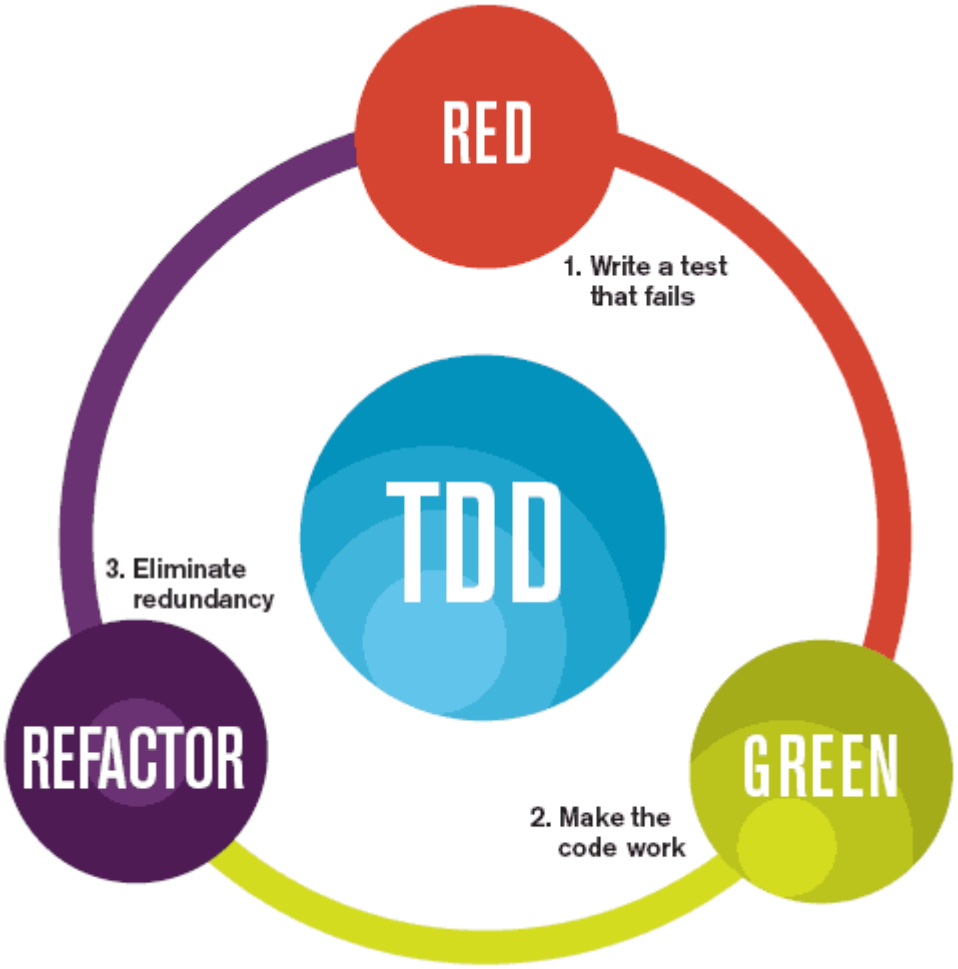
TDD (test driven development) and BDD (behavior driven development) are unique software development techniques that differ in what they are testing and how they are testing it. Despite their similar names, they serve distinct purposes.

What Is TDD?

TDD stands for test driven development. TDD is the process of writing a test for a specific portion of functionality, allowing the test to run to determine failures, and then adjusting the code as necessary to remedy the failures. Using these tests, developers can ensure that they've written functional, dependable code. Additionally, if other developers need to use components of the code, they can run tests to confirm their code's functionality.



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

TDD Process

There are several steps taken to execute test driven development:

1. The developer writes automated test cases to test the lines of code.
2. These tests are then executed to determine the location of any failures in the program.
3. Changes are subsequently applied (refactoring) to ensure that the failures are corrected and do not occur again in the future.

4. Finally, tests are executed again until they are passed without error.

Benefits of TDD

There are many benefits to using TDD instead of another style of development, such as:

- **Rework time reduced:** Test driven development does not allow new code to be written unless the existing code is successfully tested without failures. Until the failures are completely addressed and removed, the process of writing code is halted. Therefore, time spent reworking broken code is kept at a minimum.
- **Quick feedback:** As the tests focus on specific sections of code at a time, developers can receive more immediate feedback that allows them to implement changes more quickly.
- **Increased development productivity:** With TDD, the focus is on the production of functional code rather than test case design. As such, productivity increases, and development moves along smoothly.
- **More flexible, maintainable code:** Because every part of the code is tested before moving on to the next part of the software development process, the code maintains functionality and is adaptable in the future.

