Computer Science and Information Systems 15(2):449–472 https://doi.org/10.2298/CSIS170701006A

Conceptual Approach for Reuse of Test Automation Artifacts on Various Architectural Levels

Dani Almog[1], Hadas Schwartz Chassidim[1], Yaron Tsubery[2], Miroslav Bures[3], and Shlomo Mark[1]

[1] Department of Software Engineering, SCE- Sami Shamoon College of engineering
Jabotinski 84 Ashdod, Israel
almog.dani@gmail.com
hadasch@ac.sce.ac.il
marks@sce.ac.il
[2] R&D Operations Enghouse Interactive
Nehar Prat 9, Giva'at-Ze'ev, Israel
yaron.tsubery@gmail.com
[3] Department of Computer Science, Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo Namesti 13, 121 35 Praha 2, Czech Republic
buresm3@fel.cvut.cz

Abstract. When creating a test automation infrastructure, one of the main considerations for the buildup process is its efficiency. A main cause and method for improvement might come from reuse of test automation artifacts. Following that, one may ask "To what extent can the test automation artifacts be re-used?". In this paper we present a model and test automation architecture for achieving such a goal. Repository Driven Test Automation (RDTA) is a conceptual approach for the buildup process of test automation infrastructure that employs reuse of testing artifacts. This paper discusses aspects of reuse of software test automation artifacts on various levels. Then, practical implications and adjustments arising from the implementation of this new paradigm are discussed. The proposed concept is documented by a case study in an international innovative computer hardware manufacturer, one of leaders in the market. The documented results are significant and confirm the validity of the concept.

Keywords: software testing, test automation, software reuse, repository driven test automation.

1. Introduction

A brief and short version of this paper was originally presented at the Valid 2015 Conference [1]. Testing is regularly reported as a significantly expensive task in software development projects [2]. A large part of software testing costs is spent by various regression testing activities. This is because during the development of the software, we must ensure that none of the newly introduced changes in the system under test (SUT) has induced new defects. Generally, regression testing is an expensive activity that can account for a large proportion of the software maintenance budget, as well [3]. Software engineers usually add tests into existing test suites as software evolves, so these test suites grow in size during the project development, testing, and maintenance phases. This increases the costs of revalidation of the SUT and also the costs of test execution, including test set maintenance costs.

Techniques to reduce the regression tests costs by prioritization and reduction of the test set have been proposed [3,4]. However, employing these techniques can be expensive; therefore, overall regression testing costs are not significantly reduced (and often, are significantly increased). In this context, reusability of test automation artifacts can be considered as an alternative or as a complementary technique.

Addressing test automation, the concept of TAF (Test Automation Frameworks) is advised to ease the efforts of implementing test automation in new projects by achieving the goals of Reusability of test code, Maximum coverage, Recovery scenario, Low cost maintenance, Minimal manual intervention, and Easy Reporting. Today there are several popular tools that offer the TAF approach [5] such as HP QTP, HP ALM, HP UTP, Selenium, Robot Framework, RedwoodHQ, and others. RDTA approach is supposed to compliment tools leading TAFs with their different approaches. This paper will not address specifically an existing commercial infrastructure. But suggest principles and ideas, focusing on reuse of artifacts on many levels. This paper compliment and expand the TAS - Test Automation Solution - the realization/implementation of a gTAA - Generic Test Automation Architecture (providing a blueprint for test automation solutions), including test harnesses and artifacts such as test libraries) approach [6].

A survey of practitioners [4,8] shows that the main benefits of test automation are: reusability, repeatability, and effort saved in test execution. Automation can be applied to parts of the testing process by entrusting repetitive tasks to a test automation system. The main motivation of repository driven test automation (RDTA) is to reduce overall test automation implementation and maintenance costs by strong employment of the test artifact reusability concept [7]. Today, various commercial and open source tools are used for test automation. Many of these tools or frameworks are highly specialized solutions for specific aspects of testing. They are focused on different technologies or based on different test paradigms. There is a large variety of specialized test tools for test case generation, test management, test execution, and so forth. Nevertheless, except for the provision of technical interfaces between single tools, there is limited support for combining the numerous specialized tools into an integrated solution. Another consideration arising from the multiplicity of available (and actually used) test automation tools is that of challenged traceability. between-test requirements, relevant parts or functions of the SUT, actual tests on different technical levels, and reported defects [7]. It becomes much more difficult to trace all the relevant artifacts when you use different tools to store each element, each with its own repositories in its specific format.

The paper is organized as follows. In Section 2 we discuss the concept of possible reuse of test automation artifacts. In Section 3 we present the statement of the problem. In Section 4 we introduce the RDTA concept and discuss storage of test automation artifacts. In Section 5 we discuss conceptual insights into the implications of RDTA in today's modern software development arena (e.g., Unit test, Agile, integration, Service-Oriented Architecture (SOA)). In Section 6 we present the implementation of an RDTA case study. In Section 7 we discuss the results of our research, and in Section 8 we offer our conclusions.

2. Background and Related Work: Reuse of Artifacts

Analyzing our day-to-day testing activities, we may ask: how much of every action, operation, thinking, doing – is actually uniquely new. When attempting to explain the nature of the reusability concept, we may be challenged by the argument that this has all been done before and, therefore, that there is nothing new to contribute in this field. These notions are almost right–most new contributions stem from context and interpretation. For example, when designing a new test case for a specific application, memory and past experience are utilized to rearrange old knowledge into a new pattern, in order to create a new test case that should answer the new aspects we are testing. So from a conceptual standpoint, we are reusing. In this paper we will examine how much reuse is done with regard to testing artifacts. In addition, we review the extent to which we are aware of the reusable nature of our work when designing a testing artifact. Our day-to-day manual testing work flow is built out of |context| –> |concept| –> |build| –> |use|. We also review how much of a software engineer's attention/awareness is focused on the issue of reuse [9].

The reuse of artifacts is usually derived from the desire to take advantage of previously developed components and capabilities. Wang et al. [10] state that a distinction was made between Development with Reuse, which focuses on benefits gained from the utilization of reusable resources, and Development for Reuse, which aims at the creation of reusable products for the benefit of future usage. A generalized reuse model for system development was formulated, suggesting a future quantitative evaluation of reuse in a comprehensive manner [11].

Next we discuss reuse from three points of view: A. Reuse heuristics, B. System reuse frameworks, C. Reuse of Testing Artifacts.

2.1. Reuse Heuristics

Fortune and Valerdi [12] address the topic of reuse from a systems engineering perspective, a generalized framework for the reuse of systems engineering products has been proposed. This approach is based on reuse heuristics (the following is a partial list selected from the original study) [12]:

- Heuristic a: Reuse is not free, upfront investment is required.
- Heuristic b: Reuse should be planned from the conceptualization phase of programs.
- Heuristic c: Most project-related products can be re-used.
- Heuristic d: Reuse, in large part, is also an organizational issue.
- Heuristic e: Higher reuse opportunities exist when there is a match between the diversity and volatility of a product line and its associated supply chain.
- Heuristic f: Bottom-up (individual elements where make or buy decisions are made) and top-down (where product line reuse is determined) reuse requires fundamentally different strategies.
- Heuristic g: Reuse applicability is often time dependent.
- Heuristic h: The economic benefits of reuse can be described in terms of either improvement (in quality, risk identification) or reduction (of defects, cost/effort, risk, and time to market).

