process in more detail later in this document. The UX engineer may be a part of your project or a shared resource.

Business Analyst (BA)
The BA will analyse the business’s processes, documenting and defining them for the team. This is especially important when replacing an existing system or when augmenting a human-executed process.

Ancillary Roles

Project Manager (PM)

A PM is responsible for keeping the project on track and making sure the project team and stakeholders are well informed on the progress and status of the project.

Auditor

Depending on the project and organization, you may also have auditors. These are the people who will validate the software is compliant with any legal or organizational requirements, such as Sarbanes Oxley laws or government accessibility requirements.

Subject Matter Expert (SME)

It is important to identify any SMEs related to your project. These are the people who are experts on the topic(s) the software is augmenting or affecting.

Change Control Board (CCB)

For larger projects, it is often a good idea to form a Change Control Board. This group is in charge of deciding and approving major changes to the project. If you are following Agile/Scrum, this can be simplified to the product owner.

Scrum, Agile, and SDLCs

Contributor

Doug Hopkins

Agile Coach

Software Development Life-Cycle (SDLC)

The two most common software development processes:

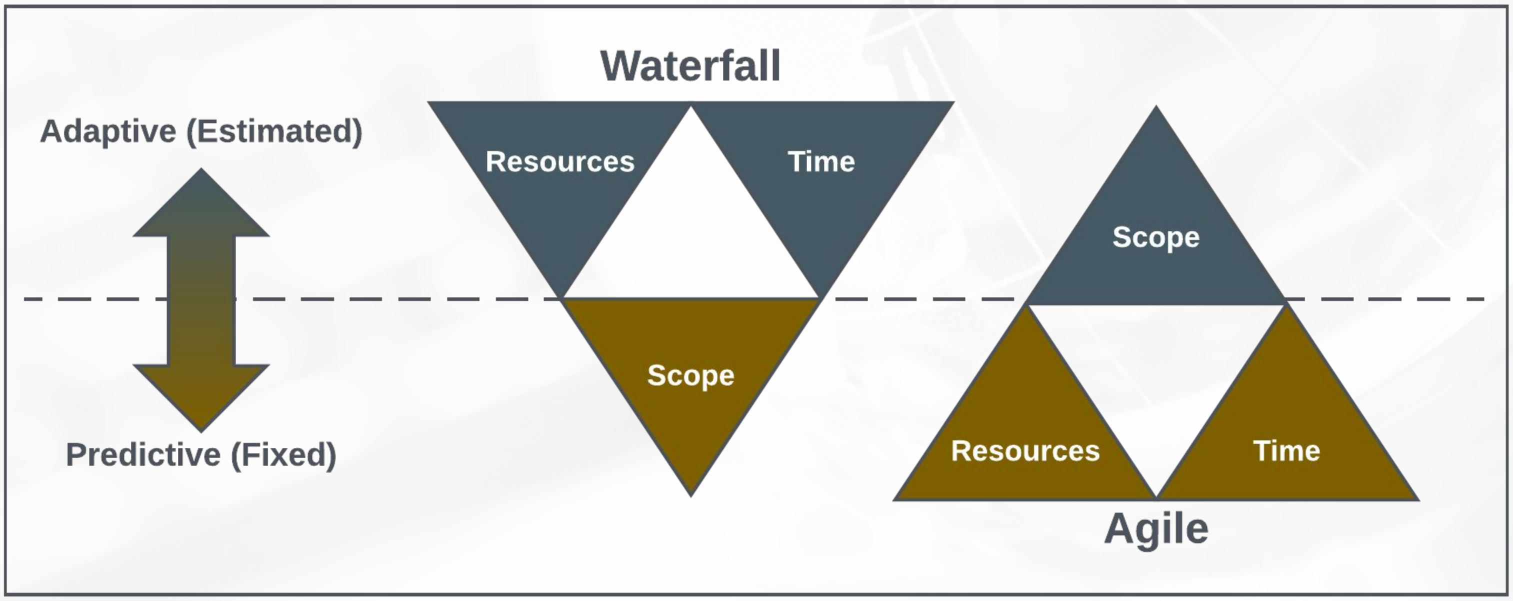
Waterfall: A legacy process, though it often seems to fit better into other non-software project processes. In this process, the team gathers all requirements up front, before beginning any development. This causes a great deal of rework when requirements evolve throughout the development process and can hinder many development best practices.

Iterative (Agile): Adopted as far back as the 70s, Agile is the process which is now generally considered the industry best practice for software development. This process takes a far more iterative approach to software development.

Software Development Life-Cycle (SDLC)

Iterative vs. Waterfall (Adaptive vs. Predictive)

Unlike Waterfall development, Agile projects have a fixed schedule and resources while the scope varies. Teams deliver high-quality, tested increments of work, meeting its own definition of “done” and progress towards its highest priority goals.



Agile Vision

“We are uncovering better ways of developing software by doing it and helping others do it.”

www.agilemanifesto.org

Organization

High-performing teams are passionate, self-organizing, cross-functional, and guided by the values identified in the Manifesto for Agile Software Development.

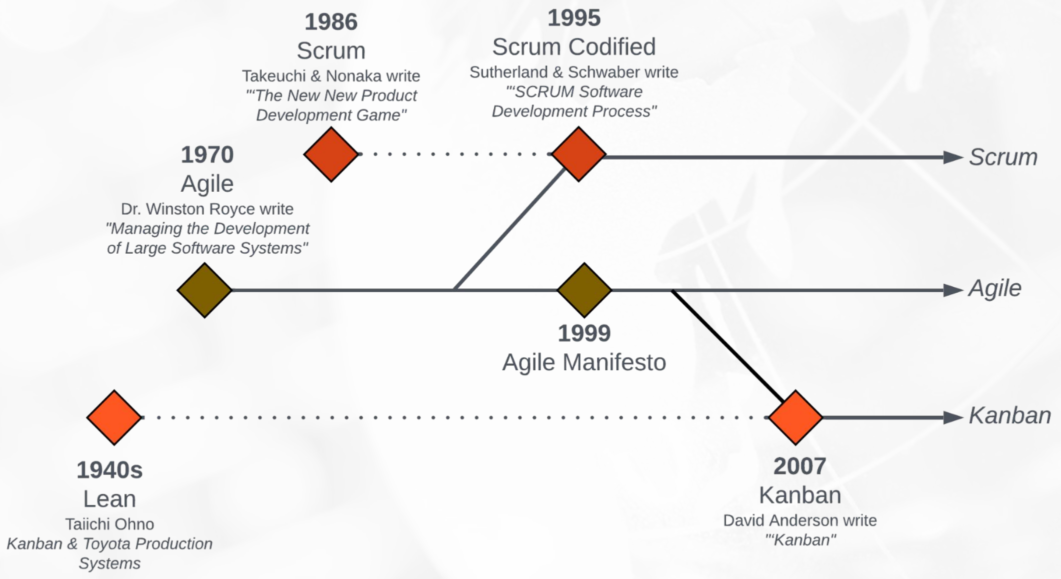
• **Individuals and interactions**	over processes and tools
• **Working software**	over comprehensive documentation
• **Customer collaboration**	over contract negotiation
• **Responding to change**	over following a plan

While there is value in the items on the right, we value the items on the left more.

Agile Frameworks & Tools

Agile has been around for a long time and has its roots in the Lean methodology. Agile is a methodology in itself, but there are also different variations to this, including the very popular Scrum and Kanban methodologies.

Though a vast oversimplification, this is a brief history of these key methods:


1986
Scrum
Takeuchi & Nonaka write
"'The New New Product Development Game"
1995
Scrum Codified
Sutherland & Schwaber write
"'SCRUM Software Development Process"
Scrum
1970
Agile
Dr. Winston Royce write
"Managing the Development of Large Software Systems"
Agile
1999
Agile Manifesto
Kanban
1940s
Lean
Taiichi Ohno
Kanban & Toyota Production Systems
2007
Kanban
David Anderson write
"'Kanban"

Scrum: Scrum is an Agile software development framework that was inspired by "The New New Product Development Game," a 1986 Harvard Business Review article by Takeuchi and Nonaka that described a flexible, holistic approach to product development. The authors used the game of rugby as a metaphor for the process, which involves small, cross-functional teams working together in short iterations or sprints to deliver a potentially shippable product. Jeff Sutherland and Ken Schwaber, who created Scrum in the early 1990s, drew on these principles to develop a framework that has since become widely used in software development and beyond.
Kanban: Kanban is a visual framework for managing software development that emphasizes the flow of work and limits work in progress. Its origins are in the Toyota Production System, where it was used in manufacturing to improve efficiency and reduce waste. In software development, Kanban helps teams manage their workflow by visualizing the status of work items and ensuring that work is delivered continuously and on demand.

