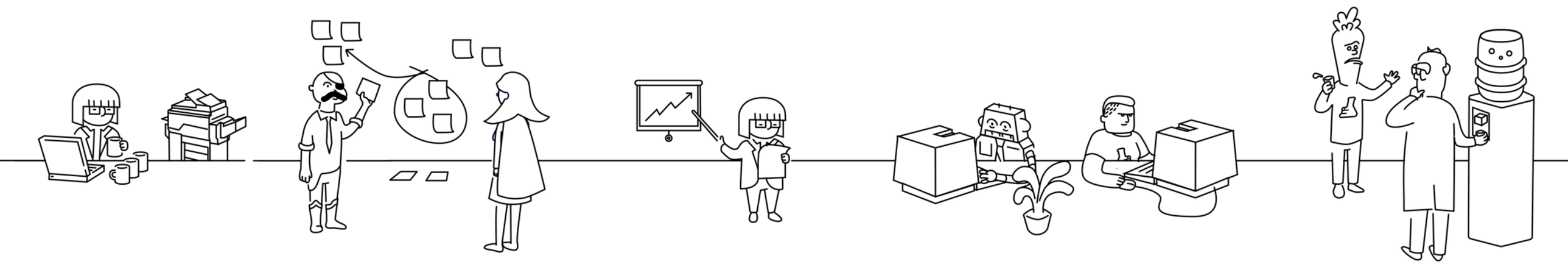
TM4J

The complete guide to Behaviour Driven Development (BDD) inside Jira

Adaptavist

Contents



Introduction to BDD

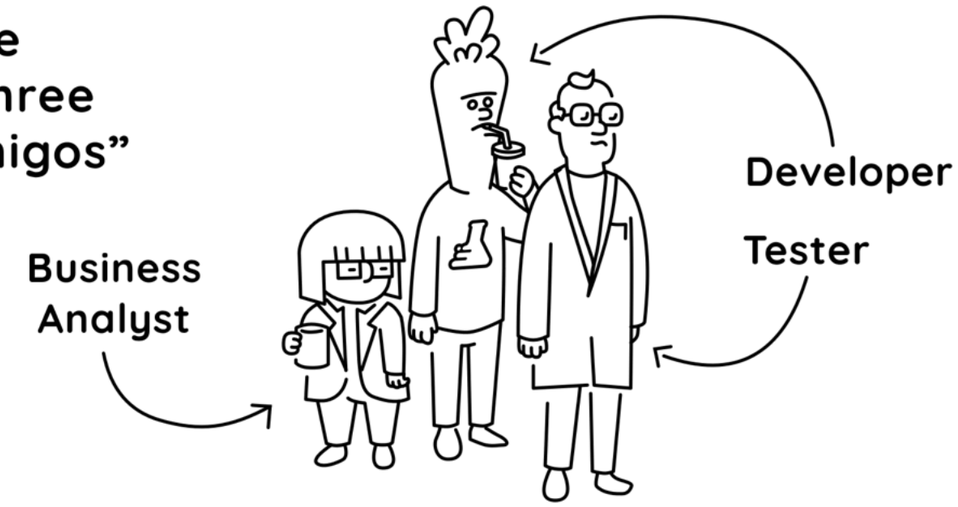
The BDD Philosophy

BDD, or “Behaviour-Driven Development” is an Agile development concept initially developed by Dan North in the mid 2000s. It emerged from an earlier method called TDD or “Test-Driven Development”.

In TDD, the focus was on ensuring that any user stories were designed and developed to allow easy and comprehensive testing. It was a great way for developers and testers to work together and communicate requirements. However, the Agile philosophy with its focus on team cohesiveness, communication and user-centricity, needed a more comprehensive way tocommunicate requirements that also included the business side.

Hence was born the concept of the “3 Amigos”, or the Developer, Tester and Business Analyst, who all need to be in sync when defining, developing and testing requirements and making sure they cover user expectations. This was the start of BDD, which is essentially a “language” in which to define test cases associated with user stories, that makes them understandable to all three team members.

