Enhancing Scaled Agility with Use Case 2.0 and BDD Gherkin

Bernard F. Clark[1]

Introduction

In this article I base my observations and opinions on my experience of applying the Use Case 2.0 Practice and Behavior Driven Development's Gherkin language, within an online products division of a major US Bank that is undergoing an Agile transformation.

"I'm not dead yet," Is a classic line from the movies that Monty Python fans will instantly recognize.
I start with this because I could win a lot of money betting on the response from Agile practitioners when I tell them I am using Use Cases in an Agile environment to great benefit.

"Use Cases? They're dead and buried!"
"That's RUP! (Rational Unified Process). They aren't agile."
"What are you thinking? Use Cases are dinosaurs."
"You should know better, Bernie."

Rarely, I get a response from an experienced coach who will not poke fun, but seek the powerful questions such as, "Now why do you think that's a good idea?", and a valuable conversation ensues.

A Very Brief History

Use Cases were invented in 1986 by Dr. Ivar Jacobson for the purpose of defining the required interactions and responsibilities of a user of a system and that system, in order to achieve a specific goal; e.g. 'Open an Account'. You can think of them as large User Stories that contain multiple possible scenarios for achieving the goal or handling errors. In one scenario, the Use Case proceeds though each step without incident (Happy Path). In another scenario, the Use Case describes an Alternative Flow for handling a special circumstance. In yet another, the Use Case may describe how a failure condition is handled. Interestingly, Ivar considered the term "User Scenarios" before it was decided to call them Use Cases. Remember the word 'scenario' for later.

In a Use Case, the 'user' is referred to as an Actor and can be a human or another system playing the Actor role (some debate that Time can be an Actor too). Either way, these functional requirements describe 'what' the Actor must do and 'what' the System must do for each scenario and, if well-written, do not contain any design and implementation information, unless such a constraint IS a requirement itself. The important thing to note is that Use Cases are User-Centric; a very good thing, ahead of their time.

1 Bernard F. Clark <bernie.clark@comcast.net>

More than a document

A system's functional capabilities can be visualized by its Use Case Model, a schematic representation in UML (the Unified Modeling Language). In its basic form this gives a clear, immediate understanding of the functional scope of a system and who is using it.

The Use Case diagram of 'stick figures and bubbles' (diag. 1) is sometimes ridiculed by those who fail to understand the power of visualization and Model Based System Engineering (MBSE). If your agile team wants a BVIR (Big Visible Information Radiator) that conveys the functional scope of what you're building and for whom, a Use Case diagram stuck on the wall is a big help.

A Use Case Model *can* be augmented to show the relationships between reusable (factored) common requirements, traceability to business processes, responsibilities across system of systems, traceability to design and test models and more. Other UML diagrams can describe dynamic aspects of the Use Case (e.g. Activity Diagrams).

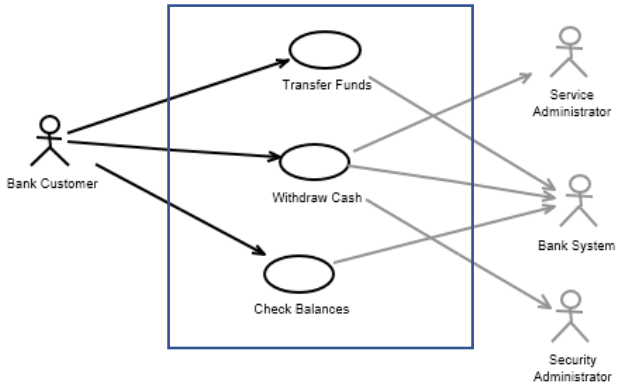


Diagram 1

I can almost hear agilists, complaining that this is a classic case of too much up-front work that violates agile principles. This can stop an initiative dead in its tracks as arguments rage about ways to model a system or which UML to use.

I agree.

