Can Clone Detection Support Test Comprehension?

Benedikt Hauptmann, Maximilian Junker, Sebastian Eder
Technische Universität München
Garching b. München, Germany

Elmar Juergens
CQSE GmbH
München, Germany

Rudolf Vaas
Munich Re Group
München, Germany

***Abstract*—Tests are central artifacts of software systems. Therefore, understanding tests is essential for activities such as maintenance, test automation, and efficient execution. Redundancies in tests may significantly decrease their understandability. Clone detection is a technique to find similar parts in software artifacts. We suggest using this technique to gain a better understanding of tests and to provide guidance for testing activities. We show the capabilities as well as the limits of this approach by conducting a case study analyzing more than 4000 tests of seven industrial software systems.**

***Index Terms*—Program Comprehension, Software Testing, Software Maintenance, Clone Detection**

I. Introduction

Creating software involves more than just creating source code. Testing consumes approximately 50% of the time and more than 50% of the total costs of software development [1]. Since tests are the central artifacts for testing, they play an important role for software development. Automating tests is expensive and does not pay off in all situations. Therefore, a lot of testing is performed manually. This is especially the case for system testing which aims to test a complete, integrated system. System tests are mostly written in natural language and are therefore difficult to analyze automatically.

Understanding tests is difficult: One challenge is the huge amount of tests. The projects we analyzed had between 72 and 1804 system tests with a length from 1 to 6 pages (A4) each.

Furthermore, tests are often badly structured. Most test suites are structured hierarchically, for example, using files and folders on a file system. Using a hierarchical structure focusses on one dimension of the relations between the test cases, consequently neglegting others. This problem is known as *the tyranny of the dominant decomposition* [2]. To understand tests, however, several concerns such as the tested use case, technical aspects, or the type of tests play an important role.

Another challenge is analyzing artifacts in natural language, for example, during maintenance or quality inspections. Whereas a plethora of tools exists to create and maintain formalized artifacts such as source code, almost no tools exist to work with artifacts in natural language in the same way.

Additionally, tests often contain common parts. This is because use cases often have to be checked several times using different input values, executing the system under different conditions or testing different exceptional cases. Also, different tests may have the same precondition or need the same parts of the system's interface. Knowledge of such similarities is useful for a lot of tasks: For example, allocating similar tests to the same tester mostly has positive effects on the execution time and quality. A tester who is familiar with the tests' structure and the necessary parts of the system interface executes tests faster and is aware of possible pitfalls. During maintenance, knowledge about where to perform changes is essential to accomplish changes consistently. Additionally, information about the substance of tests, for example preconditions, is indispensable to optimize test suites and thus helps to reduce the time for test execution. Also, during test automation, awareness of recurring parts in tests can substantially reduce development effort.

Clone detection [3], [4] is a promising method to gain understanding of software artifacts, for example, source code [5] or requirements artifacts [6]. The information of cloned parts provides assistance to understand the structure of artifacts as well as provides useful information during maintenance.

Problem: Tests are central artifacts for software development. Understanding tests is essential for testing activities such as test maintenance, test automation, or test suite optimization. Since system tests are often written in natural language it is harder to avoid redundancies compared to, for example, tests written in programing languages. Such redundancies may impair the understandability and maintainability of tests. Still, almost no work exists about understanding tests in natural language. Especially the role of redundancy has never been addressed. Up to now, there is no evidence on how much redundancy exists in manual tests, nor whether information about redundancy can be used to support testing activities or whether redundancy in real-world tests can be detected using existing tools.

Contributions: In this work, we report on an empirical study in which we evaluate clone detection for gaining understanding of tests. We apply clone detection techniques to more than 4000 system tests of seven industrial business information systems. Our ambition is two-fold: First, we analyze the extent and nature of clones in manual system tests. Second, we exemplarily investigate how far clone detection can support test comprehension by quantifying the benefit for test suite optimization and test automation.

Outline: Section II introduces the fundamental terms used in this paper. Section III gives an example of a test we analyzed in this study. Section IV introduces the study design including our research questions, the study objects, and the data collection procedure. Section V presents the results of our study. Section VI discusses the results. Section VII shows threats to the validity of this study and how we mitigated them. Section VIII gives an overview on related work. Section IX concludes the paper and gives an outline of future work.

II. Terms & Definitions

We consider a **test** as a sequence of commands including input and output data to perform a certain task with a software system as well as to validate the system's behavior. A **manual test** is a test that is performed by a human being and therefore has to be human readable, for example, written in natural language. In such a test, all inputs, the analysis of the output as well as the evaluation are performed manually without any significant tool support. In contrast to manual tests, **automated tests** are performed without manual interaction by a human. A **semi-automated test** is a test which consists of automated as well as manual parts. Although the automated parts are executed automatically, a semi-automated test cannot be executed unattended since the manual parts need manual intervention. Tests are assembled from several **test steps** which describe how to operate the software system, collect data from the software system, or evaluate the gathered results. We use the term **system testing** according to the definition of IEEE Std 610.12-1990 [7] to denote '*testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements*'.

We base the following definitions of clone detection terminology on [6]. A **test clone** is a (consecutive) substring of a test with a certain minimal length, appearing at least twice in a test suite. A **clone group** contains all clones of tests of the same system that have the same content. The **clone coverage** of a test denotes the part of a test that is covered by cloning. It approximates the probability that an arbitrarily chosen part from a test is cloned at least once. The **number of clone groups and clones** denotes how many different test fragments have been copied and how often they occur. The **blow-up** describes how much larger a test suite is compared to a hypothetical version of the suite that contains no clones.

TABLE I
Example of a manual system test

Step Description	Expected Results
Login to the system.	You are logged in.
Go to the master data dialog and create a new customer 'Frank Smith, 03/23/1983'.	Data is created. No error.
Navigate to the dialog 'View -> Reports', search for Frank Smith (use the text field 'query'). Click on button 'calculate status' afterwards.	The System calculates the status. The status is between 1 and 1.5 (text field 'status').
...	...

