

Up and Running Software – Discovery Methods

Determine what you want to do, communicate with your technical and design teams better and easier, and save time and money as a result.

About This Document:

This document was written for people who'd like to create software and would like to be introduced to some methods that might help them with that. In our experience, these are usually business owners, product owners, organizational leads, and other business stakeholders who haven't touched code. This document wasn't written for software development professionals or those who are happy with how they do discovery today.

Discovery is the process of defining what you'd like your software to do. If you were building a home, this is making the blueprints. It's a vital step as it will save you a lot of money and time (measure many times, cut once). Personally, we don't think there's one right way to do it; there's the way that works for you, your goals, your interest, and your availability. We can do this if you'd like us to, but there is a lot of value if you or some of the interested stakeholders on your team can be involved.

The following are a list of items that we normally like to define or collect when doing initial discovery on a project. We think the first three methods can be easily done by anyone, technical or not, and the rest are usually done by an analyst or someone in or near a technology field (we think anyone can do these too, but it will probably take more effort and self-education via your search engine of choice).

It's not necessary to use all of these, though some require another to be done first and doing more than one will be more thorough, leading to a better solution faster. If you're running short on time, just do the product description and low-fidelity wireframes. If you have more, do the user stories, and then ensure your wireframes account for all of them. If even more, go through the BDD process. Finally, if the interest and time is there, try out the remaining ones.

Methods:

Product description or purpose:

This basically answers the question, "Why are we building this?" This seems very simple, but sometimes this definition is skipped because it's assumed everyone knows why. The result of this could outline the benefits of the new application build, the problems it solves, the opportunities it creates, how the business/organization will use the new software, what improvements are expected (justifying the investment), and the stakeholder types (roles, logical groups of people/users) who will be using it, as well as anything that you'd like to add.

User stories:

These are high-level requirements that anyone can put together easily. Here's a model: "As a (role) I want (something) so that (benefit)." (Source: Mike Cohn). Here are a lot of good examples:

<http://www.agilemodeling.com/artifacts/userStory.htm>

Most of the time, these are defined from the perspective of a given actor/stakeholder doing the work. It's important to remember that you can treat other computer systems as actors/stakeholders as well. For example,

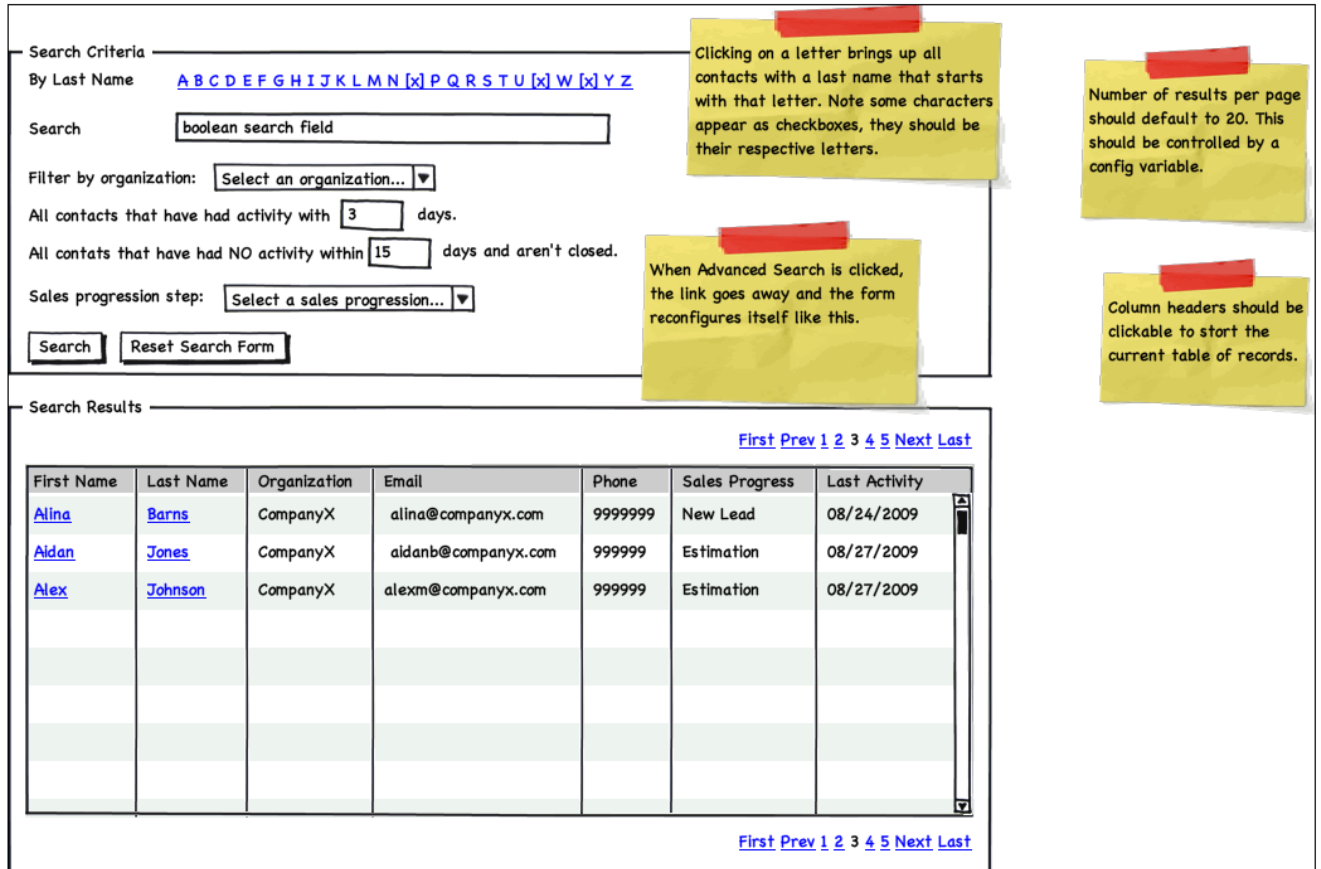
there may be user stories where the application you want to build needs to share information with other systems that already exist in your business.