The ability to recompose reusable parts is an important requirement for reuse [9]. Anticipating future reuse scenarios makes reusable parts easier to compose. Khusidman and Bridgeland [13] presented a framework of reuse and cloning techniques in software development. This work analyzed different aspects of reuse and cloning by utilizing a classification framework to define a matrix of reuse scenarios aimed at efficient reuse. A distinction may be made between "formal" reuse of object code that does not require any customization, and the "opportunistic" "cut-and-paste" reuse achieved by using and modifying fragments of existing solutions [13]. In other possible classifications we can detect potential reusable parts manually or automatically. Examples of automated detection: DUP in Eclipse (general for common code fragments) suggestions to developer of UI automated tests [14], or record UI test and structure better, utilizing the reusable parts, for instance the BlackHorse project [15].

An additional obstacle to reuse may be derived from the transition to Agile frameworks [16]. In the following section, we will attempt to generalize a reuse framework and apply its principles to test automation.

2.1. Systems Reuse Framework

It has been said that "Reuse can increase your productivity by nearly half if you avoid the common pitfalls that derail many reuse programs" [17]. This idea was made clear from the analysis of the outcome of trends in Source Lines of Code (SLOC) of the US Department of Defense (DoD) software and DoD cost in dollars per SLOC between 1950 and 2000 [17].

However, reuse in software development and testing may cause certain negative effects, such as the propagation of errors in subsequent versions of the software [18]. Comprehensive research about reuse of a test case in a safety-critical system (e.g., for a heart pacemaker) [19] concluded that, conceptually, the reuse principle may sound simple, but to implement it in a real project with hundreds of thousands of lines of code, recognizing the commonalities among the test cases, and implementing a mechanism for systematic reuse, is a huge task. Applying reuse techniques at the testing stage of a real project that involved the development of a cardiac rhythm management system led to significantly reduced efforts required to test systems. More recent studies relate reusability to Software Product Line Testing (SPLT) [20-22]. The strategy of reuse of core assets in SPLT can reduce software testing efforts during development, improve software quality, and potentially decrease the time-to-market of products and services.

Although this paper is concentrated on the reuse of artifacts, we do not focus on reuse of code or code cloning [23]. In our proposal, we place our emphasis on already mature and tested artifacts, which will be elaborated in the next section, and reuse them rather than propose the significant reuse of code.

2.2. Reuse of Testing Artifacts

The reporting on Testing reuse approach in safety-critical systems [19,23] proposes a domain-specific reuse approach. Implementing a mechanism for systematic reuse is a large and demanding task, but it may reduce overall testing efforts.

Another approach is to develop a method for using test assets (testing artifacts) that can be used multiple times on the same product asset and multiple times on related, but different product assets. A test asset will be applied multiple times to the same product asset, especially while the product asset is being developed [24].

Tiwari and Goel [25], the authors of a wide survey of the literature about the reduction of testing effort through reuse, have argued that although there are many systematic studies that deal with quality assurance techniques, virtually no literature or survey exists on reuse-oriented testing approaches. RDTA deals with the reuse of testing automation artifacts using a comprehensive multi-level reuse approach.

2.3. Motivation

The motivation behind this work is to present an orderly proposal for a new way of thinking and organizing all test automation artifacts. The proposal envisions a system for covering all known test elements and artifacts so the maximum of them will be easily available and accessible for reuse. We feel the need for presenting and suggesting a top-bottom approach of collecting, storing, and reusing the artifacts, to minimize cases of "reinventing the wheel." In order to support this suggestion, which we will present later (in Chapter 4), a case study has been conducted, based on implementing this approach.

3. Problem Statement

A software development organization that wants to implement such a solution may need to map all elements that are candidates for participating in the buildup and hierarchy of the proposed repositories. Each of these repositories will need to be defined by their properties and each must be distinguished from the others.

Additionally, while it is important, it is not so technically obvious that in order to build a comprehensive infrastructure that fulfills all broad scope and content needs for both the architecture and its supporting tools. Section 4, dealing with the RDTA novel approach, signals key possible elements and principles for the complex multi-dimension issues raised by such changes.

4. RDTA: A Novel Approach

In this section, we present our contribution to the reuse classification framework by laying out the organizational structure for the different levels, types, and candidates of the storage repositories. The classification system presented here is preliminary and may be further expanded and adapted to different technologies and software development infrastructures.

Even before we consider test automation, we are obligated to store and maintain the testing artifacts. In addition to the test repository classifications of Gupta and Prakash [26] as shown in Figure 1, we propose other levels and artifacts to be stored and maintained.

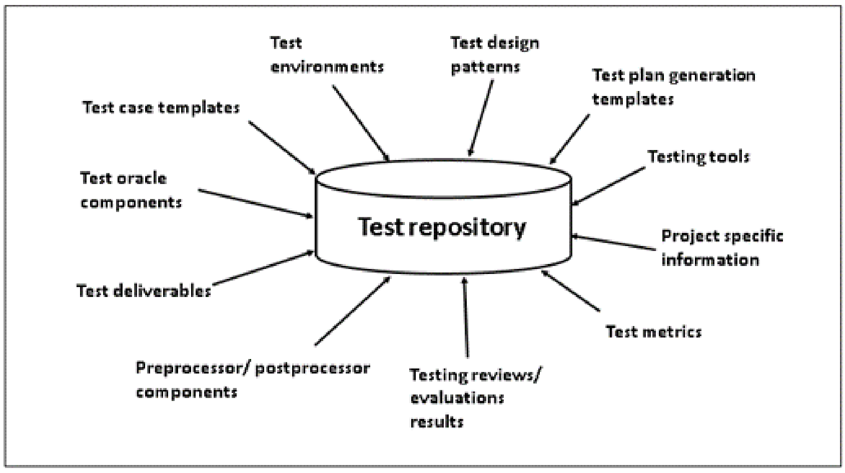


Fig. 1. Gupta and Prakash test repository

Other research and papers have provided useful resources for testing item classifications, for example, verification items [27]. Complementary to the Gupta and Prakash classifications, and in accordance with the research of many experts in the test automation industry, we propose the creation of this preliminary list of subject repositories:

- Business and Testing Repository
- Business Story Repository
- Test Cases Repository
- Basic Operational Element Repository
- Unit Test Repository
- Business Element Testing Repository
- Test Environment Repository

Additional repositories may be introduced during the progress of this research project. All these repositories support reuse when transmitted to, and propagated among, organizations and teams.

In the following paragraphs, we will name and describe each of the repositories suggested by RDTA; for some of them we will demonstrate and offer some properties relevant to the test artifacts within the repository.

