Categorising Test Smells

Bachelor Thesis

Maudlin Kummer
from
Zollikofen BE, Switzerland

Faculty of Natural Sciences of the University of Bern

25th March 2015

Prof. Dr. Oscar Nierstrasz

Dr. Mircea Lungu

Software Composition Group
Institute of Computer Science and Applied Mathematics
University of Bern, Switzerland

Abstract

The aim of this investigation into test smells was to find out how familiar developers are with test smells, the frequency of test smells and their severity in the industrial world. First of all, a taxonomy of different test smells was created and grouped according to programming principles as a basis for this study. Several interviews were then conducted to find out which test smells to include in the subsequent survey. 20 people with different industrial experience levels participated in this survey. It was hypothesised that test smells are not identified as such and that their names are unknown. The hypothesis was supported by the results of the survey. The results revealed that test smells are not quite well-known despite the fact that some of them occur rather frequently and pose severe problems.

Contents

1 Introduction

Writing test code is an essential task in the process of developing software and as such it is subject to good practices and design principles just like the writing of production code. This equal status of test code and a production code is highlighted by the emergence of test-driven development. Kent Beck [2] and David Astels [1] published books that introduce developers to test-driven development and demonstrate how test code should be written. They describe methods to avoid certain test smells without mentioning them explicitly. Thus, it is only logical that Kent Beck's [6] concept of code smells, which he contributed to Martin Fowler's book, is applicable to test code as well, resulting in so-called test smells. The idea of test smells was first introduced by van Deursen, Moonen, van den Bergh and Kok [21]. A test smell is a part of the test code which points to a design problem that can either affect the readability of the code or even its functionality. There are a lot of test smells, of which some are uncommon. Some common examples are 'Long Test', 'Test Envy', 'Multiple Assertions' and 'Slow Test' (see Chapter 6 for detailed descriptions). Gerard Meszaros [10] has compiled a wide collection of test smells, which was incorporated into this investigation. There has been similar research into the influence of experience on software testing practice, for example, by Beer and Ramler [3].

This research was conducted to find out how well-known test smells are, their frequency and severity in the industrial world. It was hypothesised that developers write and refactor tests containing test smells without really identifying them as recognised test smells, which have been theorised in literature, and that the developers are mostly unaware of the specific names given to the test smells. Unfortunately, research which is similar to this study is still limited and, as a result, there is not much pre-existent literature to relate to in this investigation into test smells.

2
Methods

Qualitative and quantitative methods were used to investigate how well-known test smells are and how they are evaluated in terms of severity and frequency in the industrial world.

2.1 Pre-Phase: Interviews

On the basis of test smells that were collected in an initial phase (see Chapter 6) an interview questionnaire was created to determine which test smells should be included in the survey (see Appendix A). The interview was conducted with four participants who had some industrial experience. Although the questionnaire was written in English the interviews were conducted in the participants' preferred language and recorded with the participants' prior consent.
The first three questions concerned personal information like programming languages and the experience of the participant. Each of the further questions consisted of a principle that is relevant to writing test code and the participant was asked to name test smells that violate the mentioned principle.

2.2 Preparation and Distribution of the Survey

After the evaluation of the interviews a survey questionnaire in the form of a quiz (see Appendix B), which consisted of 39 questions and written in English, was compiled. The survey questionnaire included nine test smells: 'Manual Intervention', 'Nondeterministic Test', 'Multiple Assertions', 'Unclear Naming', 'Mock Happy', 'Large Setup Methods', 'Not Idempotent', 'Easy Test' and 'For Tests Only' (see Chapter 6 for detailed descriptions).The survey was then put on a website [18] using the WordPress software [22] and the Wp-Pro-Quiz plug-in for WordPress [23]. The link to the survey website was distributed via e-mail to 36 individuals and 9 companies.
The participants' residence or location were considered irrelevant since the essential feature for the survey was the industrial experience of the participants. However, in the distribution process the participants were not preselected on the basis of their industrial experience because it would have been difficult to determine whether one had industrial experience or not.
The survey was presented in the form of a quiz, which could be shared on Twitter [19], to increase the

response rate. In addition, the questions were clearly formulated and as short as possible to increase their understandability. All questions were compulsory except for the ones that demanded a comment on the respective test smell. A privacy policy was included in the introduction of the questionnaire for the participants' information.
Prior to the final distribution a pilot test was conducted to disclose any technical problems and to estimate the time needed to take the quiz. Furthermore, it served the purpose of testing the understandability of the questions. The pilot test and the main survey quiz were both conducted via the website and the answers were collected anonymously by the Wp-Pro-Quiz plug-in. The data was then analysed manually and tables were formed. The whole process including the interviews took about three months.

2.3 Survey Questions

The survey questions were qualitative and quantitative. The introduction page of the survey contained three initial questions that asked the participants for two programming languages in which they were most experienced, their overall programming and industrial experience (in years). The main questions were divided into five categories:

1. Principle Questions
2. Frequency Questions
3. Severity Questions
4. Synonyms/Variants Questions
5. Comment Questions

There were 4 questions from the categories 1, 2, 3 and 5 on each of the nine test smells in the survey. Three test smells had an additional question from category 4.
The principle questions required the participants to choose from a list of three test smells the one that violates the principle named in the question. The aim was to evaluate whether the test smell is common or not.
The purpose of the frequency questions was to find out how often the participants have encountered a particular test smell. The Likert scale of five ranged from "never" to "often". The middle rate was "neutral" and the two other options were "rarely" and "rather often".
Severity questions inquired about the participants' opinion about the effect of a test smell on the quality of the test code. The Likert scale of five ranged from "not severe" to "very severe", where the middle rate was "neutral". The other options were "rather not severe" and "severe". An additional option "don't know" was provided.
Questions from the synonyms/variants category were open questions and aimed at collecting other names or variants of a test smell. The comment questions were optional and open questions. They provided an opportunity for the participants to write down any additional information they liked to share. (see Appendix B)

3 Results

3.1 Participants' Characteristics

The industrial experience of the participants was distributed as follows: 4 participants had no industrial experience, 7 had between one and four years, 2 between five and nine years and 7 had ten or more years. (Figure 3.1)

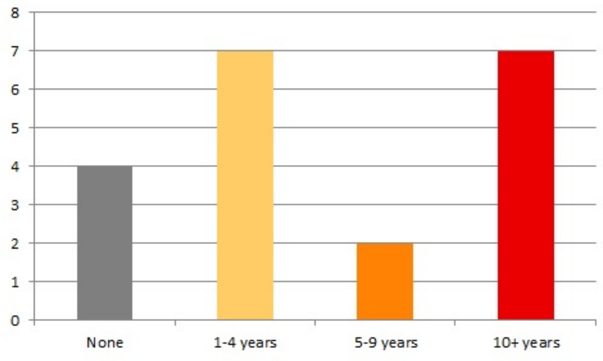


Figure 3.1: Distribution of Industrial Experience

3.2 Detailed Results

This section will present the detailed results arranged according to the categories of the questions (see Section 2.3). The results of the synonyms/variants questions were incorporated in the taxonomy of test smells (see Chapter 6) and the comment questions in the discussion of the results (see Section 4.1).

3.2.1 Principles

Mocking - *Mock Happy* and Not Affecting Environment - *Not Idempotent*:
Concerning the principle of using mock objects, 13 participants were able to identify 'Mock Happy' as a violation whereas 7 participants were unable to. The same results were observed for 'Not Idempotent' which violates the principle of not affecting the environment. (Figure 3.2)
All Paths - *Easy Test*:
The test smell 'Easy Test' was recognised as a violation of the principle that a test should test all paths by 14 participants whilst 6 were unable to. (Figure 3.2)
No Test Logic in Production Code - *For Tests Only*:
To the question asking which test smell violates the principle that test logic should not be placed in the production code, 16 participants answered correctly with 'For Tests Only' whereas 4 answered incorrectly. (Figure 3.2)
Test One Thing - *Multiple Assertions*:
18 participants correctly identified the test smell 'Multiple Assertions' as a violation of the principle that a test method should only test one thing whilst 2 participants gave a false answer. (Figure 3.2)
Communicate Intent - *Unclear Naming* and Setup - *Large Setup Methods*:
Given the principle that tests should communicate their intent clearly, 19 participants correctly answered with 'Unclear Naming' and only 1 participant gave an incorrect answer. The test smell 'Large Setup Methods' with the principle that setups should be kept appropriate and small yielded the same results as 'Unclear Naming'. (Figure 3.2)
Test Automation - *Manual Intervention* and Determinism - *Nondeterministic Test*:
All participants recognised the test smell 'Manual Intervention' as a violation of test automation and 'Nondeterministic Test' as a violation of determinism. (Figure 3.2)

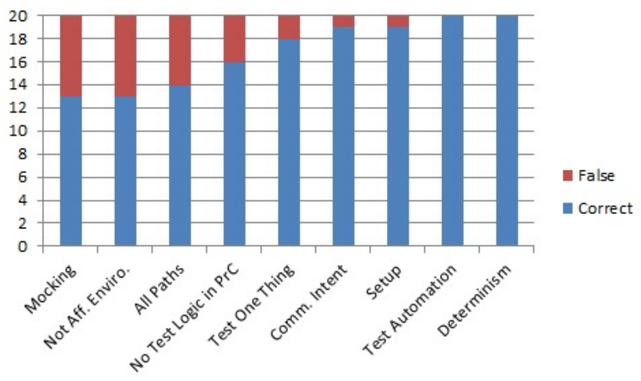


Figure 3.2: Overview of Answers to the Principle Questions

3.2.2 Frequency of Encounter

Nondeterminstic Test:
Most participants had rarely seen 'Nondeterministic Test'. The average rating was 2.35. (Figure 3.3)
Mock Happy:
Estimating the encounters with the test smell 'Mock Happy', 6 participants chose "rarely" and the same number of participants selected "neutral". The average rating was 2.7. (Figure 3.3)

Not Idempotent:
Concerning the smell 'Not Idempotent', more than half of the participants opted for "rarely" and "neutral". The average rating was 2.75. (Figure 3.3)
For Tests Only:
The majority of the participants had a neutral opinion about the frequency of 'For Tests Only'. The average rating was 2.75. (Figure 3.3)
Manual Intervention:
Concerning the test smell 'Manual Intervention', the majority of participants either chose "rarely" or "rather often". The average rating was 2.95. (Figure 3.3)
Large Setup Methods:
The question for the frequency of 'Large Setup Methods' was mostly answered with "rather often". The average rating was 3. (Figure 3.3)
Easy Test:
With respect to the question concerning 'Easy Test', most participants had seen it rather often. The average rating was 3.45. (Figure 3.3)
Unclear Naming:
Most answers were "rather often". The average rating was 3.65. (Figure 3.3)
Multiple Assertions:
The test smell 'Multiple Assertions' had been seen by all the participants but most answers tended towards "rather often". 1 participant left the question unanswered. The average rating was 3.89. (Figure 3.3)

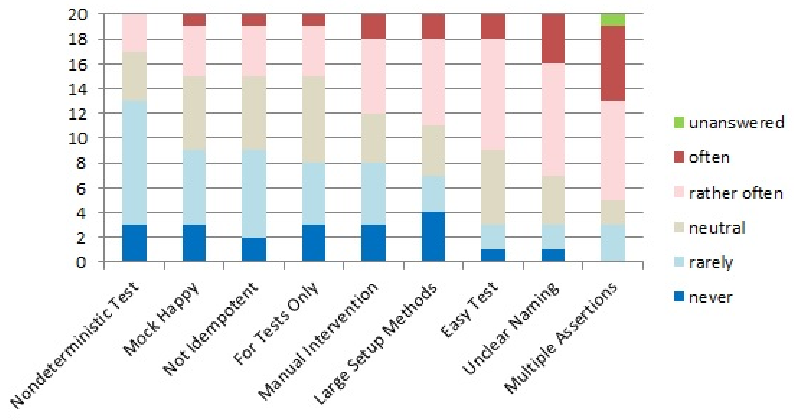


Figure 3.3: Overview of the Frequency of the Test Smells

3.2.3 Severity

Multiple Assertions:
Most answers were either "rather not severe" or "neutral". The average rating was 2.65. (Figure 3.4)
For Tests Only:
With respect to the severity of the test smell 'For Tests Only', the answers were symmetrically distributed around the mean "neutral". The average rating for this question was 3.00. (Figure 3.4)
Large Setup Methods:
The test smell 'Large Setup Methods' appears to be neither severe nor not severe since the question was mostly answered with "neutral". The average rating for this question was 3.35. (Figure 3.4)

Mock Happy:
Concerning the test smell 'Mock Happy', the majority of participants had a neutral opinion and thus, considered the test smell neither severe nor not severe. The average rating was 3.5. (Figure 3.4)
Unclear Naming:
None of the participants considered 'Unclear Naming' as "not severe". On the contrary, most participants assessed it as "severe". The average rating was 3.6. (Figure 3.4)
Easy Test:
In assessing the severity of 'Easy Test', most responses were either "neutral" or "very severe". The average rating was 3.68. (Figure 3.4)
Nondeterministic Test:
'Nondeterministic Test' was mostly judged as a very severe smell. The average rating for this question was 4.05. (Figure 3.4)
Manual Intervention:
Similar to 'Nondeterministic Test', the test smell 'Manual Intervention' was considered "very severe" by the majority of participants. 1 participant gave no answer at all. The average rating was 4.1. (Figure 3.4)
Not Idempotent:
To the question on the severity of 'Not Idempotent', three quarters of the participants chose an option from the right side of the scale, namely "severe" and "very severe". Only 1 participant left the question unanswered. The average rating for this question was 4.11. (Figure 3.4)

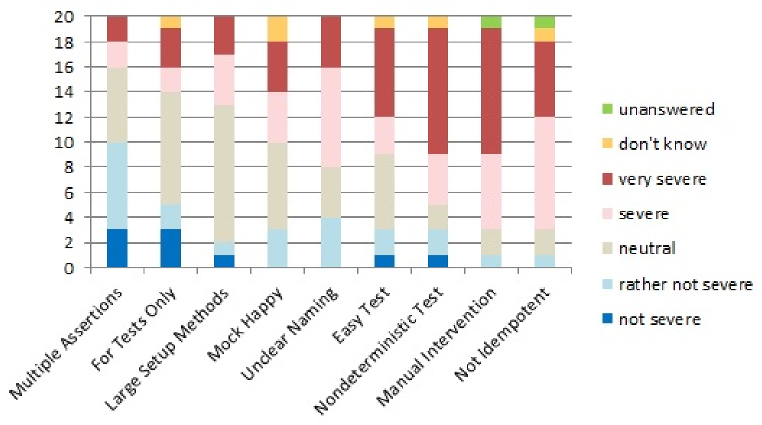


Figure 3.4: Overview of the Severity of the Test Smells

3.2.4 Frequency versus Severity

Plotting the average ratings of frequency (see Subsection 3.2.2) against the ones of severity (see Subsection 3.2.3) resulted in the following distribution:

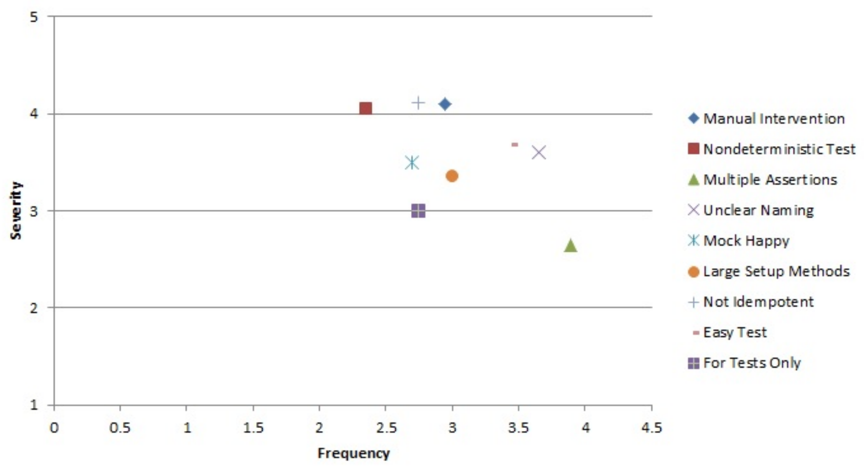


Figure 3.5: Contingency Table of the Frequency and Severity of the Test Smells

3.3 Detailed Results for Severity according to Industrial Experience Groups

Manual Intervention:
The contingency table (cross tabulation) for ‘Manual Intervention’ shows that the different experience groups answered quite similarly. However, the test smell was mostly assessed as “very severe” by the experienced participants. Interestingly, only 1 participant chose an option from the left side of the Likert scale.
This demonstrates that the majority of participants evaluated the smell as “severe” or “very severe”. (Figure 3.6)