The focus of the test case is to describe a behaviour the user should see when performing a certain action and prioritise those behaviours that contribute and meet business goals based on user needs. It has its own syntax and grammar, but is otherwise tool-agnostic.



Many BDD tools and frameworks have emerged over the years, perhaps the bestknown of which is the **Cucumber test automation tool** with its Gherkin language.

The main purpose of these tools is to link BDD cases to test automation tools so that the process can be streamlined as part of the Agile development process.

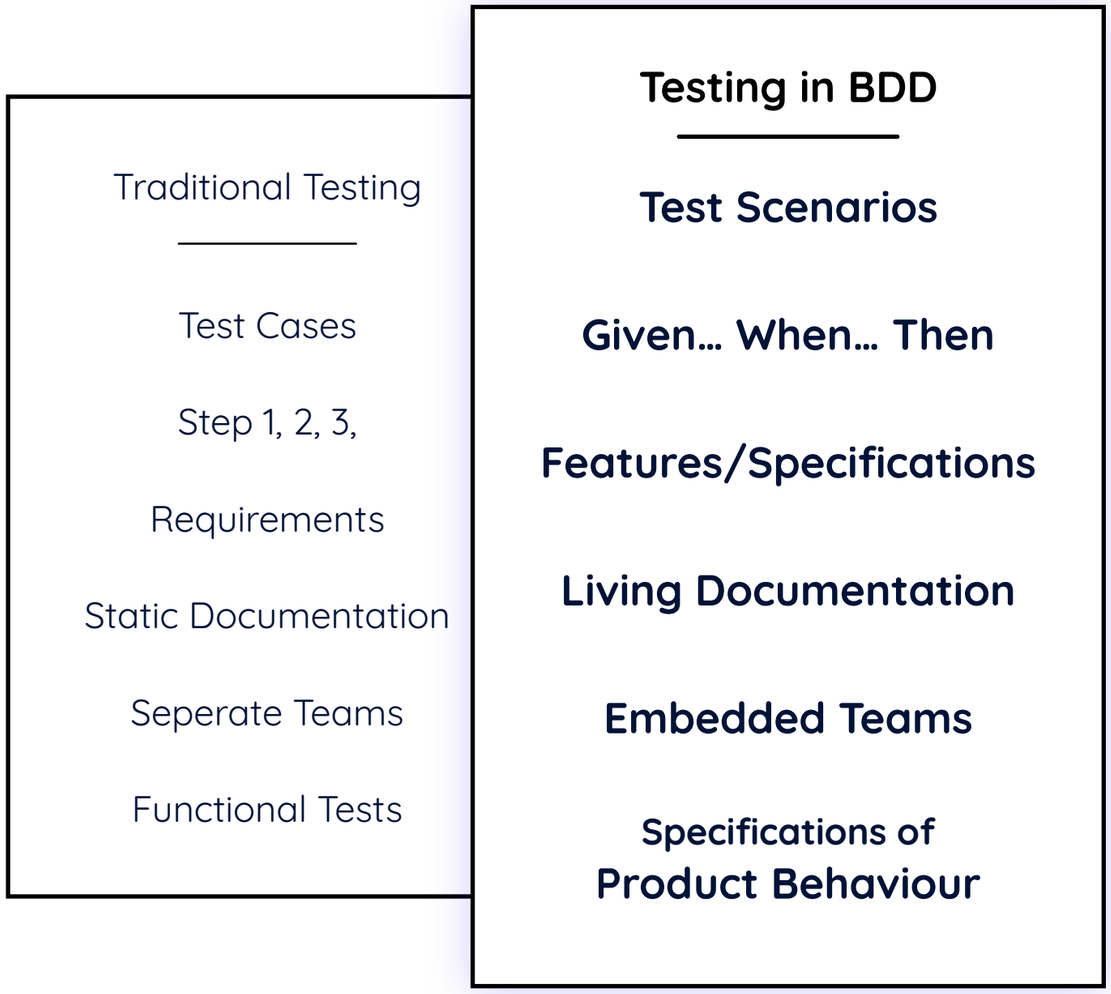
BDD vs other methods

BDD is part of a suite of Agile development methods that also includes the likes of TDD and ATDD (Acceptance Testing-Driven Development). All of these methods rely on a “test-first” concept, meaning that unit tests are written first (which would fail under the current state of the code), then enough code is developed to pass the test.

