UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Kaspar Raid

Developing and Conducting a Workshop on Approval Testing at Proekspert

Bachelor's Thesis (9 ECTS)

Supervisor: Dietmar Pfahl, PhD

Co-supervisor: Lars Eckart, MSc

Tartu 2021

Heakskiitmise testimise töötoa koostamine ja läbiviimine Proeksperdis

Lühikokkuvõte:

Bakalaureusetöö eesmärk on heakskiitmise testimise (ingl *approval testing*) töötoa koostamine ja läbiviimine ettevõttes Proekspert. Töötoa peamine eesmärk on õpetada osalejatele teadmisi ja oskusi heakskiitmise testimise rakendamiseks oma igapäevatöös. Osalejad saavad oskusi, kuidas muuta pärandkoodi paremini hallatavaks ning teadmisi heakskiitmise testimise eelistest ja puudustest võrreldes *assert*-lausetel põhineva testimisega. Bakalaureusetöö teoreetilises osas tuuakse ülevaade tarkvara testimisest ning pärandkoodi olemusest. Lisaks keskendutakse pärandkoodi ennetamisele ja tutvustatakse heakskiidu testimise metoodikat pärandkoodi paremini hallatavaks tegemiseks. Tagasisidele tuginedes võib öelda, et osalejad pidasid töötuba vajalikuks ning õppisid uusi oskusi igapäevatöös rakendamiseks.

Võtmesõnad:

Tarkvara testimine, pärandkood, töötuba, heakskiitmise testimine, ApprovalTests

CERCS: P175, Computer Science, System theory

Developing and Conducting a Workshop on Approval Testing at Proekspert

Abstract:

The bachelor's thesis at hand aims to develop and conduct a workshop on approval testing at Proekspert. The workshop's primary goal is to teach the participants the knowledge and skills to apply approval testing in their daily work. Participants will acquire skills to make legacy code more manageable and learn about the advantages and disadvantages of approval testing compared to assertion-based testing. The theoretical part of the bachelor's thesis provides an overview of software testing and the nature of legacy code. The focus is on the prevention of legacy code and the approval testing methodology is introduced to make legacy code more manageable. Based on the feedback, it can be concluded that the participants considered the workshop to be valuable and learned new skills for their daily work.

Keywords:

Software testing, legacy code, workshop, approval testing, ApprovalTests

CERCS: P175, Computer Science, System theory

Table of Contents

1. Introduction

Every software engineer, at some point in their career, has probably encountered legacy code. Legacy code has many different definitions. One way of defining is “legacy code is simply code without tests” [1:17]. Most existing projects have some degree of legacy code, which can grow over time. Working with legacy code becomes tedious and generally results in slower productivity while also increasing the risk of introducing unwanted behaviour. Adding new behaviour becomes increasingly tricky. Rewriting or refactoring is even more challenging so that existing behaviour is preserved.

Before adding new behaviour, it is essential to maintain existing behaviour [1]. A way to ensure that is to write automated tests. For this purpose, approval tests are suggested to capture the system's existing behaviour [1]. Once this is ensured, it will be safer to proceed with code changes.

The thesis at hand aims to put together workshop materials containing coding exercises and presentation materials. The primary purpose of the workshop is to teach the participants the knowledge and skills to apply approval testing in their daily work. Participants will gain skills on how to make legacy code more manageable through approval testing. They learn about the advantages and disadvantages of approval testing and how it compares to assertion-based testing. For practising approval testing, an open-source testing tool, ApprovalTests, is used.

The thesis contains five main sections. The second section provides an overview of software testing and its importance. A more detailed overview is provided of automated testing and its applications. The working principle of assertion-based tests is explained, and, in a situation where assertion-based tests may be challenging to write, approval testing is proposed. The third section provides an overview of legacy code and its possible interpretations. The risks associated with legacy code are described and how to minimise them by having automated tests. The fourth section provides an overview of approval testing and its use cases. In particular, the focus is on the usefulness of approval testing when dealing with legacy code. Approval testing advantages and disadvantages are identified compared to assertion-based testing. The fifth section provides an overview of the workshop design – target group, schedule, materials, tasks and execution. Finally, the feedback collected from the workshop participants is analysed.

2. Software Testing

2.1 Definition

Software testing is a process to verify that the software works as expected [2]. Software development usually begins with requirements analysis. The result of the analysis is an exact specification that the developed software must meet. To ensure that the software conforms to the specification, it must be verified during software testing. Software testing benefits include identifying bugs, assessing the performance and checking that added changes do not break existing behaviour [2].

2.2 Automation

Software testing can be done manually or automatically using scripted tests. Some of the advantages of automated testing over manual testing are the reusability of the tests and the speed of execution [3]. On the other hand, manual testing approach is more likely to discover new and hard-to-find bugs [4]. Automated tests are an excellent way to replace repetitive tasks, which frees up more time for manual testing [4].

Automated testing is a technique in which predefined tests are executed using dedicated frameworks. This provides a basis for better overall quality of software by making repetitive tasks automated. Automated tests are used to reduce time spent on tasks that would otherwise be performed manually. In today's agile software development world, it is crucial to have at least some kind of automation. Automation of test cases should be considered when they are [3]:

- Repeatedly executed
- Tedious or difficult to perform
- Time-consuming
- Business-critical

Although automated tests can be introduced at any stage of development, it is easier to start adding them gradually from the start. That makes it possible to increase the number of automated tests in line with software development, thus ensuring continuous test coverage.

One of the best ways of gradually building an automated test suite is by practising test-driven development (TDD). Test-driven development is a software building technique developed by Extreme Programming inventor Kent Beck [5]. Using this technique, the

software is developed by following the principle of always writing a test before adding new functionality. That ensures any added functionality is covered by automated tests.

2.3 Test Oracle

In the case of automated tests, it must be able to determine whether the tests were successful or not. One way is to write assertion-based tests. Assertion-based tests are widespread in today's software development. It is convenient to write assertion-based tests in applications if they are somehow included in the programming language used to create the application [6]. By its nature, an assertion is a function that is evaluated as true or false [7]. The function encapsulates the behaviour under test and determines the return value. If the returned value corresponds to the specified expected value, the function is evaluated as true. When testing software this way, it must be assumed that the program is working as expected. Then it is possible to write tests that validate the actual behaviour against expected behaviour. An assertion should validate a property of the program unambiguously [6]. Most commonly, assertions are written about the program's:

- Preconditions
- Post-conditions
- Return values

Although assertion-based testing is very popular, it can be challenging to use in some situations. While they are great at validating primitive output, it can be difficult to assert complex output. The program's architecture may not be very modular, resulting in different parts having too much responsibility and producing complex output. As a result, writing assertions becomes more tedious as the output becomes more complex. In this case, Llewellyn Falco, an Agile Technical Coach, recommends introducing approvals instead of assertions [8]. He is also the creator of the open-source testing tool ApprovalTests.