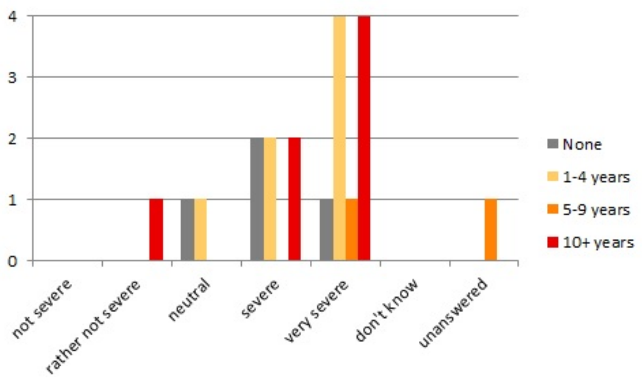


Figure 3.6: Contingency Table of the Severity of ‘Manual Intervention’

Nondeterministic Test:
For the group which had no industrial experience at all, the responses were confined to the right side of the Likert scale. On the contrary, the experienced groups answered within the complete range but still most

answers were “very severe”.
Once again, most participants tended towards assessing the test smell as “severe” or “very severe”. (Figure 3.7)

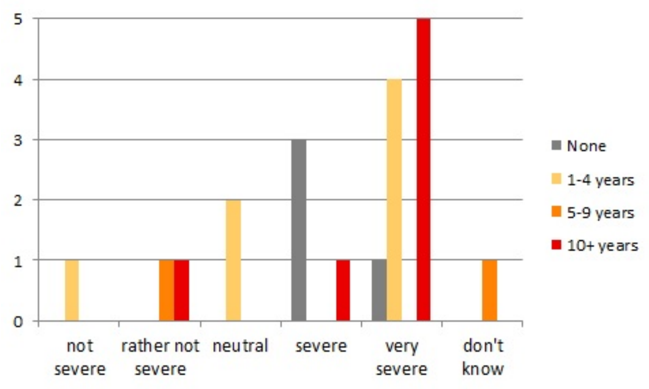


Figure 3.7: Contingency Table of the Severity of ‘Nondeterministic Test’

Multiple Assertions:
The distribution for ‘Multiple Assertions’ is right-skewed for all experience groups except for the one of 5-9 years. The participants who had no experience rated the test smell in the range between “neutral” and “very severe” whilst the experienced groups mostly opted for “rather not severe”.
This shows that most participants decided on the options “neutral” or “rather not severe”. (Figure 3.8)

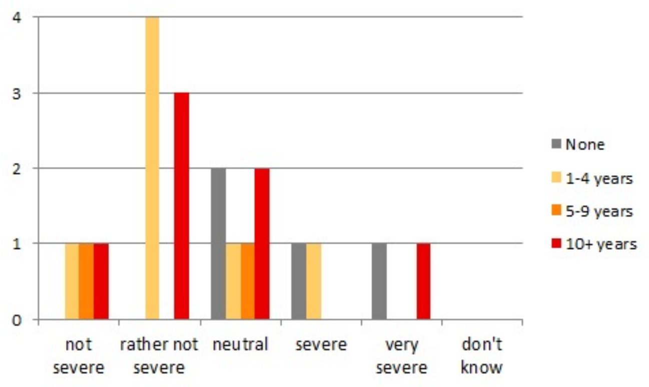


Figure 3.8: Contingency Table of the Severity of ‘Multiple Assertions’

Unclear Naming:
In all the experience groups nobody chose the answers “don’t know” and “not severe”. Within the group that had no experience and the one of 10+ years, the responses were (quite) evenly distributed. The experience group 1-4 years only chose options from the range “neutral” to “very severe” with the peak at “severe”.
The contingency table demonstrates that the majority of participants judged ‘Unclear Naming’ as a severe test smell. (Figure 3.9)

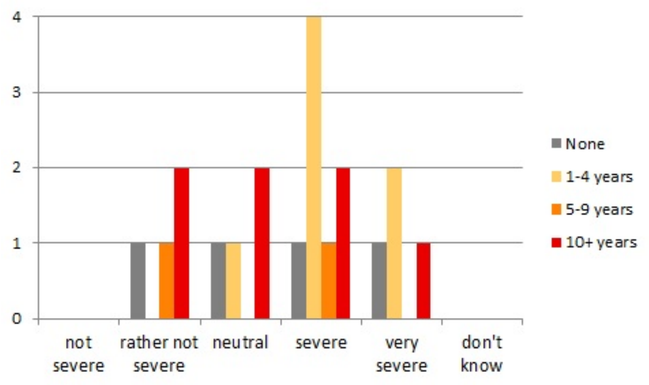


Figure 3.9: Contingency Table of the Severity of ‘Unclear Naming’

Mock Happy:
The data of the contingency table is in general left-skewed. In the inexperienced group all participants chose either “rather not severe” or “neutral”. In contrast, the majority of the most experienced group responded with “severe”. The experience groups of 1-4 and 5-9 years both reached their highest points at “neutral” and “very severe”.
Interestingly, the inexperienced participants tended towards assessing ‘Mock Happy’ as “rather not severe” whilst the more experienced participants found the smell severe. (Figure 3.10)

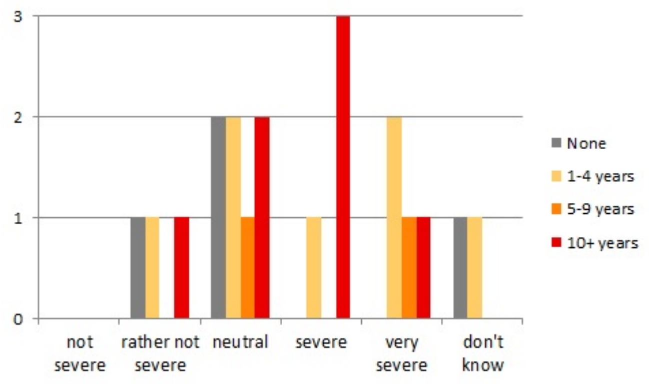


Figure 3.10: Contingency Table of the Severity of ‘Mock Happy’

Large Setup Methods:
The majority of all participants, regardless of their industrial experience, responded with “neutral”. However, only participants of the experience groups of 1-4 and 10+ years selected an option from the left side of the Likert scale.
This suggests that most participants evaluated the test smell neither as severe nor not severe. (Figure 3.11)

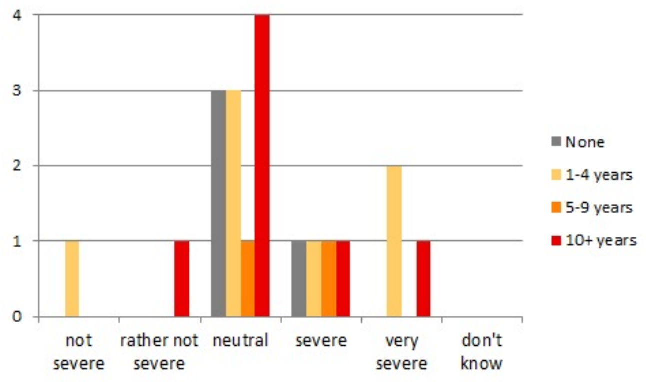


Figure 3.11: Contingency Table of the Severity of ‘Large Setup Methods’

Not Idempotent:
The results of the contingency table have a left-skewed distribution. Intriguingly, all responses except for one were confined to the range from “neutral” to “very severe”.
This demonstrates that the majority of participants assessed ‘Not Idempotent’ as “severe”. (Figure 3.12)

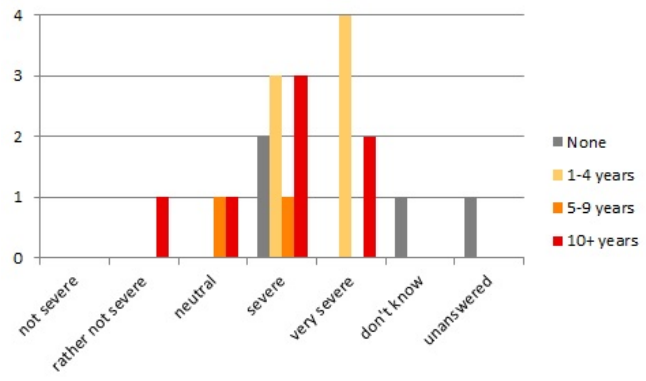


Figure 3.12: Contingency Table of the Severity of ‘Not Idempotent’

Easy Test:
Within the inexperienced group, the participants only selected options from the range “neutral” to “very severe” with the maximum at “very severe” whilst more experienced participants also chose options from the left side of the scale. The distributions of the experienced groups were similar. They were all bimodal with the peaks “neutral” and “very severe”.
It is evident that the majority of participants opted for either “neutral” or “very severe”. (Figure 3.13)

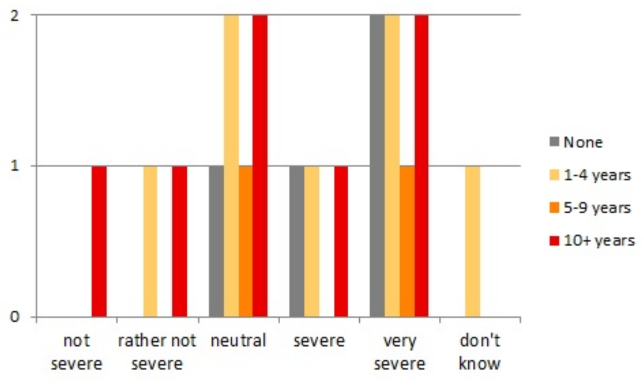


Figure 3.13: Contingency Table of the Severity of ‘Easy Test’

For Tests Only:
The contingency table of ‘For Tests Only’ demonstrates that only the experienced participants opted for the option “very severe”. Moreover, all the experienced groups mostly answered with “neutral”. Interestingly, the experience groups of 1-4 years and 10+ years exhibit their second highest number of answers at the exact opposite sides of the scale: The group of 10+ years at “not severe” and the one of 1-4 years at “very severe”.
This shows that most participants assessed the test smell ‘For Tests Only’ neither as severe nor not severe (“neutral”). (Figure 3.14)

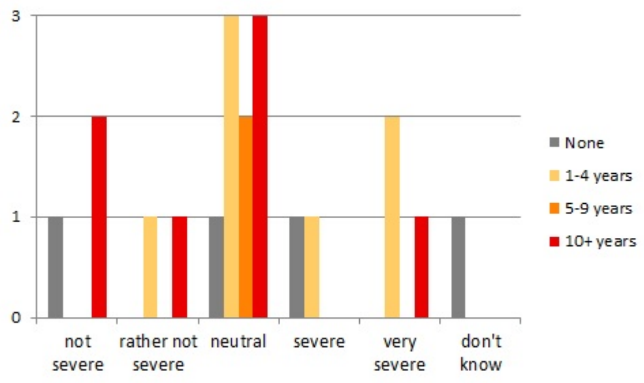


Figure 3.14: Contingency Table of the Severity of ‘For Tests Only’

4
Discussion

4.1 Discussion

The hypothesis that parts of a test code are refactored without even recognising the ‘smelly’ parts as specific test smells and that the names of test smells are not well-known was supported by the results.

The results of the principle questions showed that many developers were able to connect the violated principle with the matching test smell, which implies that developers are aware of the specific test smells and their names. However, this assumption is partly disproved by the initially conducted interviews, the comments made by some of the participants in the survey and the small number of synonyms and variants that resulted from the survey. In the interviews the participants were only given the principle and had to name test smells that violate the specified principle. Although the participants were sometimes able to describe a test smell, which is mentioned in literature, they were nearly always not able to provide the names of the test smells. This suggested that test smells are rather not well-known. Some participants of the survey openly admitted in the comments that they did not know a certain smell, which also supports the initial observation of the unfamiliarity with test smells. The discrepancy between the interview and survey results can be attributed to the fact that the names of the test smells often contain a reference to the principle. Thus, it becomes guessable which principle the test smells violate if the names of the smells are provided. For example, the test smell ‘Non**deterministic** Test’ violates the principle of **Determinism**.

Concerning the test smell ‘Manual Intervention’, most participants judged it as a very severe smell, especially the experienced developers. The reasons for this evaluation could be that tests of this kind tend to function incorrectly and that they take a lot of time to execute. Thus, they also jeopardise the good practice of testing regularly as some of the participants mentioned in their comments. Furthermore, one of the participants commented that ‘Manual Intervention’ is a minor issue since the manual part of the code can be automatised without difficulties. The occurrence of the test smell is quite evenly distributed between rare and rather often.

‘Nondeterministic Test’ was considered very severe by the majority of participants. The comments given by the participants explain this rating. It was mentioned that ‘Nondeterministic Tests’ decrease the usefulness of the tests gravely because it becomes indeterminable when a test will fail, why and whether the test code is to blame for the failure or the production code. However, one participant judged it as not severe. It could be that this participant thought of the types of tests that are not completely deterministic but are useful as well. For example, multi-threaded tests which was commented by one participant. Regarding its frequency of occurrence, the majority had rarely seen it. This could be because of the aforementioned problems that accompany the smell and render the test useless. Therefore, developers may pay more attention to avoiding non-determinism.

By reference to the severity of ‘Multiple Assertions’, the results show that most participants tend towards evaluating the test smell as a rather not severe issue. The comments suggest that the causes for this tendency are that complicated objects need multiple assertions in order to be tested and that it is a good method to test sequences and short scenarios. Another reason, which was mentioned by one participant, is the time factor. It could be that many developers in the industrial world, due to the pressure of meeting the deadline for the software release, prefer to use several assertions in one test method instead of writing a different test method for each of the assertions . This would also explain why the group that had no industrial experience mostly judged the test smell on a range from “neutral” to “very severe” since they do not have to meet deadlines for software releases. The above reasons also justify why most participants had encountered the test smell quite frequently.

The majority of participants judged the test smell ‘Unclear Naming’ as severe, especially the participants with an industrial experience between one and four years. It could be that this is due to the problems which result from the smell. These are according to the comments of the participants: Difficulties in understanding the test, misunderstandings and dependency on the co-worker who wrote the test and, as a result, the amount of time needed to understand the code. Additionally, the impact the test smell has on the maintainability of the test code could also have influenced the assessment. Despite the severity of the test smell, it seems to be quite frequent since the majority of the participants had seen it rather often. One reason could be that naming conventions are not communicated efficiently. Another explanation could also be that the programmer himself/herself who wrote the test code knows what the intention of the test is and thus, does not perceive the test name as unclear as it really is.

With regard to ‘Mock Happy’, its severity was mostly rated as neutral meaning that it is neither severe nor not severe. Interestingly, the results of the group that had no industrial experience ranged only between the evaluations “rather not severe” and “neutral”. This suggests that programmers who were probably introduced to the concept of using mock objects recently only know the advantages of it but not the disadvantages that accompany its overuse. As for its frequency, the majority had encountered the smell sometimes or rarely. It could be that developers tend to avoid an overuse of mock objects due to its disadvantages. Another reason for this result, which was mentioned in a comment, could be that an overuse of mock objects is most often an indication of a poor design of the production code. Thus, a developer will in all probability refactor the production code which will lead to a lesser need for mock objects.

‘Large Setup Methods’ were evaluated neither as a severe nor not severe test smell by the majority. It could be that this stems from the fact that large setups are sometimes unavoidable, especially if the object is complex or big as one participant mentioned in his comment. Thus, it is not very surprising that the test smell is encountered rather often.

Concerning the test smell ‘Not Idempotent’, the majority of participants considered it a severe smell, regardless of their experience. This rating is probably caused by the unforeseeable consequences which an alteration of the environment can elicit. For the same reason, most participants had rarely seen it.

Most participants either assessed the test smell ‘Easy Test’ as neutral or very severe. One reason for the selections of the option “very severe” could be, as one participant commented, that it is completely useless to test if the real behaviour of the production is being neglected in the test. However, this test smell had been seen quite often.

Regarding the severity of the test smell ‘For Tests Only’, most participants had a neutral opinion on it. This rating could originate from the following factors, which were mentioned in the comments of the participants: On the one hand, it is sometimes unavoidable to insert test logic into the production code, for example, with threaded tests. On the other hand, it is a violation of keeping a clean code and can lead to misunderstandings. This may also be the reason why most participants had encountered the smell sometimes.

The contingency table of the frequency and the severity of the test smells indicates that the less severe test smells are more frequently encountered by the developers. It could be that the less severe a test smell is, the less a developer feels obliged to refactor it or the less conspicuous the test smell becomes.

