Beyond Requirements

Analysis with an Agile Mindset

Kent J. McDonald

Agile Software Development Series

Alistair Cockburn and Jim Highsmith, Series Editors

BEYOND REQUIREMENTS

Beyond Requirements

capabilities, but you may have some freedom as to the order of implementation. In this case, a list of things to deliver—which are truly options when using impact mapping—becomes part of an overall solution. Implementing features iteratively can still be very helpful because it provides a way for the team to remove extraneous functionality. Story mapping (Chapter 14) is a helpful technique here, because you can determine the key activities that need to happen at a fairly abstract level but be very selective about what specific functionality you produce to support each of those key features. The team would define scope as measuring the established business goals and objectives, rather than as a specific list of deliverables. This can give you the latitude you need to remove features that are not truly needed by not injecting them.

This is the approach that we used for the conference submission system (Chapter 7). At the core of the matter, we were trying to support the submission process for the agile conference. We didn't have a lot of different options for completely different behavior; we had chosen a process we were going to use and we needed to support it. We found a story map helpful to organize the work because we were able to make some clear decisions about what aspects of the submission system we were and were not going to do.

So you may have several different potentially independent options to choose from, or you may have many dependent features, most if not all of which have to be included. In either case, once you have chosen to inject one feature, start with some representation of the output that represents the feature. Then, work backward to understand what you need to deliver that output.

Imagine you are working with a company that has decided to introduce a new payroll system. The new system can have a variety of outputs, including paychecks and reports. Depending on your perspective, different outputs generate value. If you are an employee, the check produced by the payroll system provides you with value. If you are the human resources manager, you may view the reports produced by the payroll system as protecting revenue. They give you information to help you take actions that reduce risk, such as discrepancies in your payment structure based on improper factors such as age, race, or sex.

It follows, then, that you need to determine who your stakeholders are and what outputs they expect from the system. Starting with stakeholders is helpful because it prevents you from generating outputs that aren't really needed. You can identify your key stakeholders by asking, "Who would care if this payroll system did not exist?" Table 5.1 shows a list you might come up with.

With this understanding of the stakeholders, you can identify the outputs that deliver the value they expect. This is where models come in handy, especially those representing displays of information or reports that stakeholders are looking for in order to answer some questions or make some decisions.

Table 5.1 *Payroll Stakeholders and Their Expectations*

Stakeholder	Expectations
Employees	If the payroll system did not exist, they might not get paid. This is probably not the case, but it certainly may be a more complicated process to get payroll completed. From the employees' perspective, the payroll system adds value because it generates checks. When employees get paid, they continue to work, so the value is in protecting revenue by keeping employees satisfied that money is still coming in the door. (Yes, I know this is kind of a stretch, but it's the one that seems to fit best.)
Payroll department	Assuming that the organization would still pay its employees even without a payroll system, the lack of a system would make the process inefficient and more prone to error; so creating a payroll system reduces costs. And to some extent, it protects revenue by reducing the risks of incorrect paychecks.
Employee relations manager	Even if you could pay employees without a payroll system, the lack of organized data makes it much more complicated to perform analysis on payroll information, which indirectly leads to increased risk for the organization, similar to the items described previously.

In the case of the commission system, the output can be something a bit more tangible, such as a paycheck, or starting a process that will result in something being shipped to a customer.

Once you understand the outputs, you can work backward to figure out what processes are needed to produce those outputs (including the rules that act in those processes) and the inputs the process needs to create the outputs. You are effectively performing analysis in the opposite direction of development, which tends to bring in the inputs of a system first, then build the process, and finally create the outputs. Said another way, because you are pulling value from the system via the outputs, you leave a hole at the beginning of the system into which features are injected.