3. Legacy Code

3.1 Definition

Legacy code is a phenomenon without an official definition. It has many different definitions. However, legacy code is a common problem in software development. Some of the possible interpretations are [9]:

- Code without tests
- Code that is hard to change
- Code with poorly written tests
- Code that many people have developed over a long period

The legacy code problem comes out in situations when code needs to be changed. Some reasons for changing the code are adding new functionality, fixing bugs, improving the design, or optimising code [1]. For all of them, existing code is modified, or new code is added. When adding new behaviour, it is even more essential to maintain the existing behaviour as it forms most of the code (see Figure 1) [1].

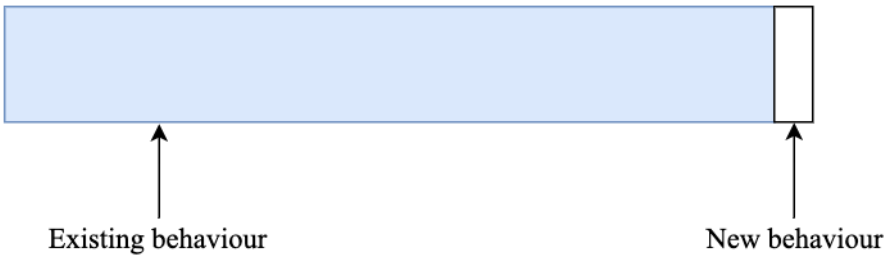


Figure 1. Behaviour distribution [1]

Because it is challenging to preserve existing behaviour, each change involves a risk [1]. In order to do that, the software needs to have some functionality to verify that. One solution is through automated tests. However, as mentioned earlier, legacy code is code without tests. According to the author, it can be concluded that maintaining legacy code becomes more challenging, which can result in a decline in productivity. Productivity can be measured in the ratio of lines of code and effort in hours spent [10]. As maintenance becomes more difficult, it takes more time to write the code. Software development slows down, which in turn hinders the achievement of business goals. Also, it increases the risk of software errors, which affects its end users.

3.2 Prevention

Most fear in adding new behaviour or modifying existing code comes from the lack of automated tests [1]. Nobody wants to inject bugs into existing code or inadvertently change existing behaviour. Without proper test coverage, it is hard to verify that the system is still working as expected. Michael C. Feathers [1] suggests that the most important thing is to get to a situation where you can be sure that you are making one change at a time. By increasing test coverage, there is a higher chance of detecting unwanted changes [1].

3.3 Improvement

The key rule is that before changing code, existing behaviour must be captured. This can be done with the help of a well-defined test suite that ideally covers every part of the system. A well-defined test suite must verify that code does what it should and detect when it does something it should not.

Feathers [1] came up with an algorithm to improve legacy code:

1. *Identify change points*

   Feathers emphasises mainly that these change points depend heavily on the architecture of the system. First, the system's structure should be made clear to make sure changes are made in the right place.

2. *Find test points*

   While finding places in the legacy code that need to be changed can be difficult, finding testing points can be just as tricky. Feathers suggests characterisation tests to lock down existing behaviour. Characterisation testing is also known as approval testing. An overview of approval testing is provided in the next section.

3. *Break dependencies*

   If there are a lot of dependencies between objects, it can be difficult to write tests. When one object needs another object, it transitively needs everything that the other object needs.

4. *Write tests*

   When change and test points are identified, it is time to write actual tests. At this point, the code changes must pass through the identified points.

5. *Make changes and refactor*

Once the tests coverings are in place, it is safer to start making changes. It is essential to run tests regularly. Also, Feathers recommends writing tests before making changes.

4. Approval Testing

4.1 Definition

Approval testing is known by several different names:

- Characterisation
- Snapshot
- Golden Master

It is a testing technique for deciding whether the test was successful by comparing the code's current output with a previously captured approved output [11]. When running an approval test for the first time, it is expected to fail due to not having an approved output. As a result, a received and approved file are created. The received file should contain the code's current output, and the approved file should be empty. By examining the received file, one can decide whether the output is expected. If so, the content can be approved by copying it to the approved file. When rerunning the test without changing code, the contents of the received and approved file are equivalent based on text comparison, and as a result, the test is successful.

4.2 Use Cases

As Michael Feathers also suggests, one of the primary use cases for approval testing is writing tests for legacy code [1]. He points out that in most cases, when testing legacy code, it is more important to determine what it does and not what it is supposed to do. The goal should not be to find all the bugs right now but to find bugs that emerge later. Approval tests document the current behaviour of the code, which help to detect unwanted effects of software changes. By having documentation of what different parts of the software do, it is possible to use it to compare it with what the system is supposed to do and make changes accordingly.

Emily Bache, a Technical Agile Coach and one of the contributors to the ApprovalTests tool, points out the following other use cases [12]:

- Code without tests that needs to be changed
- APIs that return JSON or XML
- Complex return objects
- Strings longer than one line

4.3 Composing an Approval Test

There are different tools for approval testing. One of them is the ApprovalTests tool, which is also used in the workshop. ApprovalTests or Approvals is an open-source testing tool created by Llewellyn Falco. It has support for many programming languages such as Java, C#, C++, PHP, Python, Swift, Node.js, Perl, GO, Lua, Objective-C and Ruby [13]. This tool allows simplifying assertion-based tests by replacing several assert statements with an Approvals method. Besides verifying single object output, the tool also provides methods to verify the collection of objects. The tool has dedicated methods to verify formats like XML, JSON or HTML. Parameterised tests can be written using CombinationApprovals methods, which allow the execution of a test case with different input parameters conveniently.

A typical assertion-based test consists of three steps: arrange, act and assert. It is also known as the AAA pattern [5] (see Figure 2).

```
@Test
void testAddition() {
    // ARRANGE
    int x = 1;
    int y = 2;
    // ACT
    int result = addNumbers(x, y);
    // ASSERT
    assertThat(result).isEqualTo(3);
}
```

Figure 2. Assertion-based test case

The same test case can be rewritten using the ApprovalTests Java tool by replacing the assert step with an Approvals method (see Figure 3).

```
@Test
void testAdditionApprovals() {
    // ARRANGE
    int x = 1;
    int y = 2;
    // ACT
    int result = addNumbers(x, y);
    // APPROVE
    Approvals.verify(result);
}
```

Figure 3. Approval-based test case

Execution of the approval test for the first time will fail due to not having an approved output. As a result, the received and approved file are automatically created in the directory of the test. ApprovalTests tool automatically displays the differences between the received and approved file using a diff-viewer tool (see Figure 4). Diff-viewer tools usually show

differences line-by-line, which makes it easier to detect differences. The tool used to show the differences can be configured through ApprovalTests configuration options.