4.1. Business and Testing Requirements Repository

Similar to the link between code and requirement, the RDTA approach demands a pointer link from testing artifacts to the affiliated requirement. When RDTA samples a subset of artifacts to be tested based on environment modeling, RDTA requests the ability of analysis and systematic traceability.

RDTA distinguishes between higher level business requirements and their breakdown into functional requirements and nonfunctional requirements.

Therefore, it is imperative that there be a repository where the entirety of the testing requirements is stored, maintained, and controlled. Additionally, many current test management tools contain their own test requirements are stored – and are linked and traceable to the software requirements as well as to the remaining testing artifacts. A requirement itself may build up in a hierarchical manner.

Shown below is a sample list of properties (dimensions) of a requirement artifact:

- Aggregation level
- Dependence on other requirements
- *Business affiliation*
- Classification Type (Technical, Non-Functional Testing (NFT), functional, etc.)
- Affecting/effected – up and down influence
- Maturity
- Expected test result/Oracle information (for a testing requirement)
- {Aggregation level – either it is a final detailed artifact or it is a collection of a group of artifacts.
- Maturity will refer to the previous time in which it has been used before.}

4.2. Business Story Repository

Derived directly from the store of software testing requirements is a repository of testing business stories. Software testing requirements artifact and testing business stories should be as tightly hierarchical as possible. Different aggregation levels may be represented in a repository for these testing stories or business story fragments. For example, "*The customer should be able to access the application from most popular interfaces (mobile phone, PC, remote interface, etc.) using a login procedure*." Tractability here is vital, as demonstrated in Figure 2 and Figure 3.

Shown below is a sample list of properties (dimensions) of a business story artifact:

- Business affiliation
- Aggregation level
- Classification Type (Technical, NFT, functional, etc.)
- Affecting/effected – up and down influence
- Maturity

4.3. Test Cases Repository

The test cases repository derives from the test business stories repository. Please note that test cases are very much oriented toward application/functionality, therefore requiring storage in different hierarchies that allow for different affiliations or relationships to be exposed and identified. Figure 2 presents a possible traceability matrix. This conceptual model demonstrates the need for documentation as well as management and control of all items during the testing/fixing operation. Each column presents repository categories containing other testing artifacts. The sample arrows hint at a possible dependence between the elements. Top red arrow presents the direct correlation between Def#001 and BR 1.

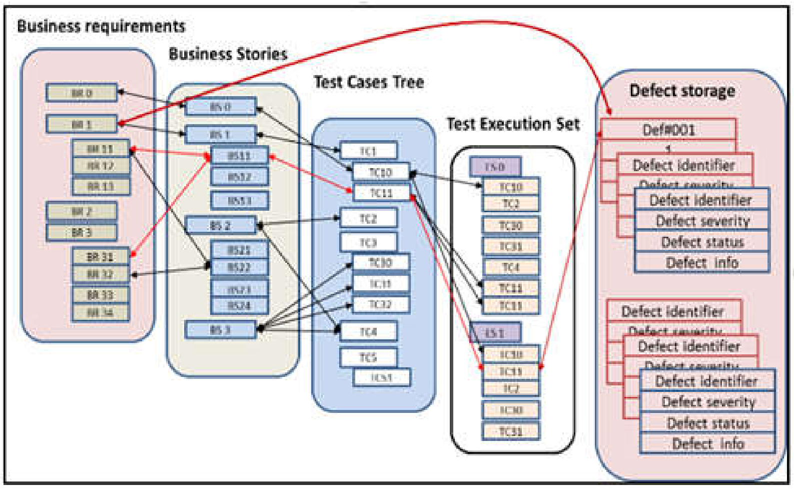


Fig. 2. Suggested test coverage matrix

A uniform test case repository is vital, so formal and structural definitions of the test case must be obtained [28]. A test case repository is provided by almost every test automation infrastructure used today as an elementary storage facility that enables quality and repeatable test automation.

These types of coverage matrices enable trackability [22], and may help support keeping track of and documenting all business and testing artifacts

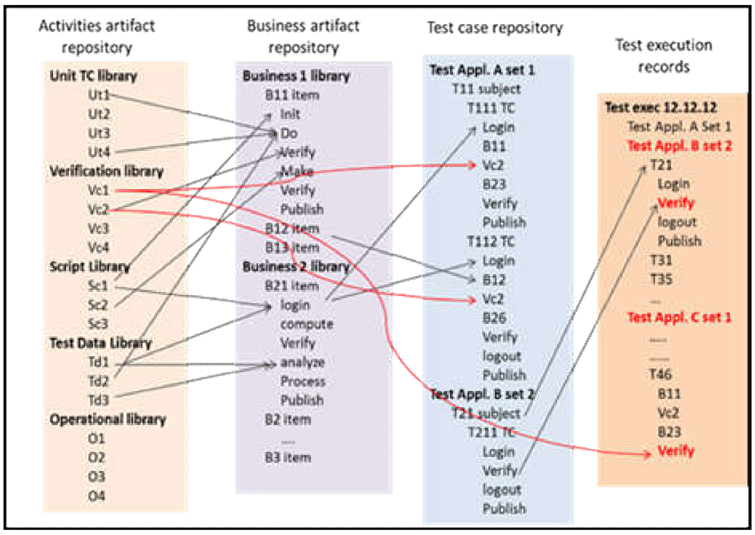


Fig. 3. Principal RDTA testing repositories build up

4.4. Basic Operational Element Repository

In order to facilitate test automation needs, we must be able to execute and operate all developed applications under conditions of control and isolation. This can be performed during the development phase or in an integration workplace before installation. RDTA divides these repositories into three categories:
1) Operational infrastructure, architectural foundation related storage, and application.
2) Business-related storage.
3) Verification of all artifacts [27], using an external mechanism to evaluate the current status of test execution.

The reusable quality of the items stems from the similarity in the basic application of the actual business behavior in the software. Each of these artifacts may be used, operated, stored, maintained, and manipulated during the testing project. More items may be added and specifically modified. The use of these artifacts is limited by resource constraints and time horizons.

An example of a basic operational elementary artifact may be a script-performing operation on test data (propagate, clean, restore, backup, and so on). Shown below is a sample list of properties (dimensions) of an operational artifact:

- Functionality
- OS
- Architecture
- Operation type

4.5. Unit Test Repository

A unit test is, by its nature, attached to the code from which it was created; therefore, we are not expecting to reuse it, we are merely concerned with changes to the actual code. Can we find another reuse formation of the unit test? Although the following idea may sound contradictive, we suggest further investigating this possibility.

Addressing the product of a unit test, by itself, may be troublesome – We are faced with a large amount of testing work that has already been completed and that could easily be executed automatically, but the original testing was hardly done in a continuous fashion [29]. An earlier attempt to reuse unit test artifacts in a system test was made by Elbaum et al. [30]. They attempted carving (identifying and storing) and replaying (CR) differential unit test cases (DUT) from system test cases. Fraser and Arcuri [31] proposed Evosuite as an automatic test suite generation for object-oriented software in an attempt to generate unit test assertions as mutant test cases, in order to reuse the already developed unit test artifacts under other possible assertions. Furthermore, Fraser and Zeller [32] suggested formation of parameterized unit tests containing symbolic pre- and post-conditions characterizing test input and test results. Influenced by the same idea – we can foresee the day that someone will invent a means to have unit tests operate in two modes, in order to accomplish the following:

- Isolate the ability to test the actual single action regardless of its integration maturity
- Operate in an integrated mode, reusing the actual test where all isolation is replaced by integration with its surroundings.