After that, some refactoring will be required to maintain code hygiene. Once that is done, another test is developed until all desired functionality has been covered. BDD operates at a slightly higher level than TDD by prioritising user actions and expected behaviour.

Both TDD/ATDD and BDD work best within a Continuous Integration (CI) setup which manages the automated build, deployment, testing, and feedback of the entire software multiple times a day, thereby minimising the risk of breaking behaviours or features due to the frequent feedback and changes.

So if you are considering using BDD, it is strongly recommended that you combine it with automated testing and CI processes and tools. One great advantage that BDD offers over simple unit tests is that it makes the test outcome independent of the implementation of the unit test of the function or module to be tested.



This achieves two main gains: it makes the code easy to change, and ensures that any assumptions made in the implementation (a default value for a variable for instance) do not have any bearing on the outcome of the test. It is of course possible to use Agile methodologies with simple unit testing without BDD or TDD and many teams do so to great effect.

Specific cases where you would not want to use BDD include times when you are developing a rapid prototype that is not expected to have high test-coverage but is mainly used to demonstrate basic functionality. Also, if CI is not part of your development process yet, using BDD might not yield particularly impressive results given the effort the team needs to put into it. It is possible, even recommended by some, to use BDD in combination with TDD or simple unit testing.

BDD in practice

The new version of Test Management For Jira (TM4J) now supports BDD using the Gherkin syntax. A typical Gherkin script takes the following form:

Given <an initial context>

When <an event occurs>

And <another event occurs>

Then <produce an expected outcome>

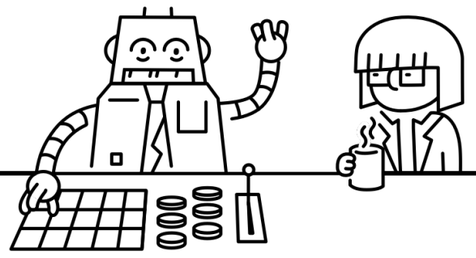
This script is linked to a user story as a test case, and the story is considered ‘Done’ when it passes all test cases via manual or automated tests.
For more details on the Gherkin syntax, please consult the Gherkin reference: https://docs.cucumber.io/gherkin/reference/

BDD in Jira

Automating your BDD process

When using BDD, you ideally want maximum integration with your existing tools, so the ideal BDD setup would include integration with test automation and CI tools as mentioned above, and would also integrate with your current Agile Management tool, e.g. Jira.

This minimises manual work and potential errors and ensures a seamless transition from requirements definition all the way to 'Done' status. The following sections will explain how to do that in Jira using TM4J.



Planning for BDD

Make sure to allow for more time for communication or feedback between team members on BDD cases and syntax, and also to sync up after test cases have been executed and share insights and results, and check the resulting behaviour together.

As a Scrum Master or Product Owner, make sure you build additional time into your sprints for the first few weeks after you introduce BDD as some test cases will inevitably have to be re-written or re-tested.