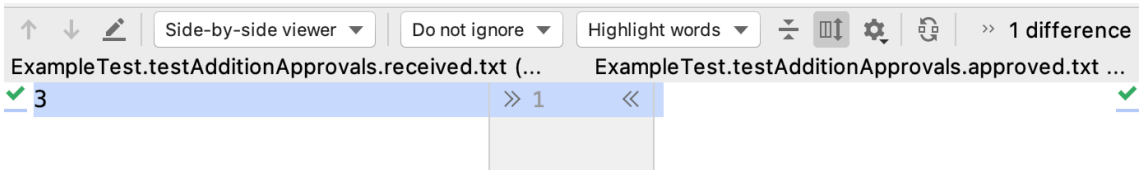


Figure 4. IntelliJ IDEA diff-viewer

By examining the content of the received file, one can decide whether the output is expected. If the output is expected, the test can be approved by copying content from the received file to the approved file. The diff-viewer tool allows copying the differences line-by-line conveniently. When rerunning the test case, the received and approved file are compared in the background. The test is successful if the content is equivalent.

4.4 Comparison to Assertion-Based Testing

Approval testing advantages and disadvantages can be identified by comparing it to assertion-based testing. To do this, they will be compared in the following categories: testing complex output and time spent for writing a test.

4.4.1 Complex Output

Complex objects require many assert statements to verify the output. As seen in Figure 5 Receipt object requires many assert statements to verify its values.

```
@Test
public void buyOneItem() {
    // ARRANGE
    SupermarketCatalog catalog = new FakeCatalog();
    Product apples = new Product("apples", ProductUnit.Kilo);
    catalog.addProduct(apples, 1.90);

    Teller teller = new Teller(catalog);

    ShoppingCart cart = new ShoppingCart();
    cart.addItemQuantity(apples, 2.5);

    // ACT
    Receipt receipt = teller.checksOutArticlesFrom(cart);

    // ASSERT
    assertThat(receipt.getTotalPrice()).isEqualTo(4.75);
    assertThat(receipt.getDiscounts()).hasSize(0);
    assertThat(receipt.getItems()).hasSize(1);
    ReceiptItem receiptItem = receipt.getItems().get(0);
    assertThat(receiptItem.getProduct()).isEqualTo(apples);
    assertThat(receiptItem.getTotalPrice()).isEqualTo(4.75);
    assertThat(receiptItem.getQuantity()).isEqualTo(2.5);
}
```

Figure 5. Assertion-based test case with complex output

The same test could be rewritten as an approval test by replacing assert statements with an Approvals method (see Figure 6).

```
@Test
public void buyOneItemApprovals() {
    // ARRANGE
    SupermarketCatalog catalog = new FakeCatalog();
    Product apples = new Product("apples", ProductUnit.Kilo);
    catalog.addProduct(apples, 2);

    Teller teller = new Teller(catalog);

    ShoppingCart cart = new ShoppingCart();
    cart.addItemQuantity(apples, 2.5);

    // ACT
    Receipt receipt = teller.checksOutArticlesFrom(cart);

    // PRINT
    String result = new ReceiptPrinter().printReceipt(receipt);

    // APPROVE
    Approvals.verify(result);
}
```

Figure 6. Approval-based test case with complex output

Instead of writing many assert statements, it is possible to store the output in a file and keep it out of the source code [12] (see Figure 7). A thing to notice is that a print step was added. That is necessary because complex output must be converted to a presentable format. The

ReceiptPrinter class converts the Receipt to a String seen in Figure 7. A printer can be a custom class like ReceiptPrinter or a utility method.

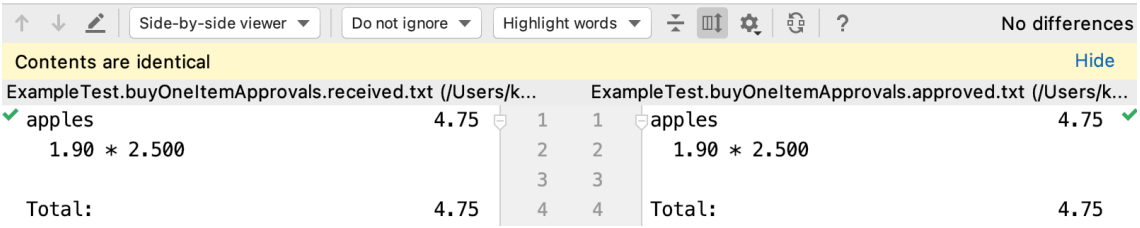


Figure 7. Received and approved output

Additionally, JSON and XML can be formatted consistently in a file, making it easier to compare against expected values [12]. Besides verifying that the values are expected, it is also possible to verify the entire output format. Code changes that cause the output to differ from the approved output will result in a test failure. However, a diff-viewer tool makes it easy to spot them (see Figure 8). Expected differences can be approved. Otherwise, a software bug is found.

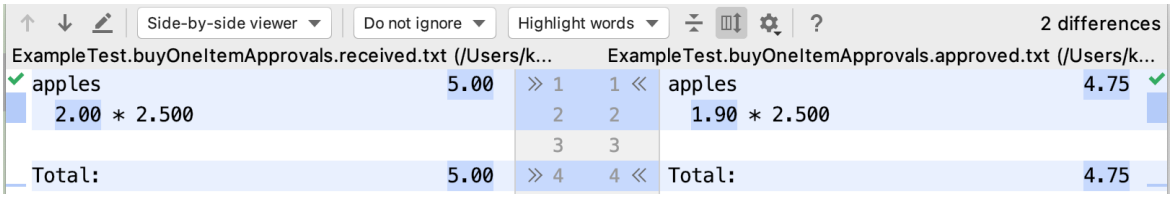


Figure 8. Differences are shown line-by-line

As previously mentioned, for legacy code, it is essential to preserve existing behaviour. With approval testing, it is possible to capture however complex output the code produces. In contrast, assertion-based tests require more effort to write the assertions and figure out the expected values.

4.4.2 Time Spent on Writing a Test

The speed at which an automated test is written is affected by factors such as choosing a descriptive name and writing the required code. The name of an assertion-based test should contain the name of the method being tested, the testing scenario and the expected behaviour [14]. Coming up with a suitable name can be pretty difficult when writing tests for legacy code without having previous knowledge about the system. Writing an approval test does not require time to figure out the expected behaviour but capture the existing behaviour. That means writing an approval test requires less time coming up with a descriptive name.

Secondly, it is possible to compare the amount of code needed to write and execute an automated test. As previously pointed out, it is possible to replace many assert statements with an Approvals method. That makes writing an approval test faster because the test writer does not have to spend time figuring out the expected values for assert statements. On the other hand, unless the result is a primitive, approval tests require the creation of a printer class or method. Time spent on creating a printer can be reduced by using libraries such as Jackson. Jackson is a Java library that provides methods to map Java objects to JSON and vice versa. That makes it possible to write a Java object as a String in JSON format. However, an undetected bug in the custom printer can also affect the resulting output.

When writing an approval test, it should be noted that the entire output is captured. This means that non-deterministic or randomly generated output values need to be removed or modified to be consistent. If that is the case, it means extra effort to deal with these values. Fortunately, the ApprovalTests tool has support for Scrubbers that can convert inconsistent values to consistent with the help of regular expressions. However, when writing an assertion-based test, it is possible to omit irrelevant values, saving time spent dealing with them.