As far as can be presently determined, no one has suggested adaptation of the unit test artifact itself into integrative mode [33]. Because we cannot allocate such a seemingly miraculous unit test tool, we will propose the principles that may lead some tool supplier to invest in this venture. Each unit test item was created with the assertion that it is replacing the actual external interface of the unit (see Figure 4). We seek a mechanism to transform the circle into a square (connected), as illustrated below. If we can do this, we can reuse (very efficiently) the forgotten unit test artifacts. In an early attempt to ease unit test automation, Cheon and Leavens [34], prepared a mechanism for a runtime assertion checker based on the formal specification of the application [35]. Microsoft Libraries present the concept of parameterized unit tests, a generalization of the established "closed" unit tests that can be turned into axioms that summarize the following aspects of the method's behavior: state change, normal return value, and exceptional return [35]. Daniel et al. [36] took another possible step in this direction when the researchers explored the possibility of reasserting broken unit tests. Figure 4 presents the needed transformation flow of isolated unit test artifacts into mature test cases, where some assertions were redirected internally [33].

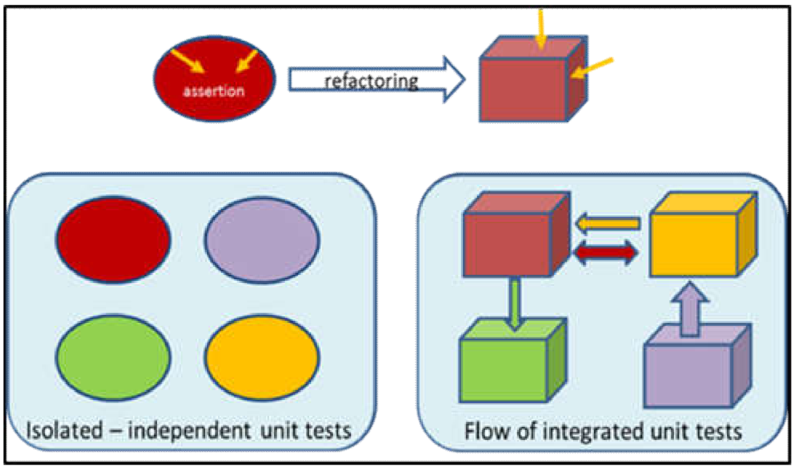

assertion
refactoring
Isolated – independent unit tests
Flow of integrated unit tests

Fig. 4. Unit test artifact transformation

Additional difficulties concerning the storage of unit test artifacts in a repository may come from the bundling with the code itself. Unit test artifacts cannot be re-used in other languages.

Shown below is a sample list of properties (dimensions) of an unit test artifact:

- Coding Language
- Unit test tool
- Context affiliation
- Complexity – Number and structure of assertions in a unit
- Maturity – has been tested, maintainability, been used
- Location (within the code)

An important way around this problem is developing and using dependency injection [37,38] or assertion injection [39].

4.6. Business Element Testing Repository

The need for the reuse of the same generic test case as part of a project scenario with a different categorical affiliation can be satisfied in most of the existing testing tools by duplicating the same formation and storing it separately. The RDTA approach will store another level of artifacts that relate to the test case business context (see Figure 3). In RDTA we try to break the test cases into smaller business artifacts that may be re-used in different test cases. We propose storing smaller elements or parts of a test case, and using those elements to construct other test cases. An example would be to use elements from existing test cases of programs that involve login procedures, and from those elements, to construct a new login procedure test case for another program.

4.7. Test Environment Repository

Creating and using the test environment repository may, for many reasons, prove to be the most challenging idea of all. These reasons include the challenges associated with saving and possibly reusing integration interfaces between systems. The idea of storing a test environment with the intention that it be re-used necessitates the need for virtual machines and storage. In many cases the test environment also maintains a ready data set for the tests, creates a test environment for each of the integrated backend systems, saves the snapshot, and then re-sets the virtual machine from the snapshot. This is the point when testers finally destroy all data that could work in specific cases. Currently it is becoming more common to duplicate test environments, especially when using public Internet resources. For example, we could use Sandbox [40]. A sandbox is a testing environment that isolates untested code changes and outright experimentation from the production environment or repository. Sandboxing protects "live" servers and their data, vets distributions of source code and other collections of code, data, and/or content, either proprietary or public; It protects live products from changes that could be damaging (regardless of the intent of the author of those changes) to a mission-critical system or which could simply be difficult to revert. Sandboxes replicate at least the minimal functionality needed to accurately test the programs or other code under development (e.g., usage of the same environment variables as, or access to, a database that is identical to that used by the stable prior implementation intended to be modified). Note that the limitations of these expansions arise from the fact that most modern sandbox mechanisms such as Docker technology [41] include within the tested environment the actual software under test applications. This is so one cannot build separate testing repositories for reuse with another software application – the RDTA intended policy

5. Application of RDTA in Concurrent Development Scheme

The rapid change of software development schemes and trend may raise the question of how the RDTA approach fits into the upcoming future of software development. This paper will not answer this particular question, but we do intend to address two key issues: Agile and SOA – in the sense of organizing the tests in a service-oriented manner, not necessarily to reuse automated tests for testing integration interfaces.

As described in Section 3, in order to facilitate easy access and usage of reusable testing artifacts, the RDTA approach mandates adding another merged level—one that stores, uses, and maintains another practical set of testing items. Business artifacts may be related to each element (or object) of the testing artifacts; these can be treated as a business portion (as opposed to technical, architectural, or other such elements).

RDTA recommends storing and maintaining the actual full context testing scripts so that during subsequent use the user will have full control of all operational and functional aspects to be tested. This approach may also assist the Continuous Delivery revolution as well, by having all test automation artifacts ready to be executed at will, similar to HARNESS TESTBEDS [42], where both local virtual machines in Virtual

Box on developer workstations and virtual machines can be run within the testing environment.

5.1. Agile and Test Automation Artifacts

An essential part of RDTA is to enable the collection of all artifacts developed along the way. We will be able to follow this by having a unit test repository (attached to the actual code) and by developing the test cases together with the software development, as dictated by the Agile model (both being derived from the same business stories). At the end of each sprint we will have the testing artifacts ready to accompany the newly developed software to its next integration and implementation stage (See Figure 5).

Product backlog is built directly from user stories (derived from requirements). At the sprint level the scrum team (in the case of a scrum model) simultaneously creates the code – with its unit test and test cases for testing the stories [43]. Altogether, at the end of each sprint a sprint regression test package should be formed to accompany the ready feature. Following this, storing all artifacts in the target repositories is essential in RDTA. Time for building test cases and a regression suite will shorten if the reuse principle is activated, especially if you have specific designated sprints for integration purposes [44]. Reuse requires uniformity of all testing and operational artifacts. Using RDTA will require having all supporting infra-structure installed. We suggest enforcing the use of the same storage and operational tools throughout the organization.