Development & Testing

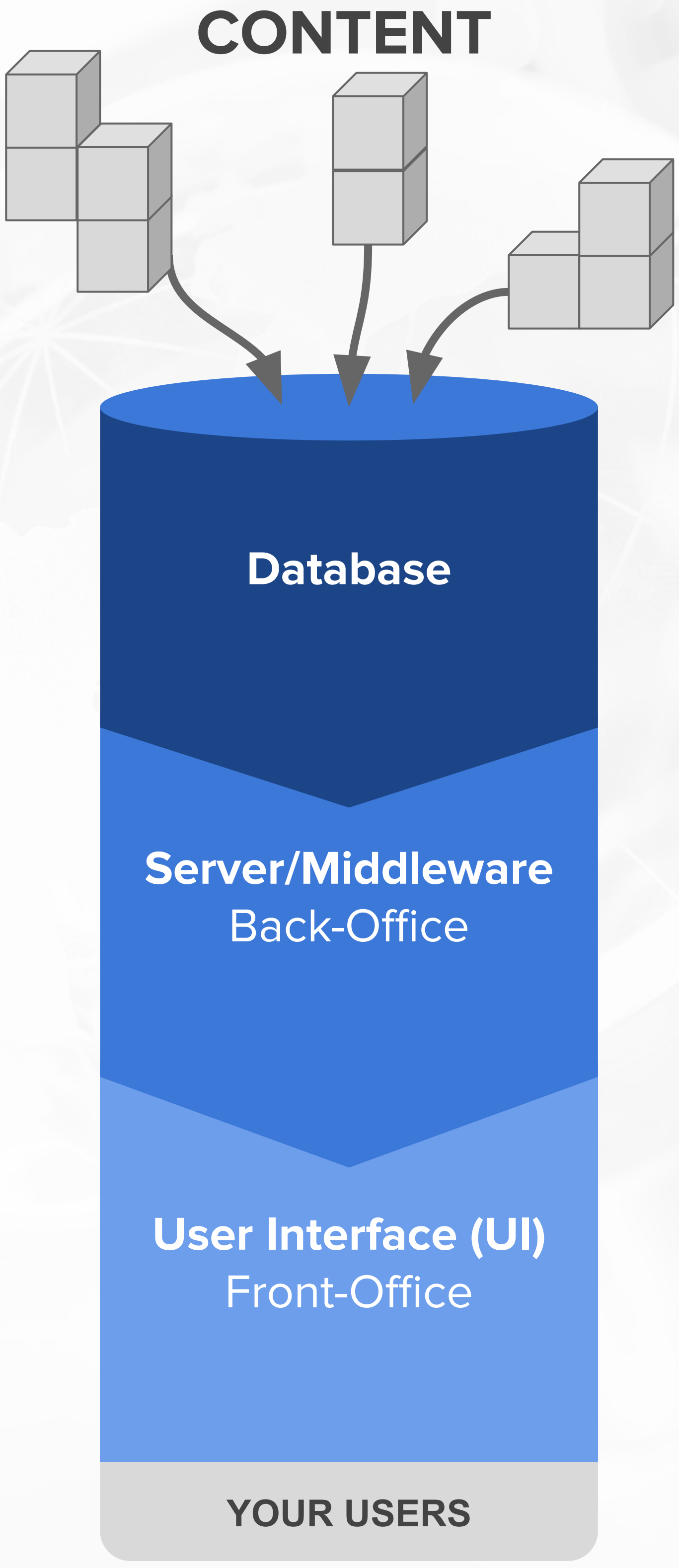
```
</script>
<!-- start header -->
<div class="header_bg">
<div class="wrap">
    <div id="content">
      <header id="topnav">
        <nav>
            <ul>
                <li class="active"><a class="scroll" href="#home">
                    Home                     </a></li>
                <li><a class="scroll" href="#service">
                    Service                  </a></li>
                <li><a class="scroll" href="#product">
                    Products                 </a></li>
                <li><a class="scroll" href="#portfolio">
                    Portfolio                </a></li>
                <li><a class="scroll" href="#team">
                    Team                     </a></li>
                <li><a class="scroll" href="#contact">
                    Contact                  </a></li>
                <div class="clear"></div>
            </ul>
        </nav>
        <div class="logo">
            <a href="./">
                <img src="web/images/logo3.png" id="logo_large" width="300">
                <img src="web/images/logo.png" id="logo_small">
            </a>
        </div>

        <a href="#" id="navbtn">Nav Menu</a>
        <div class="clear"> </div>
      </header><!-- @end #topnav -->
      <script type="text/javascript"  src="web/js/menu.js"></script>
    </div>
</div>
</div>
<!--start-slider---->
<div class="slider" id="home">
```

Technology Stack

When it comes to client software and mobile applications, technology can be grouped into three layers:

1. Database
2. Server/Middleware
3. User Interface



Database

The Database stores the content of your software.

There are many kind of databases, but three you are most likely to run into with typical applications are:

1. **Transactional Database**
   This is usually in the form of a relational database. Common technologies used here include MySQL, Oracle, Sybase, and Microsoft SQL Server.
2. **Data Warehouse**
   This stores data for larger queries, such as reporting. This can also be in the form of relational database, as seen above, but most recently there have been many "big data" databases gaining usage. These can be especially helpful in applications that require extremely large sets of data. Common technologies here include Hadoop, SAP HANA, and Oracle in-memory DB, among many other emerging technologies.
3. **Cache Database**
   Sometimes even the most highly-tuned transactional DB cannot keep up with very high volumes, or attempting to do so would create a burden on your servers. In these cases, an app may need to leverage a caching DB, often referred to as NoSQL databases. There are hundreds of options here depending on your needs and what is being stored. However, one of the most common technologies used here is Redis. If done properly, something like Hadoop can be used for this as well.

Back-Office

The back-office, or server, delivers up all the content along with handling other business functions.

Depending on your application's structure, the back-office may handle many functions. Some of the most common servers that do this are Tomcat, PHP, Node.JS, and .Net. Some of the most common functions include the following:

1. **Deliver UI**
   For a mobile or web application, the server is responsible for delivering the UI itself. Often, this takes the form of HTML5 but can also be done using other UI technologies. There are many servers, some of the most common are Apache and IIS.
2. **Deliver Content**
   Your application will need to display content in the UI. For example, an email client will have a UI the user interacts with, and within that UI is content (the emails). This content is typically delivered to the UI using Web Services. Web Services come in two flavors, REST and SOAP. The Web Service can deliver the content in many formats, such as JSON, HTML, or XML.
3. **Handle Security**
   Most applications limit user access in some way. The server typically handles user authentication and authorization.
4. **Offline Processes**
   Some applications also need to handle offline processes, like parsing and migrating files, ETL (extract, transform, and loading data) processes, data compression, or any other process a user will not want to sit and wait for.

Front-Office

The front-office, or user interface (UI), is the piece of software the user interacts with.

For most applications, there are two main UI approaches:

1. **Native App**
   This means the UI was written in code and compiled specific to a client. For Android devices, this means Java. For iOS, this means Objective-C/Switch. If you create a Windows app, then C# will likely be your choice. Native apps offer tighter device integration and better performance, but porting to multiple devices may prove to be costly.