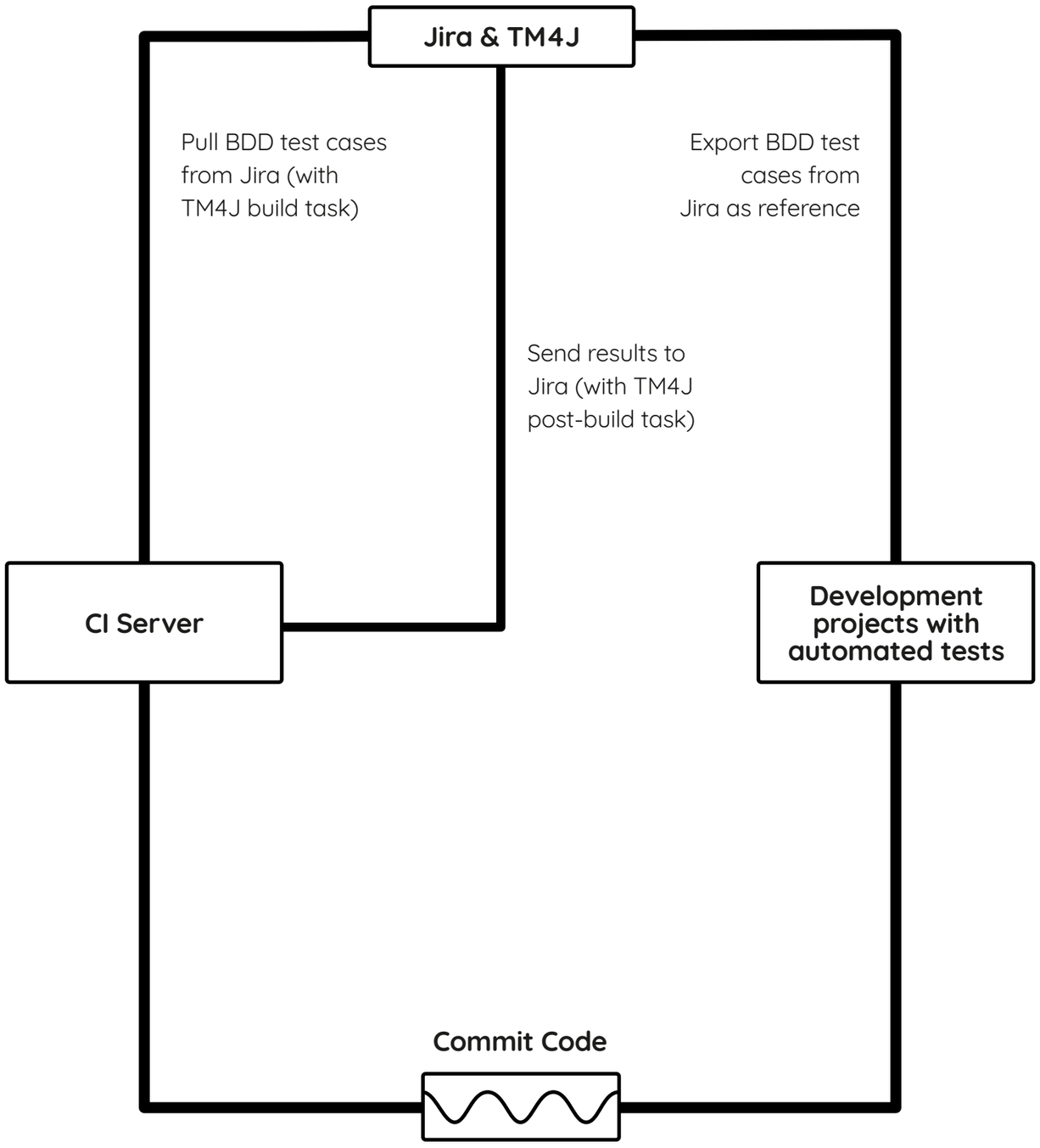
At this stage, you also need to decide whether you are going to automate your testing process. If so, you will need a test automation tool such as Cucumber. This will need to be installed and configured prior to creating your BDD test cases in Jira. In addition, you might want to use a CI tool such as Jenkins to streamline the entire process of test generation, execution, and deployment.

Once you have decided to adopt BDD in your team, it is important to bring everyone up to speed. Running a quick team session to ensure everyone is familiar with the concept and the syntax, choosing a tool that supports BDD syntax and a tool that connects it to Jira will be your main tasks here. It is important to recognise that like any new method, BDD will take some getting used to.