4.2 Detecting Test Smells

There is only a limited number of tools to detect test smells. One tool is TestLint [12], that was developed by Stefan Reichhart. His tool is able to identify several test smells including ‘Interactive Test’ and ‘Anonymous Test’. Another possibility to discover test smells is the NDepend tool [11] for the analysis of code in the .NET software framework. This tool uses metrics and CQL queries to find smells. Van Rompey, Du Bois, Demeyer and Rieger [14] describe a metrics-based approach to detect the test smells ‘General Fixture’ and ‘Eager Test’ which is based on their previous work on the significance of test smells [13]. Some of the test smells which were part of the survey will be discussed in the following in terms of their detectability.

Manual Intervention: The aforementioned TestLint tool supports the detection of this test smell. Reichhart’s [12] method uses the “Morphics” package of Squeak to discover the opening of a Morph which is, for example, a window.

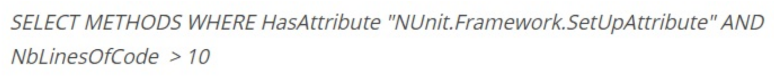
Nondeterministic Test: Since random values, specific string values and conditional logic can cause non-determinism, according to Meszaros [10], it would be a possibility to scour the code for calls to the Random class, conversions of random values into strings, if/else constructs and loops in order to detect the smell. Nondeterministic multi-threading can also be detected. As suggested by Sinha, Malik and Gupta [17], the detection method involves calculating the causality graphs of the thread events and checking that the graphs are acyclic.

Multiple Assertions: An easy way of detecting this kind of smell is to check each test method and count the number of assertions.

Unclear Naming: This test smell is detectable by the TestLint tool as well. In order to achieve this, Reichhart [12] divides the names of the test methods into smaller parts and compares all of these collected parts to the names of the classes and test methods of the production code.

Mock Happy: A way of discovering the overuse of mock objects would be, similar to the detection of ‘Multiple Assertions’, to count the number of mock objects in each test method. However, due to the fact that there is no commonly acknowledged limit value, it is quite difficult to determine at which point one can speak about an overuse of mocking.

Large Setup Methods:
Marzook [9] describes a NDepend CQL query that is capable of discovering the test smell ‘Large Setup Methods’. (Figure 4.1)



```
SELECT METHODS WHERE HasAttribute "NUnit.Framework.SetUpAttribute" AND
NbLinesOfCode  > 10
```

Figure 4.1: NDepend CQL Query to Detect ‘Large Setup Methods’

The method of counting the lines of code of a setup is not only applicable to the .NET software framework but to programming languages in general as well.

5 Conclusion and Future Work

In conclusion, the study demonstrated that test smells and especially their specific names are not quite well-known in the industrial world despite the fact that some of them occur rather frequently and pose severe problems.

From the discussion, it becomes evident that less severe test smells tend to be more frequent than severe test smells. Moreover, the estimation of the severity of some test smells depends on the experience level of the developer and can vary vastly.

Nevertheless, this investigation used only a limited sample and thus, it is difficult to draw conclusions about a whole community of programmers. In future, it might be interesting to conduct the survey with a larger number of participants and to implement a detection tool for some of the test smells.

6 Anleitung zu wissenschaftlichen Arbeiten: A Taxonomy of Test Smells

This chapter will provide a taxonomy of test smells arranged according to the principles and good practices they violate. The names written in blue are synonyms of the respective test smells that were found in books, papers *etc.* whilst the ones in red were collected from the survey. The tables present an overview of the taxonomy (Table 6.1, Table 6.2). For detailed descriptions the sections mentioned in the table should be consulted.

Principle	**Test Smell(s)**	**Section**
Test Automation	Frequent Debugging Manual Intervention	Section 6.1
Determinism	Unrepeatable Test Nondeterministic Test	Section 6.2
Test One Thing	Long Test Assertion Roulette The Giant The One	Section 6.3
Only Test One Unit	Test Envy Indirect Testing The Stranger	Section 6.4
Correct Use of Assertions	No Assertions Inappropriate Assertions Redundant Assertions	Section 6.5
Reliability	Fragile Test The Sleeper Buggy Test	Section 6.6

Table 6.1: Overview of the Taxonomy (part 1)

Principle	Test Smell(s)	Section
Encapsulation	The Inspector The Peeping Tom	Section 6.7
Performance	Slow Test	Section 6.8
No Test Logic in Production Code	For Tests Only	Section 6.9
Communicate Intent	Overly Dry Tests Anonymous Test Conditional Test Logic Overcommented Test	Section 6.10
Appropriate Use of Mock Objects	Mock Happy	Section 6.11
Independence	Interacting Tests Order Dependent Tests Generous Leftovers	Section 6.12
	External Dependencies	Subsection 6.12.1
Appropriate Setup	Excessive Setup Excess Inline Setup Fragile Fixture General Fixture Inappropriately Shared Fixture	Section 6.13
Not Affecting the Environment	Not Idempotent The Loudmouth Sloppy Worker	Section 6.14
Exception Handling	Catching Unexpected Exceptions The Secret Catcher The Greedy Catcher	Section 6.15
Diligent Testing	Test Code Duplication Lazy Test The Liar The Dead Tree	Section 6.16
	Commented Code Skip-Epidemic	Subsection 6.16.1
	Not Writing Tests Success Against All Odds The Turing Test	Subsection 6.16.2
	Happy Path Easy Test Overspecified Software	Subsection 6.16.3

Table 6.2: Overview of the Taxonomy (part 2)

6.1 Test Automation

A test should not need any manual interaction after it is run. The following test smells violate this concept:

- **Frequent Debugging [10]:** The developer has to manually debug most of the failures that occur in the test.

- **Manual Intervention [10]:** (Interactive Test [12], Manual Testing, Manual Test) The test requires a manual action by the developer. There are three types:
  - Manual Fixture Setup: In advance of running the test, a manual setup of the test environment must be done.
  - Manual Result Verification: The results are manually verified and therefore, the tests even pass if the production code returns wrong results.
  - Manual Event Injection: During the execution of the test, a manual action must be performed so that the test can proceed. (see also [16])

  A further variant that was mentioned in the survey is:
  - Works on my Machine

6.2 Determinism

Tests should be repeatable, meaning that they should render the same results if they are run several times. This principle is violated by the smells:

- **Unrepeatable Test [10]:** The test changes its behaviour after its first execution. The two possible scenarios for this test smell are Fail-Pass-Pass and Pass-Fail-Fail.
- **Nondeterministic Test [10]:** The test fails at random even if only one Test Runner is being used.

6.3 Test One Thing

A test should have some value as a documentation of the tested system. Therefore, it is reasonable to concentrate on one thing only in a test or test method. The following test smells decrease the understandability of test codes:

- **Long Test [10]:**(Obscure Test, Verbose Test [10], Complex Test [10, 15]) The test is too long in general and thus, is complicated to understand.
- **Assertion Roulette [4, 10, 21]:** If there are many assertions in a test method, it becomes difficult to determine which one caused a test failure. There are two types:
  - Eager Test: (Many Assertions [16], Multiple Assertions [15], The Free Ride [5]) A test method handles too much functionality meaning that several assertions belong to the same test method. (see also [4, 13, 21])
  - Missing Assertion Message: There are several assertions of the same kind which either have identical assertion methods or lack an assertion message. Thus, if one assertion fails, it is impossible to determine which one it is.
- **The Giant [5]:** The unit test is huge and contains many test methods although it tests only one object.
- **The One [5]:** (The Test It All [20]) This smell is a combination of the test smells ‘The Free Ride’ and ‘The Giant’. It is a unit test that consists of only one test method which, in addition, tests the whole functionality of an object.

6.4 Only Test One Unit

A unit test class should only test one unit. This principle is violated by the following test smells:

- **Test Envy [16]:** The test tests another class instead of the one it is supposed to test.
- **Indirect Testing [10, 21]:** The test method tests an object indirectly via another object.
- **The Stranger [5]:** (The Distant Relative) The test case does not belong to the unit test in which it is. It tests another object that was probably used by the tested object.

6.5 Correct Use of Assertions

A test should always assert something. Moreover, it is essential for a developer to know the possibilities of assertion so that he or she can decide on the one that is most appropriate. There are once again test smells that are a violation of this principle, which can be seen in the following list:

- **No Assertions [16]:** (Lying Test [4], Assertionless Test [12], The Line Hitter [20]) The test does not contain any assertions and does nothing else than to execute the production code and raising the code coverage.
- **Inappropriate Assertions [1]:** (Wrong Assert [15]) An inappropriate assertion is being used. For example, *assertTrue* is used to check the equality of values instead of *assertEquals*.
- **Redundant Assertions [15]:** An extra call to an assert method is made for a condition that is hard coded true.

6.6 Reliability

A test should be reliable. This concept is violated by the following smells:

- **Fragile Test [4, 10]:** (Brittle Test [16]) The test does not compile or cannot be run due to changes in test-independent parts of the production code. There are several different types according to which part of the system has changed:
  - Interface Sensitivity: Changes to the interface that is used by the tested system affect the execution of the test.
  - Behaviour Sensitivity: Changes to the production code lead to the failure of tests.
  - Data Sensitivity: Modifications to the data on which the system under test relies cause the test to fail.
  - Context Sensitivity: Changing the state or behaviour of the context in which the production code is embedded causes the failure of the test. This is the case, for example, when the code that is changed does not belong to the system under test.
  - Sensitive Equality: (The Butterfly [20]) Equality checks depend on toString method, which relies on minor details like commas or quotes. Therefore, any change to the toString method can result in a failure of the test. (see also [21])
- **The Sleeper [20]:** (Mount Vesuvius) The test is designed in a way that it will certainly fail at some point due to Date and Calendar objects whose bounds are checked incorrectly.
- **Buggy Tests [10]:** The production code is correct but its corresponding test is not, which means that there is a bug in the test code.

6.7 Encapsulation

The encapsulation principle holds true for test codes as well. Test smells that violate this are the following:

- **The Inspector [5]:** Encapsulation is disregarded by reading private fields or an extension of the class to access protected fields. The aim is to achieve better code coverage. There is a variant that slightly differs:
  - Anal Probe [20]: A test that is written in this 'smelly' way also violates encapsulation but due to unnecessary hiding of parts in the tested production code that need testing.
- **The Peeping Tom [5]:** (The Uninvited Guests) A test gains insight into the results of other tests because of shared resources.

6.8 Performance

Since it is a good practice to run tests frequently, they should run as fast as possible. There is only one test smell that violates this:

- **Slow Test [4, 10]:** (Long Running Test [16], The Slow Poke [5]) The test runs too slowly.

6.9 No Test Logic in Production Code

Production code and test code should always be kept separate. The following smell defies this principle:

- **For Tests Only [10]:** (For Testers Only [21]) The production code contains code that is strictly reserved for tests.

6.10 Communicate Intent

Tests should be understandable and thus, it is important that the intent of a test can be discerned at a first glance. In order to achieve this it is sometimes advisable, as Lundberg [8] suggests, to design tests rather DAMP [1] instead of DRY [2]. Several test smells disregard this principle:

- **Overly Dry Tests [16]:** Making the DRY principle the top priority can lead to a hardly understandable test code.
- **Anonymous Test [12]:** (Unclear Naming, Naming Convention Violation) The test name is meaningless and does not reveal the intent of it. Variants of this are:
  - The Enumerator [5]: The names of the test methods are only an enumeration. For example, test1, test2 *etc.*
  - The Test With No Name [20]: To reproduce a bug in the code a test method is added and named after the bug. For example, testForBUG1. Later, the developer needs to find the bug in the bug in order to understand the intent of the test method.

[1]**D**escriptive **A**nd **M**eaningful **P**hrases
[2]**D**on't **R**epeat **Y**ourself

Additional variants that were mentioned in the survey are:

  - Unclear Documentation
  - No Documentation
  - Name your Class

- **Conditional Test Logic [4, 7, 10]:** (Indented Test Code [10], Guarded Test [12]) This is a test that contains either boolean branching logics like if/else-statements or loops like while. Such conditionals can lead to unpredictable results and decrease the understandability of the code.

- **Overcommented Test [12]:** Since tests are a documentation themselves, comments in the test code are superfluous. Moreover, the understandability of the code can be afflicted by it if there are too many comments.

6.11 Appropriate Use Of Mock Objects

It is reasonable to use mock objects when testing. However, overuse of mock objects can harm the utility of the test code. The following test smell represents such a misuse of mock objects:

- **Mock Happy [16]:** (Mock-Overkill [15], The Mockery [5]) If one uses too many mock objects that interact with each other, the test will only test the interaction between them instead of the functionality of the production code.

6.12 Independence

Tests should be independent, especially from each other. This concept is violated by the following test smells:

- **Interacting Tests [10]:** The tests are dependent on each other. There are two variations of this smell:

  - Interacting Test Suites: The tests which are dependent on each other belong to different test suites.
  - Lonely Test: The test can only be run in a collection of tests (test suite) but not on its own since it depends on the shared fixture of another test in the collection.

- **Order Dependent Tests [1, 16]:** (Chained Tests [10], Chain Gang [20]) The tests have to be executed in a certain order due to dependencies between them.

- **Generous Leftovers [5]:** The unit test creates data which remains afterwards and is used by other tests for their own purposes. This can cause the first test to fail if it tries to reuse the data.

6.12.1 Independence from External Influences

Tests should not only be independent from each other but should also be unaffected by external influences. The following list consists of test smells that violate this principle:

- **External Dependencies [15]:** The tests depend on external factors. There are several test smells that differ in the source of influence:

- External Resources in General:
    - **Resource Leakage [10]:** The test uses finite resources. If the tests do not release them in the end, all of the resources are allocated and tests which need the resources begin to fail with time.
    - **Resource Optimism [10, 21]:** The test shows nondeterministic behaviour depending on the place and time it is run since it optimistically speculates on the state and presence or absence of external resources.
    - **Mystery Guest [10, 21]:** (External Data [16]) The test relies on information (from external resources) that are not visible from within the test. This introduces hidden dependencies and the test reader has difficulties to understand the verified behaviour.
    - **Test Run War(s) [10, 21]:** The test begins to fail when several developers run it simultaneously due to its dependence on a shared resource.
- Environment:
    - **The Local Hero [5]:** The test relies on features that are specific to the development environment. Thus, it cannot be run in another environment.
    - **The Operating System Evangelist [5]:** The unit test can only be run on a specific operating system because it depends on the environment of the operating system.
- Data:
    - **Hidden Dependency [5]:** This smell is related to 'The Local Hero'. The programmer needs to manually create data before the test can be run.
    - **The Sequencer [5]:** The unit test relies on items in an unordered list to be in the same order during the assertion.
- Web:
    - **Web-Browsing Test [4]:** To be run the test needs to establish a connection to the internet.

6.13 Appropriate Setup

The setup or fixture of tests should be kept appropriate and as small as possible. The test smells in the subsequent list disregard this principle.

- **Excessive Setup [5]:** (Large Setup Methods [9]) A mega setup is needed in order for the test to be run. There is a variation of this:
    - Refused Bequest [16]: The excessive setup is only needed by some tests in its entirety. The other tests either use it partially or do not use it at all.
- **Excessive Inline Setup [16]:** There is too much setup in each of the tests/test methods.
- **Fragile Fixture [10]:** The modification of the fixture to enable the execution of a new test causes other tests to fail.
- **General Fixture [10, 13, 21]:** (Badly Used Fixture [12]) The setup contains more elements than is needed by any of the tests.
- **Inappropriately Shared Fixture [20]:** Some tests in the fixture have no need of a setup at all. Variants of this test smell are:
    - The Cuckoo: The unit test belongs to a test case with several other tests and shares the setup with them. However, it disposes of parts or all of the setup and builds its own setup.
    - The Mother Hen: The setup does more than any of the tests need.

6.14 Not Affecting the Environment

The environment in which the test is run should remain unaffected by the test. Violations of this principle are represented in the following list:

- **Not Idempotent [16]:** (Interacting Test With High Dependency) The test alters the states of parts of the environment forever.
- **The Loudmouth [5]:** (Transcripting Test [12]) The test writes output in the console or in a global stream.
- **Sloppy Worker [5]:** (Wet Floor [20]) The test generates data but it omits dispensing with the data after finishing. This will cause itself and other tests to fail on consecutive runs.

6.15 Exception Handling

Test code must handle exceptions correctly. Mishandling of exceptions is present in the following test smells:

- **Catching Unexpected Exceptions [15]:** If unexpected exceptions are caught, the tests can pass even if an exception is thrown.
- **The Secret Catcher [5]:** (The Silent Catcher [20]) The test contains no assertions but the test needs an exception to be thrown and to be reported by the testing framework.
- **The Greedy [5]:** The unit test catches an exception and either hides the stack trace or substitutes it with a failure message that has less documentation value. In the end, the test will pass because of this.