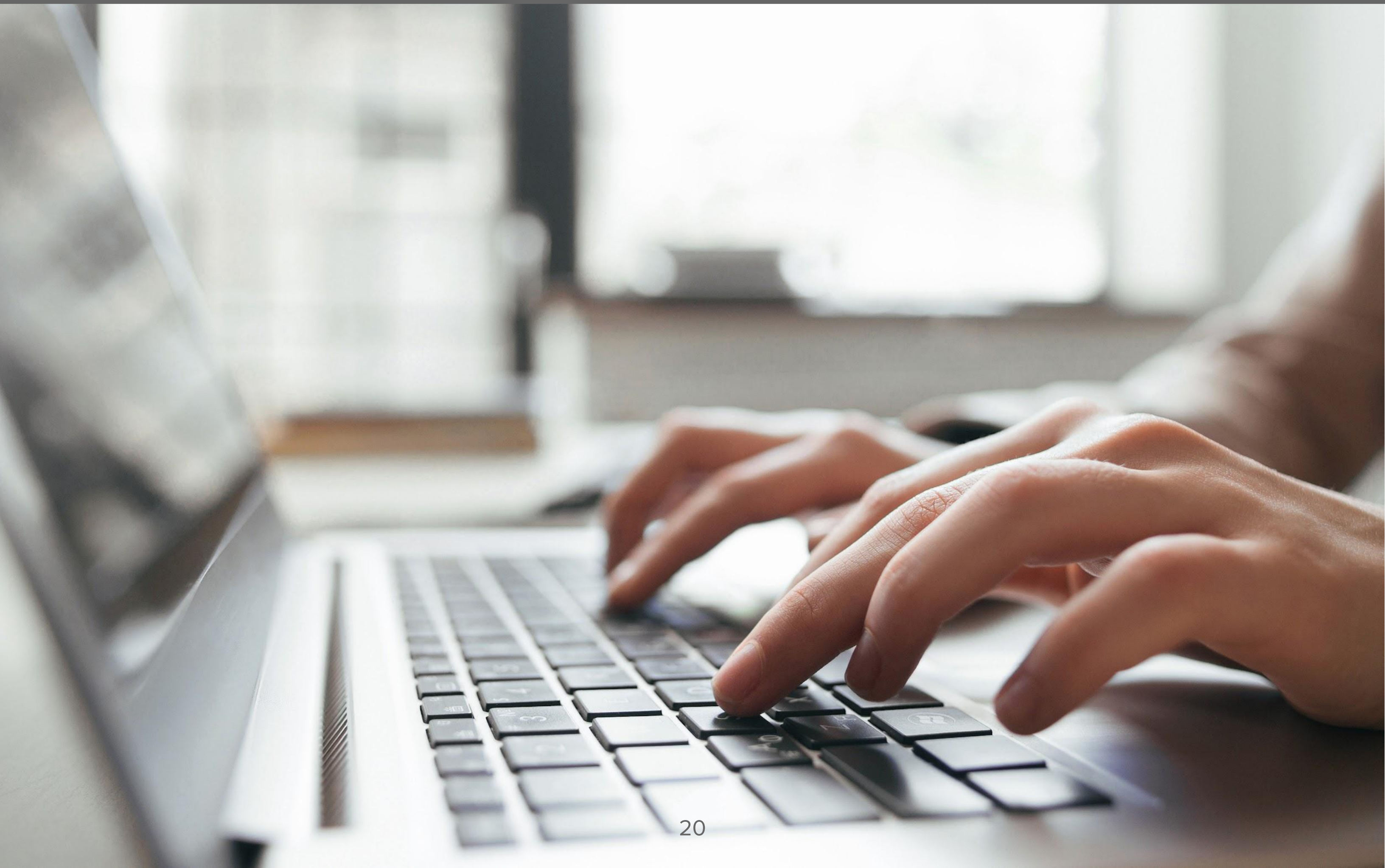
2. **Web App**
   A relatively new approach to creating applications is to use JavaScript/HTML/CSS3 and a framework such as Cordova to package the application for multiple platforms. Since the application is not "native" and therefore relies on a middleman to communicate with the device, you may have performance issues or find it difficult to interact with specific hardware features consistently. This can be a great approach for some types of applications. However, you may also find it difficult to make your app feel like an app instead of an advanced website.

There are also some technologies that allow you to program in one language but compile what is effectively a native app to achieve the best of both worlds.

Testing and QA

A key element to a successful software project is efficient and well-coordinated test processes.

Everyone in the project contributes to testing in one way or another. However, there are a few methods and types of testing we will focus on here.



Testing Methods

Dedicated Testers: A common, and slightly more traditional method of testing your software is to have dedicated testers, QA engineers. These are professional testers experienced with special automated testing software and trained on how to log efficient bug reports.

Hybrid Testers: Some development processes, such as Kanban, or for smaller teams, using hybrid testers is the better approach. In this scenario, your team members, such as your programmers or designers, also serve as the testers. This can be done as an inline part of your process or on a rotating schedule.

The method you use will depend on the team culture, project size, and development process.

Types of Testing

- **Unit Testing:** This is testing the software on its lowest level - testing each building block of the software independently. Regardless of the method of testing used, this is typically done by the programmers, as the unit tests are usually written in code and executed alongside the program's code as a part of the development process.
    - **TDD/BDD:** There are specific development techniques called Test Driven Development (TDD) and Behavior Driven Development (BDD) which unit testing plays a critical role in. Embracing and enforcing these techniques can vastly improve the stability of your software.

- **Integration Testing**: This is the testing of the software functionality, once all the code is put together and compiled. This can be as simple as opening the program and checking that it runs, or as formal as documented "test scripts" that testers follow.
    - Ideally, any bugs/issues will be logged with steps to reproduce the bug. These steps are then used after the bug is fixed to insure the issue was resolved as a part of the integration testing.
    - Each system requirement, or user story, can act as a miniature ad hoc integration test script.

Continuous Integration System

There are systems available that will automatically run your unit tests and integration tests after every code change and report results to the team. Such a valuable system is highly recommended for any software project.

Types of Testing

- **Regression Testing:** Regression testing is similar to integration testing, in that it tests the functionality. However, in this case one is testing the system as a whole, not just the elements that have been changed or fixed. In fact, the integration tests are often saved to become regression tests in the future. This is essential to ensure that changes made don't have unexpected consequences or that old bugs do not resurface. There are two types of regression testing.
  - **Manual**: Done by the QA engineers/testers.
  - **Automated**: Programed by a QA engineer, and automatically executed by the **Continuous Integration System**.

- **Compliance Testing:** This will vary, depending on your software, needs, and industry.
  - **Penetration Testing (PenTest):** This tests your system for security breaches. ISO/IEC 27000 is a common compliance standard for this, though there are many others. If needed, there is specialized software that can assist in this testing.
  - **Load Testing:** This tests that your software and hardware can sustain the anticipated load (traffic, I/O, etc.). If needed, there is specialized software, such as JMeter, that can assist in this testing.
  - **Government Compliance:** There are many government standards that may apply to your software, such as Sarbanes Oxley (SOX), Section 508, HIPAA, and many others. This is where your project auditors will come into play.

Types of Testing

- **User Acceptance Testing (UAT):** UAT is a special testing done by actual users of the system, usually the predesignated super-users, as well as key stakeholders, to confirm the delivery of the product itself. This acceptance is often an official milestone of a project and is commonly built into the project's SoW/development contract.

  This testing may take place throughout the development cycle, or it may happen at pre-designated intervals in the project. This will depend on your development process, organization culture, and user availability. However, the more frequently this is done, and the earlier in the project it is done, the better.

  For web-based software there is often a dedicated server set aside for this activity that receives software deployments at appropriate times. This helps keep your UAT team from testing against the development code, which can often be unstable.

User Interface Design & UX

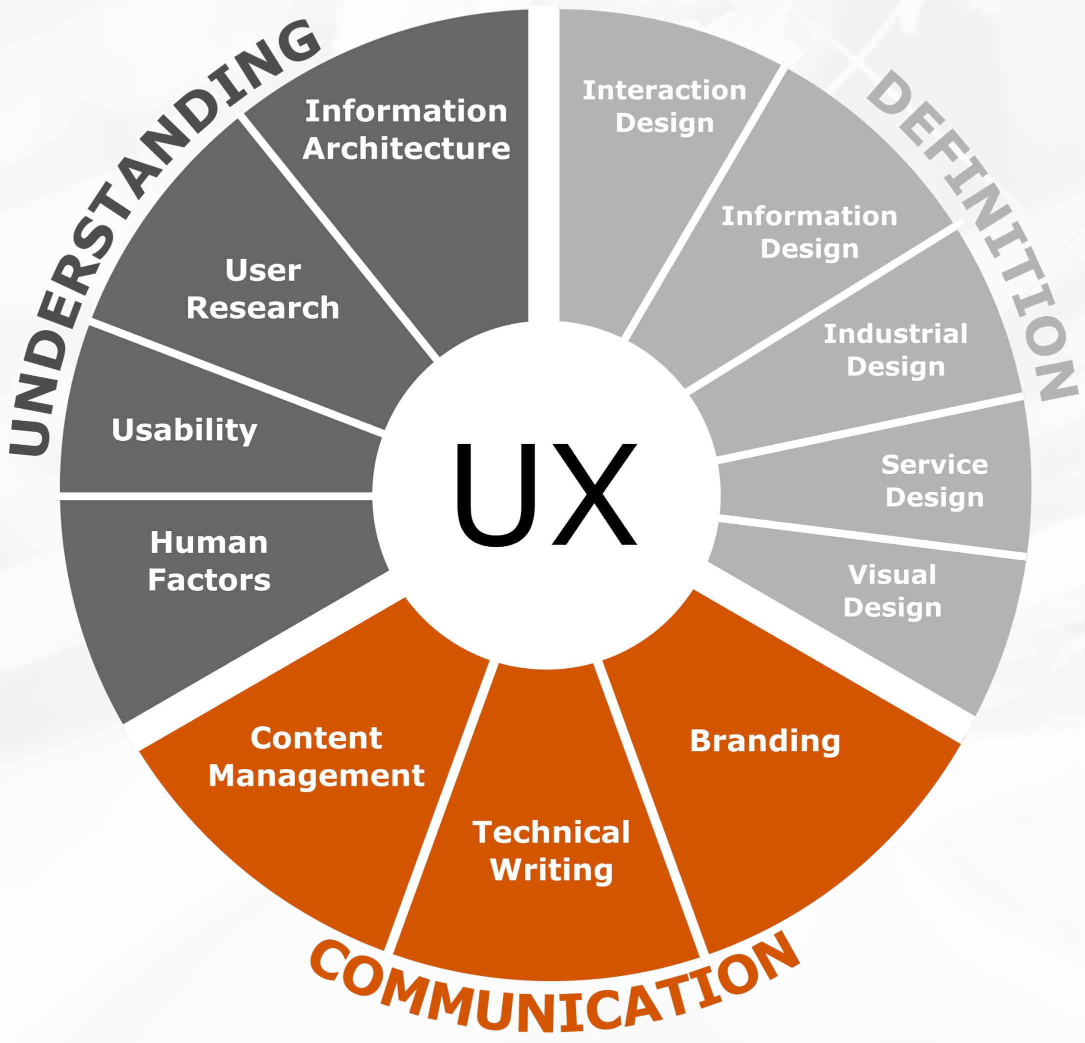
Contributor

Jason Bowden

Design & UX Leader

UX is….