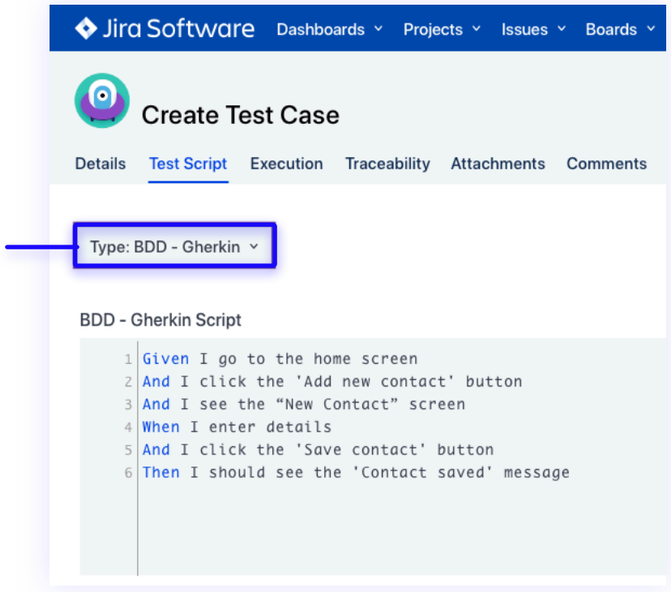
Implementing your BDD workflow

When using BDD, you ideally want maximum integration with your existing tools, so the ideal BDD setup would include integration with test automation and CI tools as mentioned above, and would also integrate with your current Agile Management tool, e.g. Jira.

This minimises manual work and potential errors and ensures a seamless transition from requirements definition all the way to 'Done' status. The following sections will explain how to do that in Jira using TM4J.



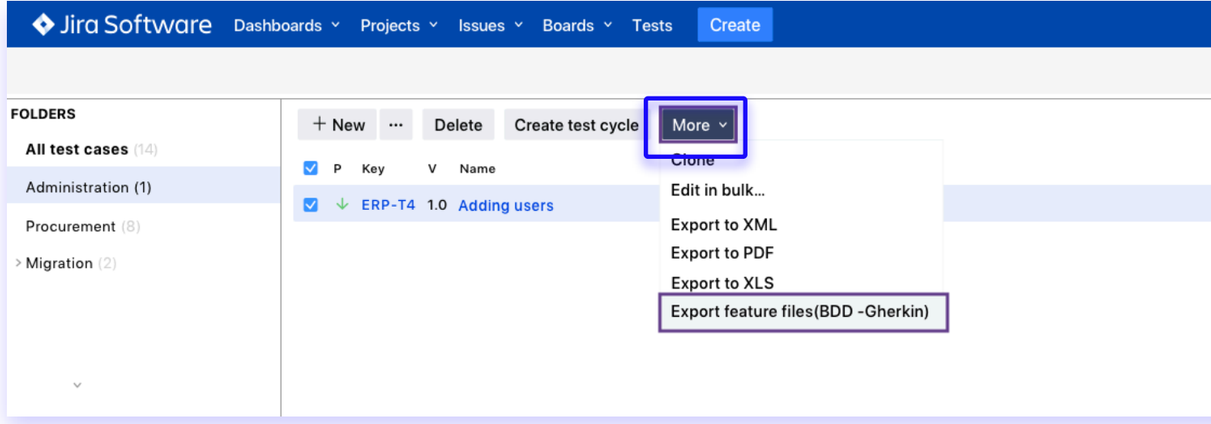
You can then start using the new feature by selecting BDD-Gherkin from the ‘Type’ dropdown menu when creating a new test case for your story.



Automating the BDD workflow still requires the use of an external test automation tool if automated testing is required, otherwise you can perform the test manually. Tools which parse the standard BDD description and link it to an automated test exist.

