# Capability Based Design

Written by Josh Rutstein
Published 5/8/2015

*When I meet with the founders of a new company, my advice is almost always, 'Do fewer things.' It's true of partnerships, marketing opportunities, anything that's taking up your time. The vast majority of things are distractions, and very few really matter to your success.*

- Evan Williams

**Summary**

Our business partners are looking for more efficient ways to get the technology solutions they need at a cost-effective price. Agile development methodology was intended to bridge the gap between big-bang requirements and build projects with the need to make modifications to requirements as we learn more about how the product might evolve. But Agile methodology has led to a different inefficient practice. We intentionally divide work into manageable chunks so that we can prioritize effort accordingly. However, the nature of the work and the evolutionary way we discuss requirements with the customer does not lend itself to efficiently detailing user stories and deriving a solution design. Often we tend to develop one-off solutions before we realize that there are other similar requirements that make it not-one-off. The current paradigm also does not equip the business with options to make decisions to scale an existing solution for a different problem. Capability Based Design will enable more efficient development, allowing the customer to make choices about solution options vs cost in a way that is not possible today.

**Use Case: Data Subsets**

Very often we are presented with very large and unmanageable datasets. Regardless of the tool or environment used to store the data, there is often a need to create subsets of this data, tagged appropriately for use in other ways. Typically, the consumer will need to tweak the logic behind this subset of data as they profile, understand and use the smaller version.

Recently we developed a profitability system that used “Average Assets” as a driver of cost. Various divisions of the organization would use this asset data to drive their high level costs from the general ledger, down to the customers that held the assets and were thus incurring cost for the organization. So if a customer held .76% of the assets they should be ‘accountable’ for .76% of the costs incurred by that group. The logic is fairly simple.

The challenge is in assigning an appropriate grouping of the total average assets dataset, to the divisional pool of dollars before the allocation math takes place. Should the equity division charge their

costs to customers that only hold fixed income assets? Should the cost of the high net worth support teams be spread across all customers of the firm? Clearly there are cases when the business will need subsets of data to align to other financial datasets, both of which should be easily reportable and identified. For performance reasons, using metadata tags for these datasets before the math process takes place, makes the algorithm perform better.

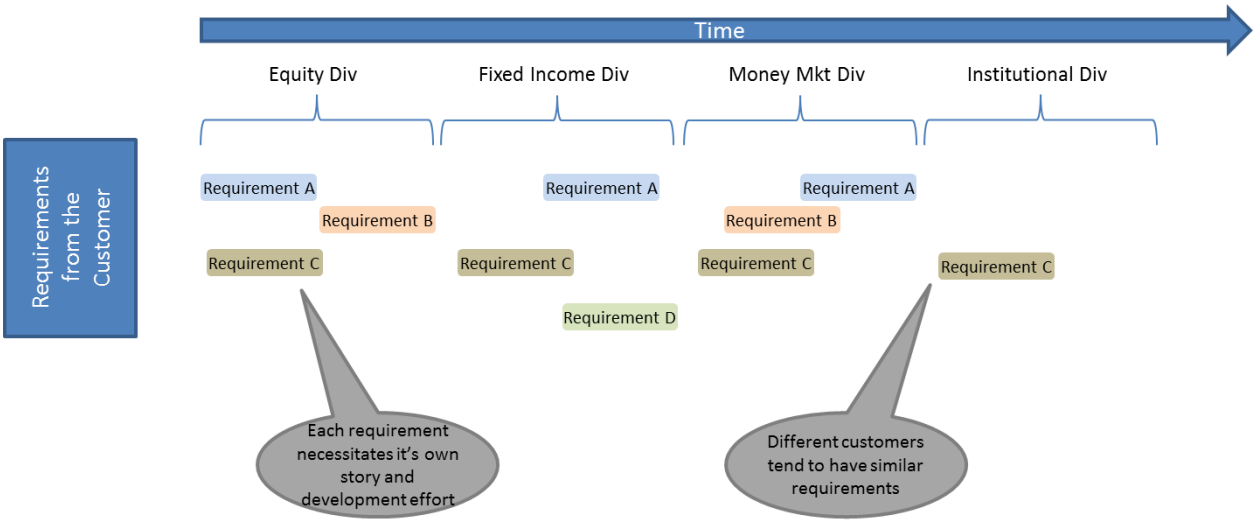
**Gathering Requirements**

In this case, there are various constituents interested in utilizing the profitability system. There might be 7 different divisions within the organization. Each has its own perspective and each has its own way of determining the formula for creating profitability results. This is common. Never is the user community perfectly homogenous. In an attempt to focus on the needs of every group, we need to make sure we hear the concerns of each. Specific meetings and requirements gathering sessions are scheduled to collect as much information as possible.

As we meet with each team their requirements naturally lend themselves to user stories.