Two letters that when put together create a term so nebulous that a Google search yields 142 million results packed with words like user experience design, user interface design, user interface engineering, usability engineering, front-end engineering, information architecture, human-computer interaction, content strategy, customer experience, prototyping, lean UX, user research, usability, and even service design. There are also a fairly ridiculous number of acronyms around: UX, UXD, UED, XD, UI, IA, IXD, CX, HCI, SC, SX, and UCD. I'm sure I've missed some. Anyway, here's a graph:



Let me clear some of this up for you:
UX is all of this. Good UX designers understand research, content strategy, navigation, user interface conventions, branding, copywriting, interaction design, and even visual design. It all falls under the umbrella of UX.

Don’t get me wrong, to us practitioners, these terms are nowhere near interchangeable. But there is an area of shared focus across each and every one of these disciplines: users.

A common definition of UX is …

> *“the process of enhancing user satisfaction by improving the usability, accessibility, and pleasure provided in the interaction between the user and the product”*

This is important because without knowing our users, we can’t know if we’re building the right product for them. The Apple app store has over a million apps in it, many of which never gain any traction because they fail to address their user’s problems. Why? They have no idea what their user’s problems are!

The whole equation

Now that I’ve overly-established the idea that UX is focused on users, I’ll say that in actuality, solving the user’s problem is only half of it. The other half of the equation is that businesses operate to make a profit and/or save money.

So, how do you marry these two possibly opposing goals?

The answer of course, is UX. Addressing both sides of the equation engages and delights users, which in turn increases engagement, which in turn grows profit and/or saves money. As I said, this is what all companies are in business to do. Sounds great right?

So, how do we do this?

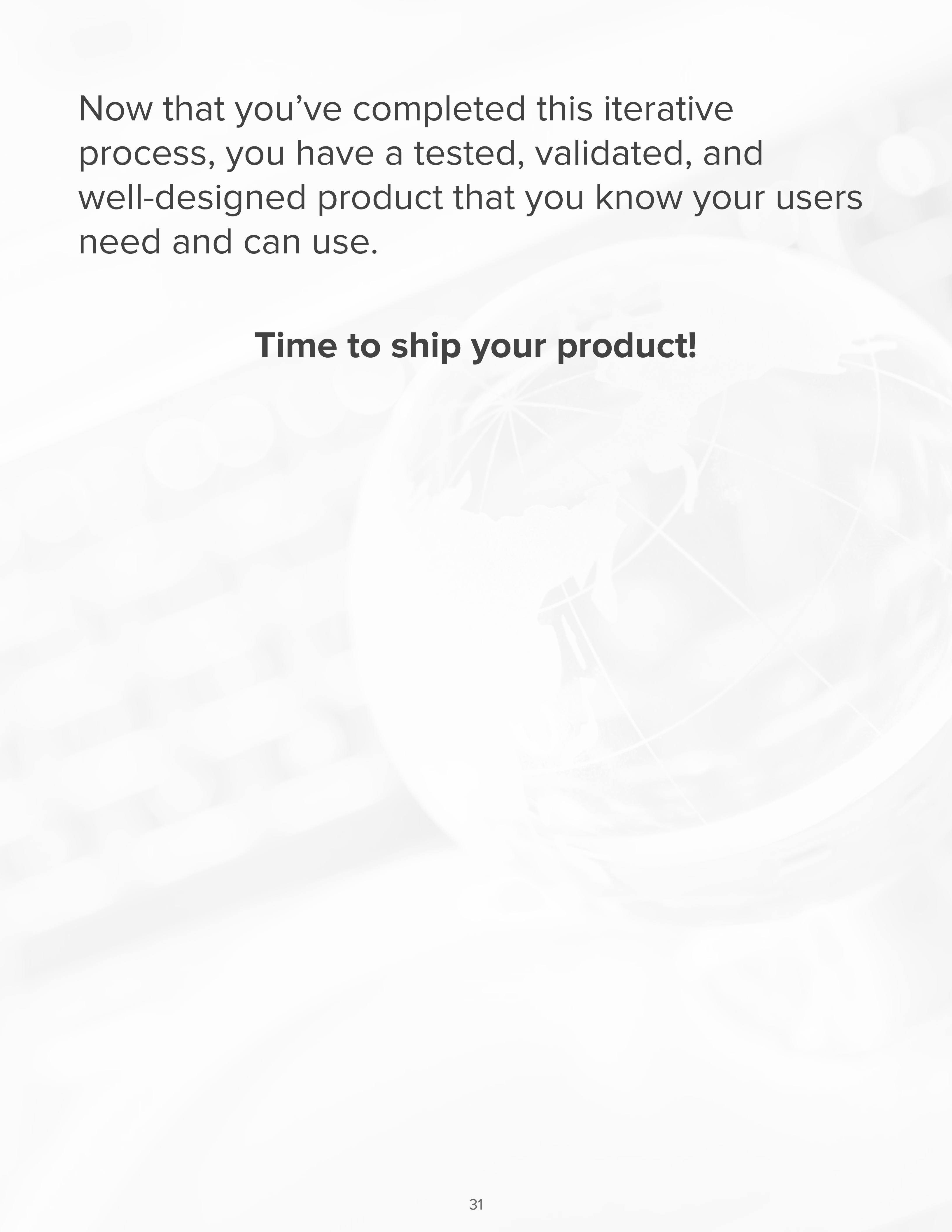
1 As I mentioned, we start with users, learning about them, learning from them, observing them and asking questions. We also learn about business objectives, existing technology decisions, budgets, and deadlines. All of this informs a strategy (the first and most important of the 5 planes of UX). As we learn about our users, we build personas — archetypal documents with demographic information, fictional names, and even representational pictures — so our development teams know exactly who we're building our product for, what their goals are, and even what they like and dislike. We also create a plan that includes metrics, to incorporate right up-front our business goals and what it will mean to achieve them.

2 Next, we can get to the root of the problem. Defining the problem is key, and our personas will inform that problem definition. Once the problem is fully understood, we can start thinking about solutions. We develop customer journeys, user flows, sketches, and wireframes. Notice we've done a bunch of work but no designs have been seen and no code has been written. Yet.

This is counter to the way many projects are worked on, but getting to design too early (before knowing your users) is the biggest pitfall to avoid! It's like building a house without knowing how many bedrooms you need. It's creates rework, expands project scope, and wastes time and money.

3 Once we get a good idea of the user's problem and we start designing solutions, it's time to validate our hypotheses. This is done through usability testing: show users your app and watch them use it. Ask them to complete tasks and track if they can do it or not. Once you are confident your app is usable, THEN you start creating polished visual design on the UI (aka user interface) — the actual screens that users will interact with.

4 During this final design (called surface in the 5 planes of UX), UX designers will use their expertise of interaction design to make sure the product is intuitive and beautiful.



Now that you've completed this iterative process, you have a tested, validated, and well-designed product that you know your users need and can use.

Time to ship your product!

Minimum Viable Product

Minimum Viable Product

If your project is creating any sort of product, be it an app or a new data warehouse, defining your minimum viable product (MVP) upfront can be the key to success.

According to Eric Ries, who popularized MVP in his book *Lean Startup*:

“The minimum viable product is that version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort.”

- Eric Ries

An MVP is a fast and safe way to vet and refine a product in contrast to conventional methods of product development, which require significant ramp-up, capital, and resource utilization.

Minimum Viable Product

In the cupcake model of a minimum viable product, we start with a smaller yet complete product. It has the appeal of a complete cake such as taste, icing, filling, etc. However, its production costs are much lower.

