Abstract

Gap Inc. Direct is using Agile methodologies for IT project delivery. This paper discusses how building systems starting from the customer facing pieces allows Gap to deliver projects faster.

1. Introduction

Gap Inc. is a leading global specialty retailer offering clothing, accessories, and personal care products for men, women, children, and babies under the Gap, Banana Republic, Old Navy, Piperlime, and Athleta brands. Fiscal 2010 net sales were $14.7 billion. Gap Inc. products are available for purchase in over 90 countries worldwide through about 3,100 company-operated stores, about 175 franchise stores, and e-commerce sites. For more information, please visit www.gapinc.com.

Gap Inc. Direct is using Agile methodologies for IT project delivery. This paper discusses how by focusing on the customer facing pieces first we’re able to deliver projects faster. The approach is generalized to show how it might be applied elsewhere.

2. What Does Reverse Order Mean

The traditional approach to building software projects starts with the foundational pieces—the data elements and the systems to create, process and edit them. From there, layer by layer, additional capabilities are added, building on the previous layers. Often, the business value of a project is delivered towards the end when for the first time the customer facing capabilities are created. Typically the clients of software delivery initiatives have to patiently wait while the pipes and plumbing are built before they get to see the stuff that’s really interesting to them being delivered, because “before you can get the great web site you asked for, we have to build the data manipulation framework <insert acronym here>”.

But what if the customer facing features could be built first? What if we started from what customers will see rather than from how the data will be created or managed? Initially, there will be “just enough” of the missing layers to get by, with the additional “pipes and plumbing” added later. Just how much has to be built first will vary, but the default working assumption is to build the minimum amount of infrastructure needed to support the target features rather than assuming the build out of all infrastructure capabilities.

This approach focuses the project from the start on the end result and provides opportunities to learn and adjust much earlier. The customer gets to see what they care about and can prioritize and potentially not build things they thought they needed, but are actually not needed after all.

3. Example – Building an E-Commerce Site

Let’s use an e-commerce platform as an example. In order to sell products online we need to have the product information, pricing, photos and attributes. There’s a system that manages promotions and discounts (e.g., take 20% off this shirt or buy one top get another for free). There is another system for displaying marketing content, a reporting system etc. If we were to follow the typical approach, we would build the content management system early and use it to create the product data to display on the site. This would establish the functionality to create, edit and approve product descriptions, shoot photography of the clothes and prepare the data for the e-commerce site. Similarly, we would create the pricing system and the promotions capabilities. We would probably have layouts of the product page showing where elements will appear and how customers will interact with the site, but it would be a while before any of that gets built—because we first need the foundation.

To the business, the value is not delivered until customers can shop the site. The focus should therefore be on building those pieces as soon as possible. This allows the client to validate the design, adjust the
requirements and prioritize what’s important in order to sell clothes online. If the content management system is perfectly built, tested and deployed to the original requirements, we may have overspent on capabilities we won’t need. Or we might be missing features that become apparent once the site is live. The same logic and concerns apply to the other layers and components.

4. Getting Started – Mock, Fake or Hack

What You Need To Get Started

Building in the usual order makes sense and feels logical – each layer built provides the foundation for the next layer and allows for straightforward testing and progression. If there’s a data feed that provides a lot of data elements, then by creating the feed and the processing, the stage is set for the next step in manipulating or displaying the data.

Building in reverse requires creativity and flexibility since the layers are not fully built. You’ll need to mock, fake or hack what you need in order to get started before all layers are built. Then once a layer is built, the outer layers have to be retested and retrofitted with the actual data to make sure they work well together and the temporary workarounds are removed cleanly.

This requires more effort to create the data and to support testing and integration and adds some risk that what was faked is too different from real world data. One risk is that the code is built to wrong assumptions. Another is that the testing is incomplete because the real life scenarios were missing from the data.

To go back to our e-commerce example, let’s say the product information was coming from a data feed from another system and that since we’re building in the reverse order the feed has not been built yet. The web site would still need a data source with the products to display. There are different approaches by which this can be done:

Hack – instead of building the system to automatically process the feeds, do a one time manual load of a data file. Once the feed system is built, the data can be loaded and the manual load can be dropped.

Fake – create the data manually – maybe just a few products, or by copying one product information thousands of times with small variations. Similarly, once the feed is built, the fake data should be removed and replaced with real data.

Mock – build code that provides randomized product information on the fly. This should be as simple as possible since the code will be disabled when the feed processing is built. The mock code will be tossed out and the data will come from the real feed. Mocking should be done at the appropriate level to minimize the impacts to the system once it’s torn out and replaced.