Test Driven Development Example

To better understand how test driven development works in development, it's helpful to go through an example. In this instance, we can define a class password to meet the condition that it must be 5–10 characters.

Your first step would be to write the code that fulfills the necessary requirements. Then you run the tests to ensure that the code is valid. For example, you create a class to test the password length, run the test, and then check to see if the output is true or false according to the conditions that you set—whether or not the length of the password is 5–10 characters. If the test returns false, then you can adjust the code as necessary.

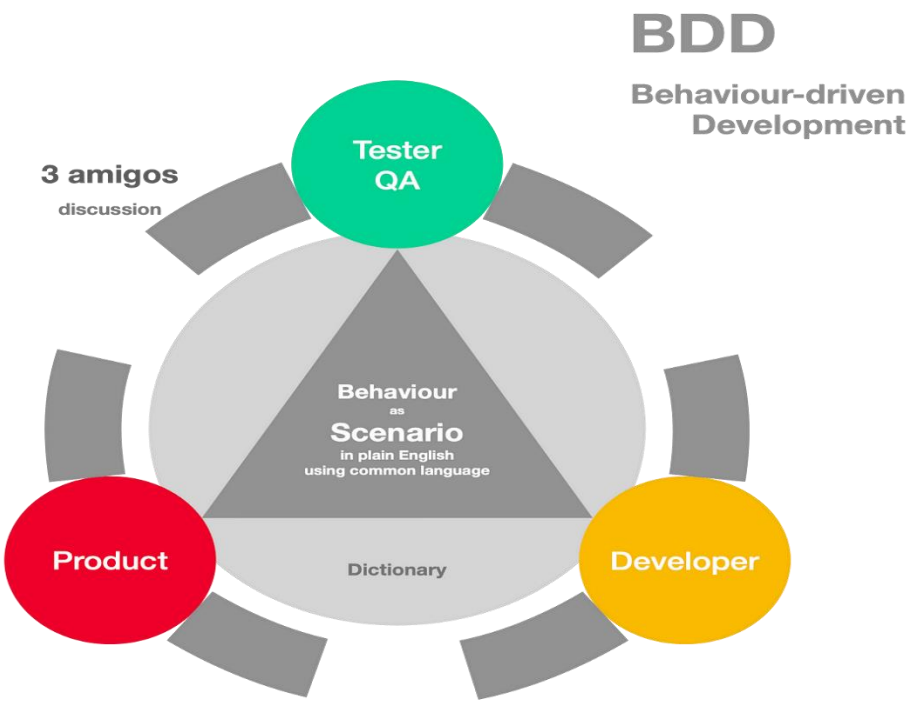
What Is BDD?

BDD stands for behavior driven development, and it is a way for teams of software developers and others to work together to narrow the distance between the business-focused team members and technical-focused people through:

- Encouragement of teamwork across roles to increase understanding of what the problem at hand is
- The use of rapid, small iterations of work done to advance feedback and enhance value flow
- Producing system documentation that is automatically checked against the system's behavior

BDD typically involves the software developers, test engineers, and project managers as well as other stakeholders. This group collaborates to develop concrete examples of acceptance criteria in a user story. Using a domain-specific language, these examples are

then described and put into a feature file. This feature file is subsequently converted into a specification that can be executed, thereby allowing developers to write actual tests that can also be executed.



BDD Process

The steps to behavior driven development are fairly simple and repeatable as necessary:

1. Behavior is described typically by utilizing a user story. This allows the team to discuss concrete examples of the new functionality so that everyone can agree on the expectations of the behavior.
2. Action is then written by turning the examples into documentation in such a way that it can be automated.
3. The test is executed to assist the developers and guide them through the development of the code.
4. The code is then created to make the action pass and make the code functional.

Benefits of BDD

There are several benefits to making use of BDD for software development, including:

- **Incorporation of user experience:** BDD focuses on the users' experience and, as such, allows the team to develop a broader perspective and make note of gaps in their understanding.