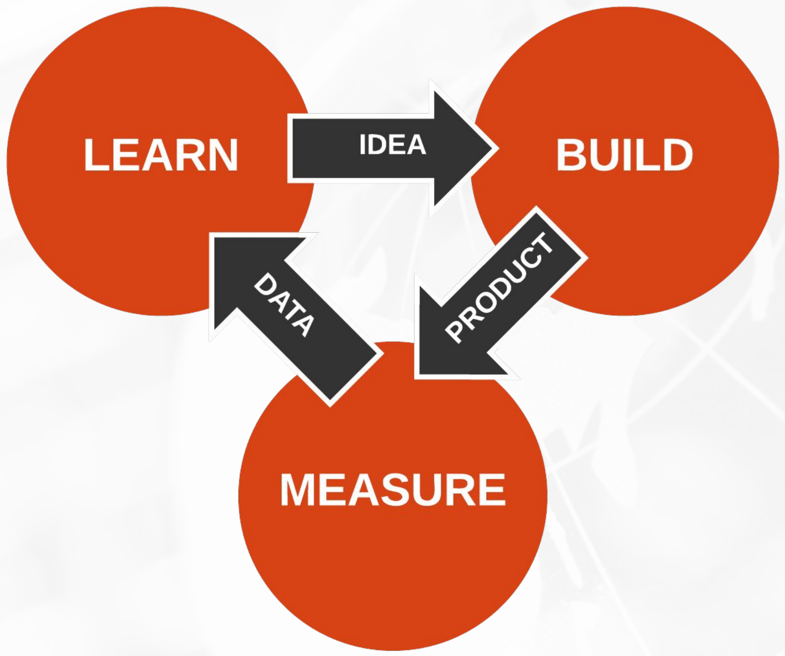
Why a Minimum Viable Product?

An MVP will be used to measure and to learn in a cost effective way:

- This will provide a feedback loop that will allow us to iterate over the product's development. This will improve it over time, as well as refine the marketing and sales messaging.
- This kind of learning, which is based on trying out an idea and validating its effect, is called validated learning.
- This is repeated until considerable insights are gained and a complete product is released.

Why MVP?

The Lean Startup Methodology encourages us to focus on releasing a product in the context of the build-measure-learn feedback loop:

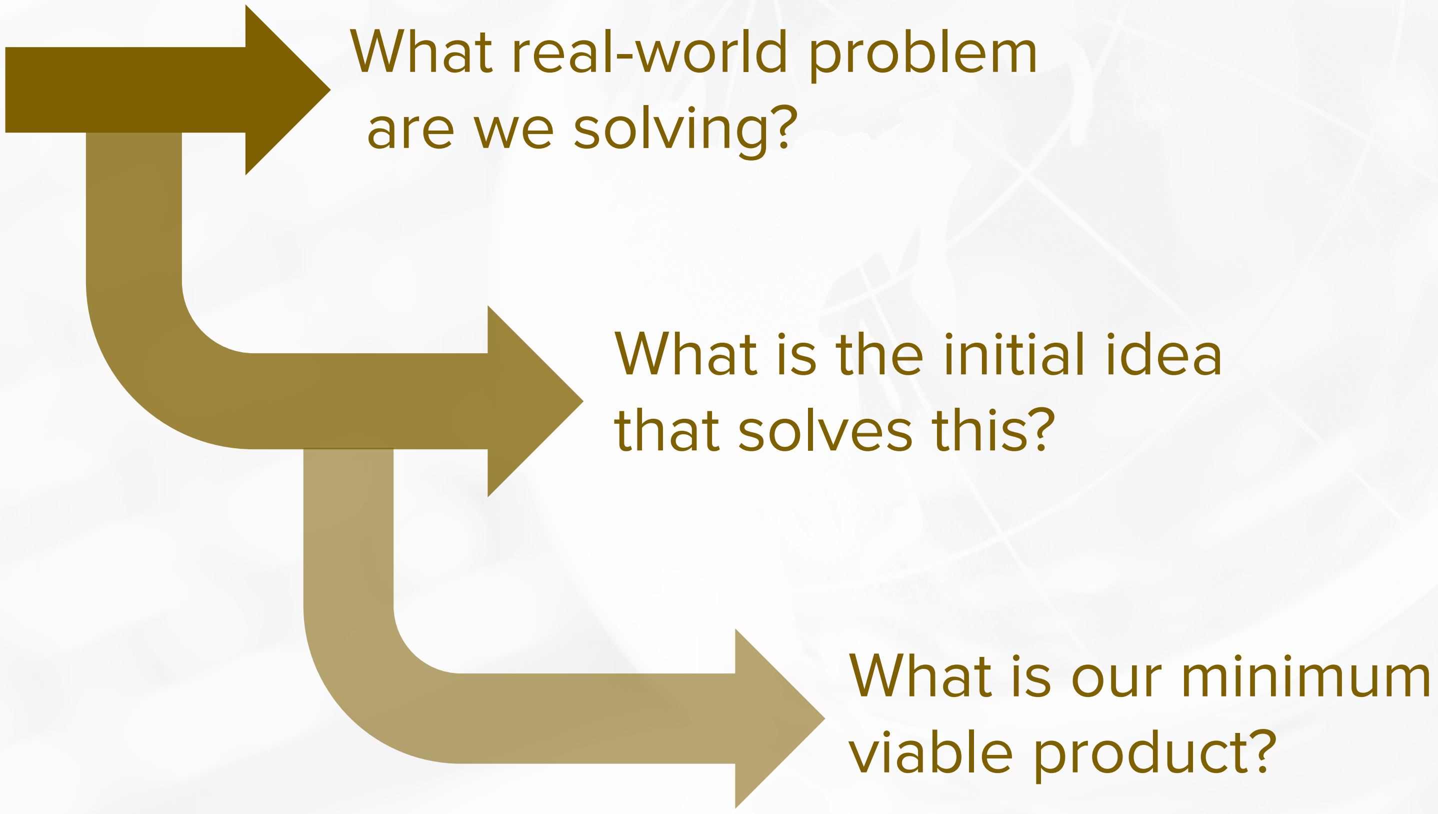


The initial step is to learn what the real-world problem you are solving is and what led to the initial idea.

- This idea can be used to build a minimum viable product, which is not a perfect product but imitates the ultimate functionality of the product.
- This gives you something to observe and measure, leading to more data, which you can learn from, to generate new ideas and build a better product.

First Step...

To start our software project, we need to determine the real-world problem we are solving, come up with an initial idea to address it, and identify the minimum viable product that will allow us to launch and test our solution.


What real-world problem are we solving?
What is the initial idea that solves this?
What is our minimum viable product?

Team Structures

Robert Blackburn

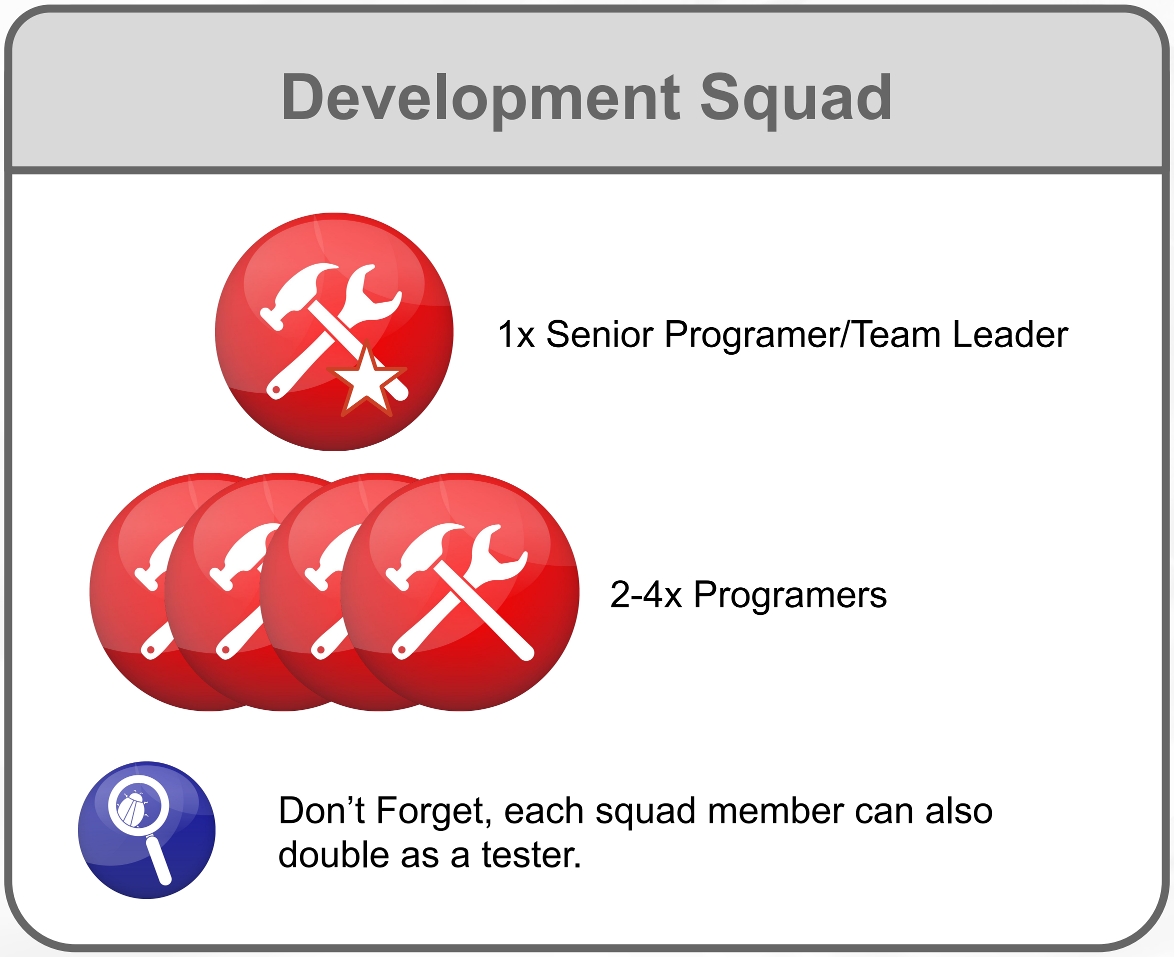
Technology Thought Leader

Software Development Teams

Now that we have looked at the roles and technologies and have explored some of the topics and techniques unique to software development, let's have a look at the common ways software development teams structure themselves.