6.16 Diligent Testing

There are several good practices that deal with how to approach the writing of test code in general. The following test smells are examples of bad testing behaviour:

- **Test Code Duplication [4, 10, 21]:** (Second Class Citizen [20]) The test code is duplicated several times. This is either done on purpose by copying and pasting the code or the same code is unintentionally written again.
- **Lazy Test [21]:** The test methods test the same method of the production code relying on the same fixture.
- **The Liar [5]:** The unit test passes but it does not verify the functionality of the object it should really test.
- **The Dead Tree [20]:** (Process Compliance Backdoor) A stub was made but the corresponding test has never been written.

6.16.1 Ignoring Failing Tests

A test should never ignore failing tests. This is violated by the following test smells:

- **Commented Code in the Test [16]:** (Under-the-carpet failing Assertion [12]) The test passes but some assertions or parts of the setup, which would cause a test failure, are commented out.
- **Skip-Epidemic [4]:** Failing tests are constantly skipped over.

6.16.2 Write Tests (First)

It is a good programming behaviour to write tests and its even better to write them prior to the production code. Violations of this concept are represented by the following smells:

- **Developers Not Writing Tests [10]:** (No Unit Tests [15]) The developers do not write any tests at all.
- **Success Against All Odds [5]:** The tests are written first but they are written in a way that they pass first rather than failing in the first execution.
- **The Turing Test [20]:** A tool generates the test cases automatically.

6.16.3 Test All Paths and Real Behaviour

The tests should test all paths and the real behaviour of the production code. The following test smells disregard this principle:

- **Happy Path [15]:** The test methods only check the behaviour that is expected from the production code.
- **Easy Tests [15]:** (The Dodger [5]) This kind of test only checks simple things of the production code but it neglects the real logic of it.
- **Overspecified Software [10]:** (Overcoupled Test) The test focuses on the structure and the expected behaviour of the software rather than on what it should accomplish. Therefore, the software must be designed in a specific way in order that the test can pass.

Bibliography

[1] D. Astels. *Test-Driven Development: A Practical Guide*. Prentice Hall PTR, New Jersey, 2003.

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

[3] A. Beer and R. Ramler. The Role of Experience in Software Testing Practice. *34th Euromicro Conference Software Engineering and Advanced Applications*, pages 258 – 265, September 2008.

[4] S. Bergmann and S. Priebsch. *Softwarequalität in PHP-Projekten*. Carl Hanser Verlag, München, 2013.

[5] J. Carr. TDD Anti-Patterns. `http://blog.james-carr.org/2006/11/03/tdd-anti-patterns/`, November 2006.

[6] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, 2007.

[7] C. Hartjes. Testing Smells - try/catch. `http://www.littlehart.net/atthekeyboard/2013/04/30/testing-smells-try-catch/`, April 2013.

[8] J. Lundberg. Dry and Damp principles when developing and unit testing. `http://codeshelter.wordpress.com/2011/04/07/dry-and-damp-principles-when-developing-and-unit-testing/`, April 2011.

[9] H. Marzook. Test Smells and Test Code Quality Metrics. `http://www.hibri.net/2009/12/09/test-smells-and-test-code-quality-metrics/`, December 2009.

[10] G. Meszaros. *xUnit Test Patterns: Refactoring Test Code*. Addison-Wesley, Boston, 2007.

[11] NDepend. `http://www.ndepend.com/`.

[12] S. Reichhart. Assessing Test Quality: TestLint. Master's thesis, Institute of Computer Science and Applied Mathematics, University of Bern, April 2007.

[13] B. Van Rompaey, B. Du Bois, and S. Demeyer. Characterizing the Relative Significance of a Test Smell. *22nd IEEE International Conference on Software Maintenance (ICSM'06)*, pages 391 – 400, September 2006.

[14] B. Van Rompaey, B. Du Bois, S. Demeyer, and M. Rieger. On the Detection of Test Smells: A Metrics-Based Approach for General Fixture and Eager Test. *IEEE Transactions on Software Engineering*, 33(12):800 – 817, December 2007.

[15] J. Schmetzer. JUnit Anti-patterns. `http://www.exubero.com/junit/antipatterns.html`.

[16] J. Scruggs. Smells of Testing (signs your tests are bad). `http://jakescruggs.blogspot.ch/2009/04/smells-of-testing-signs-your-tests-are.html`, April 2009.

[17] A. Sinha, S. Malik, and A. Gupta. Efficient Predictive Analysis for Detecting Nondeterminism in Multi-Threaded Programs. *Proceedings of the 12th Conference on Formal Methods in Computer-Aided Design (FMCAD 2012)*, pages 6 – 15, October 2012.

[18] Survey Test Smells. `http://testing-survey.iam.unibe.ch`.

[19] Twitter. `http://twitter.com`.

[20] Gishu (Username). Unit testing Anti-patterns catalogue. `http://stackoverflow.com/questions/333682/unit-testing-anti-patterns-catalogue`, December 2008.

[21] A. van Deursen, L. Moonen, A. van den Bergh, and G. Kok. Refactoring Test Code. *Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2001)*, pages 92 – 95, 2001.

[22] WordPress. `http://wordpress.com`.

[23] xeno010 (Username). Wp-Pro-Quiz. `http://wordpress.org/plugins/wp-pro-quiz/`.

A
Interview Questionnaire

Interview:

Language(s):

Programming Experience: **Industrial Experience:**

Tests should be automated:

Smell	Frequency	Severity	Syn./Var./Spec.	Solution

Tests should be deterministic:

Smell	Frequency	Severity	Syn./Var./Spec.	Solution

Test method should only test one thing:

Smell	Frequency	Severity	Syn./Var./Spec.	Solution

(Unit) Test case (class) should only test one unit:

Smell	Frequency	Severity	Syn./Var./Spec.	Solution

Correct Assertion Use:

Smell	Frequency	Severity	Syn./Var./Spec.	Solution

Tests should be reliable:

Smell	Frequency	Severity	Syn./Var./Spec.	Solution

Encapsulation:

Smell	Frequency	Severity	Syn./Var./Spec.	Solution

Tests should run fast:

Smell	Frequency	Severity	Syn./Var./Spec.	Solution

Keep test logic out of production code:

Smell	Frequency	Severity	Syn./Var./Spec.	Solution

Communicate intent:

Smell	Frequency	Severity	Syn./Var./Spec.	Solution

Mock objects:

Smell	Frequency	Severity	Syn./Var./Spec.	Solution

Tests should be independent:

Smell	Frequency	Severity	Syn./Var./Spec.	Solution

Keep Setup/Fixture appropriate and small:

Smell	Frequency	Severity	Syn./Var./Spec.	Solution

Test should not affect environment:

Smell	Frequency	Severity	Syn./Var./Spec.	Solution

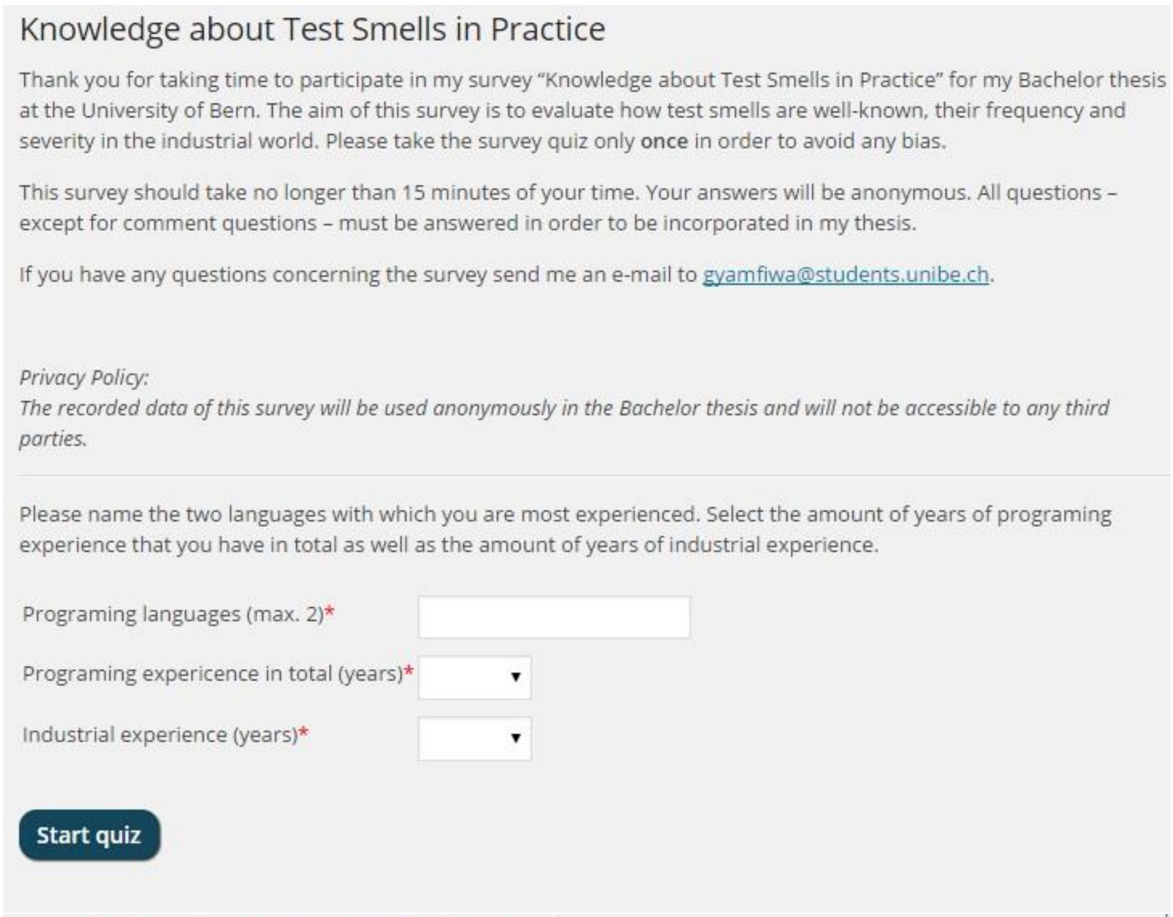
Good test behaviour:

Smell	Frequency	Severity	Syn./Var./Spec.	Solution

Tests should handle exceptions correctly:

Smell	Frequency	Severity	Syn./Var./Spec.	Solution

B
Survey Questionnaire



Knowledge about Test Smells in Practice

Thank you for taking time to participate in my survey "Knowledge about Test Smells in Practice" for my Bachelor thesis at the University of Bern. The aim of this survey is to evaluate how test smells are well-known, their frequency and severity in the industrial world. Please take the survey quiz only **once** in order to avoid any bias.

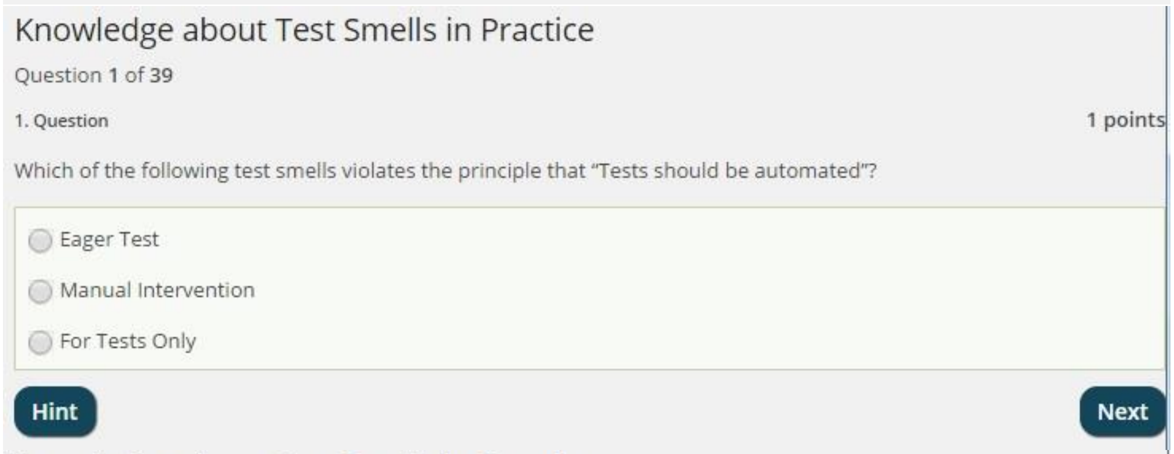
This survey should take no longer than 15 minutes of your time. Your answers will be anonymous. All questions – except for comment questions – must be answered in order to be incorporated in my thesis.

If you have any questions concerning the survey send me an e-mail to gyamfiwa@students.unibe.ch.

Privacy Policy:
The recorded data of this survey will be used anonymously in the Bachelor thesis and will not be accessible to any third parties.

Please name the two languages with which you are most experienced. Select the amount of years of programing experience that you have in total as well as the amount of years of industrial experience.

Programing languages (max. 2)*	
Programing expericence in total (years)*	▾
Industrial experience (years)*	▾



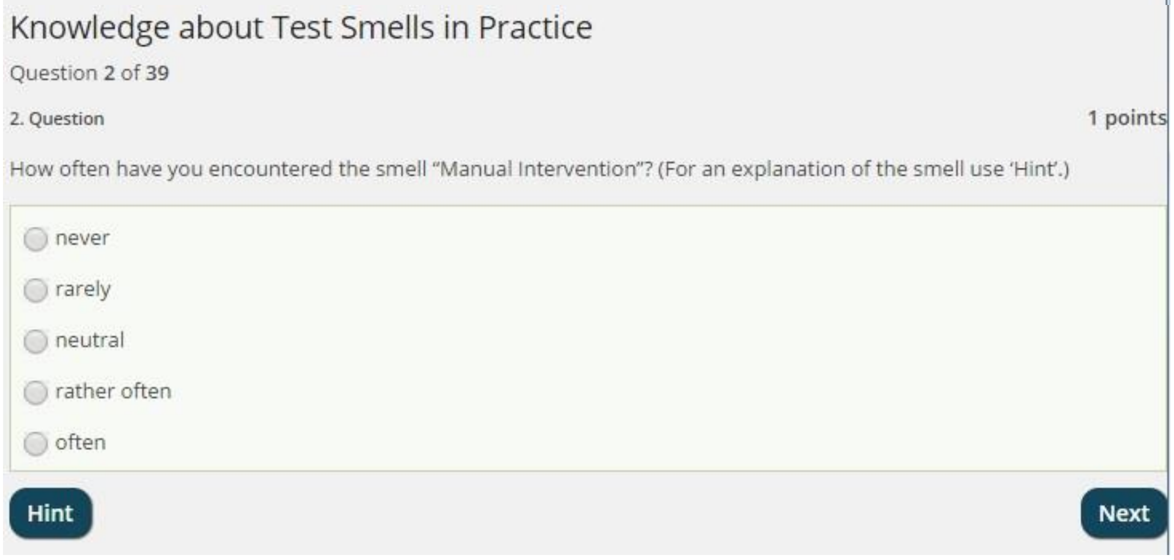
Knowledge about Test Smells in Practice

Question 1 of 39

1. Question 1 points

Which of the following test smells violates the principle that "Tests should be automated"?

○ Eager Test
○ Manual Intervention
○ For Tests Only



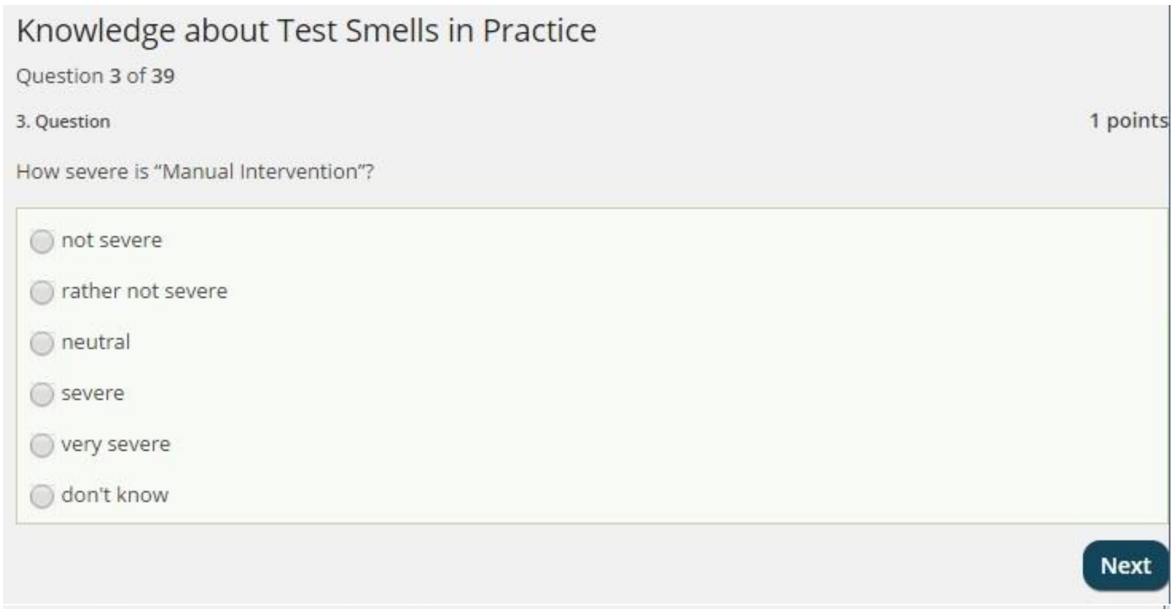
Knowledge about Test Smells in Practice

Question 2 of 39

2. Question 1 points

How often have you encountered the smell "Manual Intervention"? (For an explanation of the smell use 'Hint'.)