III. Manual System Tests

The manual system tests we have seen in industry are mainly written down in natural language. They are composed of a list of test steps to be performed. Table I shows a fictive example of a manual system test.

Test steps are written down in a table, each row representing one step. Every step consists of two parts, a step description and the expected result. The *step description* describes what the tester has to do to perform a certain task. It also includes the necessary context information and the input data, if necessary. The *expected result* describes how the tester has to verify the correct reaction of the system. It also includes the necessary output data, if necessary. The steps have to be executed sequentially, starting from the first row. A test has been successfully executed after all test steps have been performed and their expected results could be assured.

During our inspections, we noticed that the abstraction level of the test steps differs strongly between the tests. Whereas some tests use very precise descriptions (e.g., *Enter '125.7' in the text field 'search request'*), others are much more abstract (e.g., *Execute the calculation process and check if the result is plausible*), requiring implicit domain knowledge by the testers. Furthermore, the separation between step descriptions and the expected results was not always the same.

IV. Study Design

This section gives an overview of the study design. We define our research objective using the Goal-Question-Metric approach from [8]. First, we define the goal of our study. Based on the goal, we derive five research questions. After introducing our study objects, we describe how we performed the data collection to answer the research questions. After that, we briefly present details about the implementation and execution of the data collection.

A. Research Goal

The goal of this study is to analyze manual system tests of productive business information systems in order to understand if and how clones affect test maintenance and automation. Furthermore, we investigate whether information about cloning in manual system tests can be used to mitigate harmful effects and to reduce testing effort.

B. Research Questions

The study is structured by five research questions. RQ 1-3 investigate the extent and nature of cloning in manual system tests. Answering these questions enables to better understand the phenomenon of cloning in manual test cases. RQ 4 and 5 investigate if clone detection supports testing activities by fostering test comprehension. We focus on the activities test suite optimization and test automation.

RQ1 *To What Extent Exists Cloning in Manual System Tests?* The goal of this question is to assess if cloning is actually a significant phenomenon in the current practice of manual system testing. The more cloning exists, the greater is the effort to read and understand the tests. However, the

quantity of cloning is not enough to determine the nature and severity of the problems caused by cloning. Hence, RQ2 and RQ3 investigate the characteristics of clones in tests.

RQ2 *What Kind of Information is Cloned in System Tests?* Tests contain several types of information, for example, descriptions of the context or test data. Cloning different types of information causes different problems. To better understand test cloning and estimate its problems, we investigate what type of information is actually cloned.

RQ3 *Are Clones in System Tests Locally Restricted?* A use case is usually tested by several tests, for example, under different conditions, using different parameters or checking different exceptional cases. Tests which exercise the same use case are usually more alike and also stored closer together. If similar tests are stored closely together, it is more likely that during maintenance, changes in cloned parts are performed at all clones consistently. The aim of this research question is to find out if tests share common parts only with closely related tests or also with tests testing different use cases or scenarios.

RQ4 *Can Clone Detection Find Similar Parts of Tests as Basis for Test Suite Optimization?* In order to ensure that manual testing is done as efficiently as possible, it is important that nothing is tested more often than needed. Clones in manual tests could be hints that a test suite could be optimized by merging tests containing clones.

RQ5 *Can Clone Detection Indicate in What Order Tests are Automated Most Efficiently?* The success of test automation depends on the effort it takes to automate tests. As manual tests are often the basis for automated tests, redundancies in manual tests may increase test automation effort, as the same test sequences are automated several times. However, not all test steps are executed equally often. By automating frequent test steps first and leaving rarely performed steps manual, the degree of automation can be raised efficiently. In this research question, we analyze if clone detection can be used to gain the necessary knowledge about the frequency of test steps to support semi-automation of test suites.

Whereas the first three questions can be answered directly using quantitative data we will not provide a definite answer to RQs four and five. Instead, we use our finding to enable a discussion about how manual test suites can be optimized and efficiently automated using clone analysis results.

TABLE II
STUDY OBJECTS

		Length of all Tests	
Project	#Tests	(#lines)	(#words)
A	266	37,027	79,114
B	1,059	165,547	346,135
C	72	12,918	27,450
D	180	67,598	102,991
E	1,804	307,760	529,122
F	135	22,903	34,136
G	605	127,385	317,205
total	4,121	741,138	1,436,153

C. Study Objects

Our study objects are manual system tests of seven projects from the company Munich Re Group. The Munich Re Group is one of the largest re-insurance companies in the world and employs more than 46,000 people in over 50 locations. For their insurance business, they develop a variety of individual supporting software systems.

The systems of the analyzed tests provide substantially different functionality, ranging from damage prediction, over pharmaceutical risk management to credit and company structure administration. They support between 10 and 150 users each.

We chose tests written in different languages (German and English), by different teams (internal employees and external suppliers), and testing different functionalities to increase the generalizability of the study results. We included systems with web front-end, Windows fat client interface, and SAP systems. The analyzed tests included regression tests, tests for planned iterations, and tests of change requests. All these systems are already in their productive environment and all analyzed tests have been performed on the systems. For non-disclosure reasons we named the systems, from which we took the tests, system A to G. Table II shows an overview of the tests. The number of tests differs from less than 100 (system C) to more than 1,800 (system E) per system. In total, we analyzed 741,138 lines of tests containing more than 1.4 million words.

D. Data Collection Procedure

In this section we describe the data we collected in order to answer the research questions. Furthermore, we give details about how we collected the data. Most data was collected by automated analyses on the manual system tests. We performed those analyses using the open source quality assessment toolkit ConQAT[1] [9], which includes, among others, a rich toolkit for clone detection.

RQ1 *To What Extent Exists Cloning in Manual System Tests?* To answer RQ1, we performed a clone detection on all our study objects. In the beginning, we transformed the tests into plain text, which is easier to analyze. Using whitespaces and line breaks, we tokenized the tests into sequences of words. ConQAT has been used to find clones of at least 30 sequential words (in natural language) in these token streams. To find clones which differ slightly (e.g., because of inconsistent typo fixes), clones were allowed to have minor variations such that the difference (the 'gap') accounts for less that 10% of the length of the clone.