Agile Squad (Team)

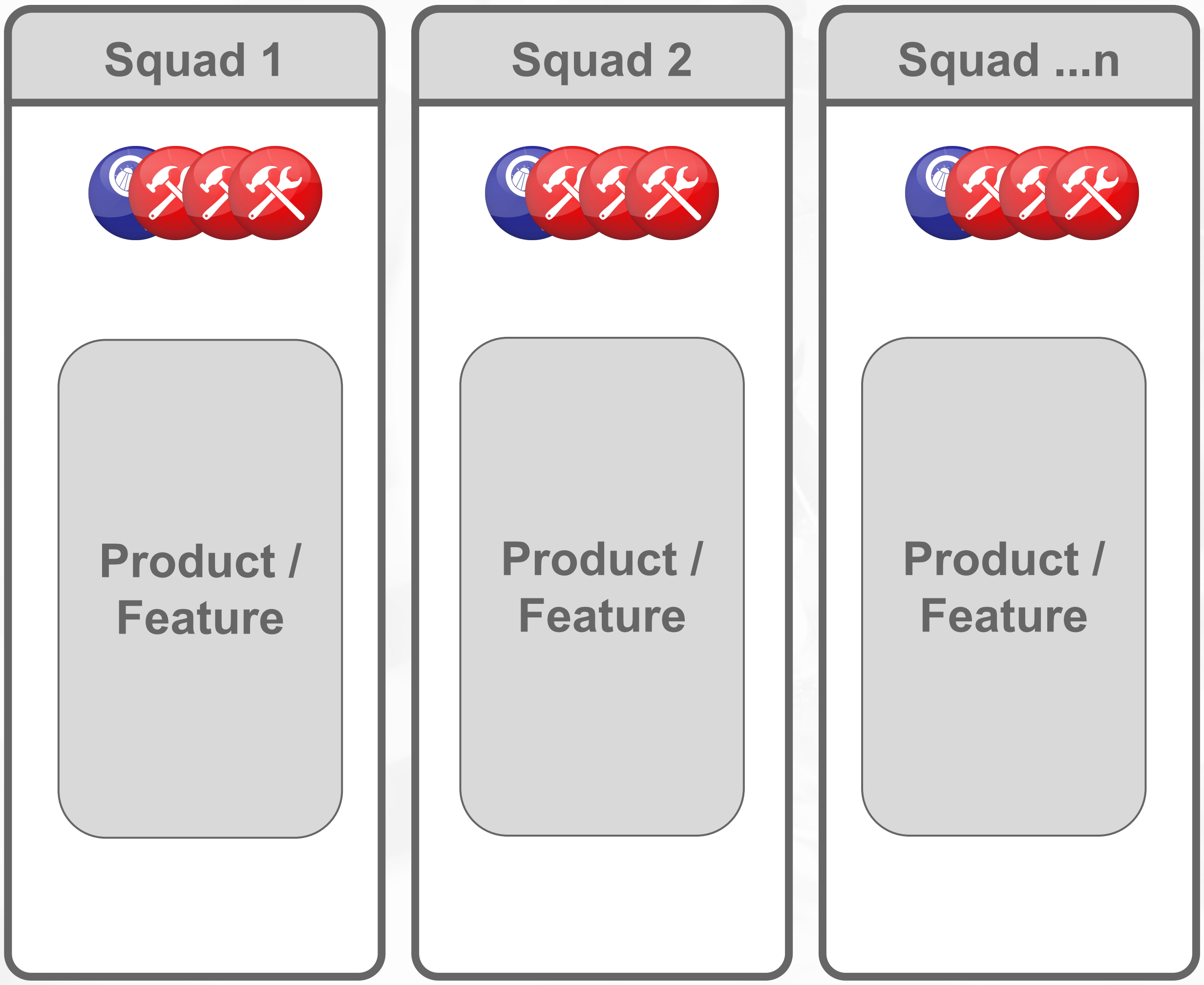
Many software efforts will form their programmers and other contributors into “squads.” The core of most squads will be the programmers themselves, usually lead by a senior programmer or team leader.


Development Squad
1x Senior Programer/Team Leader
2-4x Programers
Don’t Forget, each squad member can also double as a tester.

The makeup of squads is often specific to the organization's needs and the nature of the product. For example, an organization that needs to focus extensively on ISO 2700 compliance (like a security company) may assign dedicated security auditors to each squad.

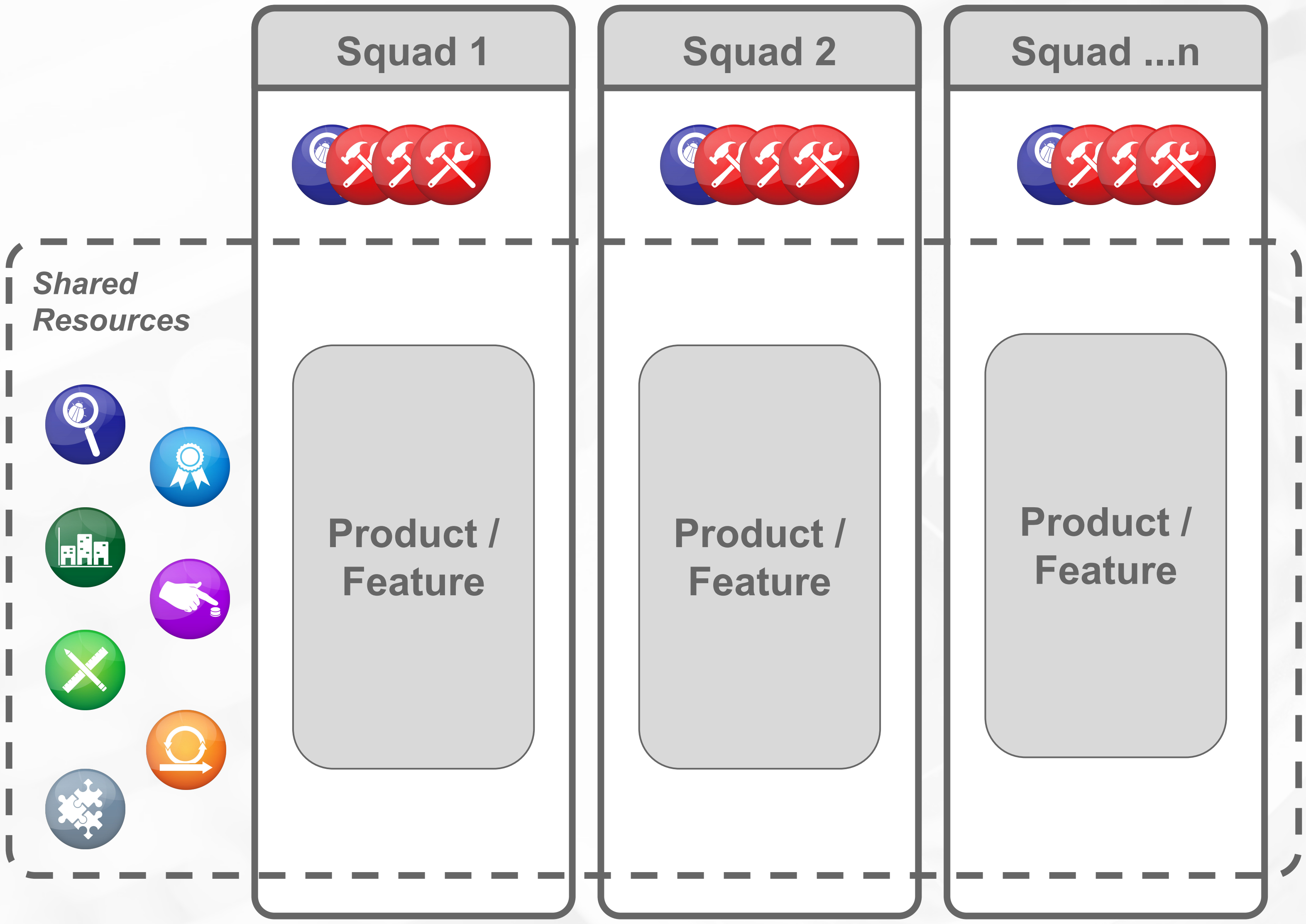
Squads in Large Organizations

For larger products or organizations, there will be multiple squads. Each squad can be assigned to a product, or for larger efforts, even to specific features or components within a product.



