User Stories

|  |  |
| --- | --- |
| **Author:**  **Created:**  **Updated:** | **8 November 2014**  **00 XXX 0000** |

# Contents

[1. Contents 1](#_Toc403249754)

[2. What’s in a Story? 1](#_Toc403249755)

[2.1. Introduction 1](#_Toc403249756)

[2.2. The structure of a story 2](#_Toc403249757)

[2.3. Telling the story 3](#_Toc403249758)

[3. The characteristics of a good story 3](#_Toc403249759)

[3.1. The title should describe an activity 4](#_Toc403249760)

[3.2. The narrative should include a role, a feature and a benefit 4](#_Toc403249761)

[3.3. The scenario title should say what’s different 4](#_Toc403249762)

[3.4. The scenario should be described in terms of Givens, Events and Outcomes 5](#_Toc403249763)

[3.5. The givens should define all of, and no more than, the required context 5](#_Toc403249764)

[3.6. The event should describe the feature 5](#_Toc403249765)

[3.7. The story should be small enough to fit in an iteration 5](#_Toc403249766)

[4. So how is this different from Use Cases? 6](#_Toc403249767)

[5. Summary 6](#_Toc403249768)

# What’s in a Story?

Behavior-driven development is an “outside-in” methodology. It starts at the outside by identifying business outcomes, and then drills down into the feature set that will achieve those outcomes. Each feature is captured as a “story”, which defines the scope of the feature along with its acceptance criteria. This article introduces the BDD approach to defining and identifying stories and their acceptance criteria.

## Introduction

Software delivery is about writing software to achieve business outcomes. It sounds obvious, but often political or environmental factors distract us from remembering this. Sometimes software delivery can *appear* to be about producing optimistic reports to keep senior management happy, or just creating “busy work” to keep people in paid employment, but that’s a topic for another day.

Usually, the business outcomes are too coarse-grained to be used to directly write software (where do you start coding when the outcome is “save 5% of my operating costs”?) so we need to define requirements at some intermediate level in order to get work done.

Behaviour-driven development (BDD) takes the position that you can turn an idea for a requirement into implemented, tested, production-ready code simply and effectively, as long as the requirement is specific enough that everyone knows what’s going on. To do this, we need a way to describe the requirement such that everyone – the business folks, the analyst, the developer and the tester – have a common understanding of the scope of the work. From this they can agree a common definiton of “done”, and we escape the dual [gumption traps](http://www.google.com/search?q=gumption+trap) of “that’s not what I asked for” or “I forgot to tell you about this other thing”.

This, then, is the role of a Story. It has to be a description of a requirement and its business benefit, and a set of criteria by which we all agree that it is “done”. This is a more rigorous definition than in other agile methodologies, where it is variously described as a “promise of a conversation” or a “description of a feature”. (A BDD story can just as easily describe a non-functional requirement, as long as the work can be scoped, estimated and agreed on.)

## The structure of a story

BDD provides a structure for a story. This is not mandatory – you can use a different story format and still be doing BDD – but I am presenting it here because it has been proven to work on many projects of all shapes and sizes. At the very least, your story should contain all of the elements described in the template. The story template looks like this:

|  |
| --- |
| Title (one line describing the story)    Narrative:  As a [role]  I want [feature]  So that [benefit]    Acceptance Criteria: (presented as Scenarios)    Scenario 1: Title  Given [context]    And [some more context]...  When  [event]  Then  [outcome]    And [another outcome]...    Scenario 2: ... |

## Telling the story

A story should be the product of a conversation involving several people. A business analyst talks to a business stakeholder[1](http://dannorth.net/whats-in-a-story/" \l "fn1) about the feature or requirement, and helps them to frame it as a story narrative. Then a tester helps define the scope of the story – in the form of acceptance criteria – by determining which scenarios matter and which are less useful. A technical representitive will provide a ballpark estimate of the amount of work involved in the story, and to propose alternative approaches. Many great ideas for systems come from the people developing them as well as the people who asked for them in the first place.