The examples are focused on getting what’s needed to incrementally build the system and elicit user feedback, but the same approach is valid for taking the code to production. It may seem a harder challenge, but again, with creativity and flexibility, it can be done. The goal is to get to production as quickly as possible with valid code. For example, let’s say products have customer review information associated with them, and the review functionality is not built yet. So the full blown review system that automatically processes the review data and displays it on the site isn’t there yet. By using the hack approach, the initial set of reviews can be loaded and taken all the way to production – without waiting for the full review system to be built. The products can go live with real reviews and customers can use them when browsing the site and shopping. The benefits here are multiple including speed to market, as there’s no need to wait for the reviews system to be completed, as well as the option to not build the reviews system at all if it turns out the feature is not useful or needed or if a higher priority item is requested.

5. Architecture

In order to build the system in the reverse order, the system architecture has to support distinct and separate layers. Otherwise there’s no easy way to focus on one layer while mocking or hacking the others. Automated testing is key as well since refactoring of the various components is expected once different layers are built. It’s important to be able to test each layer and be confident that it’s independent of others so that once the layer is built fully the impact to the outer layers is minimal and can be found early.

The organizational culture has to be supportive of this approach. In order to deliver the features as quickly as possible, some short cuts were taken. Since the tools to edit the data will not be built, updating it may require more effort. Some turbulence can be expected. Also, even when things are fully built and working, they may still need to be rebuilt and refactored since they were
quickly put together with “just enough” to support the reverse order. If the benefits are not clear and there’s no acceptance of this approach it will be hard to be successful building in reverse. The business needs to support subsequent investment to replace the “hacks” and “mocks” with robust, maintainable code even though this investment will not yield new customer-facing features.

6. Thin Slices are Still Key

The concept of a “Thin Slice” is key in Agile methodology and is still key when building in reverse. The idea is to take a vertical slice of the system, usually the smallest piece of functionality that touches as many of the layers as possible. Building out thin slices early reduces risk by forcing early integration and proving, for example, that the database can connect to the middle tier and the presentation layer. The functionality itself can be very basic and limited, perhaps just displaying one element from the database. It should be clear that building in reverse doesn’t negate the benefits of thin slicing and that it should still be done. There’s no conflict between the two. It would make sense to start with a thin slice across the systems and then build out fully in the reverse order – focusing on the end user first and working backwards.

7. Benefits

There are multiple benefits to building systems in the reverse order:

Speed – by focusing on the core functionality and delaying building anything else until absolutely the last responsible moment, the core functionality can be delivered faster.

Better Decisions – getting things in front of the customer or end user faster enables earlier learning of what’s needed, what’s working and what needs to be changed. This leads to better designed features and higher quality.

Minimizing Waste – having better information on what works and what doesn’t is useful for deciding what not to build. The waste of building, testing and deploying features that are not used is removed.

8. Risks / Downside

There are some risks and downsides to building in the reverse order and they should be considered and weighed against the benefits.

From the fake/mock/hack discussion it should be clear that the team has to be flexible. Without building the full tools to create and manage data for example, setting up test cases can be painful. The shortcuts taken require the team to work around them which means extra effort.

The culture of the organization has to be accepting of the price of the shortcuts and the risks associated. There could be surprises and discoveries when the full layers are built and the real data is different than what was used previously. Also, the hacked data may not cover all the scenarios that the real world data includes.

There are some rare projects where the scope is clearly defined upfront, and all of it must be delivered at once – a big bang launch, without the typical learning and exploration of software projects. Those projects would benefit from building the foundational pieces first and proceeding without any shortcuts.

Another call out is that this approach has the potential to incur high technical debt. By focusing on the customer facing pieces, big parts of the system may not be built fully, or may be left with significant limitations. Leaving them as is after launch will cause the technical debt to accumulate and will slow down future projects and prevent them from following the same approach. It is therefore important to allow for the refactoring that is necessary after launching with “just enough” of the functionality. The culture has to support that.

9. Conclusion

By building in the reverse order we have been able to deliver projects faster, focus on the business critical functionality and launch new capabilities that help our customers. In addition, things that we thought were
needed when we started projects turned out to not be so important and we ended up not building them. This approach is part of our Agile toolkit for delivering software projects and needs to be balanced against the risks. When applied correctly for the right project the benefits are worth the risks and downsides.

10. Acknowledgements

I would like to thank the entire team at Gap Inc. Direct for willing to do things differently, and also to Nanette Brown for her help and suggestions in writing this paper.

11. Author name(s) and affiliation(s)

- Michael Elbaz is a Director of Project Management, at Gap Inc. Direct – IT.