In conclusion, approval testing has both advantages and disadvantages compared to assertion-based testing. As the advantages have shown, approval testing is helpful for handling legacy code and complex output. However, when testing primitive output or simple behaviour, it is preferable to use assertion-based testing.

5. Workshop Design and Execution

5.1 Target Group

The workshop is intended for developers and testers with little or no previous experience with approval testing. Participants are expected to have Java programming skills. In addition, the participants should be familiar with using an integrated development environment (IDE) such as IntelliJ and Git.

5.2 Schedule

The duration of the workshop is planned to be about 2 hours. During this time, participants will be introduced to approval testing and taught the skills needed to apply this testing method in their daily work. The planned schedule of the workshop is as follows:

- Approximately 20 minutes for presentation, introduction to approval testing and how it compares to assertion-based testing. That also includes a demonstration, during which the participants are shown how to rewrite assertion-based test case using the ApprovalTests Java tool
- Approximately 90 minutes for hands-on coding exercises
- Approximately 10 minutes for retrospective, during which participants reflect on the skills learned during the workshop. The topics covered in the workshop will be repeated, and participants will be able to ask questions

5.3 Materials and Tasks

The author of the thesis created the following materials for the execution of the workshop:

- Presentation materials (Appendix I)
- Workshop project materials (Appendix II)

The workshop's primary goal is to introduce participants to approval testing and teach them how to apply it in their daily work. For this reason, Supermarket Receipt Refactoring Kata is used to perform the workshop tasks [15]. Supermarket Receipt Refactoring Kata is a coding exercise that illustrates the process of purchasing from a supermarket and receiving a receipt. The supermarket has a catalogue of different products. Each product has a price, name and unit. Every once in a while, there can be special offers for some products:

- Buy three for the price of two

- Percent discount
- Buy two for an amount
- Buy five for an amount

For every purchase, a receipt is received with the products, total price, and any applied discounts. As the code is untested, the participants are given a task to replace any existing tests with approval tests. After that, they have to make sure that all the different special offers are applied correctly.

The exercise is chosen because it characterises a real-world situation when given a task to make changes to an untested code base for which there is no previous knowledge. This provides an excellent opportunity to take advantage of approval testing and capture existing behaviour. The format of the receipt also brings out the strengths of approval testing when dealing with complex output. An example of the receipt can be seen in Figure 9.

```
toothbrush                          0.99
toothbrush                          0.99
toothbrush                          0.99
3 for 2(toothbrush)                -0.99

Total:                              1.98
```

Figure 9. Example output of the receipt

5.4 Execution

The workshop was conducted for the employees of Proekspert, an information technology company in Estonia. It was chosen because the author of the thesis is an employee of the company. Due to COVID-19 prohibitions and movement restrictions, the workshop was conducted remotely. For this purpose, Microsoft Teams was used for communication, which allowed the author to present the materials, and the participants solved exercises by sharing their screen.

The workshop started with the author showing a test case based on assertions and having the participants write down what they liked or disliked. The test case can be seen in Figure 5 on page 13. As a positive thing, the participants only pointed out that it was a pretty specific test case. On the other hand, there were many things the participants disliked about the test case. Some of the pointed-out things are listed here:

- Many assertions per test

- Quite many dependencies
- Assertions do not have clear indications of what are they for

After that, the author demonstrated how to rewrite the shown test case with the Approval-Tests Java tool. The rewritten test case can be seen in Figure 6 on page 13.

After the demonstration, the author continued with the presentation. At the beginning of the presentation, the participants were explained the aim of the workshop and what they are expected to learn from it. The author introduced the idea of approval testing and how it compares to conventional assertion-based testing, explaining why and when it should be preferred.

To ensure the contribution of all participants in the workshop, mob programming approach was applied. Mob programming is a way of developing software where the whole team contributes simultaneously, using the same computer. Participants were placed in a queue, the first in the order was the driver, and the next was the navigator. After 4 minutes, an exchange took place, where the driver went to the end of the queue, and the person who had previously given instructions moved first to write the code. The driver's job was to share his/her computer screen and follow the navigator's instructions. The navigator was responsible for giving direct instructions to the driver. After every 4-minute cycle, the previous driver had to commit and push the code to the repository, so the next driver could pull the updated code and continue where it left off. As a mob, the participants solved all the prescribed exercises under the guidance of the author. Each participant managed to be a driver and a navigator at least 2-3 times. In total, nine different approval-based test cases were written.

After solving the exercise, participants were asked again to write down things they now like or dislike about the test cases. Participants liked the following:

- Tests are easier to write and follow
- Easier to test code they have no understanding of
- Easy to test combinations of different parameters and variations
- For legacy code, approval tests provide an easy way to increase test coverage for refactoring

On the other hand, there were some things the participants disliked:

- Must have a reliable printer for output

- Might have to work through long output all at once

At the end of the workshop, the author reiterated the topics covered, and participants could ask questions.

6. Feedback

6.1 Feedback from Participants

In order to find out the participants' opinion on the topics covered, a feedback questionnaire was conducted among the participants. A copy of the feedback questionnaire can be found in Appendix III. A total of 4 participants, or 100% of the workshop participants, provided feedback. The questions were divided into two main categories. Some of the questions were designed to determine the background of the participants and their previous experience with approval testing or software testing in general. The remaining questions were directly related to the workshop's topics, the participants' opinion on approval testing advantages, and its usefulness in daily work. The workshop participants included two software developers, one tester and one technical product owner. Two of them had 0 to 2 years of testing experience, and the other two had 3 to 5 years. 3 out of 4 participants had not heard about approval testing before.

The following figures describe the background of the participants (see Figure 10 and Figure 11).

