Hunting for Smells in Natural Language Tests

Benedikt Hauptmann
Maximilian Junker, Sebastian Eder
Technische Universität München, Germany

Lars Heinemann
CQSE GmbH,
Germany

Rudolf Vaas
Munich Re Group,
Germany

Peter Braun
Validas AG,
Germany

Abstract—**Tests are central artifacts of software systems and play a crucial role for software quality. In system testing, a lot of test execution is performed manually using tests in natural language. However, those test cases are often poorly written without best practices in mind. This leads to tests which are not maintainable, hard to understand and inefficient to execute.**

For source code and unit tests, so called *code smells* and *test smells* have been established as indicators to identify poorly written code. We apply the idea of smells to natural language tests by defining a set of common *Natural Language Test Smells (NLTS)*. Furthermore, we report on an empirical study analyzing the extent in more than 2800 tests of seven industrial test suites.

Index Terms—**system testing, natural language, test smells**

I. Introduction

In many contexts, system test execution is still done manually: A human tester executes test cases written in natural language by interacting with the system under test. The alternative, test automation, is expensive and does not pay off in all situations. Moreover, test automation is sometimes hardly feasible, for example, if the system under test contains physical components such as in ATMs or embedded systems. Table I shows a fictive example of a manual system test for an ATM.

Unfortunately, manual tests are often written without software engineering best practices in mind (such as abstraction and reuse). For example, previous research found natural language system tests to contain a significant amount of redundancy (cloning) [1] which can considerably increase the costs for maintaining and executing tests.

Source code, as well as unit tests, suffer from similar problems. Time pressure or inexperience makes developers ignore well-known design rules. This results in source and unit test code that is less maintainable and understandable.

As a countermeasure, for both, source code and unit tests, so called *code smells* [2] and *test smells* [3]–[5] have been established as indicators for design flaws. In the last years, several studies have shown the negative effect of smells with respect to maintainability and code comprehension [6]–[9].

This paper applies the concept of smells to tests in natural language, for example, for manual system tests. Adopting existing unit test smells, we define smells for manual tests in natural language - *Natural Language Test Smells (NLTS)*. Furthermore, we define metrics to automatically detect these smells in natural language tests. Being able to find smells automatically facilitates a continuous quality control of test suites which helps to avoid bad quality from early on.

As a first step towards a validation, we apply our smell metrics to seven real-world test suites from industrial business information systems. The results show that every test suite we analyzed exhibits smells. Furthermore, our test smells identify smell hot spots and help to rate the overall quality of a test suite.

Problem: Tests written in natural language often do not obey well-established software engineering principles. This leads to deficits affecting maintenance and comprehension.

Contribution: First, we introduce a set of test smells for natural language tests that can be used to assess test suites with respect to its impact on maintenance and test execution. Second, we introduce ways to detect these smells automatically using static analysis. Third, we report on first industrial experiences analyzing the extent of test smells in natural language system tests in seven industrial test suites.

TABLE I
EXAMPLE OF A MANUAL SYSTEM TEST FOR AN ATM

Test: Wrong PIN Entered Twice		
	Action	**Expected Result**
1.	Put ATM card in card-reader.	ATM asks for the PIN.
2.	Enter a random bad PIN.	ATM responds that the PIN you entered is wrong and that you have only two more attempts left.
3.	Enter the same PIN again.	ATM responds that the PIN you entered is *the same wrong PIN again* and that you have only one attempt left.
	...	...

II. Related Work

To the best of our knowledge this is the first work to study smells in natural language tests. We relate our work to the research areas of *code-based test smells* and *quality assessment of natural language artifacts*.

A. Code-based test smells

The notion of a *smell* in software artifacts originates from Fowler's book on refactoring [2]. He identified a set of code smells which are symptoms of design problems negatively affecting maintainability. Inspired by Fowler's work, van Deursen et al. [3] were the first to introduce the notion of unit test smells. Meszaros et al. [4] classified these smells into two major categories. They distinguish between smells regarding (1) the *code* and (2) the *behavior* of the tests. We build on this work and transfer the notion of smells to tests written in natural language.

B. Quality assessment of natural language artifacts

Previous work by our group has analyzed the extent of the specific quality deficiency of *duplication (cloning)* both in requirements [10] and tests [1]. In this work we take a broader view and consider a variety of smells.

Several authors have proposed approaches to analyze quality aspects of natural language requirements [11]–[13]. We build on this work and use the suggested quality indicators to define smells specifically for tests in natural languages.

III. NATURAL LANGUAGE TEST SMELLS

This section gives an overview on how natural language test suites are structured. Based on this structure, we present the set of smells that we identified for natural language tests.