The squads can also be rotated to new products/features, when appropriate, to help with knowledge sharing.

Shared Resources



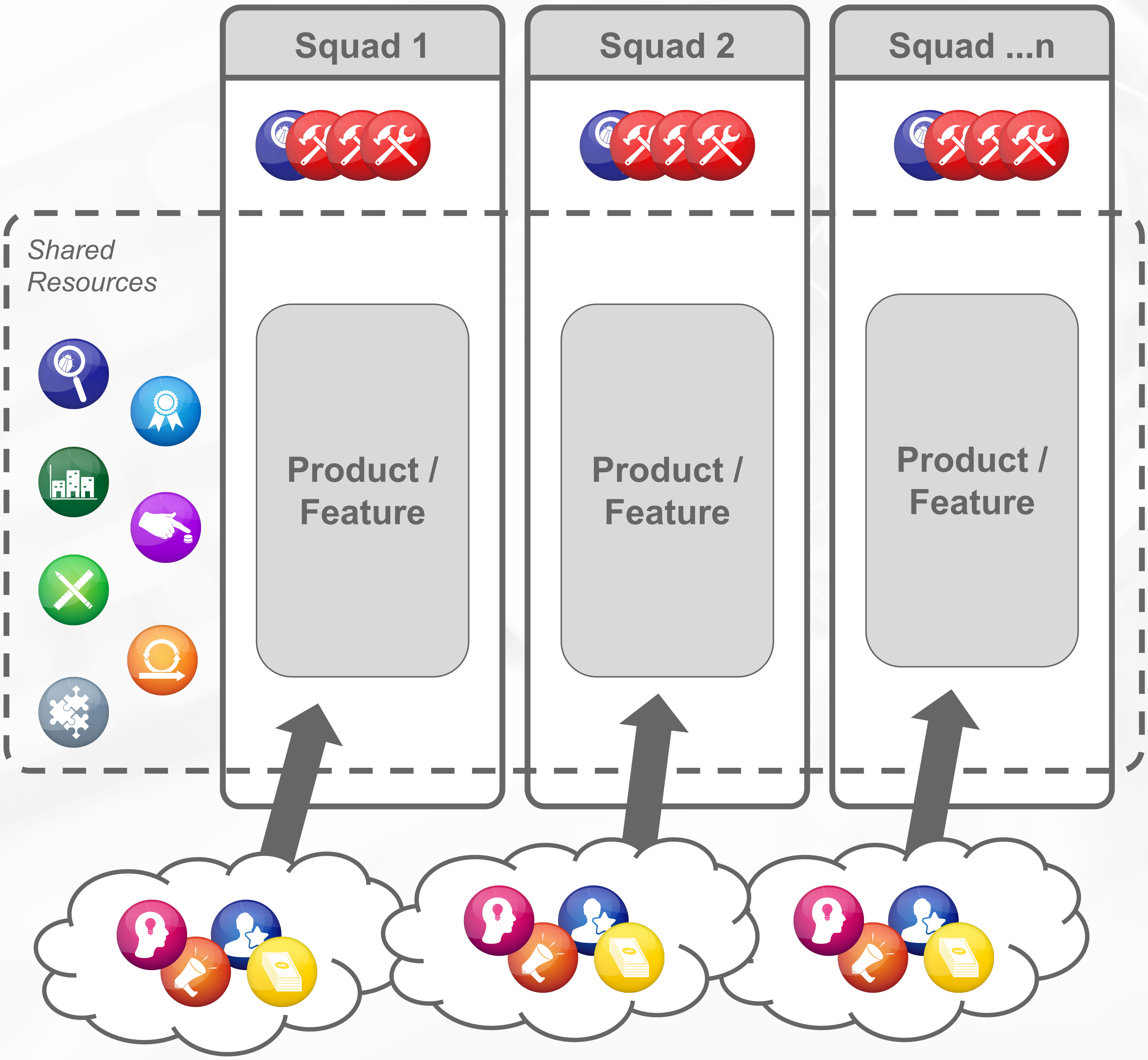
There are also other roles that would typically be shared across the squads. Examples of these can be, but are not limited to:

- UI Designers
- UX Engineers
- BAs
- Agile Coaches
- Architects
- Auditors
- QA Engineers

As mentioned before, sometimes an organization may find the demand for one of these shared roles to be high enough to warrant assigning dedicated members to each squad.

External Resources

Each project typically also has many external roles involved in the development of the product, including the project sponsors, the PMs, SMEs, and super-users:



First or Next Steps...

Next Steps

The initial steps to a project are often:

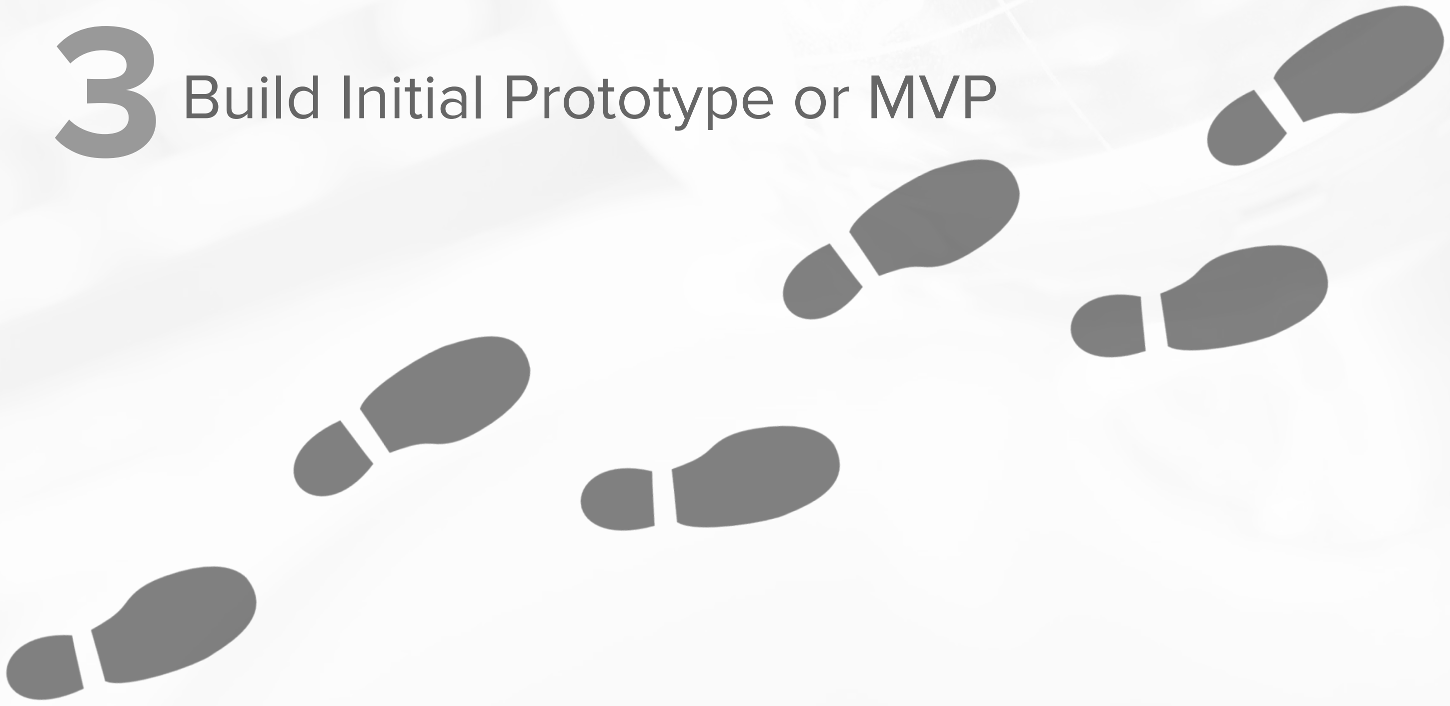
1 Project Kickoff:

- Determine project’s overall scope
- Identify the MVP
- Identify team members, special roles and stakeholders
- Create Business Requirements Document (BRD)

2 Create Project Collateral:

- Statement of Work (SoW)
- Wireframes
- Project plan

3 Build Initial Prototype or MVP



Project Kick-Off Checklist

- ❑ Determine Overall Project Scope:
  - ❑ What is in scope/out of scope?
  - ❑ Resource constraints (skill sets, telecommuting options, etc.)
  - ❑ Budget, time and resource limits or targets.
- ❑ Identify the MVP:
  - ❑ What is the minimum viable product (MVP)?
  - ❑ What features and functionality are essential for the MVP?
  - ❑ What can be deferred until later iterations?
  - ❑ What are the risks associated with the MVP?
  - ❑ What business questions do we expect the MVP to answer?
- ❑ Identify team members, special roles and stakeholders:
  - ❑ Who are the team members, and what are their roles?
  - ❑ Who are the stakeholders, and what are their roles?
  - ❑ Who will be responsible for making decisions?
- ❑ Create Business Requirements Document (BRD):
  - ❑ Identify the objectives, goals and success criteria.
  - ❑ Outline the project timeline, milestones and deliverables.
  - ❑ Describe the technical architecture and infrastructure.
  - ❑ Describe the user experience and interface design.
  - ❑ Define the testing and quality assurance process.
  - ❑ Describe the deployment process.