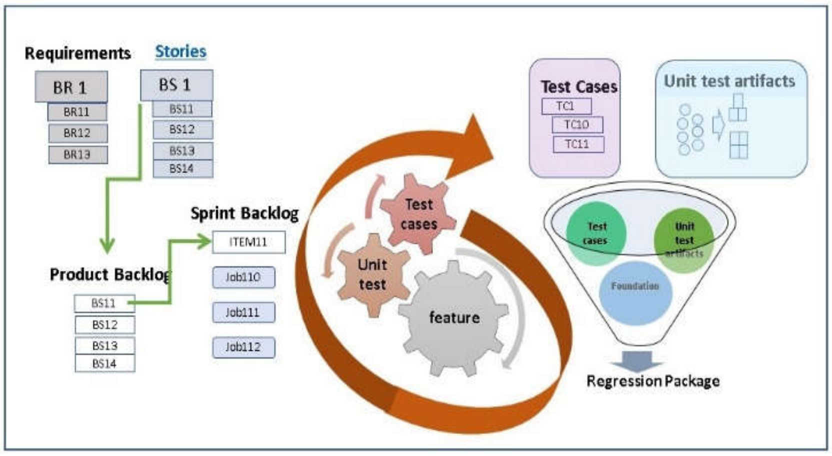


Fig. 5. Agile sprint testing artifacts

5.2. Nature of Test Automation Artifacts

The RTDA approach suggests a framework where each of the elements is categorized as a service [45] – so it can be recalled and operated independently during the progression of the testing levels. Service orientation here will mandate additional infrastructure, facilitating another technological layer. Web services by themselves will not solve all the additional requirements stated above. The complexity and multidimensional properties of the needed services demand more sophisticated types of services. Having the usage of these artifacts as an integrated part the internal flow may lead to an expanded demand to have services operated differently than “simple” Web Services.

5.3. Maintaining the Integrity of the Repositories and the Specifications

Such a complex, interconnected, and affiliated storage system must be formulated in a very practical manner. Therefore, how and where to store objects are critical issues. We must be aware of the needed efforts to maintain and upgrade all RDTA repositories on a continuing basis, all the time and not as one-time effort; substantial maintenance should be expected. Of course, we can potentially decrease this overhead by suitable annotations of the testing artifacts during their creation. Example – custom “JavaDoc comment style” attributes in unit tests. Design of repository schema could be challenging. Optimization lies, as a practical matter, between being too strict (under a fixed schema), which will minimize inconsistencies in stored data and relations between them, but will not be flexible enough to accommodate new types of artifacts or their attributes; and the opposite extreme – being too loose (under a flexible schema), which is flexible enough to store various types of ad hoc artifacts, but allows potential inconsistencies in data.

5.4. How to Store Repositories

Reflecting on the operationally practical needs for the storage requirements, the following list of storage requirements has yet to be fully researched and evaluated. It is evident that each of these requirements may influence the infrastructure to be implemented:

- Easy and efficient storage and retrieval (ease of use)
- Support for all types of items (from single data to complex executable modules)
- Support of version control
- Ability to follow complex associations between the items
- Support for dynamic hierarchy relationships
- Discoverable and presentable on multiple layers and dimensions
- Easy maintenance
- Ability to follow security requirements
- Unlimited size.

5.5. RDTA and the Test Automation Creation Work Process

Because the organization will have to adopt new work processes for implementing RDTA, we propose the following five-step work process. Assuming that each project has its own repository in the framework and that, in addition, there is a common repository for reusable artifacts (see Figure 6), the following process would apply:

1. **Analyze:** Analysis of project artifacts and the creation of a project repository.
2. **Map:** Mapping the affiliations of project artifacts to existing reusable artifacts.
3. **Inherit:** Acquiring test artifacts from the common repository for insertion into the project repository.
4. **Create:** Creating missing test artifacts at the project repository.
5. **Upload:** Uploading the missing or updated artifacts into the appropriate common repository

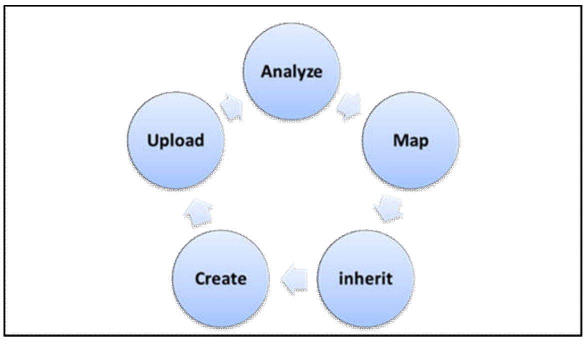


Fig. 6. RDTA operation work process five steps

5.6. Implementing RDTA

Implementing RDTA may prove to be a difficult and complicated task in light of the variability and complexity of infrastructure, organizational cultures, standards, and new quality measurements. One can foresee two different approaches for implementation:

- Top down – where management dictates, supervises, and imposes changes in production.
- Bottom up – where change develops from the bottom through limited experimental trials of one of the test automation teams, then subsequently percolates up and spreads gradually through the organization.

An additionally important aspect of implementing RDTA will be the organizational and cultural change implications, where knowledge ownership is shifting from the individual to public ownership [46].

6. Case Study – Implementation of RDTA at a Large Hardware/Software Manufacturer

This case study is a report of a successful attempt to apply RDTA philosophy to a large test automation project. The project was established on uniform platforms and infrastructure of the organization. It was limited to specific needs, tools, and technology and did not include requirements or other artifacts in an “upper” level of repositories. Note that a few details (such as economical calculation and statistics and other information, were omitted from the report at the request of the company (as a precondition for sharing information that was relevant to this study).

6.1. The Company Challenge – Background

The company is an international innovative company, leading and influencing international technology progress and development since the 1960s, and is a global leader of microprocessor manufacturers.

Hundreds of night shifts, in dozens of labs around the globe, are taking place at the company on a daily basis. During these night shifts, hundreds of thousands of business scenarios are executed for hardware components (drivers, DDRs, CPUs, etc.) over distributed systems with different configurations. To be clear, for purposes of this study, the testing of the software, not the hardware, is paramount.

6.2. Motivation and Reasons

The challenge mentioned above is managing the tests over dozens of distributed labs around the globe, in an efficient and effective way. These labs have different systems, different configurations, a variety of hardware components, and different hours of operation and testing. The company decided to seek experts to lead this effort, and approached a local leading QA and testing service to address their requirements and needs.

The initial state, before using the global services of System Awareness and Instrumental Interfaces as part of the RDTA project, is described as follows:

- Huge test automation development effort
- 25% test coverage from total needs – using sporadic scripts
- Hundreds of backup servers scattered all over the globe
- ROI = -22% (Investment = 13,200 Test Automation Dev. Hr., Payback=10,296 Testing Hr.)