We performed additional inspections to find and exclude false positives such as different versions of the same test or stereotypical parts of tests (e.g., description templates). Those inspections included a visualization of the structure of the clones (which exposed e.g. copied and archived tests), detection of identical tests, a manual inspection of clones, and interviews with the project members. False positives were then excluded by configuring filters in ConQAT.

[1] www.conqat.org

Apart from qualitative results about the structure and distribution of clones in the test suites, the main metric we gained here was the *clone coverage*, which is the percentage of the test suite that is subject to cloning. Furthermore, we calculated the *blow-up* of the tests introduced by cloning.

RQ2 *What Kind of Information is Cloned in System Tests?* To answer RQ2, we performed a manual inspection of a sample of manual tests as well as of a sample of the clones. We randomly selected samples of 10 complete tests per project and 20 clones per project. The aim of the inspections was to categorize the information contained in both samples and compare the results. Therefore, we developed a simple categorization scheme. Our scheme is based on the IEEE-829-2008 standard [10] for system test documentation which describes, among others, the type of information that should be included in system tests.

We designed the scheme to be detailed enough to capture the most important types of information as well as simple enough that it can be applied to a large sample of text in acceptable time. The categories of our scheme are:

- **Actions:** Description how the tester has to stimulate the system. Actions are usually located in the *step description*. The granularity ranges from abstract (e.g., *Execute function XY*) to concrete (e.g., *Click on the button ABC*).
- **Checks:** Expectation of the system's behavior which has to be verified by the tester. Checks are usually located in the *expected result*. The granularity ranges from abstract (e.g., *The result is positive*) to exact values (e.g., *Expect message 'Everything has been performed successfully.'*).
- **Input Data:** Input required to execute the test. The granularity ranges from abstract (e.g., *a date max. three month ago*) to exact values (e.g., *'03/23/1983'*).
- **Output Data:** Output expected by the system. The granularity ranges from abstract (e.g., *between 1 and 2*) to exact values (e.g., *'1.4'*).
- **References to other Tests:** References within the same test or between different tests such as prerequisite (e.g., *Test XY has to be executed before*) or jumps to other test steps (e.g., *execute step 3-5 from test XY*).
- **Environmental Needs:** Test environment needed for the execution such as hardware, software, or test data.

Using the scheme, we went through the samples (tests and clones) and counted what types of information could be found. To compare both values, we normalized the assignments to a per word base.

RQ3 *Are Clones in System Tests Locally Restricted?* To answer if cloning only exist between tests of the same use case or also appear between tests of different functionalities, we analyzed the relationships of the clones found by answering research question 1.

The tests we analyzed in this study have been organized in a hierarchical structure similar to the structure of the functionality of the system under test. Starting by main function groups, the tests are further separated by their sub-functionalities. The lowest hierarchy level always represents a specific use case or scenario. Therefore, tests which are stored in the same directory always test the same functionality whereas tests in different directories test different functionalities.

To analyze the clone dependencies, we organize the tests in a graph structure. We create a node for every test and an edge between two nodes if there exists at least one clone between the corresponding tests. In this graph, we analyze all outgoing edges of every node. We classify edges between two tests as *Intra Use Case Dependency* if they are stored in the same directory in the file system. If they are not in the same directory, and therefore not testing the same use case, we classify this edge as *Inter Use Case Dependency*. The ratio between intra and inter use case dependencies forms the result of this analysis.

RQ4 *Can Clone Detection Find Similar Parts of Tests as Basis for Test Suite Optimization?* If two tests contain large common parts, it might be the case that the same tests steps are unnecessarily executed twice. Those tests could be merged to a new test accomplishing the goals of both tests. This could reduce the overall size and the execution time of the test suite.

During manual inspections of the tests, we noticed that cloned parts are often located at the beginning of tests and comprise actions to bring the system into a certain state, for example, by creating a defined set of test data.

In order to obtain an estimation of the minimization potential of clone information, we adapted the clone detection in a way that it finds clones at the beginning of tests which have a significant length (at least 50 words). We inspected such clones, as well as corresponding tests, to assess if the tests could be merged. The number and length of such clones were used as an indicator for the minimization potential. Furthermore, we calculated the relative reduction of overall test size enabled by test suite optimization.

Optimized test suites might have a shorter runtime because common parts of the tests are only executed once. However, test suite optimization is a complex topic that has to be performed carefully considering domain knowledge. This research question just estimates the potential of test suite optimization.

RQ5 *Can Clone Detection Indicate in What Order Tests are Automated Most Efficiently?* Manual test cases are often the input for creating automated tests. If a significant amount of clones exists in manual tests not all test steps are executed equally often. Some steps are performed several times within the same test or among different tests whereas other steps are unique and therefore performed just once. By automating popular test steps first and leaving rarely performed steps manual, the degree of automation could be raised efficiently.

In order to quantify the benefit of semi-automated test execution, we perform a clone detection to determine the execution frequency of test steps. If a test step is covered by a clone, the number of clones in the clone group indicates how often the step will be executed by running the whole test suite. If a test step is not affected by cloning, it is unique and therefore executed just once.

During manual inspections, we noticed that not all clones span over complete test steps. Some clones affect just parts of