However, I also ask skeptics to please bear with me when I claim there is merit in applying some of these techniques at the right time in the right way. Relationships need to be understood at the appropriate time when developing large systems (albeit a lightweight, agile way), and the agile community does not have to reinvent the wheel simply because such techniques have been around a long time. In short, "Don't throw the baby out with the bathwater."

For example, Customer Journey analysis and Story Mapping (all good stuff), have a strong correlation with 'Finding Actors and Use Cases', 'Telling Stories' and 'Slicing' from Use Case 2.0. My current client already has a very rich set of Actors and Use Cases for each Value Stream. This investment provides a great starting point for 'agile' collaborative discovery of scenarios when a new Portfolio-level Epic and its initial Features need to be prepared for planning.

Documents?

Use Case Documents are one of the greatest concerns for agile practitioners. Use Cases can be documented in a number of styles: Word documents in a step-wise narrative, Word tables, Excel spreadsheets or perhaps entered into a specialized tool. The esteemed Alistair Cockburn provided some of the most popular techniques for organizing the complex combinations of basic flows (happy paths) and alternative flows, exception flows, pre and post conditions et al. Thank you Alistair!

The Steps in a Use Case also provide an opportunity to reference any Business Rules (that can be maintained in a rules catalog). Likewise, any special NFRs (Non-Functional Requirements) that specify requirements such as Usability, Reliability, Performance, Scalability, Portability et al.) can be referenced.

Cockburn's numbering schemes of the Use Case documents were/are entirely logical, if somewhat complex, and the final works are frequently detailed, correct and, usually, complete masterpieces.

Not at all agile, of course, if you try and write it all up front!

It must be said that Ivar and his colleagues recognized this way before Agile became a term, let alone popular. Use Cases can, and should, be developed in stages. In RUP, Use Cases can be simply outlined and significant scenarios from an Architectural or Business risk perspective can be focused upon and delivered in early Iterations (yes, RUP has iterations) then other scenarios and Use Cases can be prioritized and added. Unfortunately, many business that I am aware of had a tendency to practice Big Upfront Requirements with signoffs and freezes. Very waterfall-ish… and helped give Use Cases (and RUP) a bad name, IMHO. Perhaps you can glean at this point, though, that the ingredients for an Agile approach were all there… and more.

What do we really need?

Quality, Speed to Market, Fast Feedback

As Agile exponents, we need to be able to deploy correct, useful functionality as soon as we need, not necessarily as soon as possible (Some customers don't want constant changes; e.g. users may need formal training for a new version).

To achieve this, we must be able to size our potentially-deployable (tested & demo'd) solution into small chunks that either fit into an iteration (sprint) or Kanban 'development cycle/iteration'.

To achieve that, we need to be able to describe and prioritize our needs in terms of useful, suitably-sized Scenarios. We have a choice as to how we discover and capture those Scenarios.

Finding Scenarios

If we are in a position where we have the luxury of organically growing the system scenario by scenario, then we might not have the need nor desire to capture the scenarios in an overall 'framework' such as a requirements model containing use cases, rules and NFRs.

BDD analysis techniques, Customer Journey analysis and Story Mapping may be all that is required to get the show on the road and we can work from stickies, cards or a spreadsheet.

The stories are the promise for a conversation where requirements will emerge. Perhaps we aren't concerned about missing infrequently-used, low-value scenarios. There might be no need at this stage to worry about implementing Business Rules consistently across systems because there are no other systems to worry about. Likewise, there might not be a need to think about which micro-services we should be reusing; architecture is emergent. We could rely on using BDD Gherkin and a tool like Cucumber to capture our requirements as automatable tests. Look in there and at the code if you want to know how it really works.

Managed Requirements

What if we are a medical device developer? By law, we must have Features documented as an 'intent' with an expected outcome, that can be used for Validation. We can't simply throw some ideas onto stickies and get after it, unless we are spiking and prototyping solutions that will not be deployed to production, and 'no', you can't document the intent after the fact!