- Equity Division
    - Overhead: Carve out expenses for Finance, HR, infrastructure and allocate using all equity division managed assets based on relative share
    - Management Expense: Carve out management fee direct expenses and allocate using Equity division assets but on a product by product basis. Where domestic products are assigned a complexity factor of 1, Foreign developed products 1.2 and foreign emerging country products 1.7
- Fixed Income Division
    - Overhead: Carve out expenses for Finance, HR, infrastructure and allocate using all fixed income division managed assets based on relative share
    - Management Expense: Carve out management fee direct expenses and allocate using Fixed income division assets but on a product by product basis. Where muni products are assigned a complexity factor of 1, corporate debt products 1.5 and foreign debt products 2.2
- ETC.

Each of these 4 requirements would normally spawn 4 separate user stories. Each has its own very specific user acceptance criteria, test plan, migration plan, etc.




**Design Process**

The design process is a thought exercise to solve a requirements challenge with a combination of business processes and data processing. There are two ways to do design work.

1. Individual developers are assigned stories in an agile environment and design their own solution to the problem. This works well when teams are co-located and the developers are sitting near other developers and the customer.
2. A central design team exists to coordinate a variety of different requests. This works well in a distributed environment or when the developers are separated from the customer or when consistency across a very large system necessitates someone dedicated to design work.

In our data subset use case let's assume a team does not employ scenario 1, because we know this will most likely lead to 4 different solutions for essentially the same requirement. If they are lucky, that team of co-located developers might meet and consciously recognize the 4 different approaches, picking one as the preferred solution. This is not guaranteed however, and in small projects, might not make a difference as MVP or speed to market is the primary goal.

As this is a paper rooted in the design process as a discipline, our focus is on the approach in scenario 2. There are issues with this methodology, where Agile can create inefficiencies. Depending on the prioritization of the various stories, code independent standalone solutions are possible. It is likely that the second division would benefit from functionality created for the first, and some synergies could be exploited. However, because user stories are coded over time, development sequencing is anathema to the overall process. We start coding based on the requirements for the division first interviewed and the entire QA and UAT test plan is tailored as such. Then there is the possibility for significant redesign and rework to modify that first solution when we later decide to scale the design for other groups: Scalability is not a concern when designing for a single requirement. If 7 divisions use the same functionality we make changes to the same source code through 6 iterations with a round of

incremental testing for each. Because the same code is modified, it creates challenges to migrating to production. Either we do it one division at a time with a round of smoke testing for each or hold implementation until the last group leveraging the original code base is complete: More inefficiency.

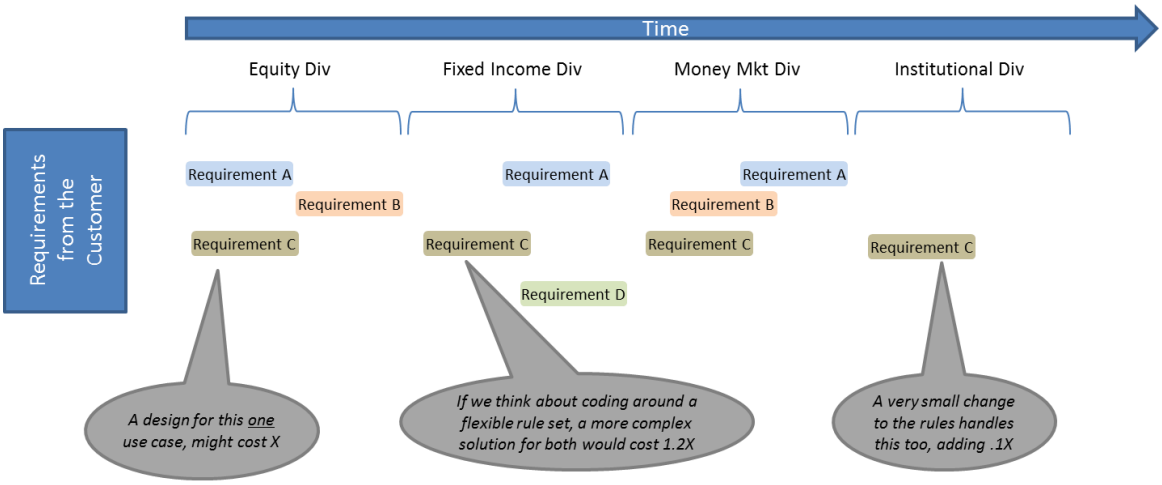
**A New Way**

Assume that your user would sacrifice customization in exchange for cost and speed to market. We should focus on core functionality and help the user unlock the potential of the platform. At the very least, assume that your customer would appreciate a cost-based choice of customization vs the ability to modify their usage of a tool based on an existing capability.

Consider an example from the past. Twitter in its original form was a text based platform and the 140 character postings contained few capabilities. The point was to post messages and the team focused on stability of the platform. The user community leveraged the platform and created de facto capabilities on its own. On August 23, 2007, the Twitter hashtag was born. Invented by Chris Messina (then with the consulting firm Citizen Agency, now an open web advocate for Google), the first tweet with a hashtag read as follows: "how do you feel about using # (pound) for groups. As in #barcamp [msg]?" (1) The user community developed the hashtag as a means to categorize tweets before Twitter ever developed a product around it. Today the Hashtag is ubiquitous to nearly every social media platform.

The power of the community using a product, trumps the unproven desire for specific features every time. It is a much better way to guide the backlog.