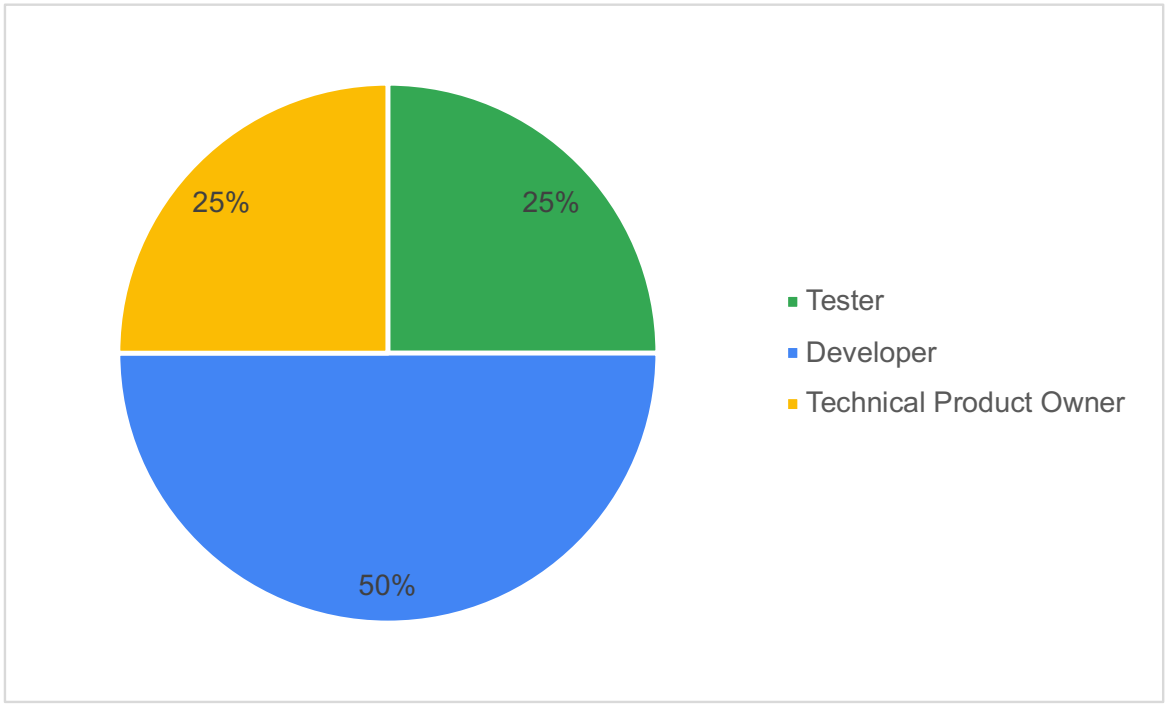


Figure 10. The role of the participants in the company

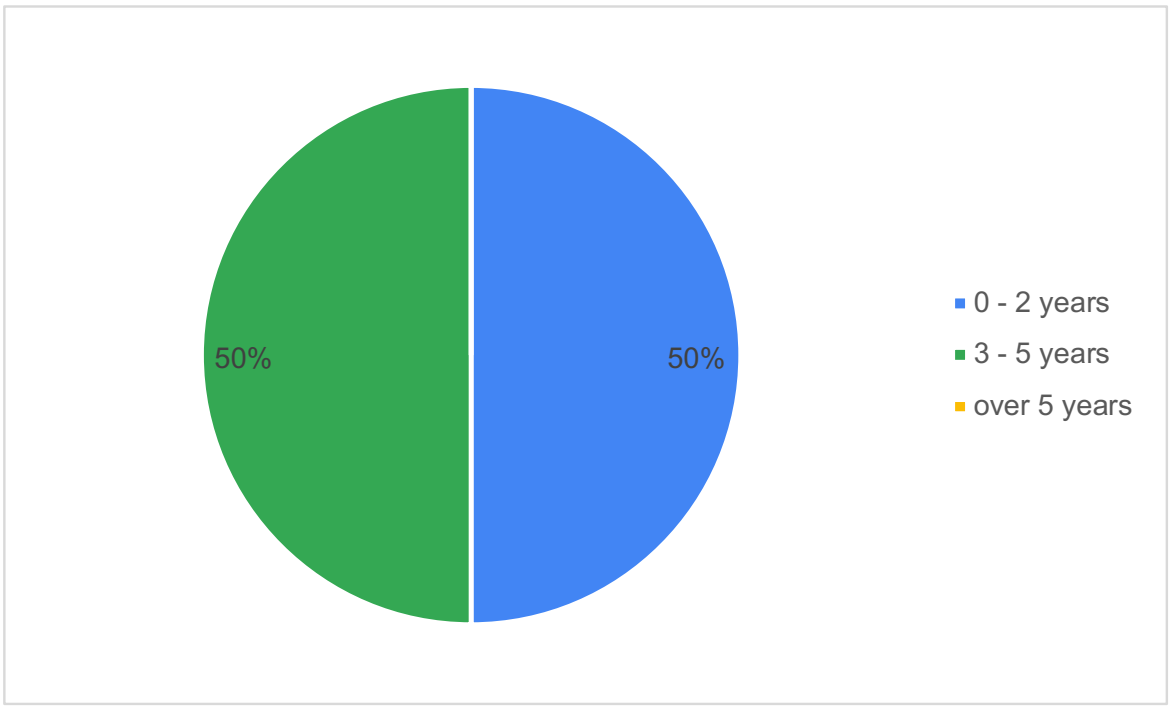


Figure 11. Previous experience with software testing

The following figure shows the distribution of the participant's opinion on topics covered in the workshop (see Figure 12).

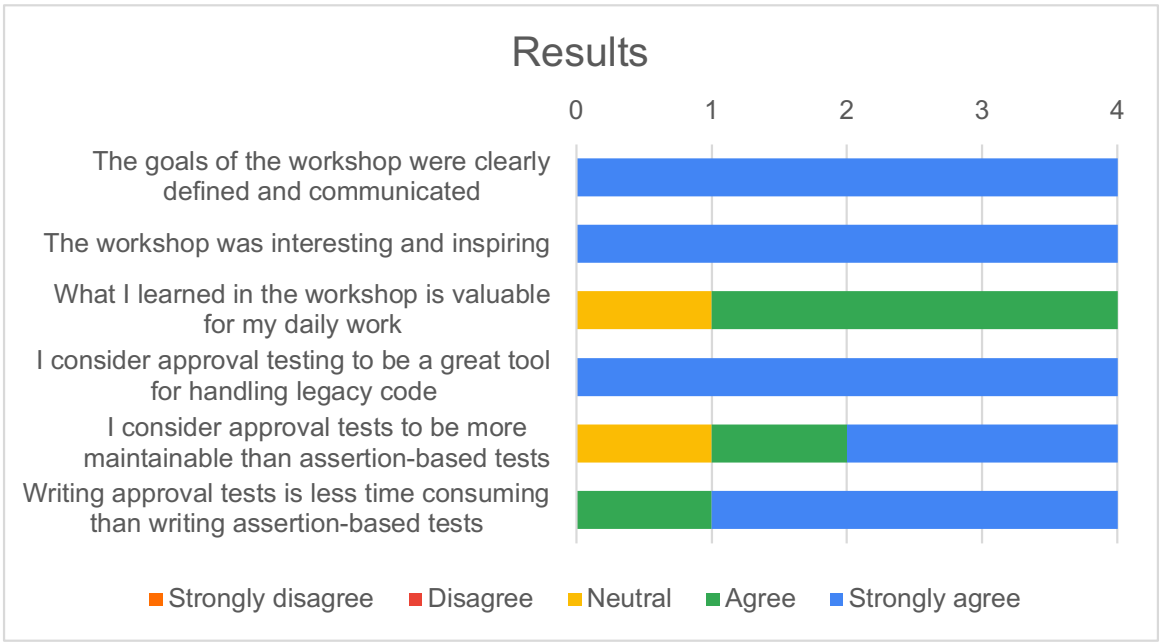


Figure 12. Results

6.2 Analysis

Figures 10 and 11 show that the background of the workshop participants was very different. Some participants had relatively little experience with software testing, as well as those who had been involved in it for some time. Also, participants included people with very different roles, from software developers and testers to the product owner. Most participants also had no previous experience with approval testing, making them good candidates for the workshop.