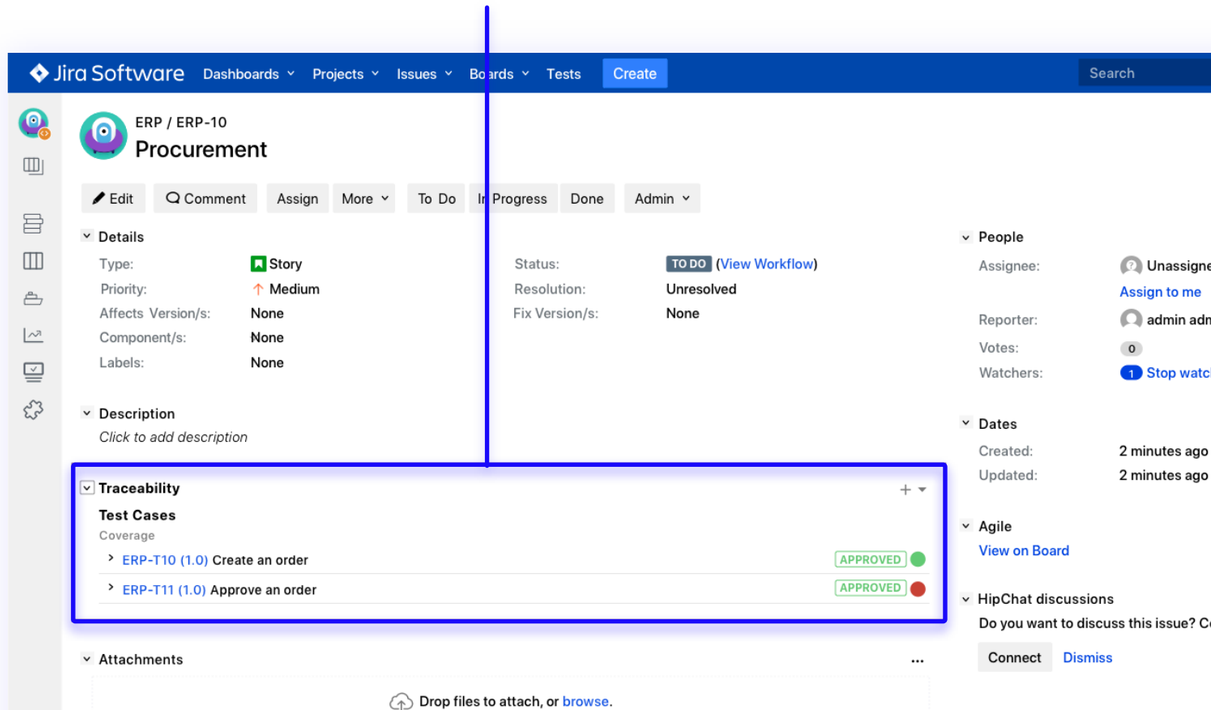
The automated testing tool will be triggered by the ‘given’ and ‘when’ statements, open the software, execute the test and record the outcome, then compare it to the description in the ‘then’ portion of the script. If the behaviour checks out, the test is passed.

The recommended best practice is for the “3 Amigos” to sit down at the start of a sprint and agree on all BDD test cases by writing and reviewing the Gherkin scripts. Once these have all been signed off, they can be exported simultaneously by selecting them and choosing “More” → “Export feature files (BDD-Gherkin), which generates a feature file parsable by Cucumber.



You will need to import the feature file into your automated test project and then implement the step definitions, which is the code that will execute according to the sequence of steps defined in the feature files.

Once the tests have been executed, the TM4J API, used for example by the Jenkins plugin, can re-import the test results back into Jira and publish them in the test case under the “Execution” tab, showing each test case and its status. A traffic light-style colour next to the test case will also show its status under the “Traceability” section of the main story page in Jira.



Tips and pitfalls

More than BDD

TM4J not only supports BDD but other test and development methods just like Waterfall or hybrid models. Just choose the ‘plain text’ or ‘step by step’ test case type and use your own language/ guidelines to describe the test case.