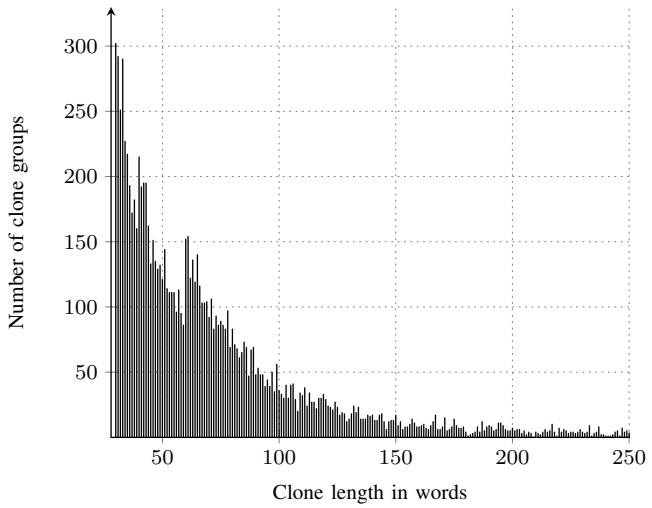

Fig. 1. Distribution of Clone Lengths

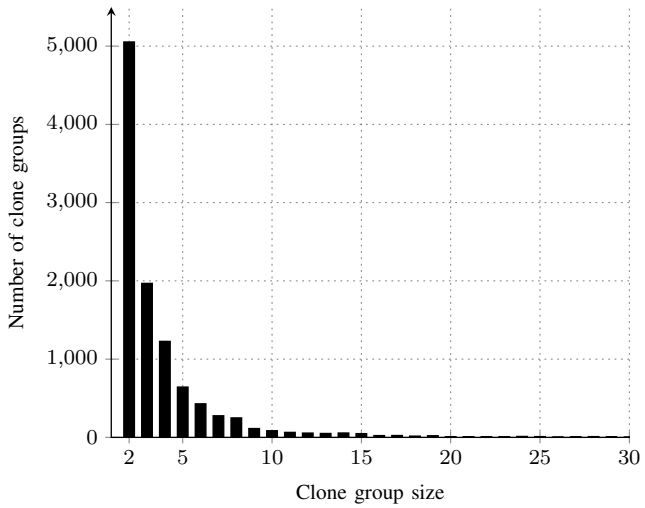

Fig. 2. Distribution of Clone Group Sizes

a test steps. This is usually the case if test steps are created by merging or cutting other, already existing test steps. Therefore, this test steps are not suitable for a direct transition into automated test steps. To solve this problem, we adapted our clone detection to find clones of either whole test steps or at least significant shares of steps. Using the results of the clone detection, test steps could be separated (if necessary) into test steps which are more suitable for a direct automation.

Based on the results of the clone detection, we compute the frequency distribution of the test steps. Furthermore we create an extrapolation in which we put the number of already automated test steps into context with the thereby achieved automation of the whole test suite. Assuming that all analyzed tests (which consist of test steps) are executed equally often, we estimate the automation benefit by first automating more frequent test steps and later rarely executed test steps.

E. Study Implementation and Execution

The case studies were performed on PCs with Windows and Linux operating systems equipped with Intel Core 2 Duo CPUs with 2.4 GHz, and 4 GB of RAM each. The time for the clone detection was between a few seconds up to one hour per project.

V. Results

This section presents the results based on the research questions.

A. RQ1: To What Extent Exists Cloning in Manual System Tests?

RQ1 investigates the extent of cloning in real-world system tests. The results are shown in Table III. The clone coverage varies from 43.3% (project C) to 85.9% (project G). The tests of 6 out of 7 projects have a clone coverage over 50% (every project except C).

The relative blow-up varies from 50% (project C) to 258% (project G). 4 out of 7 projects have a relative blow-up over 100% which means that the test artifacts have more than double the size they could have. The tests containing the most clones are more than two and a half times longer as they would have to be (project G with 258%).

TABLE III
Clone Detection Results

Project	#Clone Groups	#Clones	#Cloned Words	Clone Coverage	Blow-Up
A	674	2,113	51,749	65.4%	114%
B	2,513	7,774	201,575	58.2%	75%
C	175	563	11,890	43.3%	50%
D	804	2,797	57,362	55.7%	78%
E	4,424	24,816	381,120	72.0%	138%
F	309	1,028	20,146	59.0%	82%
G	1,754	11,594	272,628	85.9%	258%

Figure 1 shows the distribution of lengths (in words) of all clones found. Short clones are more frequent than long clones. However, 503 clone groups have a length greater than 250 words. The longest clone group has a size of 1019 words spanning 8 test steps.

Figure 2 shows the distribution of the number of clones per clone group. Small clone groups with two clones are more frequent than clone groups with the size of 3 or higher. However, 104 clone groups containing more than 30 clones were detected.

B. RQ2: What Kind of Information is Cloned in System Tests?

This question investigates which type of information is cloned in real-world system tests. We subjected the sample of 70 test cases (24,880 words) and 140 clones (6,998 words) to our categorization schema. Since the size of both samples differed, we report on the relative frequencies. Table IV shows the assignment rates per 10,000 words. The columns *overall samples* and *clones samples* show the absolute number of assignments per category. The relative difference between the categories of both samples is shown in the column *difference (relative)*.

TABLE IV
CLASSIFICATION OF CLONED INFORMATION

Category	Samples		Difference
	Overall	Clones	(relative)
Actions	445.7	557.3	+25%
Checks	446.9	694.5	+55.4%
Input Values	176.8	137.2	-22.4%
Output Values	121.8	101.5	-16.7%
Environmental Needs	32.2	50	+55.3%
References	28.5	5.7	-80%

Checks occurred the most often (446.9 for all tests and 694.5 for the clones) followed by *actions* (445/557). *Inputs* and *outputs* have been rated as 176.8/137.2 and 121.8/101.5. The categories with the least number of cases are *environmental needs* (32.2/50) and *references* (28.5/5.7).

The highest relative difference between the two samples is in the category *references* with 80% less cases in the clone sample as in the overall sample. The numbers for *checks* and *environmental needs* are slightly more than 50% higher for clones as for the overall sample (+55.4% and +55.3%) whereas *actions* have 25% more cases in the clone sample. *Input* and *output values* have been assigned 22.4% and 16.7% less to cloned as to the overall sample.

C. RQ3: Are Clones in System Tests Locally Restricted?

This question investigates if clones are restricted to tests of the same use case or do also appear between tests of different functionalities. To answer this question, we create a graph representing the dependencies between tests and calculate the ratio between intra and inter use case dependencies.

Figure 3 visualizes the dependencies of project A as *edge bundle view* [11]. The rings on the outside reflect the hierarchical organization of a test suite. The innermost level of rings represents the directories in which the tests are stored whereas every pin on the inside of the ring represents a single test. Dependencies between tests are indicated by lines.