○ never
○ rarely
○ neutral
○ rather often
○ often

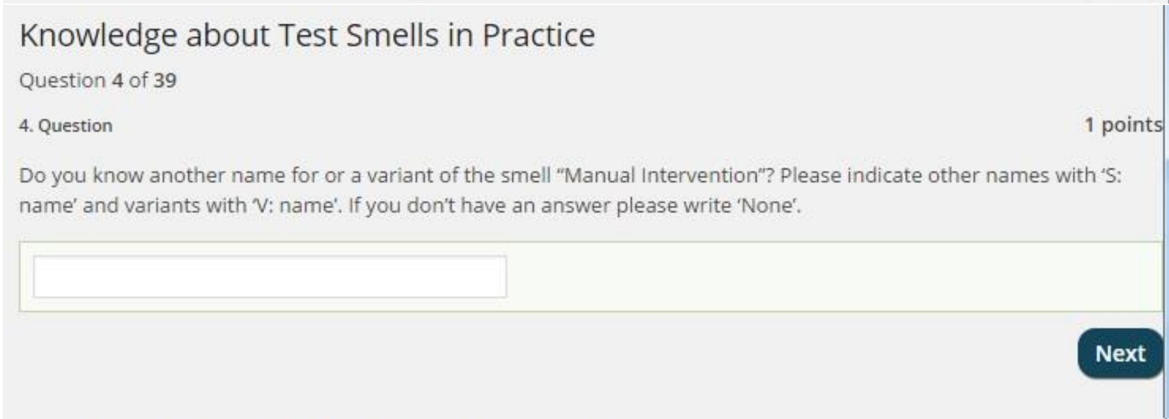

Knowledge about Test Smells in Practice

Question 3 of 39

3. Question — 1 points

How severe is "Manual Intervention"?

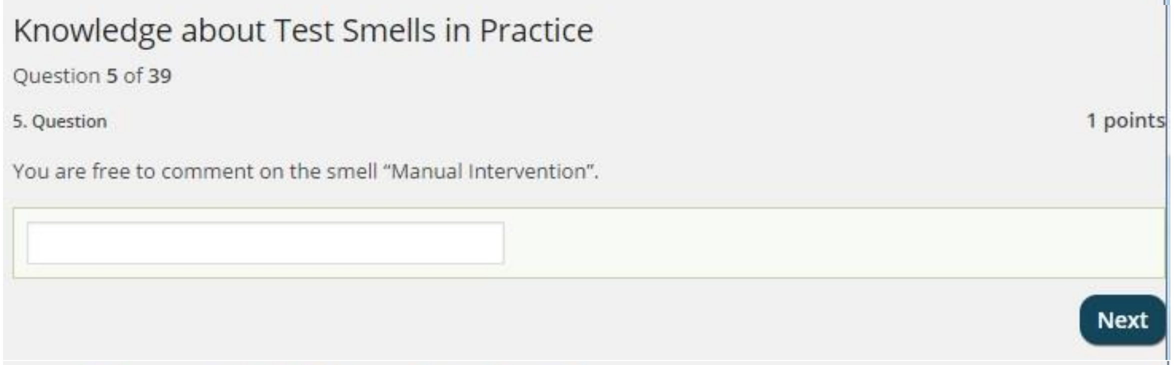
- not severe
- rather not severe
- neutral
- severe
- very severe
- don't know


Knowledge about Test Smells in Practice

Question 4 of 39

4. Question — 1 points

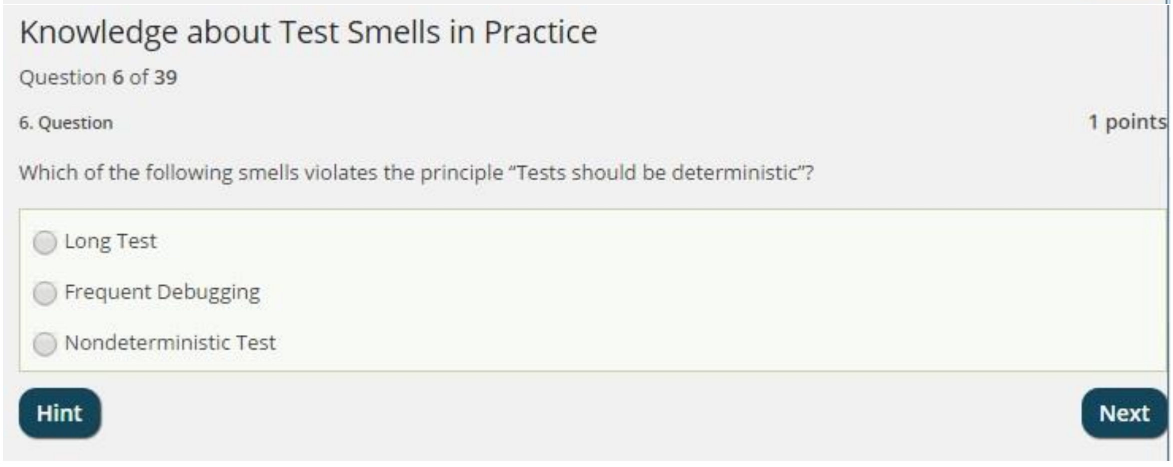
Do you know another name for or a variant of the smell "Manual Intervention"? Please indicate other names with 'S: name' and variants with 'V: name'. If you don't have an answer please write 'None'.


Knowledge about Test Smells in Practice

Question 5 of 39

5. Question — 1 points

You are free to comment on the smell "Manual Intervention".

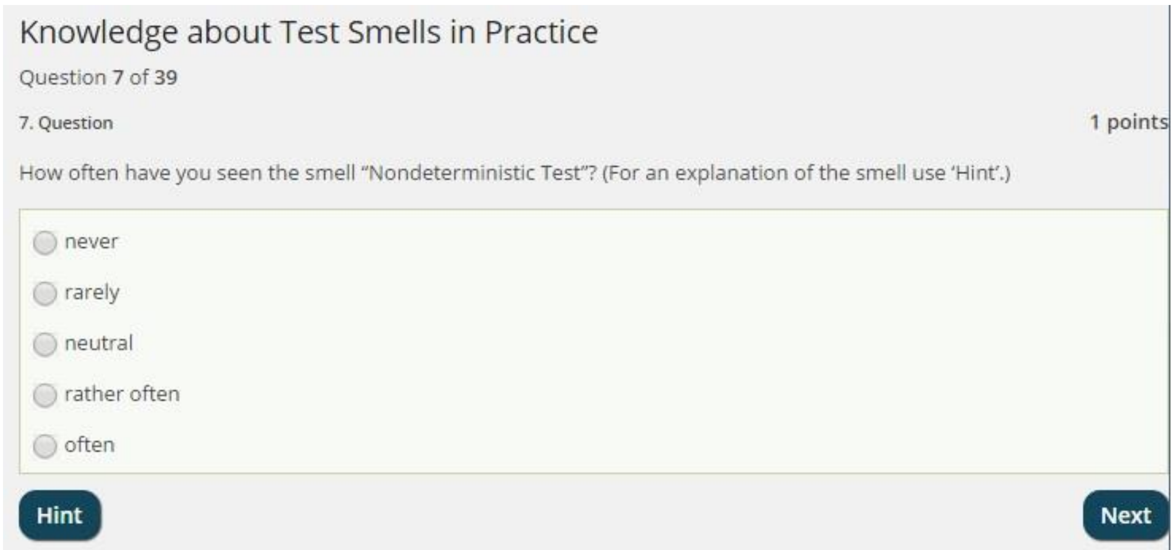

Knowledge about Test Smells in Practice

Question 6 of 39

6. Question — 1 points

Which of the following smells violates the principle "Tests should be deterministic"?

- Long Test
- Frequent Debugging
- Nondeterministic Test



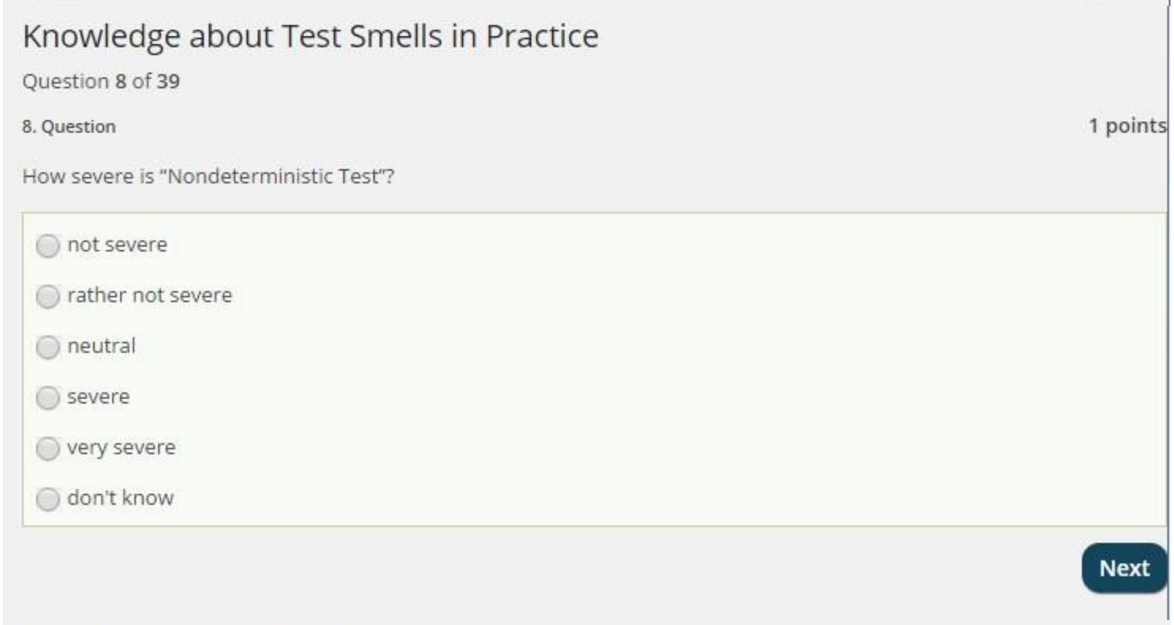
Knowledge about Test Smells in Practice

Question 7 of 39

7. Question 1 points

How often have you seen the smell "Nondeterministic Test"? (For an explanation of the smell use 'Hint'.)

- ○ never
- ○ rarely
- ○ neutral
- ○ rather often
- ○ often



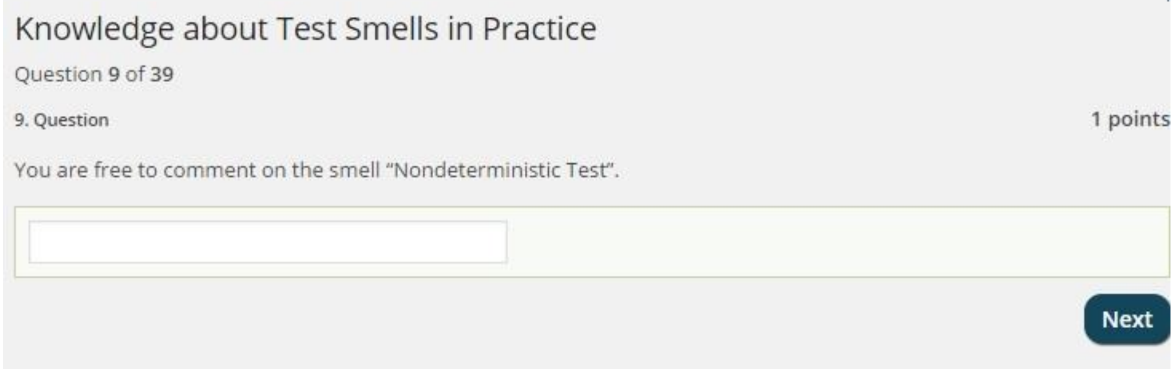
Knowledge about Test Smells in Practice

Question 8 of 39

8. Question 1 points

How severe is "Nondeterministic Test"?

- ○ not severe
- ○ rather not severe
- ○ neutral
- ○ severe
- ○ very severe
- ○ don't know

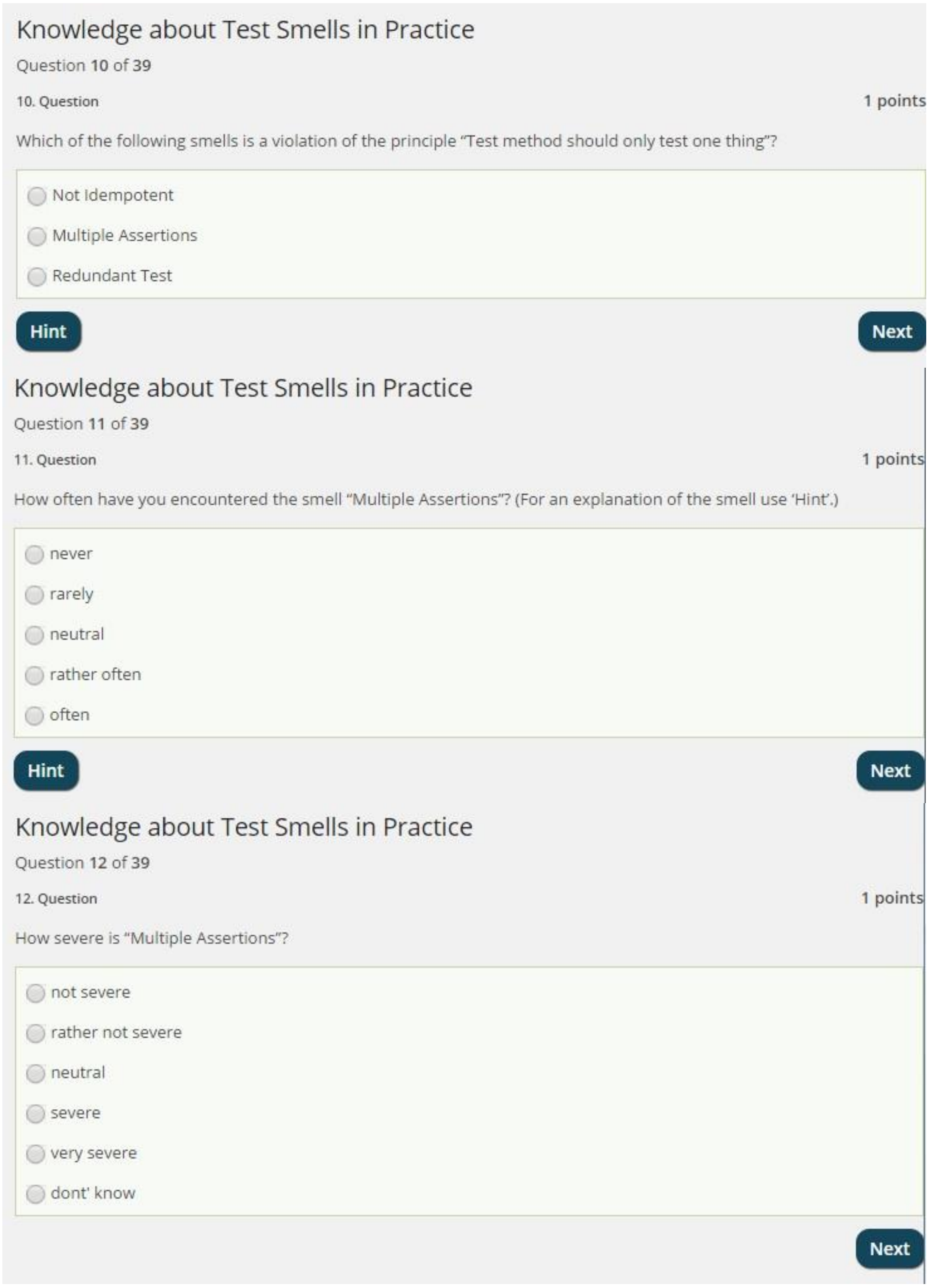


Knowledge about Test Smells in Practice

Question 9 of 39

9. Question 1 points

You are free to comment on the smell "Nondeterministic Test".



Knowledge about Test Smells in Practice

Question **10** of **39**

10. Question — **1 points**

Which of the following smells is a violation of the principle "Test method should only test one thing"?