Low-fidelity wireframes, pictures, white-boarded screens, drawings on napkins, or [whatever you'd like to call them or however you'd like to approach them]:

Our customers, whether technical or not, find a lot of value in creating low-fidelity mockups and then doing some paper prototyping using the mockups they arrive at to perfect them. The process is an easy one to start and do, and the results end up answering a lot of the other questions to be answered too, such as the data you're going to collect and the interaction you're looking for. "A picture is worth a thousand words" is the applicable proverb.

Examples:

- Here is one low-fidelity mock-up:



Search Criteria

By Last Name [A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [W](#) [X](#) [Y](#) [Z](#)

Search

Filter by organization:

All contacts that have had activity with days.

All contats that have had NO activity within days and aren't closed.

Sales progression step:

Clicking on a letter brings up all contacts with a last name that starts with that letter. Note some characters appear as checkboxes, they should be their respective letters.

Number of results per page should default to 20. This should be controlled by a config variable.

When Advanced Search is clicked, the link goes away and the form reconfigures itself like this.

Column headers should be clickable to sort the current table of records.

Search Results

[First](#) [Prev](#) [1](#) [2](#) [3](#) [4](#) [5](#) [Next](#) [Last](#)

| First Name | Last Name | Organization | Email | Phone | Sales Progress | Last Activity |
|-----------------------|-------------------------|--------------|---------------------|---------|----------------|---------------|
| Alina | Barns | CompanyX | alina@companyx.com | 9999999 | New Lead | 08/24/2009 |
| Aidan | Jones | CompanyX | aidanb@companyx.com | 999999 | Estimation | 08/27/2009 |
| Alex | Johnson | CompanyX | alexm@companyx.com | 999999 | Estimation | 08/27/2009 |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

[First](#) [Prev](#) [1](#) [2](#) [3](#) [4](#) [5](#) [Next](#) [Last](#)

After looking at it, you can see that it's reasonably detailed, which is one of the important outcomes of this approach. If you can be sure the wireframe is thorough, meaning it presents all the data you need for that step and you have wireframes for every step, this information will give a software developer a great deal of insight into the exact data you're working with and how those interactions should flow. This works far better and faster than words alone, which really opens the door for downstream errors.

- If you've not received it, please request this file from us: "Up and Running Software - The Development Process.pdf". Pages 6-8 and Appendix A present more examples and commentary about this process.

Here are a couple of links about the paper prototyping process:

- Google: <https://www.google.com/search?q=how+to+paper+prototype>
- Jakob Nielsen: <http://www.nngroup.com/articles/paper-prototyping>. "The most common estimate is that it's 100 times cheaper to make a change before any code is written [...]"