Similarly, what if we are a major bank? We will likely be audited at some point, and a clear definition of WHAT the System is supposed to do is essential. We need a baseline of our requirements and a matching set of tests. We also need to make sure that complex Business Rules are consistently implemented across diverse channels and platforms. We usually don't want a different version of rules for the phone app versus the browser. We also may be implementing Micro-Services and need to know where they are integrated. In my experience, mining BDD Feature files alone, to understand the big picture and the functional structure of the system is not so straightforward.

Then suppose we are no longer the small start-up that was happy to go with the flow and handle all our requirements embedded in stories, rules being understood via conversation and institutional knowledge. We now have hundreds of people and multiple products referencing common business rules implemented across apps and browsers on a variety of devices and things are getting overlooked.

Use Case 2.0

The important thing to understand is that a Use Case is a structured collection of Scenarios. It always has been. What Use Case 2.0 does is provide guidance on how to structure the Use Case Flows and to ensure that each flow (scenario) has at least one Test Case.

Then we can batch one or more flows into an implementable Backlog Item, called a Slice. The Slice can be treated just like a User Story. Indeed, my current client expresses the Slice as a User Story in Jira. It can be sized and prioritized and the Test Cases become the Acceptance Criteria.

I have used the word 'Scenario' several times in this article. There may be some ambiguity that needs resolving. When I use 'Scenario', I am referring to a conditional path to a Use Case goal or a failed attempt to reach the goal. The condition(s) for entering a Flow are expressed in statements such as:

At [Examine Withdrawal Threshold], When the Withdrawal Amount would result in the Actor's Checking Account Balance being less than the Checking Account Balance Warning Threshold, ….flow steps go here. (e.g. The System provides a Warning Message).

The Scenario describes a Generic Instance of Use. If I want to define a concrete test and plug in an actual value for the Checking Account Balance Warning Threshold (e.g. $500), then I call this an Example or a Specific Instance of Use. For those that use Gherkin, the terms ‘Scenario’ and ‘Example’ are synonymous, which irks me. I prefer to be able to explain it thus: “In this Scenario, the intended withdrawal amount is going to cause the account holder’s balance to fall below a preset threshold and the system has to provide a warning. Here’s an Example …. “.

The majority of requirements for my client are enhancements or additions that have a context within the existing functional offering. By using the existing Use Case structures, it is relatively easy to identify impacted or new scenarios. As user stories are elaborated and delivered during a sprint, the Use Case provides a place to persist the emerging requirements for future reference (e.g. next version, audit) rather than embedding them in Jira stories that get lost over time.

With Use Case 2.0, my client’s Business Analysts (providing assistance to Product Owners) can start with a very lightweight, simple bulleted outline of just the Happy Path and make that into a Slice and get the team going. They can then ask (just like BDD advocates) what else could happen at any given step in that Flow and elaborate the Use Case with additional Scenarios as needed. The system is built incrementally by value on cadence and can be deployed on demand just like any agile system should be.

A tool such as Jama can be used to manage the baselined requirements for a given Value Stream (product line). The Baseline for a given Release is brought into a Managed Workspace, where the impacted or new Scenarios that have been identified via agile analysis can and focused on without the need to manage an unwieldy Use Case document.

Adding BDD Gherkin (Given, When Then)

One of the challenges I used to have with Use Cases was ensuring that the Post Conditions were complete. All too often the Post Conditions only applied to the Happy Path. What about the nuances of a successful Alternative Flow? What about the outcome from an Exception?

Likewise, I have always had problems with ambiguity regarding Acceptance Criteria in User Stories.
The arguments run the gamut from,
“Just put the high-level criteria in there; use 3 to 5 bullet points’ to define the scope of the story,”
to
“The Acceptance Criteria ARE the requirements. Put it all in there!”
It can become a religious war.

When we create a User Story, or a Use Case Slice (They are exactly equivalent), we don’t want to do Big Upfront Requirements; we want the right amount to facilitate Sizing and Prioritization (The detailed Conversation will ensue if necessary). To do that, we need Acceptance Criteria that describe the Scenario in generic, not specific terms.