Figure 12 shows that the participants unanimously agreed that the workshop's purpose was clearly defined and communicated. This confirms that the workshop's aim was well set, and the participants understood why the workshop was being conducted. Based on the exact figure, it can also be seen that all participants found the workshop interesting and inspiring. From this, it can be concluded that the topics covered were new to the participants, and they would also consider applying the learned topics in their work. According to the author, this also justifies the workshop execution and is confirmed by the following question, according to which 3 out of 4 participants agree that the topics learned are helpful in their daily work. One of the participants remained neutral on this question, but this may also be justified because it is likely that the participant's day-to-day work is not directly related to software development or testing.

Figure 12 also shows that all participants found approval testing to be a handy tool when dealing with legacy code. That is a good sign that participants have previous experience dealing with legacy code and understood how to apply mentioned testing methodology.

Figure 12 shows that 3 out of 4 participants agreed that approval-based tests are more maintainable than assertion-based tests. As all the participants had previous testing experience, this is a good indicator that approval-based tests are more maintainable. The same questions could be asked from the participants after they have had time to implement approval tests in their projects, as this would provide even better confirmation. In addition, all participants agreed that writing approval tests is less time-consuming.

Overall, the author considers workshop execution to be successful. None of the participants expressed their discontent regarding the execution or materials of the workshop. According to all participants, the length of the workshop was about right. In the “free form” feedback, participants indicated that the workshop had a good setup and were grateful for putting in the time and effort.

6.3 Future Improvements

Based on the feedback collected, the workshop was successful. Participants did not point out that the workshop should be improved in any way. According to the author, there could have been more participants, but it is understandable, given that the workshop was conducted remotely. However, the materials are reusable, and the workshop may be held several times.

7. Summary

The goal of the thesis was to develop and conduct a workshop on approval testing at Proekspert. The primary goal of the workshop was to introduce the participants to approval testing by teaching skills to apply in their daily work. During the workshop, the participants were explained problems arising from legacy code and how to make them manageable through approval testing. Participants also learned the advantages of approval testing compared to assertion-based testing. The feedback questionnaire revealed that the participants considered the workshop valuable, and they learned new skills to apply in their daily work. The participants unanimously agreed on approval testing usefulness when dealing with legacy code. The feedback also showed that writing approval tests takes less time than assertion-based tests. In addition, the participants considered approval tests to be more maintainable. Overall, the workshop execution went smoothly. According to the author, there could have been more participants, but the materials are reusable, and the workshop may be held several times.

8. References

[1] Feathers, M. C. Working Effectively with Legacy Code. *Prentice Hall Professional Technical Reference*. Upper Saddle River, NJ, 2005

[2] What is software testing? https://www.ibm.com/topics/software-testing (07.05.2021)

[3] Automation Testing Tutorial: What is Automated Testing? https://www.guru99.com/automation-testing.html (09.01.2021)

[4] Itkonen, J., Lassenius, C., Mantyla, M. How Do Testers Do It? An Exploratory Study on Manual Testing Practices. *3rd International Symposium on Empirical Software Engineering and Measurement,* pp. 494-497, 2009

[5] Beck, K. Test Driven Development: By Example. *Addison-Wesley Professional*, 2003

[6] Malloy, B. A., Voas, J. M. Programming with Assertions: A Prospectus. *IT Professional*, pp 53-59, 2004

[7] Voas, J. M. How Assertions Can Increase Test Effectiveness. *IEEE Software,* vol. 14, no. 2, pp. 118-119, 1997

[8] Falco, L. Approve is the new Assert. https://blog.approvaltests.org/2008/10/approve-is-new-assert.html (06.04.2021)

[9] Carlo, N. What is Legacy Code? Is it code without tests? https://understandlegacycode.com/blog/what-is-legacy-code-is-it-code-without-tests/ (12.03.2021)

[10] Abrahamsson, P., Moser, R., Pedrycz, W., Sillitti, A., Succi, G. A. Case Study on the Impact of Refactoring on Quality and Productivity in an Agile Team. *Balancing Agility and Formalism in Software Engineering*, pp. 252-266, 2008

[11] Linders, B. Approval Testing with TextTest. https://www.in-foq.com/news/2017/02/approval-testing-texttest/ (16.04.2021)

[12] Gee, T., Henney, K. 97 Things Every Java Programmer Should Know: Collective Wisdom from the Experts. *O'Reilly Media, Inc,* 2020

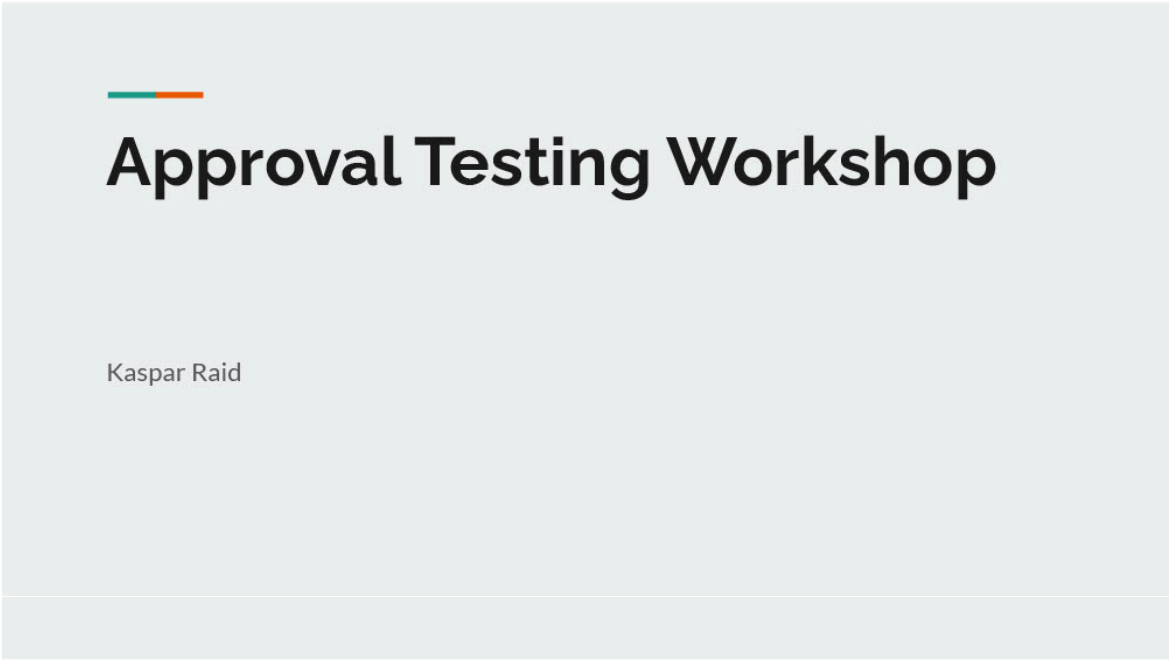
[13] ApprovalTests homepage. https://approvaltests.com/ (18.04.2021)

[14] Reese, J. Unit testing best practices with .NET Core and .NET Standard. https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices (02.05.2021)

[15] Bache, E. SupermarketReceipt-Refactoring-Kata. https://github.com/emilybache/SupermarketReceipt-Refactoring-Kata (15.04.2021)

Appendix

I. Presentation Slides



Goals

- Learn features of Approval Tests Java library
- Replace assertions with approvals
- Testing complex output
- Locking down logic in untested (legacy) code

REVIEW ASSERT BASED TEST CASE

write down what you liked and didn't like

DEMO TIME

What is Approval Testing?