Blackburn Labs

It's Time to Get Started!

Schedule Your Kick-Off Meeting Today

www.BlackburnLabs.com
info@blackburnlabs.com
401.515.5115

Create Project Collateral

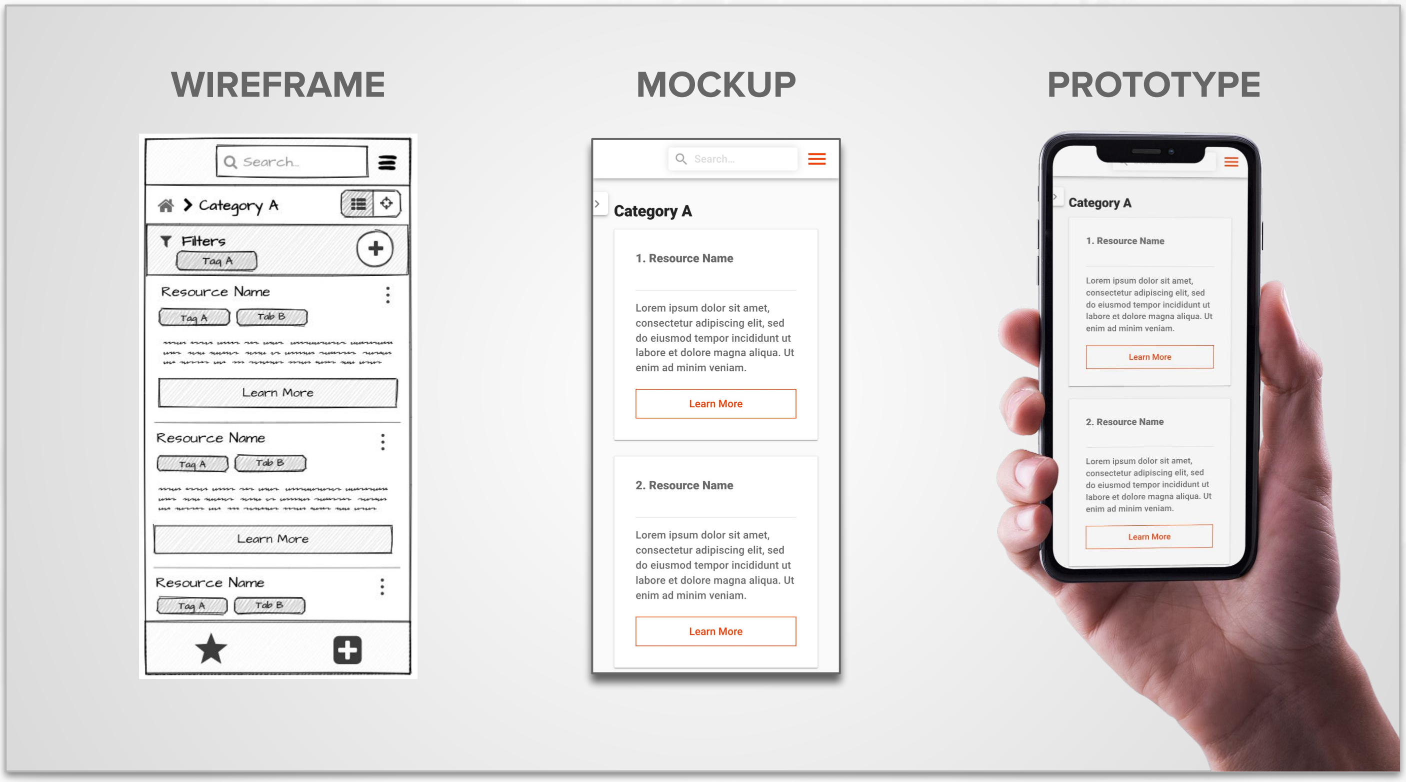
After the kick-off meeting(s), you are ready to create the necessary collateral to begin your project.

Examples of common collateral are:

- **Wireframe**
- **Mockups**
- **Prototypes**
- **Technical Design (UML, ERDs,...)**
- **Statement of Work (SoW)**

Wireframes vs Mockups vs Prototypes

People often confuse the difference between wireframes, mockups, and prototypes.


WIREFRAME
MOCKUP
PROTOTYPE
Search...
Category A
Filters
Tag A
Resource Name
Tag A
Tab B
Learn More
1. Resource Name
2. Resource Name
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam.
Learn More

Each of these documents help describe the user interface, UX, and flow of your app. However, each has a different focus and purpose.

Wireframes vs Mockups vs Prototypes

Wireframes: Wireframes are lightweight “line drawings” of key screens of your app. You can use many different tools to do this. Popular tools are LucidChart or MockFlow. However, simply using Google Slides or plain old pen & paper is often just as effective. Whatever you are most comfortable with and can work with the most fluidly.

Don’t try to wireframe all screens, just the important ones. Just like the test scripts, it is also better to get an initial draft of the wireframe to the team and stakeholders quickly to start gathering feedback.

Mockups: As you gather feedback, itr may become clear you need a more detailed visual than a wireframe. This is when you may want to create a mockup or work with the team’s designer to make a mockup. Tools like MockFlow can also be great for mockups, but often graphic editing tools like Photoshop or Gimp are used for this.

Prototype: Sometimes a new feature is doing something so unique, we need a prototype to effectively understand it or verify our approach to it.

What is a wireframe?

A wireframe is an unstylized graphical representation of the different states or pages of your system, site, and/or app.

- This can be as simple as a line drawing on a napkin, but it can be as sophisticated as an interactive drawing.
    - Even static line drawing wireframes can be placed into a flowchart-like diagram to show the user flow within the app.
    - For examples of interactive wireframes: https://mockflow.com/samples

- The intent is to keep these simple and to avoid getting bogged down in the aesthetic details.
    - Because we are avoiding writing code or pixel-perfect design mockups, we can change the wireframes more freely, allowing for more fluid and responsive design discussions.
    - It is far cheaper and faster to change a wireframe then to redo mockups or update code in a prototype.

What is a wireframe?

- Wireframes can then be used as the basis for the next steps:
  - Gives a graphic designer or UX engineer an idea of the overall flow, allowing them to select only key pages to mockup instead of mocking up the entire app.
  - Gives developers a big-picture view of the app, allowing them to make better architectural decisions and more easily break up tasks.

- Wireframes can even be used to show to potential investors or stakeholders, to encourage buy-in, or simply to show progress.

Statement of Work (S0W)

Compiling all the material from the kick-off meeting, and all the material generated as a follow-up to the kick-off, the development team will typically create one or more statements of work (SoW) options for consideration by the product owner or business sponsors.

The document may include some of the following elements:

- Primary Use Cases
- In Scope/Out of Scope
- Outstanding Questions
- Cost/Resource Estimates
- Key Deliverables/Milestones

Do not over document!

It can be easy to fall into the trap of trying to document every aspect of the product or application. This is often wasted effort.

The goal at this stage of the project is to document enough to allow the team to get started, **and no more**.

This can be difficult when obligations need to be made and contracts for resources signed.

- You'll need to collaborate closely with your contracting firm to generate just enough documentation that all parties are confident the correct product will be built to everyone's satisfaction.

This is why finding the **right** partner is so important.

Notes:

Notes:

About Blackburn Labs

This document was prepared by the staff, and partners, of Blackburn Labs.

At Blackburn Labs, we have committed, motivated, and experienced programmers, architects and process specialists. As an award-winning software designer and creator, Robert and his team have the uncanny ability to provide solutions to address a client's business, application, or software engineering needs with leading-edge technical skills and valuable business experience.

Robert W. Blackburn
CEO & CTO

Have a look at the full list of solutions, technologies, and processes we use everyday to solve clients' software architecting, building, and maintenance needs on our website:

www.BlackburnLabs.com

Feel free to contact us to set up a consultation:

info@blackburnlabs.com

401-515-5115

info@blackburnlabs.com

401.515.5115