A. Test Model

A test suite consists of a set of tests which are composed of a sequence of steps. A step consists of an action and its expected result, both written in natural language (see Figure 1). Furthermore, tests may use abstraction mechanisms such as ways to fork tests or reuse test steps (indicated by *TestCall* in Figure 1).
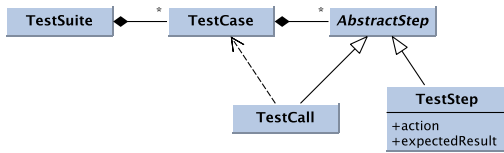

Fig. 1.   Test Model as UML Diagram

B. Test Smells

The list of smells is the result of our experiences in quality evaluation of natural language system tests that we have been conducting in an industrial environment for the last two years. For each smell, we describe which testing activity (*e. g.*, test comprehension, execution or maintenance) is impaired by it.

Smell 1: *Hard Coded Values*.
Tests contain lots of 'magic numbers' or Strings (*e. g.*, test data or names of user interface elements).
⇒*Maintenance:* It is difficult find out where to perform changes if hard coded values have to be changed.

Smell 2: *Long Test Steps*.
A test step is very long.
⇒*Comprehension:* A step's intention is difficult to grasp.

Smell 3: *Conditional Tests*.
Tests are very complex and contain conditional logic which is phrased in natural language.
⇒*Comprehension:* It is hard to understand the test's intention.
⇒*Correctness:* Complex tests are more likely to have bugs.

Smell 4: *Badly Structured Test Suite*.
The structure of the test suite does not follow the structure of the tested functionality.
⇒*Comprehension:* It is difficult to find out which functionality a test is verifying.

⇒*Execution:* It is difficult to select tests for a test plan.
⇒*Maintenance:* If functionality changes, it is difficult to find out which tests have to be adapted.

Smell 5: *Test Clones*.
Tests contain similar parts (*e. g.*, introduced by copy&paste).
⇒*Comprehension:* Test sequences which are similar but not identical are not easy to distinguish. It is not easy to grasp a test's intention.
⇒*Maintenance:* The effort to maintain duplicated parts of tests increases. Furthermore, it is difficult to find out where maintenance has to be performed.

Smell 6: *Ambiguous Tests*.
The test is underspecified and leaves room for interpretation.
⇒*Comprehension:* It is not clear what the idea of a test is.
⇒*Execution:* Multiple test executions of the same tests are not comparable. The test execution becomes indeterministic.

Smell 7: *Inconsistent Wording*.
Domain concepts are not used in a consistent way (*e. g.*, several names are used for the same domain concept).
⇒*Comprehension:* It is difficult to detect similarities of tests.

IV. MEASURING NATURAL LANGUAGE SMELLS

In order to quantify the degree to which a test suite is affected by test smells we define metrics measuring the extent of the smells for every test. Furthermore, we propose thresholds for the metrics based on which we consider a smell as present. We define those thresholds based on our experience in industrial quality assessment of tests over the last years.

Smell 1: *Hard Coded Values*.
We calculate the amount of hard coded values relative to the length of a test (based on the number of words). We consider words as hard coded values if they are in quotation marks or consist of just numbers. A test contains a smell if more than 10% of its words match our definition of hard coded values.

Smell 2: *Long Test Steps*.
We count the number of words of every test step's action and expected result. A test contains a smell if it contains at least one action or expected result with more than 50 words.

Smell 3: *Conditional Tests*.
We count the number of conditional words for every test. We consider a word as conditional if it indicates a case differentiation phrased in natural language[1]. A test contains a smell if it contains at least one of those words.

Smell 4: *Badly Structured Test Suite*.
We remove all stop words[2] from the text. The remaining words are normalized by reducing them to their word stem[3]. On this cleaned up text, we calculate a test's most dominant concepts for every test case using the *term frequency-inverse document frequency (TF-IDF)* metric [15]. Furthermore, we assume that a test suite is hierarchically structured in folders and sub folders. Tests which are testing a similar functionality (and

---

[1]we used the words: *if*, *whether*, *depending*, *when*, *in case*
[2]*e. g.*, *a*, *and*, or *how*
[3]we used the stemming approach introduced by Porter [14]

therefore share dominant concepts) should not be located in different folders. A test contains a smell if there is an overlap of dominant concepts with other tests located in different folders.

Smell 5: *Test Clones.*
Following the definition from [1], we consider a test clone as a substring of a test with at least 30 words appearing at least twice in a test suite. To find clones which differ slightly (*e. g.*, because of inconsistent typo fixes), clones are allowed to have minor variations such that the difference (the gap) accounts for less than 10% of the length of the clone. A test contains a smell if it contains at least one clone.

Smell 6: *Ambiguous Tests.*
Similar to smell 3, we count the number of vague words[4] in every test. A test contains a smell if it contains at least one of those words.