This will likely be an iterative process. The stakeholder will have an idea of what they want but will usually not know how much work will be involved, or how that work will be allocated. With the help of the technical and testing experts, they will understand the cost/benefit trade-off of each scenario and can make a judgement about whether they want it. Of course this is also balanced up against the other requirements: is it better to cover more edge cases in this story, or to move on to another story?

Sometimes the development team will simply not know enough to be able to make even a ballpark estimate. In this case they may choose to carry out some investigative work, known as a “spike”, in order to understand more about the requirement. (I will cover planning in more detail in a future article.)

# The characteristics of a good story

Using the example from the article [Introducing BDD](http://dannorth.net/introducing-bdd), let’s look at the requirements for withdrawing cash from an ATM:

|  |
| --- |
| Story: Account Holder withdraws cash    As an Account Holder  I want to withdraw cash from an ATM  So that I can get money when the bank is closed    Scenario 1: Account has sufficient funds  Given the account balance is \$100   And the card is valid   And the machine contains enough money  When the Account Holder requests \$20  Then the ATM should dispense \$20   And the account balance should be \$80   And the card should be returned    Scenario 2: Account has insufficient funds  Given the account balance is \$10   And the card is valid   And the machine contains enough money  When the Account Holder requests \$20  Then the ATM should not dispense any money   And the ATM should say there are insufficient funds   And the account balance should be \$20   And the card should be returned    Scenario 3: Card has been disabled  Given the card is disabled  When the Account Holder requests \$20  Then the ATM should retain the card  And the ATM should say the card has been retained    Scenario 4: The ATM has insufficient funds  ... |

As you can see, there are a number of scenarios to consider, some related to the account balance, others about the card and yet others about the ATM itself. Let’s dissect the story to determine whether it is any good.

## The title should describe an activity

The title of the story, “Account Holder withdraws cash”, describes an activity that the account holder wants to carry out. Until we implement this feature, the account holder won’t be able to withdraw money from the ATM. Once we have delivered it, they will. That gives us an obvious starting point for determining what “done” looks like.

If we had a title like “Account management” or “ATM behaviour”, we would have to look harder to understand when we were finished, and the edges would be a lot more fuzzy. For instance, “account management” might incorporate applying for a loan, and “ATM behaviour” might encompass changing the PIN number on my cash card. The story title should always describe actual behaviour by a user of the system.

## The narrative should include a role, a feature and a benefit

The template *“As a [role] I want [feature] so that [benefit]“* has a number of advantages. By specifying the role within the narrative, you know who to talk to about the feature. By specifying the benefit, you cause the story writer to consider *why* they want a feature.

It gets interesting if you find the feature won’t actually deliver the benefit attributed to it. This usually means you have a missing story. There is one story with the current feature, which delivers a *different* benefit (and is therefore still useful), and a hidden story whereby you will need a *different* feature to deliver the benefit described.

The example story tells us there is an Account Holder who *cares* about the feature being delivered, so we know where to start exploring what it should do.

## The scenario title should say what’s different

You should be able to line up the scenarios side by side, and describe how they differ using only the title. In our example, we can see that the scenario descriptions say *only* what’s different between each scenario. You don’t need to say “an account holder withdraws money from an account with insufficient funds and is told they are unable to fulfull the transaction”. It’s obvious from the title whether this is the scenario you care about, compared to the others.

## The scenario should be described in terms of Givens, Events and Outcomes

This is the single most powerful behavioural shift I have seen in teams adopting BDD. Simply by getting the business users, the analysts, the testers and the developers to adopt this vocabulary of “given/when/then”, they discover that a world of ambiguity falls away.

Not all scenarios are this simple. Some are best represented as a sequence of events, described as: *given [some context] when [I do something] then [this happens] when [I do another thing] then [this new thing happens]* and so on. An example is a wizard-style website, where you step through a sequence of screens to build up a complex data model. It is perfectly appropriate to intermingle sequences of events and outcomes, as long as you get into the habit of thinking in these terms.