ROI calculation = [(Payback - Investment)/Investment)] *100

6.3. Project Participants (Actors, Stakeholders)

People leading the case study: Automation Team Leader and Junior Test Automation Engineer.

Stakeholders: Developers, system engineers, lab managers, practical engineers, integrators, QA test automation members in dozens of labs (power, performance, integration, EV, SV) around the globe.

6.4. Project Progress

The team implemented the Agile approach in this project. More than 250 automation scripts are added every month – 3000 scripts a year. The scripts are executed on more than 4500 systems scattered in more than 20 labs; and are used by approximately 500 test engineers, system engineers, and integrators.

6.5. New Test Automation Architecture

The new test automation architecture was created in the project. This architecture is centered on a global centralized repository that can be accessed by various groups in different units throughout the organization. Each group is obligated to upload its newly updated testing artifacts into the global repository, so unified taxonomy has been published to all participants. Another key factor was the usage of a unified homemade dashboard enabling all stakeholders to communicate and address testing and quality measurements in uniform language and terminology.

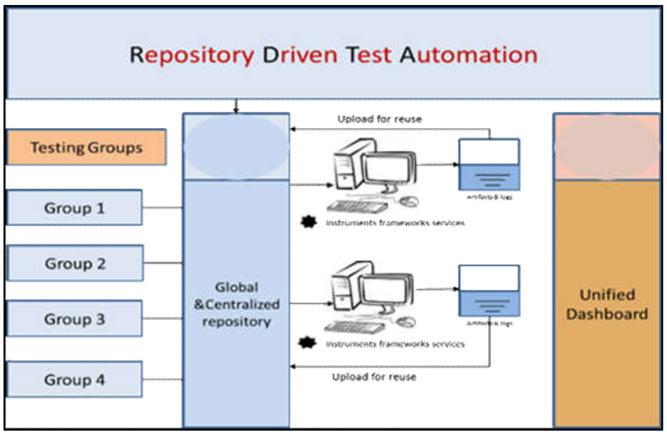

Repository Driven Test Automation
Testing Groups
Group 1
Group 2
Group 3
Group 4
Global &Centralized repository
Upload for reuse
artifacts & logs
instruments frameworks services
artifacts & logs
instruments frameworks services
Upload for reuse
Unified Dashboard

Fig. 7. RDTA project architecture.

Figure 7 presents the architectural buildup and approach of the infrastructure involved in the project – note the global and international aspects of this implementation.

Figure 8 presents a selected sample of the implemented repositories. It is apparent that these repositories address the commonly shared elements of all test automation project elements.

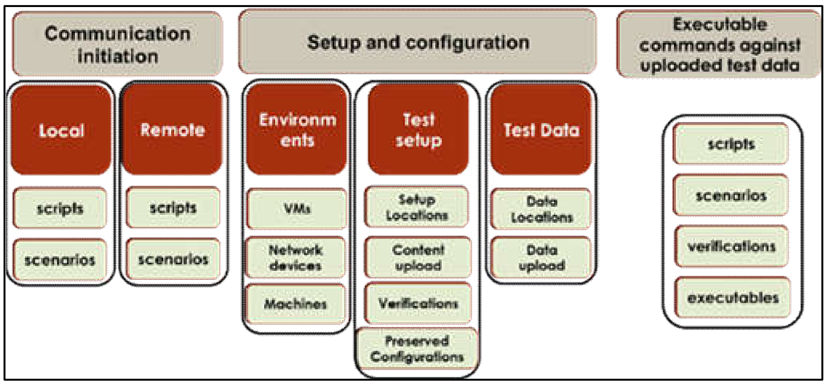


Fig. 8. Selected implemented repositories

The focus of this implementation is reuse of initiation and usage of required elements such as infrastructure, communication, test lab environment, and setup. These elements are shared by all projects regardless of a specific domain.

6.6. An Example of a Search for Reusable Artifacts

Process:

1. Instruments Framework
   ► Detailed information and examples can be found in company's SharePoint.
   ► Testers all around the globe can connect to the Instruments Framework Service.
2. Supply a sequence of commands and receive the generated code as an artifact.
3. Sequences and Generated Code can be uploaded into the repository and can then be re-used.

System Awareness Service:
Detailed information and examples can be found in the company's internal SharePoint. Testers all around the globe can connect to the System Awareness Service, supply a Host or a Target name, or both, and receive a detailed environmental review as an artifact.

A single service is being re-used by thousands of the company employees around the globe.

Example:

The tester supplies:

```
(Python) Agilent → Init → Connect → GetConfig → GetIP
```

The tester receives the following code:

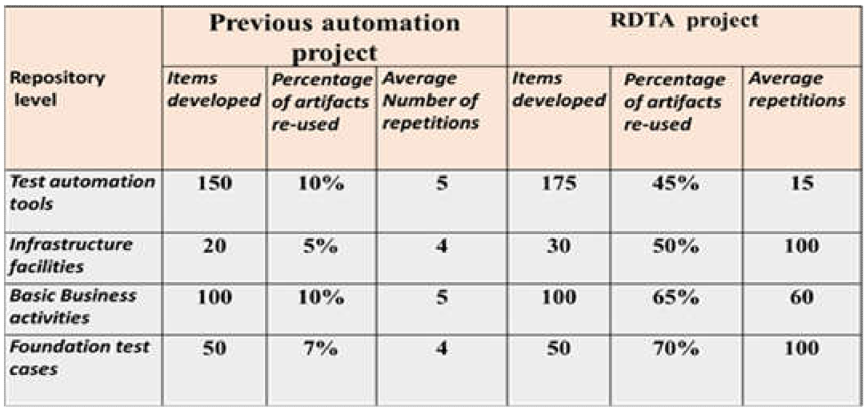
```
agilent=instrumentsLayerService.createPythonInstrument('A
gilent')
agilent.instrument.SetAddress(str(0xc0))
agilent.Init()
resConnection=agilent.Connect()
resConfig=agilent.GetConfig(config)
resIP=agilent.GetIpAddress()
```

This code sample was presented here, showing the challenges facing implement such an approach in large company.

6.7. Project Outcomes

Table 1 presents the actual workload and a comparison between a previous test automation project of similar volume and the RDTA project. The achieved direct benefits can be easily seen by reuse (%) and average number of repetitions.

Table 1. Project results



Repository level	Previous automation project			RDTA project		
	Items developed	*Percentage of artifacts re-used*	*Average Number of repetitions*	*Items developed*	*Percentage of artifacts re-used*	*Average repetitions*
Test automation tools	150	10%	5	175	45%	15
Infrastructure facilities	20	5%	4	30	50%	100
Basic Business activities	100	10%	5	100	65%	60
Foundation test cases	50	7%	4	50	70%	100

All ROI calculations appear as reported by the customer and the only comparison of the proposed approach with alternative approaches can be seen at the table I (previous automation project), together with principal outcomes reported by the team.