Table V shows the quantitative results. The columns *Intra Use case (absolute)* and *Inter Use case (absolute)* show the absolute values of intra and inter use case clone dependencies. The proportion between intra and inter use case dependencies is shown in the columns *Intra Use case (relative)* and *Inter Use case (relative)*. The relative amount of inter use case dependencies varies between 4.26% (project E) and 90.93% (project C). 2 out of 7 projects have inter use case dependencies around 5% whereas 4 out of the remaining projects are between 80% and 90% in the same category. Just one project had almost as much intra as inter use case dependencies (49.17% to 50.83%). The average of inter use case clones (based on the sum of the absolute values) is 87.75%.

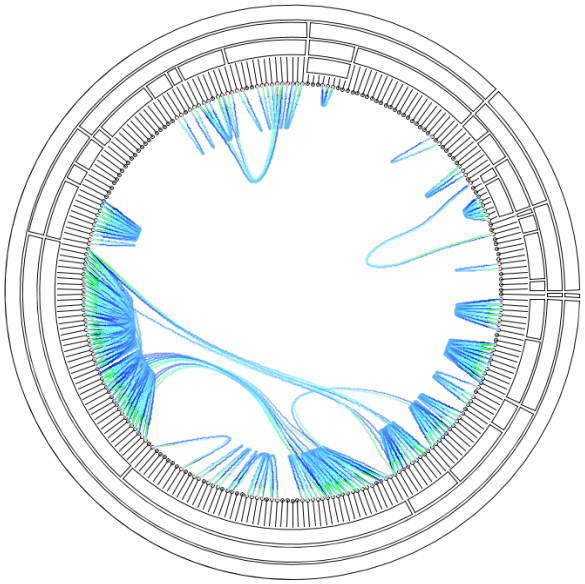

Fig. 3. Dependencies Between Tests of Project A

TABLE V
DEPENDENCIES BETWEEN TESTS BY CLONING

Project	Intra Use Case		Inter Use Case	
	(absolute)	(relative)	(absolute)	(relative)
A	290	94.77%	16	5.23%
B	1,300	19.97%	904	80.03%
C	90	95.74%	4	4.26%
D	792	19.97%	3,173	80.03%
E	4,237	9.07%	42,475	90.93%
F	118	49.17%	122	50.83%
G	248	10.76%	2,056	89.24%
total	6,805	12.25%	48,750	87.75%

D. RQ4: Can Clone Detection Find Similar Parts of Tests as Basis for Test Suite Optimization?

This questions investigates if clone detection supports test suite optimization. Table VI shows the results of the adapted clone detection we performed to find clones as indicator for test consolidation. The column *#clone groups* shows the number of clone groups per project. The maximal and average size of the clone groups are shown in the columns *Clone Group Size (max)* and *Clone Group Size (avg)*. The maximal and average lengths of the clones are shown in the columns *Clone length (max)* and *Clone length (avg)*. The column *Savings (relative)* shows the relative savings of the test word sizes which could be achieved by eliminating the redundancy.

TABLE VI
CLONE DETECTION RESULTS (TEST SUITE OPTIMIZATION)

Project	#Clone Groups	Clone Group Size		Clone Length		Savings (relative)
		(max)	(avg)	(max)	(avg)	
A	25	4	2.24	224	93	2.88%
B	1154	14	2.98	2943	117.05	24.63%
C	3	3	2.33	146	45	0.99%
D	10	13	4.3	127	44.7	1.19%
E	377	49	4.56	1391	73.35	10.78%
F	4	2	2	86	36.886	0.82%
G	191	13	2.96	6303	218.6	13.29%

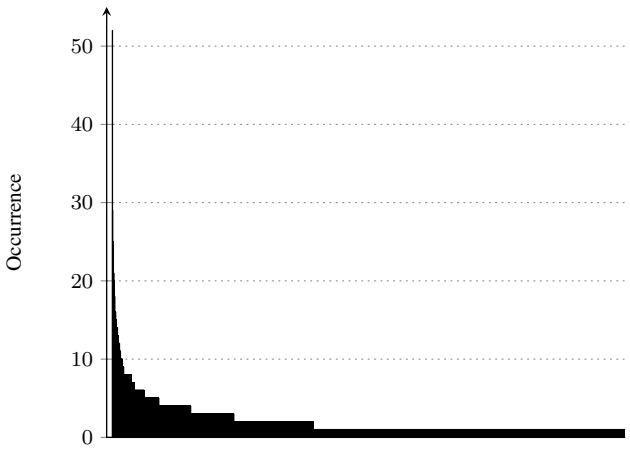

Fig. 4. Distribution of the Test Step Frequency for Project A

The number of clone groups varies from 3 (project C) to 1154 (project B). The maximal cardinality of the clone groups varies between 2 and 49 clones (project F, E) whereas the average cardinality is between 2.24 and 4.56 (project A, E). Considering the clone length, the possible (relative) size reduction indicated by these results varies between 0.82% (project F) and 24.63% (project B). The relative savings of 3 projects are around 10% and 25% whereas the remaining 4 projects' relative savings were around 3% and lower.

E. RQ5: Can Clone Detection Indicate in What Order Tests are Automated Most Efficiently?

This question investigates if clone detection supports efficient test automation. To answer this question we performed a tailored clone detection which finds clones of whole test steps or significant parts of test steps. The size of the clone group therefore indicates how often the according test step would be performed by executing the tests.

Figure 4 exemplarily shows the distribution of the clone group size of project A. We found 2761 potential test steps of which 39.3% are occurring twice or more often. The most frequent test step occurs 52 times.

Figure 5 exemplarily shows the extrapolation bringing together the relative progress of automation of test steps (x-axis) with the thereby achieved automation of the whole test suite (y-axis) for project A. By automating frequently executed test steps first, 25% (the lower quantile) of all test executions could be performed automatically by automating 5.3% of the test steps. 50% of the test executions (the median) could be performed automatically after automating 18.3% of all steps. To automatically perform 75% of all necessary step executions (the upper quantile), 46% of all test steps have to be automated. Since 60.7% of all test steps are unique and therefore have to be executed just once, the growth of automatically executed test steps starts to growth linearly after 39.3% of all test steps have been automated.