Spot the Examples

We typically use models to describe the outputs and processes and the inputs used to create them. These models are helpful for creating a shared understanding with everyone involved in delivering the features, but they are rarely sufficient. In the words of George E. P. Box, "All models are wrong, but some are useful." One way to make the overall understanding of a model clearer is to add examples. Examples serve two purposes. First, they provide concrete guidance in very specific situations to people who tend to ask, "Yes, but what about this situation?" Second, examples give the team a way to test the models, make sure they account for different situations that may occur, and build a shared understanding about what a feature should do. Examples also describe requirements and tell when a feature has been delivered successfully, giving your team a head start on your tests. Examples tend to provide a more useful description of the requirements than other means, based on anecdotal evidence and my conversations with developers. If given a choice between a set of examples and a set of textual requirements, developers I have worked with almost always refer primarily to the examples.

In an approach that Matts calls "Break the Model," a team establishes a model they believe appropriately represents the domain in which they are working and use it to produce the desired outputs. Then examples are cited to determine whether the model, as it currently exists, will support that situation. If it does, great; we keep the example around as a convenient way to describe the desired behavior of the system. If the model does not support the example and the example is actually relevant (meaning the example is actually likely to happen, and it's a situation we are interested in addressing), we revise the model to represent the new understanding of the domain to account for that example. We then provide both the relevant examples and the model to the rest of the team as a description of what needs to be built for that particular feature. One thing to keep in mind here is the Key Example pattern described by Gojko Adzic in his book *Specification by Example*. That idea is to focus on key examples and avoid repetitive examples that do not show anything new. Don't keep examples that don't add any new information to the set you already have.

When we built the submission system, we described all of our features in terms of examples. Those examples were used as the basis for automated acceptance tests, and I still use them as a reference for what was and was not accounted for. For example, recently a question came up from the current conference chair about what he could control with respect to whether the submission system was "open" or "closed." I was able to go into our source code repository and find the pertinent examples (shown below). The examples indicated that the conference chair can control when people can submit new session proposals, edit their session proposals, or edit their accepted session proposals.

```
Feature: Edit conference dates
  As a conference chair
  I want to change session submission deadlines

  Background:
    Given I am logged in as "Connie"
    Then session submissions should be open
    And session edits should be open
    And accepted session edits should be open
```

```
Scenario: Update Open Date
  When I change the session submission start date to 1 day from now
  Then session submissions should be closed

Scenario: Update Close Date
  When I change the session submission end date to 1 day before now
  Then session submissions should be closed

Scenario: Update Edit Date
  When I change the session submission edit date to 1 day before now
  Then session edits should be closed

Scenario: Update Accepted Session Edits Start Date
  When I change the accepted submission edit start date to 1 day from now
  Then accepted session edits should be closed

Scenario: Update Accepted Session Edits End Date
  When I change the accepted submission edit end date to 1 day before now
  Then accepted session edits should be closed
```

The thought process surrounding Feature Injection is a huge influence on how I approach analysis on all projects, and while I very rarely mention the name to the teams I'm working with, I'll often introduce the ideas. Most of the people I talk to say things like "Yes, that makes a lot of sense. Why didn't we do that before?" or "Yeah, we do that, with these few tweaks." Starting with the value you want to deliver, using that value to decide what feature to build next, and describing that feature through the use of real-life examples proves to be a simple, effective way to build the right thing, and not build things that are not right.

Minimum Viable Product

Eric Ries introduced the concept of minimum viable product in his writings on Lean Startup. This is the most straightforward description he provides (*The Lean Startup*, Chapter 6):

> A minimum viable product (MVP) helps entrepreneurs start the process of learning as quickly as possible. It is not necessarily the smallest product imaginable, though; it is simply the fastest way to get through the Build-Measure-Learn feedback loop with the minimum amount of effort.
>
> Contrary to traditional product development, which usually involves a long, thoughtful incubation period and strives for product perfection, the goal of the MVP is to begin the process of learning, not end it. Unlike a prototype or concept test, an MVP is designed not just to answer product design or technical questions. Its goal is to test fundamental business hypotheses.

The main purpose of MVPs as defined by Ries is learning what customers find valuable, not necessarily delivering value to customers. Additionally, learning is trying to figure out business as well as product design and technical questions, and in the early stages it is most likely focused on the business questions.