Main project outcomes reported by the team:

- Reduced costs
- Low development effort
- Used internally and externally
- Programmers reuse the building blocks made for the QA and testing teams in their Unit Tests
- The indicators show that more than 85% of the building blocks are being re-used
- Approximately 500 test engineers, system engineers, and integrators are using the test automation scripts
- Used for validating at customer sites
- 1.7M business scenarios are covered (>85% coverage)
- Agility and Uniformity of tests
- Dozens of backup servers only
- User and Role Management
- Transparency – Centralized Reporting
- ROI = 904% (cost of investment = 4,400 Test Automation Development Hours, gain from investment = 44,176 Testing Hours).

6.8. RDTA Project Recommendations

These are general and related to the process recommendations of the project team:
The team members and the active stakeholders made the following recommendations:

- Keep your services generic and flexible
- Focus on test automation uniformity
- Prepare a user guide
- Use mirror servers
- Synergy and sharing between internal groups
- Dashboard and transparency.

7. Discussion of Results

As declared in chapter 3, the motivation behind this work was to propose an orderly new way of thinking and organizing all test automation artifacts. Answering the requested need of the industry for a method that can cover all known test elements and artifacts so the maximum of them will be readily available and accessible for reuse. The example provided by the company (a mechanism to search and extract existing items) may present an important aspect of the project – finding a way to expose and share elements previously developed. This, of course, aligns with the theme of RDTA about reuse. In accordance with the discretion that was requested by the company, the outcome of this project was presented at a high level only. There are four key advantages to be shared:

- The process enforces organizational cultural change.
- The process opens the ground for potential improvements
- The project was shown to be economical where the ROI was easily proved
- The project created a base for adjustable software development methodologies.

On the other hand, the project has presented some challenges to be considered:

- The process is difficult to implement
- The project requires tight managerial support
- The hierarchal structure is the most challenging aspect because the diversity between units and their contextual view point
- This process enforces organization cultural change

A significant limitation of this case study originates in its partial approach; (the project did not include requirements or other artifacts in an “upper” level of repositories). We hope that one day we will have the opportunity to present a case study of an organization that covers all aspects of RDTA, expanding the scope that was presented in Section 4.

8. Conclusions

This paper presents a new conceptual approach to test automation – RDTA. This approach focuses on the reuse principle for test automation artifacts. The RDTA approach presents a hierarchical multi-level repository that encapsulates different test artifacts, regardless of a specific technology or tool. We have suggested a list of possible storage entities (repositories), each presenting another aggregation level of test automation artifacts. The final results should produce test automation infrastructure that supports the reuse of artifacts. A case study of a large implementation project adapting concepts from RDTA with a significant result was reported – proving the benefits and ROI of such a configuration and architecture. We ought to evaluate the generality of results from the case study presented, since the RTDA present an approach it is possible that implementation in different company size, organizational profile and methodology - may lead to another settings, tools and solutions. We also have to be aware of the need to adapt RDTA concept to the modern CICD.

This work did not address an important aspect of RDTA: the implication and influence on the organizational culture and change in management process. We believe this aspect of the RDTA concept merits additional research efforts.

In order to further transition from concept to practice, each subject and proposition presented here should be addressed and developed into an organizational strategy and framework to reduce costs. More broadly, we envision the creation of international sharing schemes for the purpose of resource and performance amplification. Further development of the criteria for the selection of services and the evaluation of RDTA benefits are required.

References

1. Almog, D., Schwartz-Chassidim, H., Mark, S., and Tsubery, Y.: RDTA – Repository Driven Test Automation- A new look into reuse of test automation, presented at the 7th Int. Conf. Adv. Sys. Test. Valid. Lifecycle, Barcelona, Spain, Nov. 15-20, (2015)
2. Myers, G.,: The Art of Software Testing. New York: John Wiley & Sons, (2004)
3. Malishevsky, A., Rothermel, G., and Elbaum, S.,: Modeling the cost-benefits tradeoffs for regression testing techniques in Software Maintenance, in Proc. Int. Conf. Software Maintenance, IEEE, Montëal, Canada, pp. 230-240. (2002)
4. Rafi, D. M., Moses, K. R. K., Petersen, K., and Mäntylä, M. V.: Benefits and limitations of automated software testing: Systematic literature review and practitioner survey, in Proc. 7th Int. Workshop Automation of Software Test, Zurich, Switzerland, Jun. 2-9, (2012)
5. Vizulis, Valdis, and Edgars Diebelis. "Self-testing approach and testing tools." Datorzinatne un informacijas tehnologijas(2012): 27.
6. ISTQB Certified Tester Advanced Level Syllabus Test Automation Engineer https://www.istqb.org/downloads/send/48-advanced-level-test-automation-engineer-documents/201-advanced-test-automation-engineer-syllabus-ga-2016.html
7. Almog D., and Tsubery Y.,: How the Repository Driven Test Automation (RDTA) will make test automation more efficient, easier & maintainable, in Proc. 8th India Software Eng. Conf. ACM, Bangalore, India, Feb. 18-20, pp. 196-197. (2015)
8. Cleland-Huang, J., Gotel, O. C. Z., Hayes, J. H., Mäder, P., and Zisman, Z.,: Software traceability: trends and future directions, in Proc. Future Software Eng. Hyderabad, India, May 31-Jun. 6, (2014)
9. Parsons, J., and Saunders, C.,: Cognitive heuristics in software engineering applying and extending anchoring and adjustment to artifact reuse, IEEE Trans. Software Eng., vol. 30, no. 12, pp. 873-888, Dec. (2004)
10. Wang, G., and Rice, J.,: Considerations for a Generalized Reuse Framework for System Development, in INCOSE Int. Symp., Denver, CO, Jun. 20-23, pp. 2178-2394. (2011)