BBD can’t fix all!

Like any software development “philosophy”, BDD is not without its pitfalls. It is quite possible to write weak or unpassable tests in BDD, and the process can be frustrating to some teams.

Don’t clog up your stories with code!

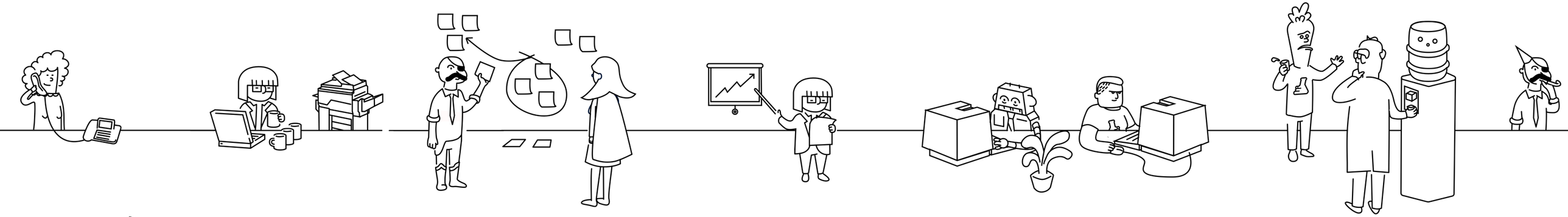
Another common pitfall is to get too technical and start adding code snippets or other technical details to stories. Given that the main point of BDD is to facilitate communication between technical and less technical team members, this can lead to even more confusion and frustration with the BDD process.

Dont dispense with documentation just yet!

Using BDD can sometimes entice teams to dispense with a requirements gathering and documentation process altogether, assuming that requirements will be captured in the stories in enough detail. Although this can work for some teams, it should not be assumed that any team using BDD for the first time will automatically be able to dispense with a formal documentation of requirements. There is no right or wrong answer here, so it’s important to experiment with keeping the old process, or possibly a hybrid of the two, before thinking about replacing it with BDD altogether.

Writing stories is still tricky!

BDD’s focus on user behaviour and system response can sometimes be a double-edged sword as it requires the involvement of end users in writing stories, or at least a very knowledgeable ‘user advocate’ such as a particularly customer-focused product owner. If access to end users is not available or not enough, the team might risk a misinterpretation and misimplementation of stories and test scenarios.



What before how!

Product owners or other team members using BDD for the first time can sometimes fall into the trap of describing implementation details in the story, ie specifying the ‘how’ not the ‘what’ of the desired user behaviour. This robs the developers of the opportunity to decide on the best technical solution to the problem at hand and can lead to inconsistencies across implementation in different parts of the system or even worse, problems when a desired behaviour needs to be changed further down the road.

Collaborate, collaborate, collaborate!

Finally, possibly the most dangerous pitfall of all is that BDD requires more collaboration than other, more traditional methodologies. Not all teams, especially developers, will be used to this amount of collaboration and some might complain that their time is being wasted communicating with users/product owners, testers, and other stakeholders when they should be spending more time writing code. This is a cultural and mindset shift that needs to take place and will require patience and coaching. Ultimately, it might result in some team members deciding to leave, or in the team deciding to abandon BDD altogether, but in most cases it can be overcome with time and practice.