Smell 7: *Inconsistent Wording.*
We calculate the amount of inconsistent synonym usage. First, similar to smell 4, we remove stop words and normalize words to their word stem. Second, we group words having the same meaning using the Java API for WordNet Searching[5] and calculate the most frequently used synonym for every group. Third, going through every test step, we decide for every word if there is a synonym in the test suite which is more often used. A test contains a smell if it contains at least one word which is not the most frequently used word in its synonym group.

V. EXPERIMENTAL INDUSTRIAL APPLICATION

To test the idea of natural language test smells we apply our metrics to seven real-world test suites from industrial projects. With this experiment we have the following two aims: First, we want to validate if our smell metrics are sufficient to find actual smells. Second, we want to quantify the extent of our smells in real-world test suites.

A. Study Objects

Our study objects are system tests in natural language of seven projects from the two companies Munich Re and Cassidian. *(note: clearance of second company still pending)*
The Munich Re Group is one of the largest re-insurance companies in the world. For their insurance business, they develop a variety of individual information systems. The systems of the tests we analyzed provide substantially different functionality, ranging from damage prediction, over pharmaceutical risk management to credit and company structure administration.

Cassidian, an EADS company, is a worldwide leader in global security solutions and systems, providing products and services to civil and military customers around the globe. The tests we analyzed here stem from ground support systems used, for example, for mission planning.

All analyzed systems are in productive use and provide graphical user interfaces such as web front-ends or fat client interfaces. The analyzed tests included regression tests and tests of change requests and are performed frequently. For non-disclosure reasons we named the test suites A to G (see Table II).

TABLE II
STUDY OBJECTS

Test Suite	#Tests	#Words
A	72	27,450
B	1,804	529,122
C	135	34,136
D	605	317,205
E	42	9,990
F	102	144,249
G	117	93,547
total	2,877	1,155,699

B. Tool Support

All analyses of the tests were implemented on top of ConQAT[6], a modular open source software quality assessment toolkit. ConQAT provides the functionality for processing the natural language documents as well as for presenting analysis results as HTML reports. The input for our analysis is a complete test suite given as Excel or Word files. The result is a tree representing the hierarchical organization of the test cases along with the metric values. Furthermore, we visualize the extent of smells as tree maps. This allows a top down analysis of potential *smell hot spots*.

C. Results and Interpretation

For every smell metric, we performed manual inspections to validate the findings. We inspected metric hot spots as well as cold spots and adapted our configuration if necessary. After that, we calculated the number of tests which are affected by smells using our threshold (see Figure 2).

We found every smell in every test suite, however, the extent of the smells differs between the test suites. For example, the number of tests containing conditional or vague words is quite similar between test suites. However, the percentage of tests affected by cloning or bad structure differs strongly.

We tried to verify the meaningfulness of the result by assessing the test suite's quality manually. And indeed we found that test suite A, which performed best regarding the smell *bad structure*, is structured in a very clean way. It is organized in a two level hierarchy where the first level represents use cases and the second level the tests. The organization of the other test suites is considerably less clear.

Moreover, test suite E is the result of a quality improvement initiative and forms a rework of a part of test suite C. In this rework, a clone detector has been used to find similar parts which then have been manually extracted to reusable units. However, the appearance of other smells decreased as well. This leads to the suspicion that cleaning up one smell positively affects other smells as well.

---

[4]we used the words: *similar*, *better*, *similarly*, *worse*, *having in mind*, *take into account*, *take into consideration*, *clear*, *easy*, *strong*, *good*, *bad*, *efficient*, *useful*, *significant*, *adequate*, *fast*, *recent*, *far*, *close*
[5]http://lyle.smu.edu/~tspell/jaws
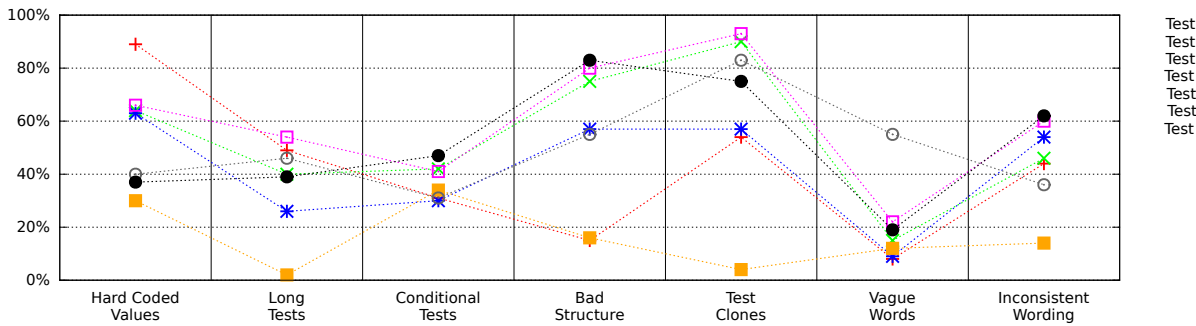
[6]http://www.conqat.org/

Fig. 2.   Percentage of Tests Affected by Smells.