○ Not Idempotent

○ Multiple Assertions

○ Redundant Test

Hint **Next**

Knowledge about Test Smells in Practice

Question **11** of **39**

11. Question — **1 points**

How often have you encountered the smell "Multiple Assertions"? (For an explanation of the smell use 'Hint'.)

○ never

○ rarely

○ neutral

○ rather often

○ often

Hint **Next**

Knowledge about Test Smells in Practice

Question **12** of **39**

12. Question — **1 points**

How severe is "Multiple Assertions"?

○ not severe

○ rather not severe

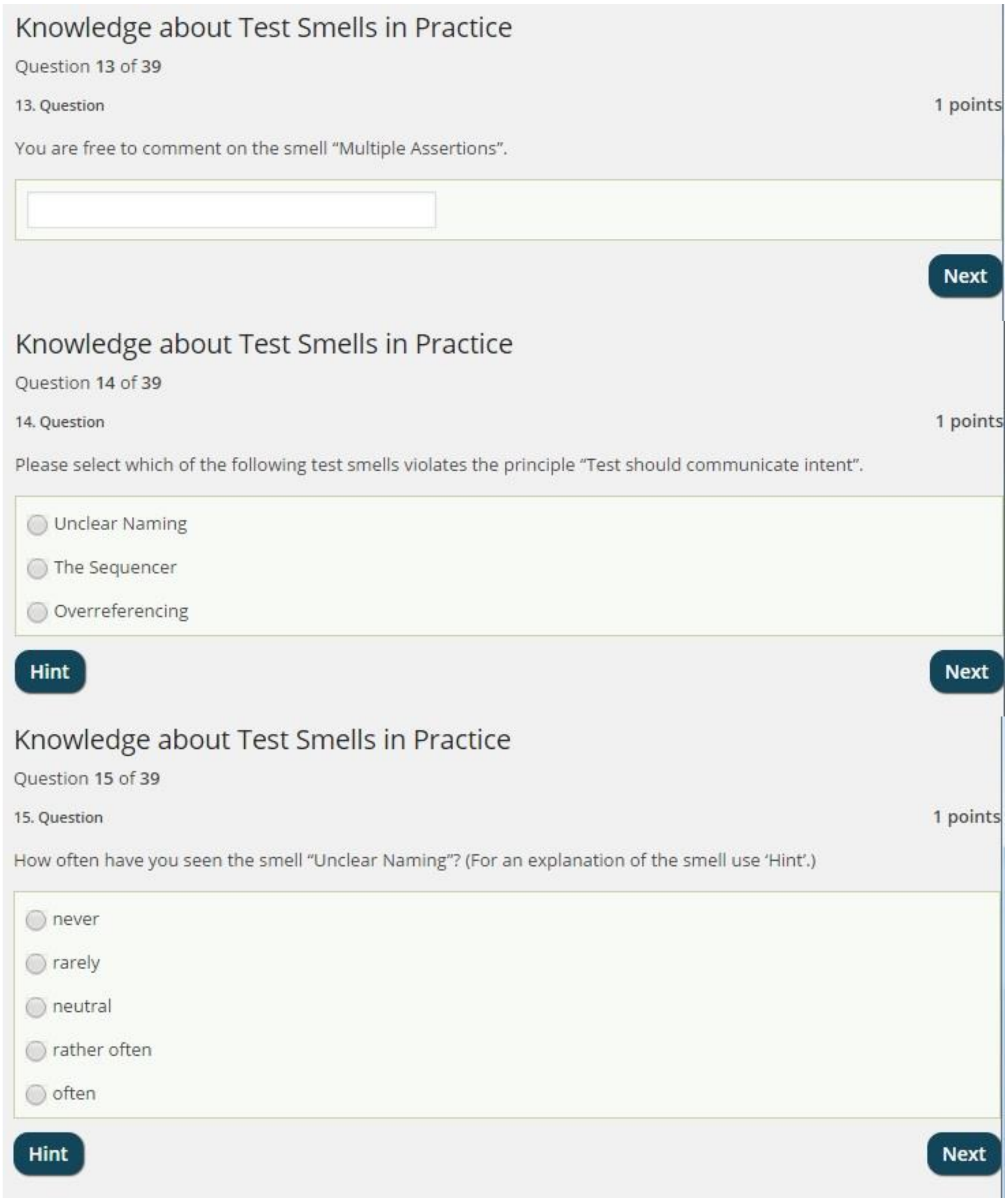
○ neutral

○ severe

○ very severe

○ dont' know

Next



Knowledge about Test Smells in Practice

Question **13** of **39**

13. Question — **1 points**

You are free to comment on the smell “Multiple Assertions”.

Next

Knowledge about Test Smells in Practice

Question **14** of **39**

14. Question — **1 points**

Please select which of the following test smells violates the principle “Test should communicate intent”.

○ Unclear Naming
○ The Sequencer
○ Overreferencing

Hint **Next**

Knowledge about Test Smells in Practice

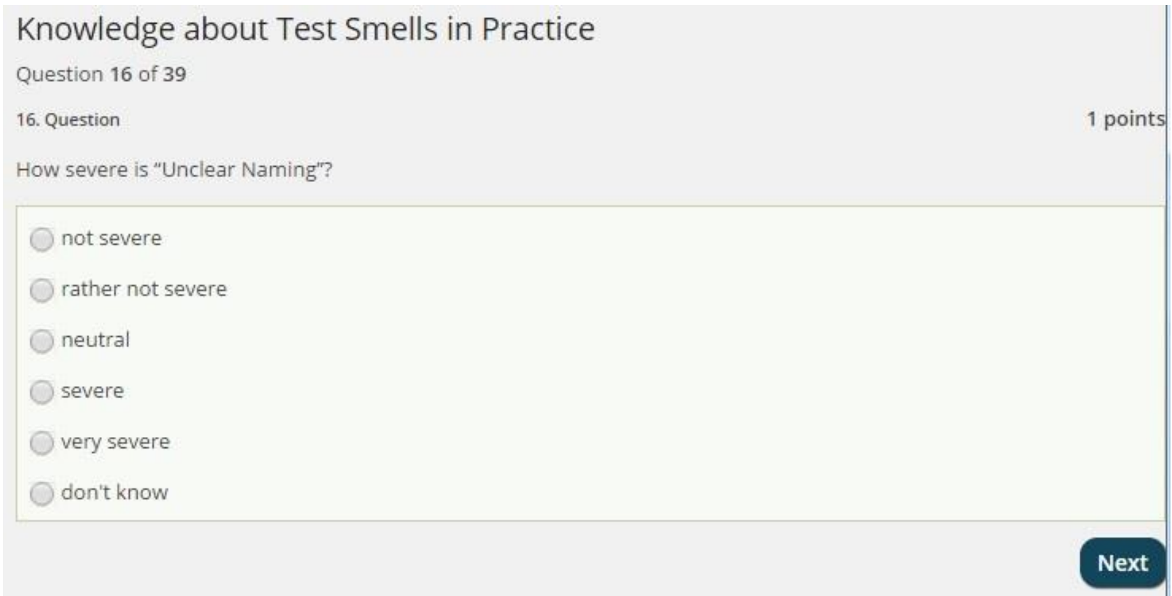
Question **15** of **39**

15. Question — **1 points**

How often have you seen the smell “Unclear Naming”? (For an explanation of the smell use ‘Hint’.)

○ never
○ rarely
○ neutral
○ rather often
○ often

Hint **Next**



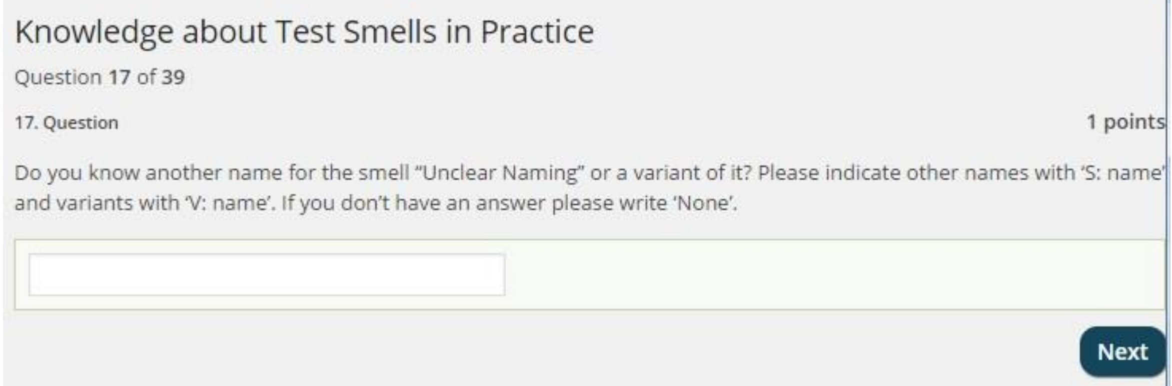
Knowledge about Test Smells in Practice

Question **16** of **39**

16. Question **1 points**

How severe is "Unclear Naming"?

○ not severe
○ rather not severe
○ neutral
○ severe
○ very severe
○ don't know

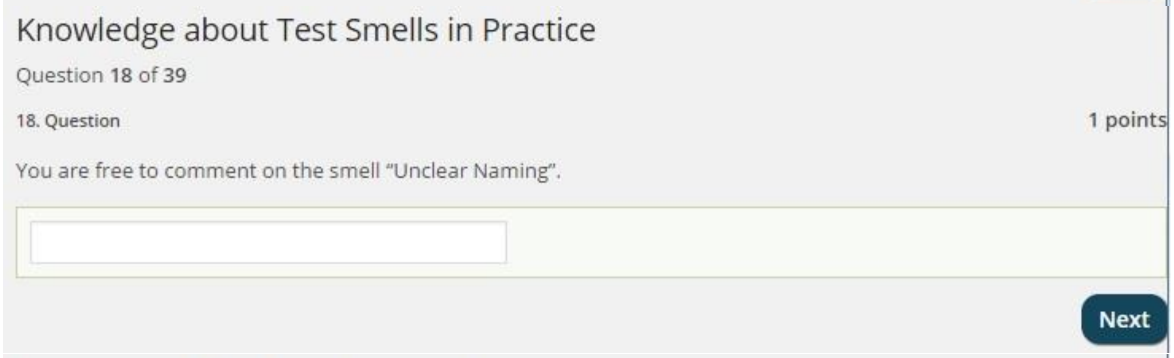


Knowledge about Test Smells in Practice

Question **17** of **39**

17. Question **1 points**

Do you know another name for the smell "Unclear Naming" or a variant of it? Please indicate other names with 'S: name' and variants with 'V: name'. If you don't have an answer please write 'None'.

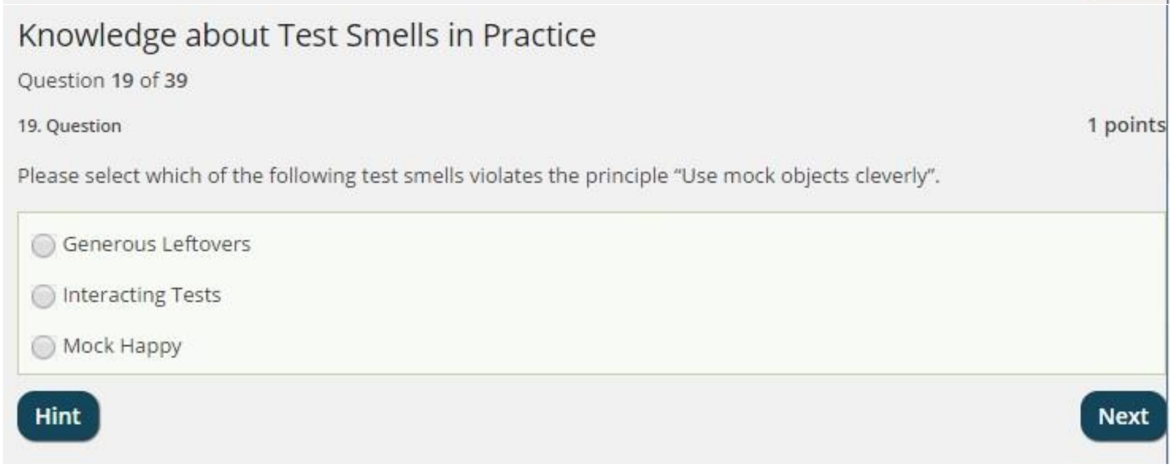


Knowledge about Test Smells in Practice

Question **18** of **39**

18. Question **1 points**

You are free to comment on the smell "Unclear Naming".



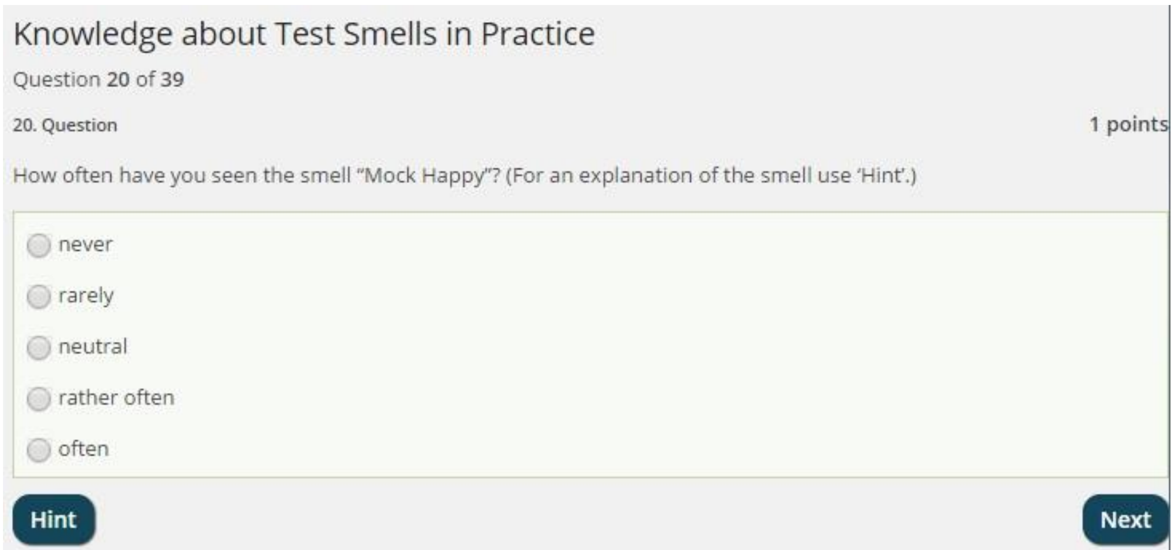
Knowledge about Test Smells in Practice

Question **19** of **39**

19. Question **1 points**

Please select which of the following test smells violates the principle "Use mock objects cleverly".

○ Generous Leftovers
○ Interacting Tests
○ Mock Happy



Knowledge about Test Smells in Practice

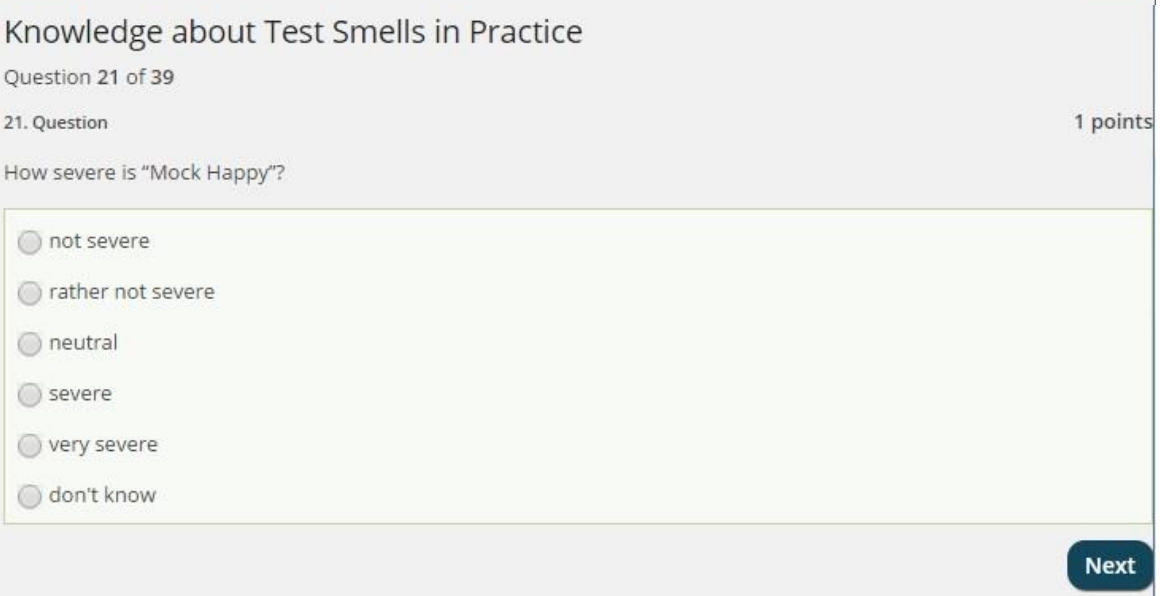
Question 20 of 39

20. Question

1 points

How often have you seen the smell "Mock Happy"? (For an explanation of the smell use 'Hint'.)