Approval Testing *also known as* **Golden Master Testing** or **Characterization testing**

Assertion-based	**Approval**
Assert behaviour of an application → multiple **assert** statements for each aspect of the system	**Capture** and **store** behaviour of an application → present output for **approval**

Problem with Assertions

```java
@Test
void testAddition() {
    // ARRANGE
    int x = 1;
    int y = 2;
    // ACT
    int result = addNumbers(x, y);
    // ASSERT
    assertThat(result).isEqualTo(3);
}
```

Complex output

```
{
  "items" : [ {
    "product" : {
      "name" : "apples",
      "unit" : "Kilo"
    },
    "price" : 1.99,
    "totalPrice" : 4.975,
    "quantity" : 2.5
  } ],
  "discounts" : [ ],
  "totalPrice" : 4.975
}
```

```java
// ASSERT
assertThat(receipt.getTotalPrice()).isEqualTo(4.975);
assertThat(receipt.getDiscounts()).isEmpty();
assertThat(receipt.getItems()).hasSize(1);
ReceiptItem receiptItem = receipt.getItems().get(0);
assertThat(receiptItem.getProduct()).isEqualTo(apples);
assertThat(receiptItem.getPrice()).isEqualTo(1.99);
assertThat(receiptItem.getTotalPrice()).isEqualTo(2.5 * 1.99);
assertThat(receiptItem.getQuantity()).isEqualTo(2.5);
```

Parts of an Unit Test

ARRANGE	ACT	ASSERT
set-up objects	execute code	assert outcome is expected

Parts of an Unit Test (APPROVAL)

ARRANGE	ACT	PRINT	APPROVE
set-up objects	execute code	capture and present	compare output

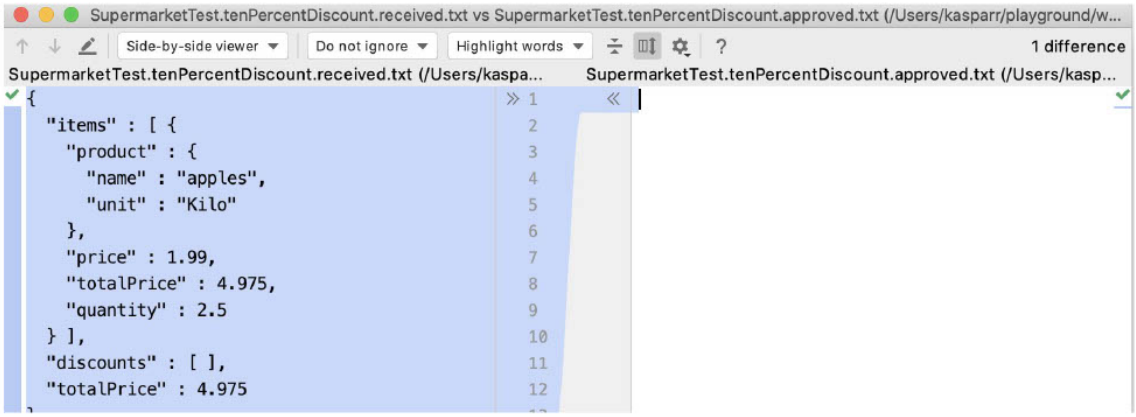
Printer

- Design your own printer
- Use standard serializer library such as Jackson

How often do you just run the code and check the output?

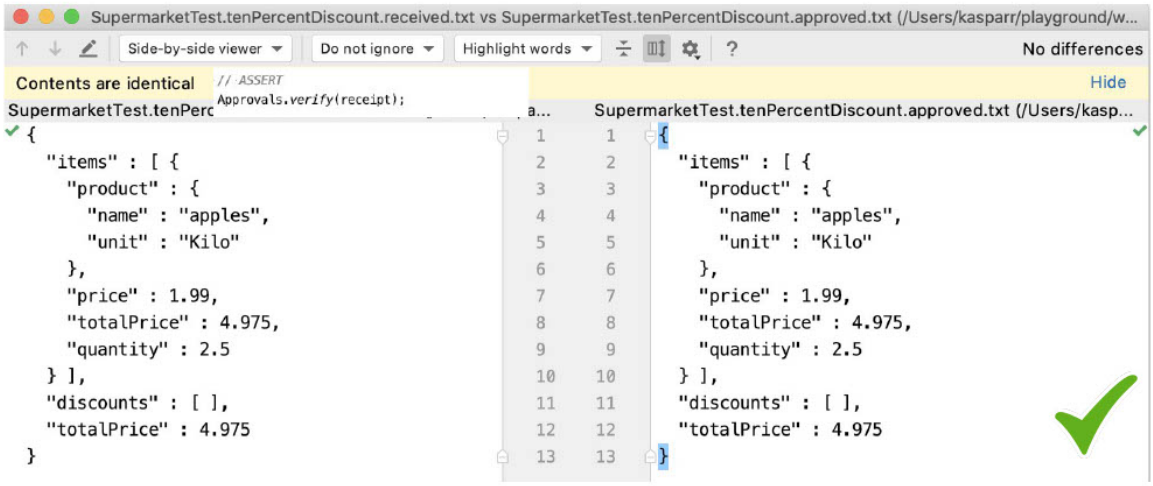
RECEIVED

APPROVED

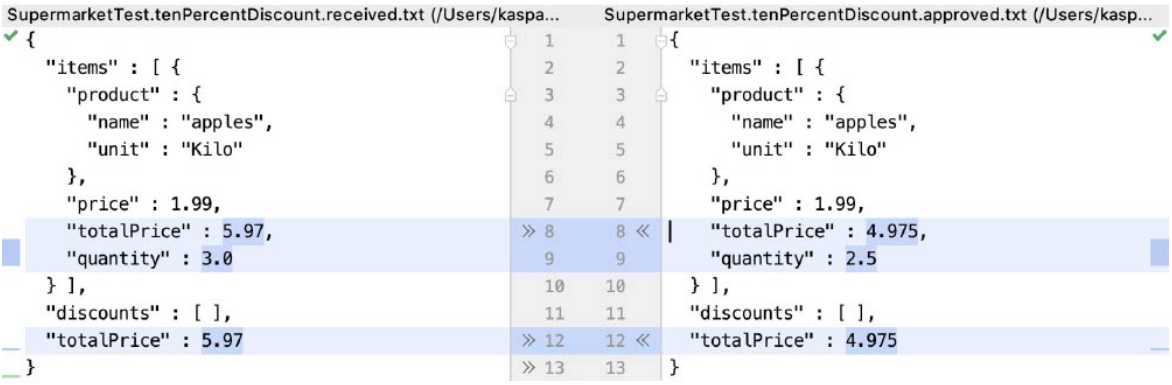


RECEIVED

APPROVED



RECEIVED **APPROVED**



Use Cases

- Verifying objects that require more than a simple assert
- Maps, Collections
- Long Strings
- Log files
- XML
- HTML
- Json
- Legacy Code