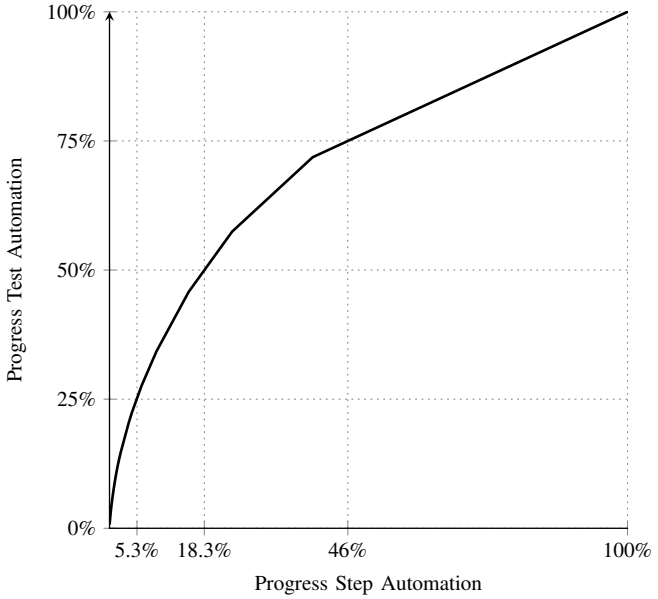

Fig. 5. Extrapolation of the Automation Efficiency for Project A

TABLE VII
EXTRAPOLATION OF AUTOMATION EFFICIENCY

Project	Percentiles (25%)	Percentiles (50%)	Percentiles (75%)	Start of lin. growth (at steps automated)
A	5.3%	18.3%	46%	71.9% (39.3%)
B	3%	13.5%	39.7%	78.5% (43.3%)
C	4.3%	15%	45.2%	69.8% (33.8%)
D	1.4%	9.7%	42.6%	68.2% (72%)
E	0.6%	5.9%	26%	48.5% (43%)
F	4.1%	15.4%	44.5%	71.5% (36.6%)
G	2.5%	10.8%	31.3.%	87.9% (53.4%)

Table VII summarizes the same metrics for all study objects. The columns *Percentiles (25%)*, *Percentiles (50%)* and *Percentiles (75%)* show the relative amount of test step executions automated by automating 25%, 50% or 75% of the test steps. The column *Start of linear growth* shows at which relative test step automation degree *(at steps automated)* the additional value of test step automation will start to grow linearly.

The 25% percentile differs from 0.6% to 5.3% of the test step automations. The 50% percentile (the median) ranges from 5.9% to 18.3%. The 75% percentile of test step execution automation is reached between 26% and 46% (project E and A each time). The start of linear growth ranges from 48.5% automated test step executions (at 43% steps automated of project E) to 87.9% (at 53.4% steps automated of project G).

VI. DISCUSSION

This section discusses the results based on the research questions.

A. RQ1: To What Extent Exists Cloning in Manual System Tests?

The results show that cloning is common in real-world system tests. Although the amount of cloning differed, clones

appeared in all analyzed projects in a significant amount (clone coverage >40%). Furthermore, we found many clone groups with a size greater 5 (see Figure 2). This means, when maintaining one of those parts of tests, at least four other parts need to be maintained, too. Finding out where to perform changes can be challenging if clones are spread across multiple tests (see RQ3). Therefore, depending on the locality of the cloned regions, clones can be a threat for test comprehension.

In the interviews we conducted to adjust the clone detection results, we noticed that the tools with which the tests have been created and maintained have a substantial influence on the amount of cloning. In several projects, the tools did not provide the necessary abstraction capabilities to avoid cloning or the test creators were not trained to use these mechanisms properly.

B. RQ2: What Kind of Information is Cloned in System Tests?

Action and check commands appear significantly more often in clones than in tests in general. In contrast, inputs and outputs appear less often in clones. The reason for this is that test data is frequently covered by the gaps of clones. Hence, clones of one clone group often differed in the input and output data whereas the sequences of commands were mostly alike. This indicates a potential maintenance problem, as test data is not explicitly managed. This makes it difficult to assess what a test is actually testing.

Our results show further differences in the cloning of different types of information. For example, checks appear noticeably more often in clones than actions. However, we could not find an explanation for that. Furthermore, environmental needs appear frequently in clones. This indicates a similar problem as with the test data. For a consistent handling of such needs across tests, this information should be managed seperately from the test step description. The fact that references are cloned rarely is not surprising, as common information has already been externalized in such cases.

C. RQ3: Are Clones in System Tests Locally Restricted?

In 5 out of 7 projects we found a significant amount (>50%) of inter use case clone dependencies. This indicates that in more than half of the cases of redundancy in tests (in 4 projects more than 80%), there exists at least one other test containing similar parts but testing a different use case or scenario.

During the interviews with the domain experts, we figured out that many inter use case dependencies can be traced back to dependencies in the requirements of the corresponding use cases. If there was a dependency relationship between two use cases (e.g., use case A must be executed before use case B), there also were clones between the corresponding tests. Therefore, the locality of clones in system tests depends on how the tests are structured and how many dependencies between the tested use cases exist.

Furthermore, clones which are spread across many tests often addressed the creation of test data. The cloned parts guide the tester to create a certain type and amount of data which is used in the subsequent test steps. This implies that, if tests require the same type of test data, it is very likely that they have cloned test steps for the data creation.

The same is true for test steps dealing with the system interface. Independent of the use case under test, many clones that affected more than one use case contained system interface related information, such as navigation within the user interface. Therefore, if several tests require the usage of the same parts of the system interface, they are likely to have clones.

Inter use case clones hinders the comprehensibility of test suites because such clone dependencies between tests cannot be seen easily. Making those relations explicit improves the comprehensibility of test suites.