- **Cost-effectiveness:** Because BDD sets priorities for users, developers, and investors alike, it allows for resources to be used optimally in the development of programs.
- **Simple cross-browser testing:** BDD focuses on behavior, which means that it gives you an ideal framework for cross-browser testing.

Behavior Driven Development Example

Behavior driven development follows the "Given-when-then" framework. For example, if your site has a language translation feature and provides translations of the entire webpage in Italian, Spanish, and French, you could encounter scenarios such as the following:

- **Scenario**: Receive Messages in my Set Language (Italian)
    - **Given** I am the user "fmallo"
    - **When** the system sends the message "Invalid Login"
    - **Then** I should see the error message "Login non valido"
- **Scenario**: Change the language I use
    - **Given** I am the user "dmessina"
    - **When** I set my language to "Spanish"
    - **Then** my language should be equal to "Spanish"

Difference Between TDD and BDD

The primary differences between TDD and BDD lie in what is being tested and how. BDD predominantly focuses on the end user's standpoint in its testing of the application behavior, whereas TDD focuses on smaller sections of functionality to be tested by itself. Additionally, BDD involves a larger group of people—project managers, developers, and test engineers who come together to develop the behavior examples. As such, a great deal of communication is needed before anything is implemented. On the other hand, TDD can be done by a sole developer without external input from project managers or stakeholders.

Using Ranorex for Behavior Driven Development

Ranorex DesignWise has been built for a BDD framework. With DesignWise, you can create tests faster with less redundancy, less risk, and less coding. Then, you can important them into Ranorex Studio to run the BDD tests.

Using Ranorex for Test Driven Development

If test driven development is more suited for your project, Ranorex also has you covered. Whether you want to conduct functional testing or regression testing or any number of developer-led testing for your TDD, Ranorex Studio presents you with the ideal platform to automate all of your testing.

Improve Testing Automation with Ranorex

Ranorex Studio is a versatile and powerful tool for developers and novices alike, with an array of features that allow you to create and run tests without the need for complex coding or manual testing.

What Is Behavior-Driven Development?

Behavior-driven development (BDD) is a framework for software development in which teams create tests in plain language without code. Creating test scenarios in simple English allows all members of the development process to gain an understanding of the testing process and results.

Stemming from test-driven development, which allows developers to work with multiple selections of test data, this collaborative framework is a way to synthesize test-driven development practices and make them more accessible to everyone involved. Typically, it is implemented with Cucumber software and written using Gherkin language, which defines the behavior of applications. It has become increasingly popular among software testing companies due to its simplistic nature.

Using a Behavior-Driven Development Framework

A behavior-driven development framework uses conditional logic to perform tests, applying an “if, when, then” model. First, software testing team members and various stakeholders assemble to determine the purpose and desired outcome of a test run. They then define the requirements for the test runs as structured scenarios and use those scenarios as a guide for automated testing.

After implementing the scenario, developers revise and refactor the software as necessary, developing the functionality until the software passes the testing scenario successfully. Owners, testing teams, and stakeholders all must play an active role in the process of a behavior-driven development framework.

Advantages of BDD Framework

Easy Collaboration

Collaboration is one of the most integral benefits of BDD automation. As part of the behavior-driven development framework, communication is encouraged among members of the software development project every step of the way. Expectations for software features are clearly stated, as are the steps for making sure the software is functional. Third-party investors who have little knowledge of technology can remain informed about the testing process without the need for translation.

Simplicity

The straightforward nature of behavior-driven development is a huge draw to many software testing companies. An advantage of BDD frameworks is that behavior-driven development is easy for any novice to learn, regardless of whether they are a coder or entrepreneur.

Quick Feedback

Rather than implementing large test runs that test the entirety of a software product’s features, behavior-driven development runs tests in shorter iterations. This incremental progress results in more frequent communication among collaborators. Rapid feedback is an essential component of behavior-driven development, and it allows for continuous improvement with each new recalibration of a software product.

Centered on User Needs

BDD benefits development companies by centering the testing process on its intended audience. With user needs in mind, BDD makes the finalized product easy for users to understand, navigate, and operate. Thinking about user intuitiveness is an often missed aspect of software development, but the presence of non-technical members who can contribute ideas in the testing process gives a unique perspective that developers can utilize.