The Gherkin language provides a relatively simple syntax that can describe:

1. The necessary pre-conditions or Context for the Scenario (GIVEN).
2. The conditions that cause the Scenario to be exercised (WHEN)
3. The expected Outcome (THEN)

In the example about Balance Threshold, the Acceptance Criteria might be written as:

GIVEN an ATM User has specified an amount to withdraw
WHEN the Calculated Post-Withdrawal Balance LT Checking Account Balance Warning Threshold
THEN provide the ATM User with a suitable warning message that their Checking Account Balance Threshold has been exceeded.

Note that there are no specific values here. The outcome is clearly understandable by anyone and all the necessary information is in place for the agile team to create specific Gherkin tests (and there will be several for this) that can be put into a tool like Cucumber for the creation of automated tests.

The best of both worlds

What is the advantage of using Use Cases in agile enterprise? Why bother with the perceived additional 'overhead' of Use Cases? Here are a few benefits:

- For organizations that already have made a valuable investment in Use Cases (+ Business Rules and NFRs), Use Case 2.0 provides a logical pathway to Agility.
- Can be combined with customer journey and story-mapping techniques to provide a consistent way to align and ensure coverage of Scenarios pursuant to a common User Goal.
- Emphasizes that the System is delivered in Incremental Slices that comprise layers from Business
  Process through to Deployment (MBSE).
- A baselined, persistent set of validated requirements that can serve as the starting point for common understanding of the impact of any new initiative (Epics and Features) that may change and/or add to the product offering.
- Produces backlog items that can be sized and prioritized, just like User Stories, or be expressed as User Stories.
- Use cases provide a readily-understandable 'framework' or context for Business Rules and NFRs.
- Supports lightweight outlines/activity diagrams to facilitate value-based elaboration.
- Provides clarity regarding the Acceptance Criteria for Scenarios.
- Scenario-based approach keeps Use Cases aligned with BDD tests to aid in test coverage

Applying in an Agile Enterprise

In a 'Scaled Agile' enterprise, opportunities often begin life as big ideas (Epics). The following diagram shows a stepwise approach to applying Use Case 2.0 and BDD Gherkin.