- never
- rarely
- neutral
- rather often
- often



Knowledge about Test Smells in Practice

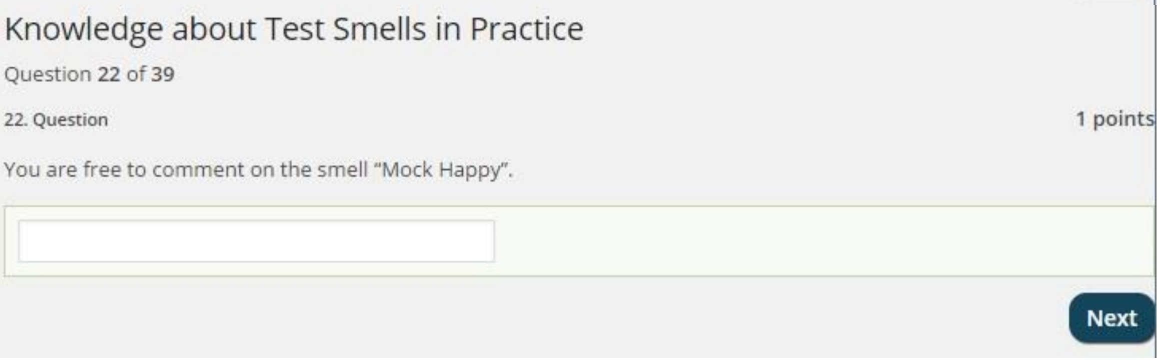
Question 21 of 39

21. Question

1 points

How severe is "Mock Happy"?

- not severe
- rather not severe
- neutral
- severe
- very severe
- don't know



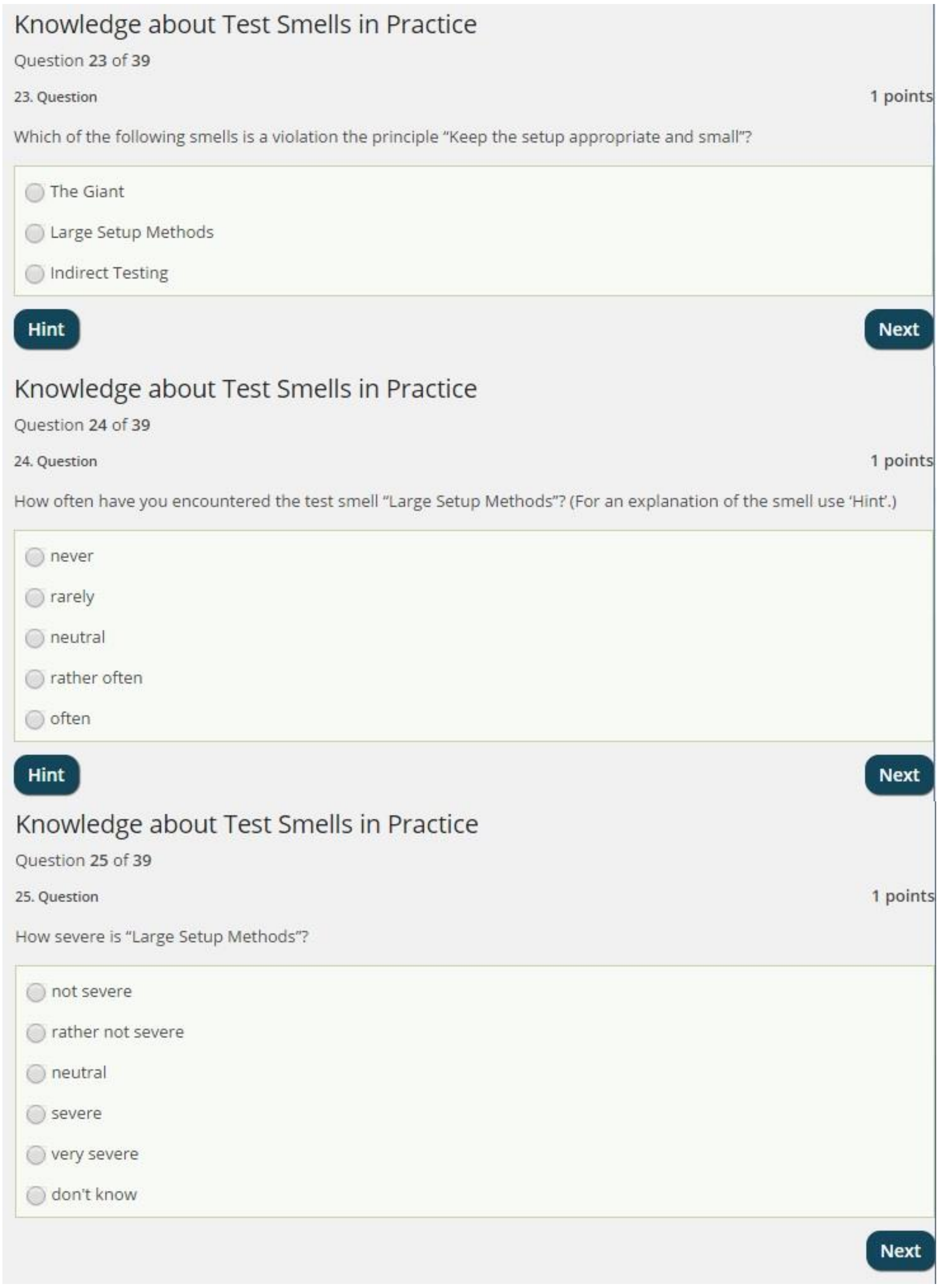
Knowledge about Test Smells in Practice

Question 22 of 39

22. Question

1 points

You are free to comment on the smell "Mock Happy".



Knowledge about Test Smells in Practice

Question 23 of 39

23. Question — 1 points

Which of the following smells is a violation the principle "Keep the setup appropriate and small"?

- The Giant
- Large Setup Methods
- Indirect Testing

Knowledge about Test Smells in Practice

Question 24 of 39

24. Question — 1 points

How often have you encountered the test smell "Large Setup Methods"? (For an explanation of the smell use 'Hint'.)

- never
- rarely
- neutral
- rather often
- often

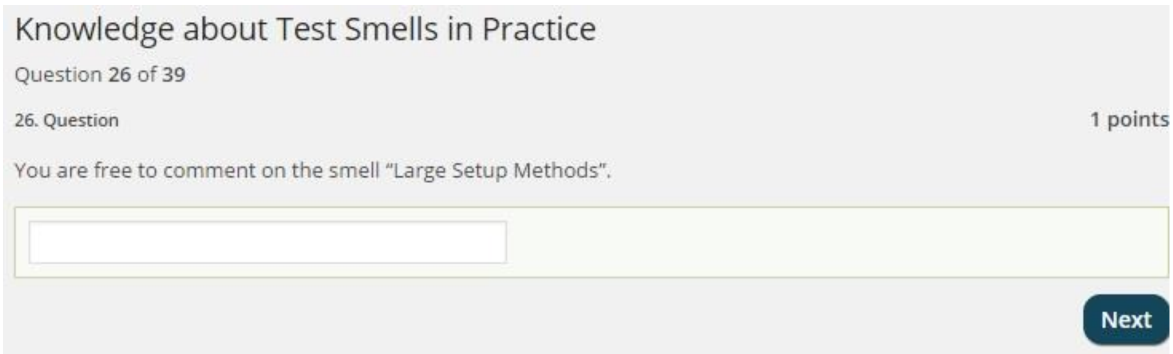
Knowledge about Test Smells in Practice

Question 25 of 39

25. Question — 1 points

How severe is "Large Setup Methods"?

- not severe
- rather not severe
- neutral
- severe
- very severe
- don't know



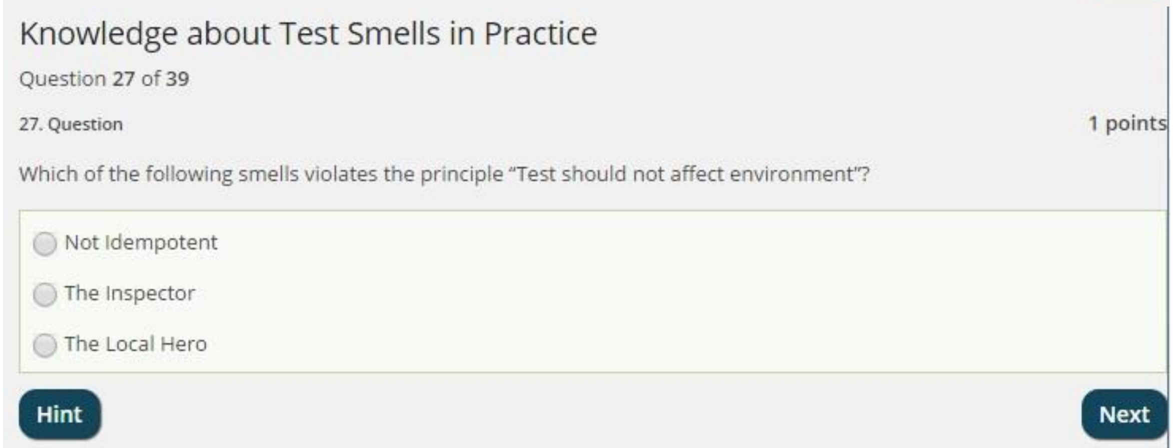
Knowledge about Test Smells in Practice

Question 26 of 39

26. Question

1 points

You are free to comment on the smell “Large Setup Methods”.



Knowledge about Test Smells in Practice

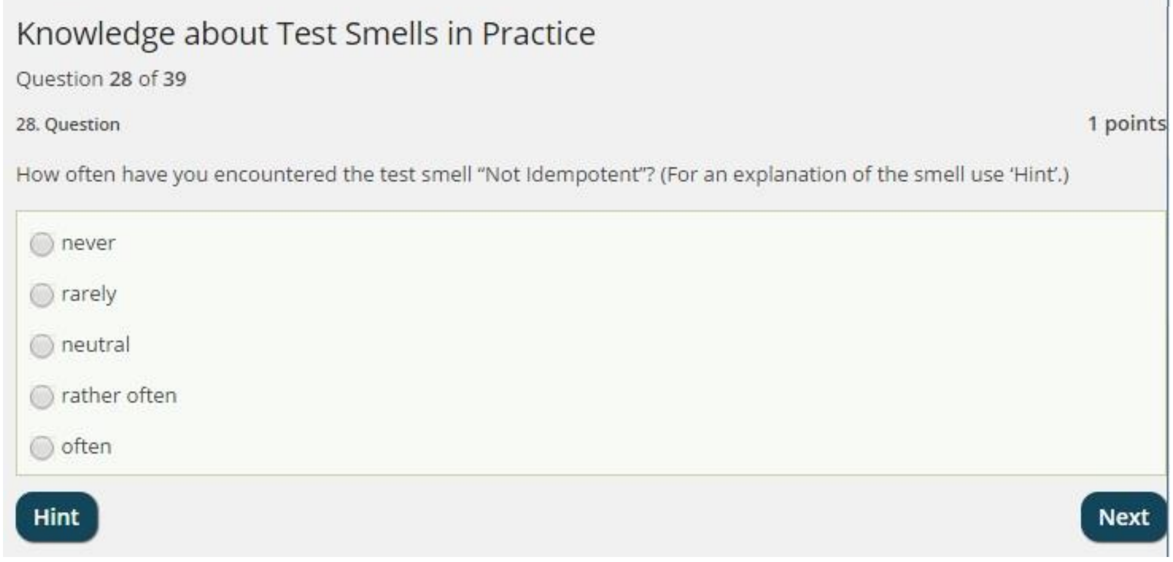
Question 27 of 39

27. Question

1 points

Which of the following smells violates the principle “Test should not affect environment”?

- Not Idempotent
- The Inspector
- The Local Hero



Knowledge about Test Smells in Practice

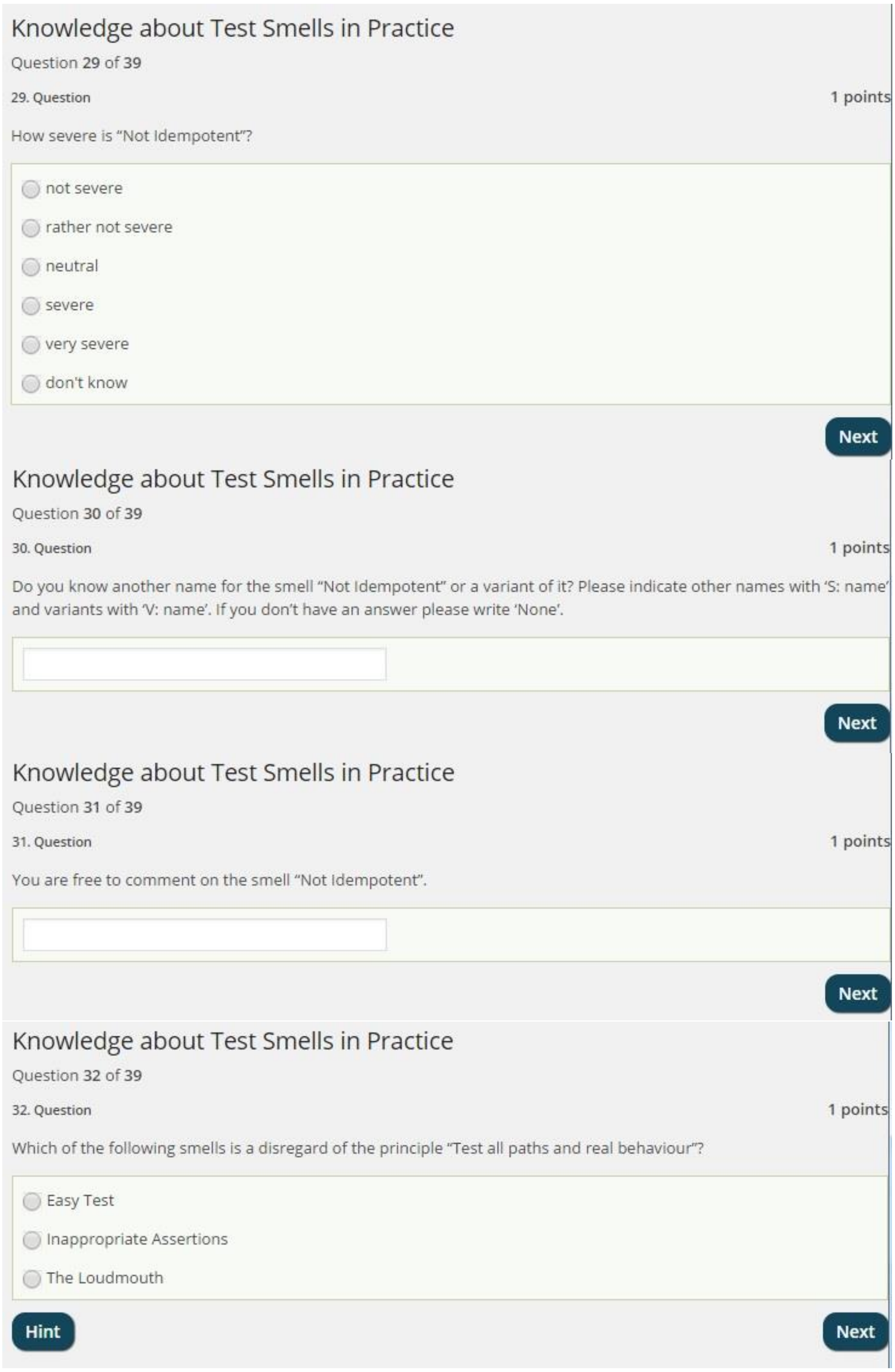
Question 28 of 39

28. Question

1 points

How often have you encountered the test smell “Not Idempotent”? (For an explanation of the smell use ‘Hint’.)

- never
- rarely
- neutral
- rather often
- often



Knowledge about Test Smells in Practice

Question 29 of 39

29. Question — 1 points

How severe is "Not Idempotent"?

○ not severe
○ rather not severe
○ neutral
○ severe
○ very severe
○ don't know

Next

Knowledge about Test Smells in Practice

Question 30 of 39

30. Question — 1 points

Do you know another name for the smell "Not Idempotent" or a variant of it? Please indicate other names with 'S: name' and variants with 'V: name'. If you don't have an answer please write 'None'.