Fewer Test Runs in Less Time

The advantages of behavior-driven development testing include the efficiency of cutting out unnecessary tests. Only tests that are an essential part of the development process are run. The tests are purpose-driven with the intent to spot errors and correct them before deployment. Because more bugs are caught, there are fewer wasteful tests, saving software development companies time and money. It improves the quality of code, makes maintenance easy, and minimizes project risk.

Our Behavior-Driven Development Testing Tools

Seamless UI Automation

Automated behavior-driven development testing tools streamline the testing process by equipping software testing teams with the ability to create feature files for smooth test runs. Tests can be built without any programming or code. The benefits of BDD automation are numerous, including saved time and fewer test runs, but the most important is that companies receive a greater return on their investment. Behavior-driven development results in finished software products being deployed sooner and with fewer missed bugs.

Multiple Integrations

Ranorex testing software can be integrated with programs such as Jenkins, Jira, and TestRail. Complete continuous integration, rapid automation, and fast test run results make working with these tools significantly easier and accelerate the building process. Our AI-powered tools identify objects reliably and accurately so that intelligent code completion is functionally sound. Combining your go-to software development tools with Ranorex maximizes your resources, resulting in quality products.

Customizable Reports

Custom reports at the end of each test run give software testing teams the data they need and none of the information they don't. With customized test result reports, the BDD framework application is easy to apply, and the reports are easy to decipher. Users can receive reports for each test run or each test case, and they can filter reports by categories such as specific actions. Reports are sent conveniently to users' emails and can be generated as HTML, PDF, or JUnit files.

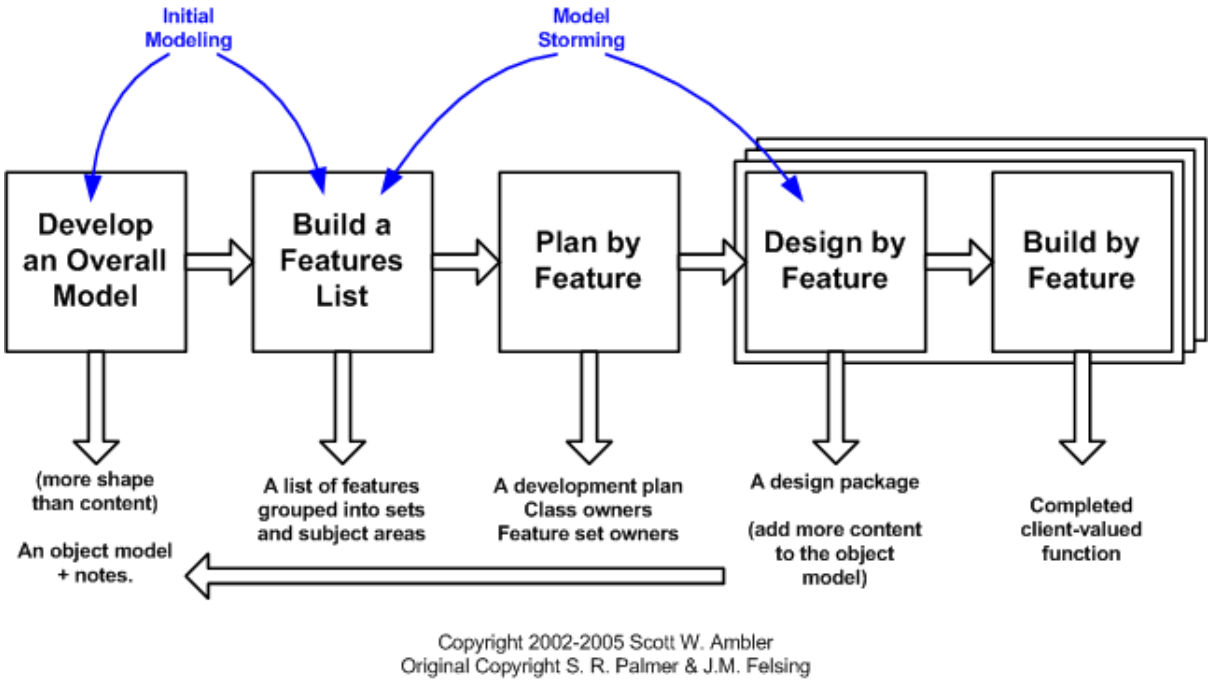
FEATURE DRIVEN DEVELOPMENT:

An Agile methodology for developing software, Feature-Driven Development (FDD) is customer-centric, iterative, and incremental, with the goal of delivering tangible software results often and efficiently. FDD in Agile encourages status reporting at all levels, which helps to track progress and results.