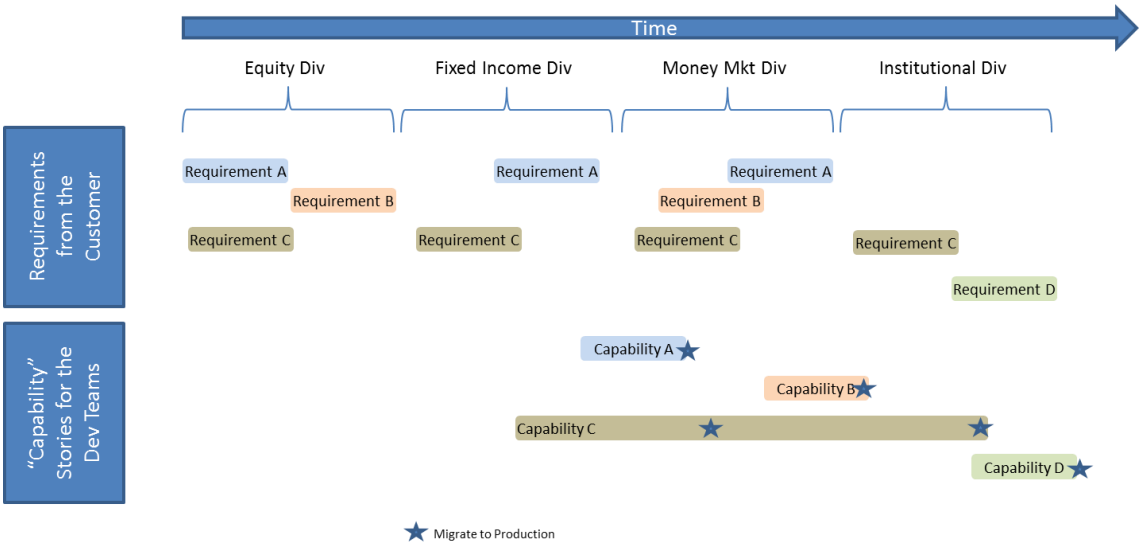
In our hypothetical use case, we would approach the process of creating development stories differently. As requirements come in and we see the overlap of a need for a "Data Subset & Weighting Engine" we note that as a core capability needed for the system. The fact that multiple areas will be using one shared capability makes this piece of work a priority: It provides the biggest bang for the buck. This capability is one story. We create a design for the capability that extends over multiple use cases. The design would focus on enabling the core data manufacturing capability, but scaled in a way that offers flexibility beyond the requirement. We continue to refine the design under an Agile process based on what we learn as we continue to work with the user community. However, as we meet with each new group to gather requirements, we start all discussions by telling them which capabilities are already in the works. In this case we inform them of the capability to allocate via flexibly defined data subsets. Helping the customer to understand how their requirements fit into the existing design helps us both understand their requirement and refine the design. The user knows the business process best. Further, there are always options to modify the business process that may be more cost effective than customizing code. The customer decides on the approach and funds its effort.




Design a Capability for Requirement C at a cost of 1.3X vs separate solutions at 4X

Regardless of the business cases for the Data Engine, the QA team can build test scenarios for the capability without needing to know all the required potential combinations. Success criteria are based on the capability to be delivered, which should be a superset of desired and unanticipated functional demand. In practice, the business may decide on more uses for this core capability than were intended when it was created (ie hashtag). Other Divisions might determine that their own unique system requirements can be best solved by the existing engine with a slight change in business process, thus negating any marginal development cost.

As part of UAT testing we train the user community how the engine works. The customer employs their own specific rules to affect data manipulation and evaluate the end result. It is the responsibility of the user to determine if their specific cases are satisfied by this capability. From a technical perspective, the focus is on delivering that capability, the usage of the tool is irrelevant. Success is based on the capability, not nuances of how the customer may use it.




## Conclusion

Enabling a development team to focus on a capability creates crystal clear criteria of what is acceptable for the definition of done. We are 'done' with that capability when its functionality is proven to work and is usable in production. More capabilities may or may not be necessary and can evolve during the design and development phases in an Agile framework. But the team is not bogged down with worrying about whether all of the user community's unique scenarios are completely solved by each capability. It may turn out that simple is better and some of those requirements are no longer needed. Whatever we do, make it fast and flexible and work with the customer to leverage each capability to its fullest potential.

Capability Based Design enables professionals who typically bridge the gap between the customer and the developer to clarify a functional design separate from the requirement that spurred it. The designer's role is to concoct solutions that are easily coded and solve most of the user requirements. If we profess to follow the 80:20 rule, we should accept that spending extra time accounting for every possible scenario in a requirements gathering session only promotes waste. The primary goal is to deliver incremental solutions that meet multiple objectives at scale. Thus we enable the customer to make a decision of whether they are willing to pay for further enhancements (customization) to each of those capabilities. This lets us deliver specific capabilities that scale at a cost effective price.

Do fewer things but make each thing do more than you can anticipate.

**Citations**

1) https://gigaom.com/2010/04/30/the-short-and-illustrious-history-of-twitter-hashtags/