D. RQ4: Can Clone Detection Find Similar Parts of Tests as Basis for Test Suite Optimization?

In at least three projects, we found significant portions (>10%) of cloning at the beginning of test cases. Such cases are indicators that the test suite might be optimized by consolidating test cases. However, not in every case such an optimization is possible or reasonable. There might be good reasons why specific tests are kept separate and redundancy and inefficiency are tolerated. For example, those tests might be part of different test sets which are executed at different times. On the other hand, there might be several other opportunities for test suite optimization that are not captured by our analysis (e.g. because the common part is not at the beginning).

In the remaining four projects the amount of cloning at the beginning of tests was rather low (<3%). The test suites of those projects were also smaller than the other ones. A possible explanation for the discrepancy in the results is that those tests are more diverse and do not include as many tests per use case as the other projects. Thus they do not have as many similar starting sequences.

Nevertheless, at least for the three projects (B, E, G), the clone detection can be helpful in identifying many possible candidates for an optimization. The cloning information is especially helpful in the context of large test suites where information about redundancies and optimization potentials is otherwise hard to retrieve.

E. RQ5: Can Clone Detection Indicate in What Order Tests are Automated Most Efficiently?

The execution frequency of test steps varies considerably among different steps. In project A, executing the whole test suite, some steps are executed only once, others over 50 times. If we assume that all tests are executed equally often and all steps are equally costly to automate, the overall degree of automation can be quickly increased by prioritizing frequently executed steps over infrequently executed steps. Across the projects, the benefit is different. Still, to reach 50% of automation no more than 18.3% of the steps need to be automated (only 5.9% in project E). To reach 75% automation in project E, 26% of the steps need to be automated. However, to benefit from such an approach, a testing framework which allows implementing semi-automated tests is necessary.

Since our study objects are written down in natural language, not all semantically identical test steps might be found using syntactic clone detection. Thus, the values have to be regarded as a lower bound.

Nevertheless, the results indicate that test automation can benefit significantly from clone detection at least in the presence of a semi-automation approach through the ordering of the steps to be automated.

VII. Threats to Validity

In this section, we discuss threats to the internal and external validity of the study and how we mitigated them.

A. Internal Validity

The results of the clone detection (RQ1) can be biased by individual experiences and preferences of the researcher that tailored the clone detection. We mitigated this risk by inspecting random samples of the results by at least two researchers.

The categorization in RQ2 has not been performed on all tests but just on samples of the tests and clones. Selecting samples can potentially introduce inaccuracy. However, we selected the samples randomly, which is a commonly accepted way to address this issue.

Since the categorization in RQ2 has been performed manually, it is subjective to some degree. We addressed this risk by performing the categorization by three researchers which worked in close cooperation. Borderline-cases have been discussed between all researchers to compensate deviant interpretations.

To validate the results of RQ4 and RQ5, random samples of the findings have been manually inspected by researchers. However, the researchers have not been particular domain experts nor have they been users of the systems under tests. To substantiate the results, the findings have to be inspected by dedicated domain experts. Therefore, we consider the results of RQ4 and RQ5 not as incontrovertible facts but as basis for domain experts to target their work.

In RQ5, we calculated the automation efficiency based on the execution frequency of test steps. However, the execution frequency is just one factor to determine the automation efficiency. There are additional factors that have to be considered, such as the execution time of a test step or its individual automation costs. Therefore, we consider the results of RQ5 as a first step towards a comprehensive cost model for test automation.

B. External Validity

All tests we analyzed have been from the same company testing applications of the same application domain. It is possible that tests look different in other application domains or other companies because they use different tools and processes to create and maintain their tests. To make our results more generalizable, the study has to be repeated using tests of different application domains created by different organizations.

VIII. Related Work

Maintenance of tests: Maintaining tests during the software maintenance process is considered important [12]. In [13], [14] 11 so called *test smells* are enumerated, denoting poorly designed tests. Much effort has been spent in detecting test smells in xUnit tests [15] or TTCN-3 tests [16], [17]. However, none of these concepts have been transferred to system tests or tests in natural language.

Test suite visualization: There are studies aiming to support test comprehension using visualizations [18], [19]. In these approaches, static and dynamic code analysis is used to reconstruct the structure and composition of test suites. However, most of these techniques are based on unit tests and therefore not applicable to tests in natural language.

Test suite optimization: Many techniques evolved to reduce the size of test suites regarding the number of tests, whilst the test suite's coverage of code, requirements, or fault detection is kept high [20]–[23]. Our goal, however, is not to present a specific optimization technique but rather to explore if information gained by clone detection can be used as input for a test suite optimization, regardless of the used technique.

Test automation: During the last decades, much effort has been put into the automation of tests. We do not focus on automatically generating test data, like [24]–[26], nor on creating fully automated tests, like [27]–[30]. Based on the results of this study, we propose to use semi-automated tests to increase the execution efficiency of tests. Based on our experience, this method faces better acceptance in environments where many manual tests already exist, because not everything has to be automated in one single leap, but can be automated in an incremental manner.

Clone detection: There are several approaches to detect clones in code [3]–[5]. There are only few studies on cloning in other artifacts than code, for example in requirement documents [6]. The presented analyses base on the clone detection techniques proposed in the mentioned sources. Moreover, some effort has been put into commonalities detection: Clustering algorithms like [31] search for documents with common topics and plagiarism detection algorithms, like [32], search for commonalities between a given document and a set of other documents, while we are also considering commonalities within one document.

IX. Conclusions and Future Work

Tests are the central artifact for testing and therefore play a key role for software development. They often contain recurring parts. Knowledge about redundancy in tests can be beneficial during maintenance, optimization or automation. However, this knowledge does often not exist explicitly. This is especially the case for tests represented in natural language such as manual system tests.

We have conducted an industrial case study analyzing more than 4000 manual system tests to (1) reveal to what extent tests from industry are affected by cloning and (2) if clone detection can be utilized to support tasks. From this study, we draw the following conclusions:

Clones in manual system tests do exist in a significant amount. The tests of all analyzed projects have a clone coverage between 43.3% and 85.9%.

Cloning does not affect all type of information in tests the same way. Test commands occur more in clones as they do in tests in general. Clones often differed just in test data which was covered by the gaps of the clones.