MVPs are a technique that your team can use to carry your discovery activities forward through your delivery process. It's a key component of the Build-Measure-Learn loop because it's the thing that teams build in order to gather feedback and learn from it.

That's all stated from the startup context. What value can the MVP concept bring to IT projects? Primarily, the MVP can be used when testing solutions. We may find that we get a lot of information from our stakeholders about what they believe their needs are, and we may even pull some data from a variety of sources that tells us how our stakeholders behave, but at the end of the day the most effective way we may find to see if our solution is really going to satisfy our stakeholders' needs is to build it, try it, and see what happens.

The idea of the MVP, even in the IT project context, means that the purpose of an iteration is to either learn or earn. Initially your team is trying to learn about the problem and solution by doing things to validate assumptions, address risks, or make progress toward your desired outcome. Your team produces some output that while not a complete solution does provide information for the purpose of testing your hypothesis. In effect, you're saying, "We think this is going to work, but we won't know until we try it, so let's see what happens."

Your team may be able to provide something for stakeholders at the end of an iteration, get feedback, and have a pretty good idea of whether it will work or not. The benefit of this approach is that you can get feedback sooner because you don't have to build a solution that's conceptually complete, just enough that you can get feedback on it. It's important when the intent of a sprint is learning that you have a specific question in mind that you are trying to answer. This can be stated as your sprint goal so that it's clear to your entire team.

What does an MVP useful for feedback in an iteration setting look like? I worked with a team recently that was building new analytic capabilities for an organization that traded securities. The team was working on an effort to incorporate a new source of data into an existing data warehouse, and one of the first things they needed to do was verify that they could successfully merge data from the new source into an existing source. They selected a view of the data that represented a simple listing of securities but contained attributes from multiple sources. Instead of worrying about creating a pristine report or moving the data through all the various architectural layers, they chose to start by associating data from the new source into the existing data and generating a query using Excel. They were able to show the data they expected to see in the final report without spending a lot of time designing the report or building all of the background infrastructure perceived as necessary to automatically integrate the data in the long run. They had to determine whether they could combine data from the disparate sources correctly. Doing it in this manner let them immediately identify where they had some logic corrections to make, and they were able to find that out in a couple of weeks. Had they taken the typical approach to building up a data warehouse, they might not have uncovered the issues they found until months down the road, because they wouldn't have been able to isolate where the problem occurred. In addition, they were able to quickly get meaningful feedback from their stakeholders about which attributes were really needed and specific rules on how to match data from the disparate systems.

On the other hand, your team may figure out that the only way to truly know whether an MVP will work is to let stakeholders use it in actual day-to-day business. In this case, the MVP may represent a more complete change than the version that was demoed at the end of a sprint.

In the preceding analytics example, this type of MVP happened when the team released one full-fledged report to its stakeholders. Here, they were trying to find out if the reporting interface was helpful to the stakeholders and if the organization of the reports made sense. This one report would still be useful, but real value would be generated when multiple reports were available. However, by delivering this one report first, the team learned a great deal about the stakeholders' needs, and the stakeholders gained a better understanding of their needs and the reporting capabilities.

Either way, understanding the MVP idea keeps teams from feeling compelled to define too much information up front and encourages them to see delivery as a way to validate assumptions and test hypotheses.

The other important aspect of the MVP concept is the implied focus on speed. You seek the minimum viable product (or change to an asset) because it means you can get it done sooner, get it in front of stakeholders sooner, and get feedback sooner. All of that means that you aren't wasting time heading down a road that leads nowhere; and if you are, you aren't wasting nearly as much time on that dead end.

Minimum Marketable Features

Minimum marketable feature (MMF) is a similar concept that came out a few years before the idea of MVP. While the ideas are similar, there are some significant differences.

Given our focus on delivering value, we may find it helpful to organize our work based on how much value we're providing. In 2004, Mark Denne and