FDD allows teams to update the project regularly and identify errors quickly. Plus, clients can be provided with information and substantial results at any time. FDD is a favorite method among development teams because it helps reduce two known morale-killers in the development world: Confusion and rework./p>

First applied in 1997 during a project for a Singapore bank, FDD was developed and refined by Jeff De Luca, Peter Coad and others. The original project took 15 months with 50 people, and it worked; it was followed by a second, 18-month long, 250-person project./p>

Since then, it's become a pragmatic approach ideal for long-term, complex projects looking for a simple but comprehensive methodology. While Scrum and new variations of Agile are more widely recognized methods (especially outside of software development), FDD can be a good option for software development teams looking for a structured, focused Agile methodology that can be scaled across the product organization and will deliver clear outcomes.



How Does FDD Work?

Typically used in large-scale development projects, five basic activities exist during FDD:

- Develop overall model
- Build feature list
- Plan by feature
- Design by feature
- Build by feature

An overall model shape is formed during the first two steps, while the final three are repeated for each feature. The majority (roughly 75%) of effort during FDD will be spent on the fourth and fifth steps – Design by Feature and Build by Feature.

However, the difference is that once a goal has been identified, teams following FDD organize their activities by features, rather than by project milestones or other indicators of progress.

How is a Feature Defined?

In FDD, each feature is useful and important to the client and results in something tangible to showcase. And because businesses appreciate quick results, the methodology depends on its two-week cycle.

Stages of Feature-Driven Development

Stage 0: Gather Data

As with all Agile methodologies, the first step in FDD is to gain an accurate understanding of content and context of the project, and to develop a clear, shared understanding of the target audience and their needs. During this time, teams should aim to learn everything they can about the why, the what, and the for whom about the project they're about to begin (the next few steps will help clarify the how). This data-gathering can be thought of as stage 0, but one that cannot be skipped. To compare product development with writing a research paper, this is the research and thesis development step.

Once teams have a clear understanding of their goals, the targeted audience and their current (and potentially, future) needs, the first named stage in FDD can begin: Developing an Overall Model.

Develop an overall model

Continuing the research paper metaphor, this stage is when the outline is drafted. Using the “thesis” (aka primary goal) as a guide, the team will develop detailed domain models, which will then be merged into one overall model that acts as a rough outline of the system. As it develops and as the team learns, details will be added.

Build a features list

Use the information assembled in the first step to create a list of the required features. Remember, a feature is a client-valued output. Make a list of features (that can be completed in two weeks’ time), and keep in mind that these features should be purposes or smaller goals, rather than tasks.

Plan by Feature

Enter: Tasks. Analyze the complexity of each feature and plan tasks that are related for team members to accomplish. During the planning stage, all members of the team should take part in the evaluation of features with the perspective of each development stage in mind. Then, use the assessment of complexity to determine the order in which each feature will implemented, as well as the team members that will be assigned to each feature set.

This stage should also identify class owners, individual developers who are assigned to classes. Because every class of the developing feature belongs to a specific developer, someone is responsible for the conceptual principles of that class, and should changes be required to multiple classes, then collaboration is necessary between the owners of each to implement them.

And while class owners are important to FDD, so are feature teams. In feature teams, specific roles are defined, and a variety of viewpoints are encouraged. This ensures that design decisions consider multiple thoughts and perspectives.

Design by Feature

A chief programmer will determine the feature that will be designed and build. He or she will also determine the class owners and feature teams involved, while defining the feature priorities. Part of the group might be working on technical design, while others work on framework. By the end of the design stage, a design review is completed by the whole team before moving forward.

Build by Feature

This step implements all the necessary items that will support the design. Here, user interfaces are built, as are components detailed in the technical design, and a feature prototype is created. The unit is tested, inspected and approved, then the completed feature can be promoted to the main build. Any feature that requires longer time than two weeks to design and build is further broken into features until it meets the two-week rule.

Conclusion

Feature-Driven Development is a practical Agile approach suited for long-term, complex projects. It is a suitable choice for development teams seeking a simple but structured Agile method that is scalable and delivers predictable results.