Next

Knowledge about Test Smells in Practice

Question 31 of 39

31. Question — 1 points

You are free to comment on the smell "Not Idempotent".

Next

Knowledge about Test Smells in Practice

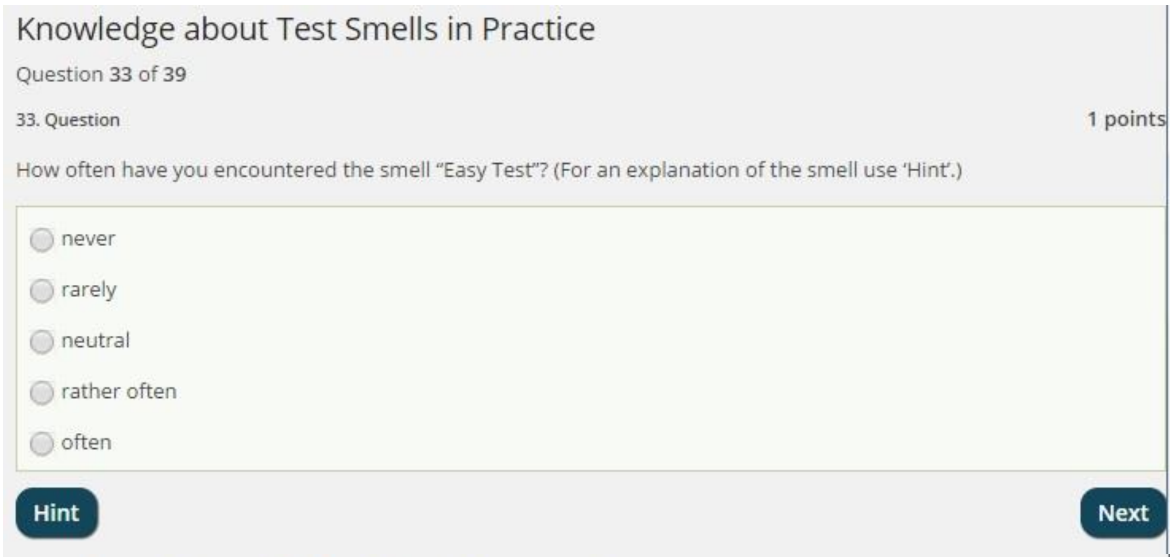
Question 32 of 39

32. Question — 1 points

Which of the following smells is a disregard of the principle "Test all paths and real behaviour"?

○ Easy Test
○ Inappropriate Assertions
○ The Loudmouth

Hint — Next



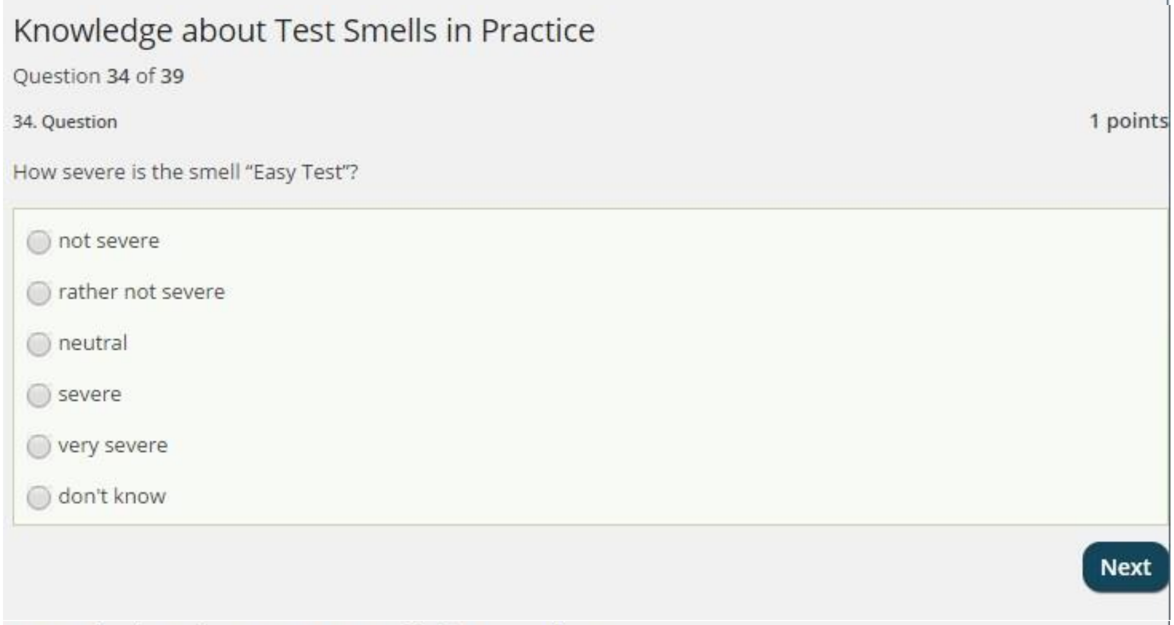
Knowledge about Test Smells in Practice

Question 33 of 39

33. Question — 1 points

How often have you encountered the smell "Easy Test"? (For an explanation of the smell use 'Hint'.)

- never
- rarely
- neutral
- rather often
- often



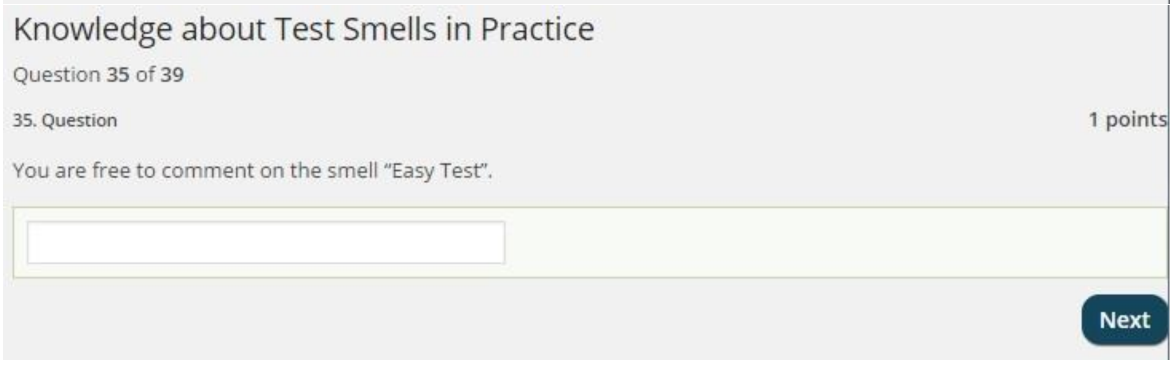
Knowledge about Test Smells in Practice

Question 34 of 39

34. Question — 1 points

How severe is the smell "Easy Test"?

- not severe
- rather not severe
- neutral
- severe
- very severe
- don't know

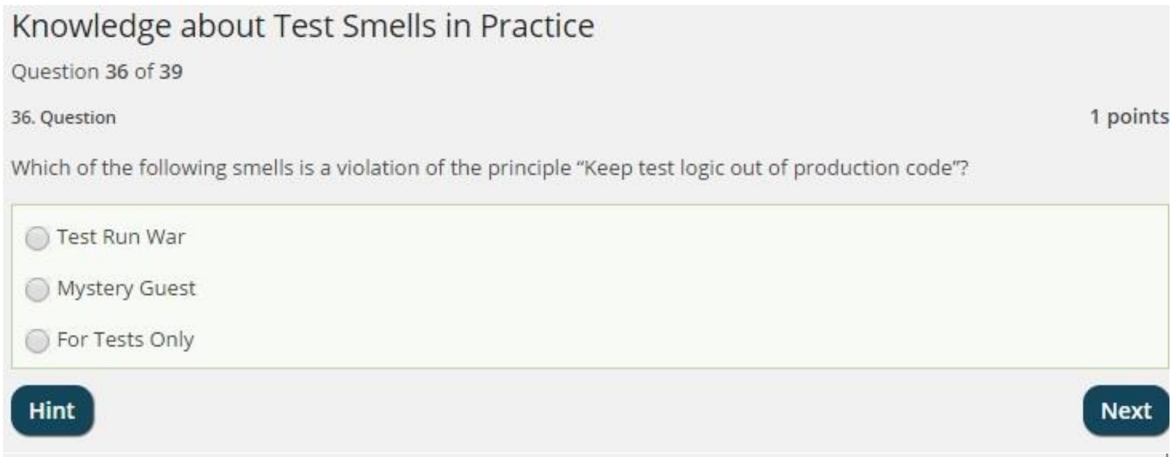


Knowledge about Test Smells in Practice

Question 35 of 39

35. Question — 1 points

You are free to comment on the smell "Easy Test".


Knowledge about Test Smells in Practice

Question 36 of 39

36. Question

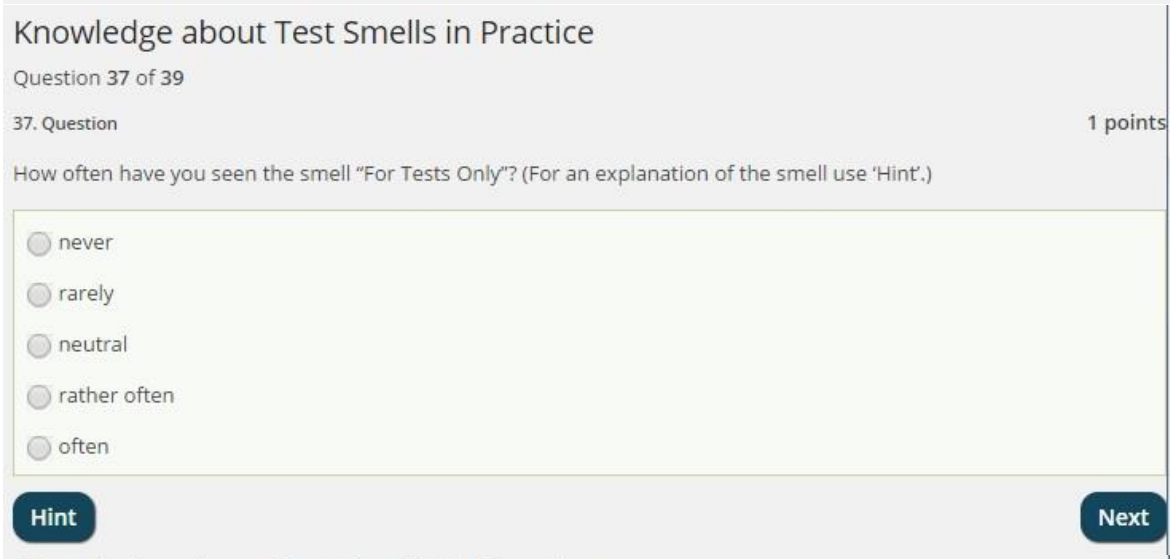
1 points

Which of the following smells is a violation of the principle “Keep test logic out of production code”?

○ Test Run War

○ Mystery Guest

○ For Tests Only


Knowledge about Test Smells in Practice

Question 37 of 39

37. Question

1 points

How often have you seen the smell “For Tests Only”? (For an explanation of the smell use ‘Hint’.)

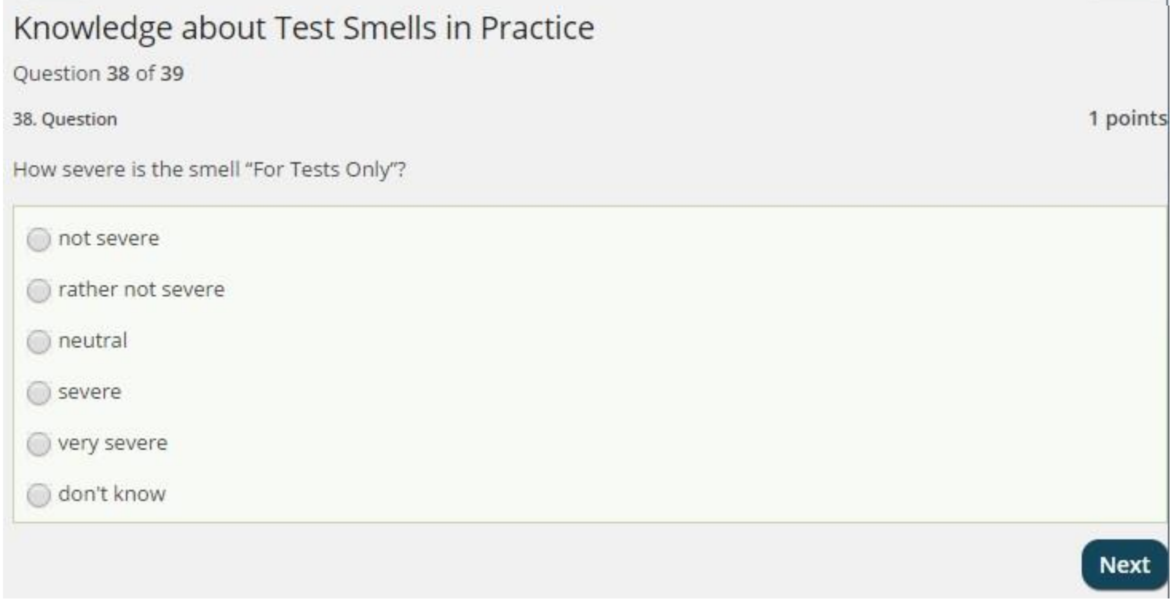
○ never

○ rarely

○ neutral

○ rather often

○ often


Knowledge about Test Smells in Practice

Question 38 of 39

38. Question

1 points

How severe is the smell “For Tests Only”?

○ not severe

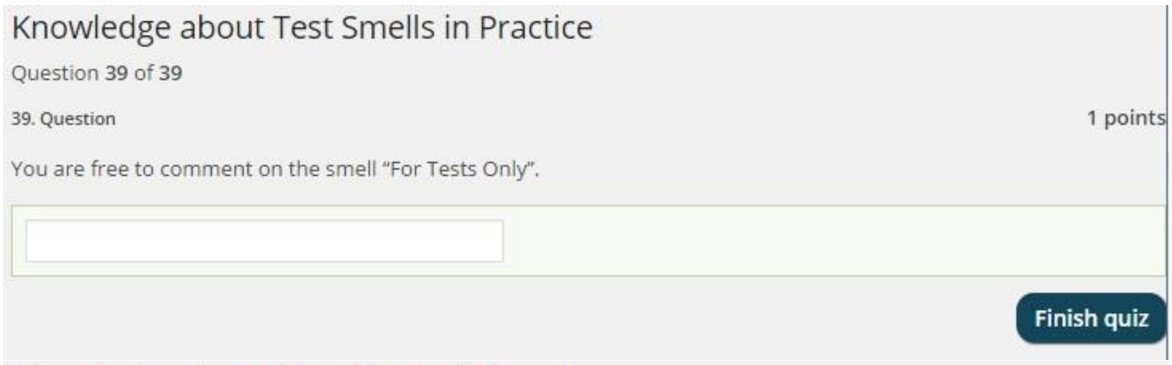
○ rather not severe

○ neutral

○ severe

○ very severe

○ don't know

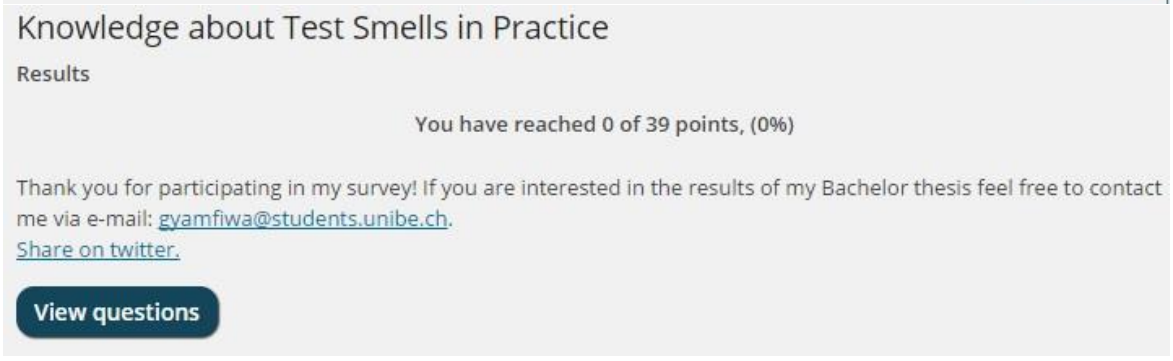

Knowledge about Test Smells in Practice

Question 39 of 39

39. Question

1 points

You are free to comment on the smell "For Tests Only".


Knowledge about Test Smells in Practice

Results

You have reached 0 of 39 points, (0%)

Thank you for participating in my survey! If you are interested in the results of my Bachelor thesis feel free to contact me via e-mail: gyamfiwa@students.unibe.ch.

Share on twitter.