VI. Discussion

The results of our experiments show that all proposed smells are present in all analyzed systems. Moreover, we have seen a clear variance between the test suites regarding the ratio of tests affected by a particular smell. Most importantly, for the system for which we know that they are well structured or that concrete quality improvements regarding maintainability have taken place, a lower extent of smells compared to the other systems was observed.

While our smell analysis showed interesting results for the analyzed systems, it cannot be safely used for an absolute assessment of the quality of a test suite. This is due to the thresholds and parameters used for the smells which we chose from our own (*i. e.*, limited) experience with the maintenance of natural language tests. To increase objectivity of our smell analysis, such thresholds and parameters should be determined by a benchmark, taking into account a large set of different projects, which remains an important direction for future work.

Finally, we concentrated on smells for which we can define measures that can be determined *automatically*. However, quality deficiencies such as *bad naming* can hardly be found by a tool. Manual analysis is still required to cover such quality aspects. However, we consider our automated analysis as complimentary to those inspections and advocate the use of both assessment methods for a comprehensive quality assessment.

VII. Conclusion and Future Work

In this paper we introduced a set of test smells for tests written in natural language. In addition we defined measures to automatically assess the extent of these smells. We applied our metrics to seven industrial test suites demonstrating the applicability of the smells and associated measures. The study revealed that the smells are an indicator for quality issues in manual tests as they coincided with our assessment of the study objects. Furthermore, in one case the results also mirrored the quality improvement introduced through rework of the test suite.

However, our list of smells is not complete in the sense that it captures all quality aspects of natural language tests. In fact, it exclusively focuses on automatically measurable aspects to facilitate *continuous quality monitoring*. Monitoring quality checks continuously helps to uncover quality issues early and therefore helps to prevent bad quality from the beginning.

To focus on other aspects besides maintainability, a comprehensive quality model would be required. However, our list of smells is a first step in this direction and can help to get a list of concrete quality deficiencies for manual test suite. As future work, we plan to define additional smells covering more aspects of natural language test quality. The long term goal is to develop a comprehensive quality model with a connection of high-level quality attributes (*e. g.*, maintainability) to concrete measures of the textual test cases. This approach has been successfully pursued for modeling and assessing software product quality [16].

References

[1] B. Hauptmann, M. Junker, S. Eder, E. Juergens, and R. Vaas, "Can Clone Detection Support Test Comprehension?" in *ICPC'12*, 2012.

[2] M. Fowler and K. Beck, *Refactoring: improving the design of existing code*.   Addison-Wesley, 1999.

[3] A. van Deursen, L. Moonen, A. van den Bergh, and G. Kok, "Refactoring Test Code," in *XP'01*, 2001.

[4] G. Meszaros, S. Smith, and J. Andrea, "The test automation manifesto," in *XP'03*, 2003.

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

[6] M. Abbes, F. Khomh, Y.-G. Gue andhe andneuc, and G. Antoniol, "An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension," in *CSMR'11*, 2011.

[7] F. Khomh, M. Di Penta, and Y.-G. Gueheneuc, "An exploratory study of the impact of code smells on software change-proneness," in *WCRE'09*, 2009.

[8] A. van Deursen and M. Leon, "The video store revisited thoughts on refactoring and testing," in *XP'02*, 2002.

[9] G. Bavota, A. Qusef, R. Oliveto, A. De Lucia, and D. Binkley, "An empirical analysis of the distribution of unit test smells and their impact on software maintenance," in *ICSM'12*, 2012.

[10] E. Juergens, F. Deissenboeck, M. Feilkas, B. Hummel, B. Schaetz, S. Wagner, C. Domann, and J. Streit, "Can clone detection support quality assessments of requirements specifications?" in *ICSE'10*, 2010.

[11] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami, "An automatic quality evaluation for natural language requirements," in *REFSQ'01*, 2001.

[12] I. Hussain, O. Ormandjieva, and L. Kosseim, "Automatic quality assessment of srs text by means of a decision-tree-based text classifier," in *QSIC'07*, 2007.

[13] W. Wilson, L. Rosenberg, and L. Hyatt, "Automated analysis of requirement specifications," in *ICSE'97*, 1997.

[14] M. Porter, "An algorithm for suffix stripping," *Program*, 1980.

[15] K. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, vol. 28, no. 1, 1972.

[16] S. Wagner, K. Lochmann, L. Heinemann, M. Kläs, A. Trendowicz, R. Plösch, A. Seidl, A. Goeb, and J. Streit, "The quamoco product quality modelling and assessment approach," in *ICSE'12*, 2012.