The extent of clones in system tests is not limited to single use cases or scenarios. Tests significantly share parts with other tests testing different use cases. 5 out of 7 projects had far more (>50%) dependencies to tests of other use cases as to tests testing the same use case.

Using clone detection to support test suite optimization does not work for all projects. In 3 out of 7 projects, candidates for test consolidation could be found. In those cases, the results are promising. They indicate a potential reduction of the overall test size between 10.78% and 24.63%.

Clone detection can provide helpful information to target test automation effort. Using information about clone cardinalities, 50% of all test step executions could be performed automatically by automation just 5.9% - 18.3% of all test steps.

However, some important research questions are still unanswered. We are planning on investigating them in future work:

Tools and methods to avoid redundancy in tests: In this study, we have analyzed the dimension of cloning in manual system tests. Furthermore, we have shown tasks in which uncovering reused parts of tests can be used gainfully. However, we didn't investigate how to reduce clones in natural language tests. Creating a notation and tool support to define tests which reduce cloning and make reuse explicit would be helpful to conserve the information of reuse for activities such as test suite optimization or automation.

Comprehensive cost model for test automation: By answering RQ5 we made a first step towards a cost model for test automation. We estimated the value of test steps by determining their execution frequencies. However, the execution frequency is just one attribute among others. To create a comprehensive cost model for test automation, the effect of other attributes has to be investigated and considered in the assessment of test steps, too.

ACKNOWLEDGMENTS

The authors would like to thank Daniel Veronika Bauer, Méndez Fernández, Lars Heinemann, and Jonas Eckhardt for their helpful comments.

REFERENCES

[1] G. J. Myers and C. Sandler, *The Art of Software Testing*. John Wiley & Sons, 2004.

[2] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton, Jr., "N degrees of separation: multi-dimensional separation of concerns," in *ICSE*, 1999.

[3] C. K. Roy and J. R. Cordy, "A survey on software clone detection research," *SCHOOL OF COMPUTING TR 2007-541, QUEENS UNIVERSITY*, 2007.

[4] R. Koschke, "Survey of research on software clones," in *Duplication, Redundancy, and Similarity in Software*, ser. Dagstuhl Seminar Proceedings, R. Koschke, E. Merlo, and A. Walenstein, Eds., 2007.

[5] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do code clones matter?" in *ICSE*, 2009.

[6] 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*, 2010.

[7] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," New York, USA, 1990.

[8] V. Basili, G. Caldiera, and H. Rombach, "The Goal Question Metric Approach," *Encyclopedia of Software Engineering*, vol. 1, 1994.

[9] E. Juergens, F. Deissenboeck, and B. Hummel, "Clonedetective - a workbench for clone detection research," in *ICSE*, 2009.

[10] IEEE, "IEEE Standard for Software and System Test Documentation," 2008.

[11] D. Holten, "Hierarchical edge bundles: visualization of adjacency relations in hierarchical data." *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, 2006.

[12] R. D. Craig and S. P. Jaskiel, *Systematic Software Testing*. Artech House, Inc., 2002.

[13] A. v. Deursen, L. Moonen, A. v. d. Bergh, and G. Kok, "Refactoring test code," in *XP*, 2001.

[14] G. Meszaros, *xUnit test patterns: Refactoring test code*. Addison-Wesley, 2007.

[15] 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," *Software Engineering, IEEE Transactions on*, vol. 33, no. 12, 2007.

[16] H. Neukirchen and M. Bisanz, "Utilising code smells to detect quality problems in ttcn-3 test suites," in *Testing of Software and Communicating Systems*, ser. LNCS, vol. 4581.

[17] G. Din, D. Vega, and I. Schieferdecker, "Automated maintainability of ttcn-3 test suites based on guideline checking," in *Software Technologies for Embedded and Ubiquitous Systems*, ser. LNCS, vol. 5287.

[18] B. Van Rompaey and S. Demeyer, "Exploring the composition of unit test suites," in *ASE Workshops*, 2008.

[19] M. Breugelmans and B. Van Rompaey, "Testq: Exploring structural and maintenance characteristics of unit test suites." in *WASDeTT-1*, 2008.

[20] T. Chen and M. Lau, "A new heuristic for test suite reduction," *Information and Software Technology*, vol. 40, no. 5–6, 1998.

[21] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," *ACM Transactions on Software Engineering and Methodology*, vol. 2, 1993.

[22] J. Jones and M. Harrold, "Test-suite reduction and prioritization for modified condition/decision coverage," *IEEE Transactions on Software Engineering*, vol. 29, no. 3, 2003.

[23] Y. Yu, J. A. Jones, and M. J. Harrold, "An empirical study of the effects of test-suite reduction on fault localization," in *ICSE*, 2008.

[24] B. Korel, "Automated software test data generation," *Software Engineering, IEEE Transactions on*, vol. 16, no. 8, 1990.

[25] R. DeMilli and A. Offutt, "Constraint-based automatic test data generation," *Software Engineering, IEEE Transactions on*, vol. 17, no. 9, 1991.

[26] J. Edvardsson, "A survey on automatic test data generation," in *Proceedings of the Second Conference on Computer Science and Engineering in Linköping*, 1999.

[27] A. Belinfante, J. Feenstra, R. de Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, and L. Heerink, "Formal test automation: A simple experiment," 1999.

[28] Z. Xiaochun, Z. Bo, L. Juefeng, and G. Qiu, "A test automation solution on gui functional test," in *INDIN*, 2008.

[29] R. V. Binder, "Testing object-oriented software: a survey," *Software Testing, Verification and Reliability*, vol. 6, 1996.

[30] G. Meszaros, S. Smith, and J. Andrea, "The test automation manifesto," in *Extreme Programming and Agile Methods*, F. Maurer and D. Wells, Eds. Springer Berlin / Heidelberg, 2003.

[31] J.-R. Wen, J.-Y. Nie, and H.-J. Zhang, "Clustering user queries of a search engine," in *WWW*, 2001.

[32] F. Culwin and T. Lancaster, "A review of electronic services for plagiarism detection in student submissions," in *LTSN*, 2000.