Legacy Code

- Code without tests
- Code that is hard to change
- Code with poorly written tests
- Code that many people have developed over a long period

I need to change this code but it has no tests

Possible solutions

- The cross fingers approach - what could go wrong?
- Study the code thoroughly before making any changes
- Lock down the existing behaviour

Golden Master offers us a place to get started, and support while we start breaking things apart.

- **J.B. Rainsberger**

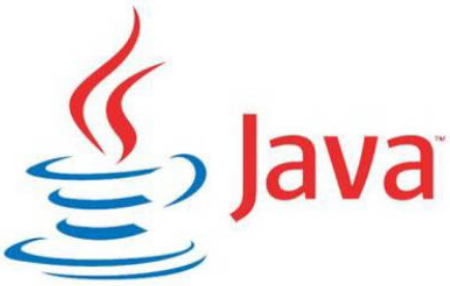
Why Approvals?

- Less time consuming than writing assert based tests
- Quickly increase test coverage
- More humane way to test complex output
- Easier to maintain

Approval Tests

https://approvaltests.com/

- .Net
- C++
- Java
- Lua
- NodeJS
- Objective-C
- Perl
- Python



https://github.com/approvals/ApprovalTests.Java

Features

- Approvals.verify()
- Approvals.verifyAll()
- Approvals.verifyXml()
- Approvals.verifyJson()
- Approvals.verifyHtml()
- Approvals.verifyException()

Features

- CombinationApprovals.verifyAllCombinations()
- Up to 9 different parameters

Number of Parameters	Variations per Parameter	Total Combinations
2	5	25
3	3	27
3	4	64
4	5	625
5	6	7,776
9	9	387,420,489

```java
@Test
void addCombinations() {
    CombinationApprovals.verifyAllCombinations(
            this::addNumbers,
            new Integer[]{1, 2, 3},
            new Integer[]{2, 3, 4}
    );

}

private int addNumbers(int x, int y) {
    return x + y;
}
```

```
[1, 2] => 3
[1, 3] => 4
[1, 4] => 5
[2, 2] => 4
[2, 3] => 5
[2, 4] => 6
[3, 2] => 5
[3, 3] => 6
[3, 4] => 7
```

Reporters

Mac	Windows

`BeyondCompareMacReporter.INSTANCE,`
`DiffMergeReporter.INSTANCE,`
`KaleidoscopeDiffReporter.INSTANCE,`
`P4MergeReporter.INSTANCE,`
`KDiff3Reporter.INSTANCE,`
`TkDiffReporter.INSTANCE,`
`IntelliJReporter.INSTANCE,`
`VisualStudioCodeReporter.INSTANCE`
	`TortoiseDiffReporter.INSTANCE,`
	`BeyondCompareReporter.INSTANCE,`
	`WinMergeReporter.INSTANCE,`
	`AraxisMergeReporter.INSTANCE,`
	`CodeCompareReporter.INSTANCE,`
	`KDiff3Reporter.INSTANCE,`
	`IntelliJReporter.INSTANCE,`
	`VisualStudioCodeReporter.INSTANCE`

Configuring reporters

Class and method level - @UseReporter(IntelliJReporter.class)

Package level

```
public class PackageSettings {
    public static IntelliJReporter UseReporter = IntelliJReporter.INSTANCE;
}
```

Supermarket Receipt Kata

The supermarket has a catalogue of different products.

Each product has a price, name and unit.

Following discounts:

- Buy three for the price of two
- Percent discount
- Buy two for an amount
- Buy five for an amount

Task

- Your task is to replace any existing tests with Approvals tests.
- Make sure that all the different discounts are applied correctly.

Mob programming

- https://mobti.me/approvalworkshop
- **Navigator** - writes code (also shares his/her screen)
- **Driver** - gives instructions to navigator
- Interval duration **3** minutes
- After each cycle commit and push

LET'S CODE!

REVIEW APPROVAL BASED TEST CASE

write down what you liked and didn't like

Tests produce inconsistent output?

- Tests depend on time
- Auto generated ids
- Etc...

Possible solutions

- Provide a fixed clock
- Mock inconsistent method (every time you mock, you actually test the mock)
- **Scrubbers (the Approvals way)**

RegexScrubber

```
@Test
void scrubNumbers() {
    String input = "Auto generated number is: " + new Random().nextInt( bound: 100);
    Approvals.verify(input, new Options(new RegExScrubber( pattern: "\\d+",  replacement: "NUMBER")));
}
```

DateScrubber

- DateScrubber.getSupportedFormats()

```
Approvals.verify( response: "2021-03-30T13:12Z", new Options(DateScrubber.getScrubberFor("2021-03-30T13:12Z")));
```

What you should have learned today

- Features of Approval Tests Java library (verify, verifyAllCombinations, Scrubbers, Reporters)
- Replace assertions with approvals
- Testing complex output
- Locking down logic in untested (legacy) code

Thank you for attending :)

II. Workshop Materials

“Approval Testing Workshop” Materials, ZIP file

https://drive.google.com/file/d/1UlWpVtbTz3z03Wwq9RuyzyjJYoNnxrzo/view

III. Feedback Questionnaire

Approval Testing Workshop Feedback

Thank you for attending the workshop!

*Required

1. My role in the company (for example: developer, tester) *

_____

2. Previous experience with software testing *

Mark only one oval.

- ◯ 0 - 2 years
- ◯ 3 - 5 years
- ◯ over 5 years

3. The goals of the workshop were clearly defined and communicated *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

4. The workshop was interesting and inspiring *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

5. What I learned in the workshop is valuable for my daily work *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

6. I had previously heard about approval testing *

Mark only one oval.

- ◯ Yes
- ◯ No

7. I consider approval testing to be a great tool for handling legacy code *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

8. I consider approval tests to be more maintainable than assertion-based tests *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

9. Writing approval tests is less time consuming than writing assertion-based tests *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

10. Was the workshop length too long, too short or about right? *

Mark only one oval.

- ◯ Too long
- ◯ About right
- ◯ Too short

11. Free form

IV. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, Kaspar Raid,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Developing and Conducting a Workshop on Approval Testing at Proekspert,
supervised by Dietmar Pfahl.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Kaspar Raid
04/05/2021