1. Product Manager has an Epic idea that ‘maps’ to a part of the existing business.

2. The Epic and proposed Features are entered in a tool such as Agile Central.

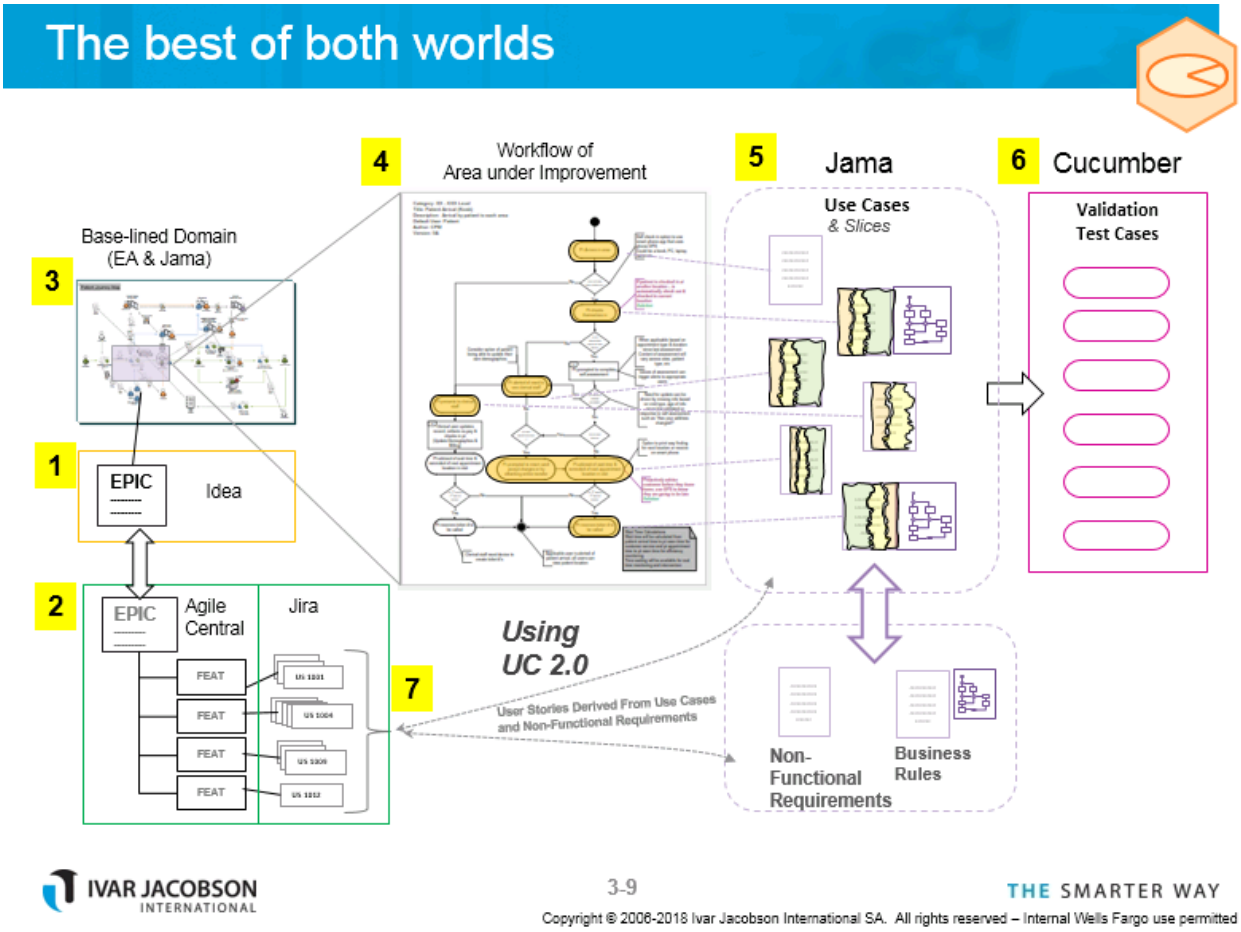
3 & 4. Before User Stories are created, the Business Context is analyzed and the Use Cases identified

5. The Use Case Outlines are analyzed and Key Scenarios (Flows) identified and combined into Slices.

6. Slices have Acceptance Criteria expressed in Gherkin that can spawn Validation Tests.

7. Slices are converted into User Stories and put into Jira. The Jira Stories are linked to Jira ‘Epics’ that bridged to the Agile Central Features.

Note: When User Stories are planned into Sprints for development, the Conversations lead to requirements being captured in the Flows in Jama, as well as BDD tests that go into Cucumber.



In Closing

In my experience with Use Case 2.0 to-date, I can cite very positive responses from agile teams. Here are a few examples.

“We have just started using UC2.0 in our project and it has been great! Discussing the use case scenarios and their impact on requirements has made our workflow smoother. From a design perspective, there is a better understanding of the work needed as well as the amount of work. I look forward to continuing using this process.” Senior Producer

“UC2.0 approach helped in defining the stories much better and it was possible to identify where we needed richer details which led to more efficient refinement sessions with Dev and QA. It also reduced the time spent on ambiguities and slicing stories to manageable stories that could be completed in one sprint.”

“Using UC2.0 is saving time by identifying impacted Use Cases before just diving into creating stories. Now we are INVESTING in our stories. They are well thought out for capturing our requirements.”
VP Project Management.

I hope you found this overview useful. If you need a logical framework to manage requirements in an agile way, perhaps because it is difficult to either find them in old Jira Stories or mine them from BDD feature files, I recommend that you take a hard look at Use Case 2.0 and Gherkin.