For these pitfalls, we recommend two things:

Ease your team in

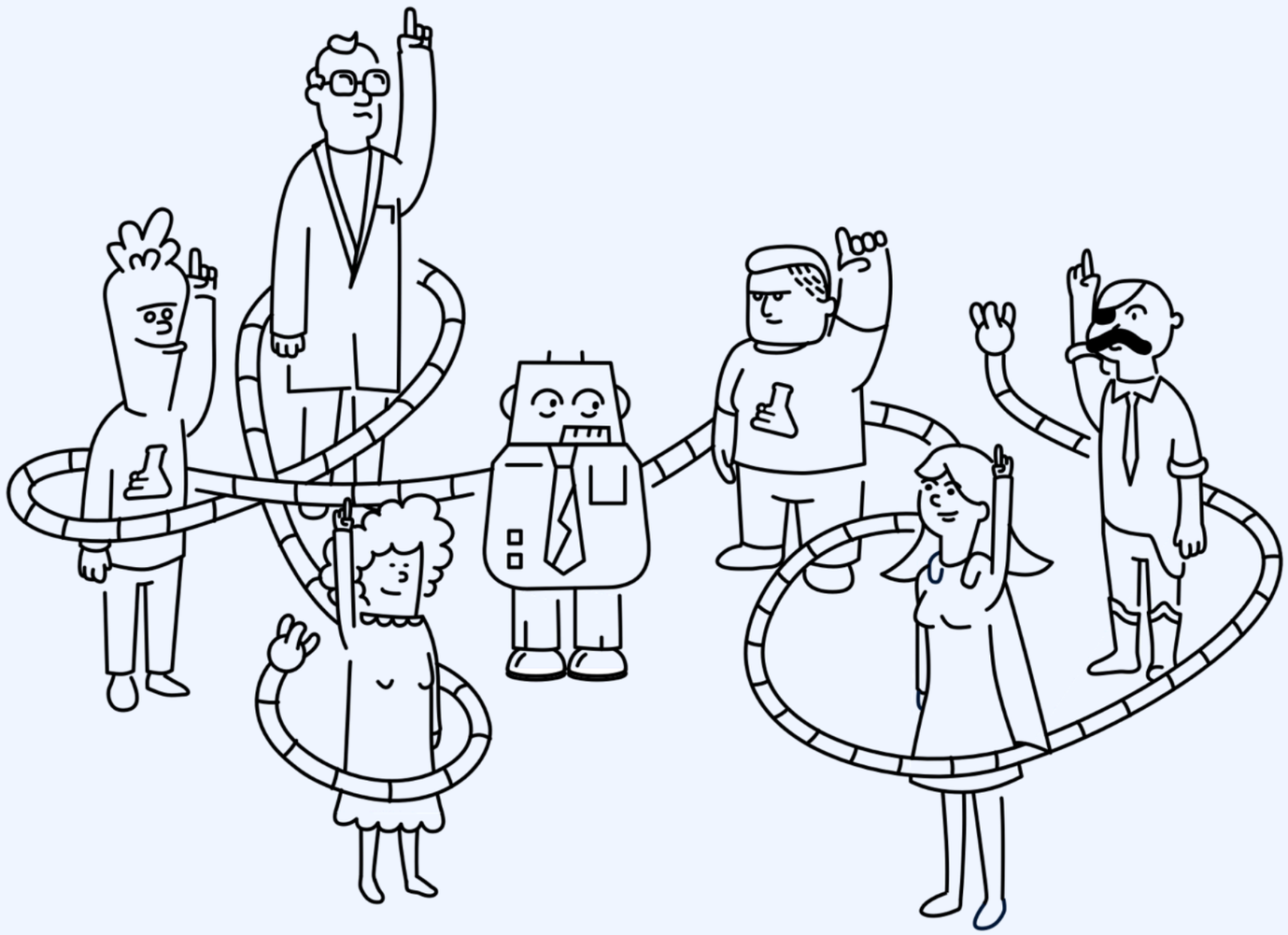
Try implementing BDD for a small part of a sprint first, maybe with the most straightforward story you can think of. Then, as the team get comfortable, gradually increase the use of BDD across all stories. Consult a BDD manual such as “BDD in Action” by John Ferguson Smart.

Get everyone speaking Gherkin

Understanding the BDD philosophy and ensuring the team have mastered the Gherkin syntax will go a long way towards improving your BDD performance.

Unlock BDD testing in Jira with TM4J

Install for FREE Now!



TM4J

Adaptavist