Other notes related to this:

- Helpful tool: <http://balsamiq.com>. This makes constructing these wireframes really easy.
- Usually, you, the person making the mockups, are not your user. However, in the case of your system, it may be that you are designing this exactly for yourself. We write this so you can think about it, and pick the path that you'd like (design for many users/people, or just for yourself or select users). Remember to make mock-ups for all of your user types. For example, we've noticed that people who haven't done this before tend to skip the administrative mockups.
- If you want to really wrap your mind around the users you're making the software for, please use your search engine of choice to read about "personas" a bit.
- Before getting too into the tech or polished user interfaces (high-fidelity UIs, like Photoshop files), we think it's great to get sign-off on the low-fidelity or high-fidelity wireframes. Once those are set, you can start exploring the best technical approach, including who should do the work and how. Having these will make your discussions with your technical contacts a lot easier and more productive, and this whole process should save you a lot of money because you'll really know what you want at this point.
- Have you addressed all of your user stories? After creating these and doing paper prototyping, are there any user stories that you'd like to remove? Would you like to add any?
- Once you have your wireframes defined/built, you can go back to the user stories you created and reference the relevant wireframes in the user stories. You can do this on the wireframes too (annotate them with the user stories they represent) if you want. This will help map out what visuals tie in with the user stories.

Behavior driven development (BDD):

BDD is an approach where the stakeholders and developers define a common language on how stories should be handled in an application (scenarios). You could think of them as a set of acceptance tests for the software. Basically, it has to do "this" and "that" and we'll know it's working like it should. BDD provides guidelines for how these specifications need to be defined, in a structured format (but still plain English), so that everyone involved can understand what's needed. They are also structured enough that test frameworks can utilize the specifications as well. If done thoroughly, these bring the probability of incorrect assumptions very close to zero.

Stated differently, this defines all scenarios for a user story, every possible situation within that story. If a set of conditions ("this") has a different outcome ("that"), it should have its own scenario. This expands on user stories in a defined manner so that all combinations of conditions that can happen are addressed in the software and can be tested according to the business rules of the company. These are a good next step after the user stories are defined.

Here are a couple of helpful resources:

- A good introduction by the gentleman who coined the term "BDD": <http://dannorth.net/introducing-bdd>. We think the ATM example that's over halfway into it is a nice one in particular.
- A useful example for someone new to BDD: <http://www.integralist.co.uk/posts/guide-to-js-testing.html>
- A more detailed introduction using a framework: <http://jasmine.github.io/2.0/introduction.html>

In the examples presented in those links above, please do not give attention to the programming elements and structuring code; that's for the developers to do. The main focus for you is the natural language of the stories and following the rules for how they need to be defined.

The data:

It's helpful to think about the data you want to capture. To get started, you may have a system that you use now, and running an export of that or gathering the important reports are a couple of things you can do. There are formal methods for data organization, such as https://www.google.com/?gws_rd=ssl#q=ERD+example and https://www.google.com/?gws_rd=ssl#q=data+dictionary+example. It's not necessary to put those together at this point. It's just great if you can assemble or list out all the data in a spreadsheet that you think you'd like to capture with this system, which should be reflected in the low-fidelity interfaces eventually.

Functional requirements:

People approach this differently, and we won't provide formal methods here. We think the purpose of this is to define what isn't easily definable in the wireframes and user stories, though annotating those wireframes with this information is recommended if it will fit nicely. Usually, a more technical person is tasked with this work.

For one approach, these functional requirements could list out all the features you'd like, present the goals of those features, and present any special notes about those features that you think are important. Here are a couple of prompts that may help you write more information:

- 1) What are your pain points or problems today, and have we addressed that with the documentation?
- 2) What are the opportunities that we'd like to pursue, and have we addressed those with the documentation?

Use cases:

These are not the same as user stories. These are the core workflows that the actors (personas, user groups, roles, etc.) of your system go through to accomplish work. They can be demonstrated through wireframes as well, but having a breakdown of the key use cases helps identify the most important things your users do in the system. Helpful link: http://www.gatherspace.com/static/use_case_example.html. Usually, a more technical person is tasked with this work.

Next Steps:

I hope the information provided above is useful, and you're able to use these to create the system you want. If you have questions, requests, or feedback, we'd be pleased to hear from you at any time.

Thank you and Respectfully,

Ian McKilligan

Ian McKilligan, CEO
Up and Running Software, Inc.
Cell: 906-281-2627
Email: ian@upandrungingsoftware.com