One interesting emergent behaviour is that the quality of the conversation changes. You will quickly discover that you have missed out an assumed given (“well *of course* the account is overdrawn”), or forgotten to verify an outcome (“well naturally the account holder gets their card back”). I observed this on one particular project where the lead developer told me he could sense the analysts and developers were talking at cross purposes, but couldn’t see a way to demonstrate this to them. Within a few days of introducing the given/when/then vocabulary, he could see a dramatic improvement in the quality of their interactions.

## The givens should define all of, and no more than, the required context

Any additional givens are distracting, which makes it hard for someone looking at the story for the first time – whether from the technical or business side – to understand what they need to know. Similarly any missing givens are really assumptions. If you can get a different outcome from the givens provided, then there must be something missing.

In the example, the first scenario says something about the account balance, the card and the ATM itself. All of these are required to fully define the scenario. In the third scenario, we don’t say anything about the account balance or whether the ATM has any money. This implies that the machine will retain the card *whatever* the account balance, and *whatever* the state of the ATM.

## The event should describe the feature

The event itself should be very simple, typically only a single call into the production code. As discussed above, some scenarios are more complicated than this, but mostly the scenarios for a story will revolve around a single event. They will differ only in the context (the givens) and the corresponding expected outcomes.

## The story should be small enough to fit in an iteration

There are no hard and fast rules about how you do this, as long as you break it down into demonstrable chunks. In general if there are more than about five or six scenarios, a story can probably be broken down by grouping similar scenarios together.

We can’t tell from the ATM example how many more scenarios there are for this story, but I am suspicious that there may be several more. Essentially we have three “moving parts” in this story, namely the account balance, the status of the cash card and the state of the ATM. We could get into more detail with the cash card: what if it is out of date, so I can’t use it to withdraw cash but the ATM still returns it to me? What if the ATM breaks partway through the transaction? What if my account has an overdraft facility?

It might be better to break the story out into several smaller stories:

* Account Holder withdraws cash (assumptions: ATM is working and card is valid)
* Account Holder withdraws cash with invalid card *(assumptions: ATM is working)*
* Account Holder withdraws cash from flaky ATM *(assumptions: card is valid)*

Although this may seem artificial, it allows you to demonstrate progress in business terms and gives you more data points to track with. The important thing is always to break the story out along business lines by scenarios (and making the assumptions explicit) rather than along technical lines (e.g. doing the database stuff in this iteration and the GUI stuff in the next iteration). This is so the business can see demonstrable progress on their own terms rather than just taking your word for it.

# So how is this different from Use Cases?

There are use cases and there are use cases. I’m a huge fan of the way Alistair Cockburn describes use cases (as opposed to the over-engineered versions I have encountered on RUP-as-waterfall projects). Given that I don’t have much experience on use case-driven projects, I’ll leave it to others to draw the comparisons.

Certainly I agree with his process of starting at the lower precision (of an outcome, or goal) and working towards higher precisions, taking in more exception scenarios as you go. In BDD, this means starting with the business outcome, and working through high level functional areas to drill into specific stories with acceptance criteria.

In reality, it doesn’t matter what your process is to identify and elaborate requirements. It’s fine for you to write requirements documents if that helps you organise your thoughts. It’s *not* fine, however, to pass those documents along as if they actually encapsulated all your thoughts, because they don’t. Instead, you should put the requirements document, or stack of use cases, to one side and start over defining stories from business outcomes, safe in the knowledge that your hard work has meant that you have all the answers in your head – or at least a good enough understanding to outline the work as you currently see it.

# Summary

Behaviour-driven development uses a story as the basic unit of functionality, and therefore of delivery. The acceptance criteria are an intrinsic part of the story – in effect they define the scope of its behaviour, and give us a shared definition of “done”. They are also used as the basis for estimation when we come to do our planning.

Most importantly, the stories are the result of conversations between the project stakeholders, business analysts, testers and developers. BDD is as much about the interactions between the various people in the project as it is about the outputs of the development process.