11. Bures, M.,:Automated testing in the Czech Republic: the current situation and issues, in Proc. 15th Int. Conf. Computer Systems Technol.Ruse, Bulgaria,Jun. 27-28, pp. 294-301.(2014)
12. Fortune, J., and Valerdi, R.,:A framework for reusing systems engineering products, Sys. Eng. vol. 16, no. 3, pp. 304-312 Sept. (2013)
13. Khusidman, V., and Bridgeland, D. M.,:A Classification Framework for Software Reuse, J Object Technol. vol. 5, no. 6, pp. 43-61, Jul. (2006)
14. Filipsky, M., Bures, M., Jelinek, I.,:Creating Smart Tests from Recorded Automated Test Cases, in New Contributions in Information Systems and Technologies. Springer International Publishing, pp. 773-780. (2015)
15. Carino, S., Andrews, J. H., Goulding, S., Arunthavarauaj, P., and Hertyk, J.,:BlackHorse: creating smart test cases from brittle recorded tests, Software Qual. J. vol. 22, no. 2, pp. 293-310, Jun. (2014)
16. Turk, D., France, R., and Rumpe, B.,:Assumptions underlying agile software development processes, arXiv preprint arXiv:1409.6610, Sept. (2014)
17. Boehm, B.,: Managing software productivity and reuse," Computer, vol. 32, no. 9, pp. 111-113, Sept. (1999)
18. Weyuker, E. J.,: Testing Component-Based Software: A Cautionary Tale, IEEE Software, vol. 15, no. 5, pp. 54-59, Sept. (1998)
19. Poonawala, M., Subramanian, S., Tsai, W. T., Vishnuvajjale, R., Mojdehbakhsh, R., and Elliott, L.,:Testing Safety-Critical Systems-A Reuse-Oriented Approach, in Proc. 9th Int. Conf. SEKE, Madrid, Spain, Jun. 1997, pp. 271-278. (1997)
20. McGregor, J. D.,:Testing a software product line, in Testing Techniques in Software Engineering, Berlin, Germany: Springer, pp. 104-140. (2010)
21. Bosch, J.,: Design and use of software architectures: adopting and evolving a product-line approach. Upper Saddle River, NJ: Pearson Education, (2000)
22. Condron, C.,: A domain approach to test automation of product lines, in Int. Workshop Software Product Line Test, 2004, CiteSeerX. vol. 2004, p. 27, Aug. (2004)
23. Roy, C. K., Zibran, M. F., and Koschke, R.,:The vision of software clone management: Past, present, and future (keynote paper), in Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), Software Evolution Week-IEEE Conference on, Feb. 2014, pp. 18-33. (2014)
24. McGregor, J. D.,:Verification and validation issues and implications for reuse. in Workshop on Verification and Validation of Models and Simulations, Foundations' 02, Proceedings, Oct. (2002)
25. Tiwari, R. and Goel, N.,:Reuse: reducing test effort, SIGSOFT Softw. Eng. Notes, vol. 38, no. 2, pp. 1-11, Oct. (2013)
26. Gupta, M., and Prakash, M.,: Possibility of Reuse in Software Testing, in 6th Ann. Int. Software Testing Conf. India, Bangalore, India, Dec. 1-2, (2006)
27. Almog, D., and Heart, T.,:Developing the Basic Verification Action (BVA) Structure Towards Test Oracle Automation, in Comp. Intell. Software Eng. (CiSE), IEEE 2010 Int. Conf.. Wuhan, China, Dec. pp. 1-4. (2010)
28. Almog, D., and Heart, T.,:Test Case Definition: A New Structural Approach, in Cast009, Colorado Springs, CO, Jul. 13-16, (2009)
29. Robinson, B., Ernst, M. D., Perkins, J. H., Augustine, V., and Li, N.,:Scaling up automated test generation: Automatically generating maintainable regression unit tests for programs, in Proc. 2011 26th IEEE/ACM Int. Conf.. Nov 2011. pp. 23-32. (2011)
30. Elbaum, S., Rothermel, G., Kandun, S., and Malishevsky, A. G.,: Selecting a Cost-Effective Test Case Prioritization Technique, Software Qual. J., vol. 12, no. 3, pp. 185-210, (2004)
31. Fraser, G., and Arcuri, A.,:Evosuite: Automatic test suite generation for object-oriented software, in Proc. ACM SIGSOFT Symp. and 13th Eur. Conf. Found. Software Eng. (FSE). Szeged, Hungary, Sept. pp. 416-419.(2011)

32. Fraser G., and Zeller, A.,:Generating parameterized unit tests," in Proceedings of the 2011 International Symposium on Software Testing and Analysis, Toronto, ON, Canada, Jul. 17-21, 2011, pp. 364-374.
33. Y. Tsubery and D. Almog, "Reuse of Unit Test Artifacts – Allow Us to Dream, Agile Record, vol. 16, pp. 49-52, (2013)
34. Cheon, Y., and Leavens, G.,:A simple and practical approach to unit testing: The JML and JUnit way, in ECOOP 2002—Object-Oriented Programming, Berlin: Springer, pp. 1789-1901.(2006)
35. Tillmann N., and Schulte, W.,:Parameterized unit tests, ACM SIGSOFT Software Eng. Notes, vol. 30, no. 5, Sept. (2005)
36. Daniel, B., Dig, D., Gvero, T., Jagannath, V., Jiaa, J., Mitchell, D., Nogiec, J., and Tan. S. H.,:ReAssert: a tool for repairing broken unit tests. in Proc. 33rd Int. Conf. Software Engineering, Honolulu, HA, May 21-28, pp. 1010-1012.(2011)
37. Vare, C.,:Dependency injection for unit testing, Available from: http://www.javaranch.com/journal/200709/dependency-injection-unitesting.html.2016.04.20
38. Polites, J.:Dependency Injection, in The Tao of Testing, Available from: http://www.polites.com, 2013.05.13
39. Maxwell, A.,: Assertion injection, Available from: http://redsymbol.net/articles/assertion-injection, 2014.04.20
40. Gangadhar, K., McKinney, D.,: Sandbox support for metadata in running applications. U.S. Patent No. 8,782,604. 15 Jul. (2014)
41. Fink, J.,:Docker: a software as a service, operating system-level virtualization framework, Code4Lib J. vol. 25, Jul. (2014)
42. Rutherford M. J., and Wolf, A. L.,:A case for test-code generation in model-driven systems, in Proc. 2nd Int. Conf. Generative Programming Component Eng. 2003, Erfurt, Germany, Sept. 22-25, pp. 377-396.(2003)
43. Crispin, L.,: Agile Testing in Real Life. Book your training with Díaz & Hilterscheid!, p. 14. the only reference I find is a pdf written by S.S.F. Pragmatic, with no reference information. (2011)
44. dos Santos, A. M., Karlsson, B. F., Cavalcante, ., A. M., Correia, I. B., and Silva. E., Testing in an agile product development environment: An industry experience report, in 12th Latin-American Test Workshop (LATW), 2011, Mar. 27-30, pp. 1-6.(2011)
45. Woolf, B.,: Exploring IBM SOA technology & practice : how to plan, build, and manage a service oriented architecture in the real world, Gulf Breeze, FL: Maximum Press, (2008)
46. De Long, D.,: Building the knowledge-based organization: How culture drives knowledge behaviors, Centers for Business Innovation–Working Paper, May (1997).

Dani Almog is a senior researcher and lecturer on Software Quality and test automation. Contributing Shamoon Collage of Engineering (SCE) and Ben Gurion University (BGU). Currently the main topic of his studies are Software quality testing and fundamentals addressing software engineering needs. Dani has very vast experience in the industry – former test automation managing director for Amdocs product development division. Dani is serving at the Advisory Board of ITCB (Israeli Testing Certification Board) and leads the ISTQB academia research group stream.

Hadas Chassidim is a faculty member at Shamoon College of Engineering's Department of Software Engineering. She holds B.Sc., M.Sc. and Ph.D degrees in Industrial Engineering from Ben-Gurion University (2013). Her research deals with software quality and testing, human-computer interaction, usable privacy and security. Hadas is the head of software quality development track at Software Engineering department in Beer Sheva campus.

Yaron Tsubery is an IT executive with excessive experience in managing QA & Testing organizations conducting large-scale, complex, real-time system testing at various sectors, Yaron has more than 20 years in Software development, QA and Testing field as a Test Engineer, Customer Support Engineer, Testing TL, and Testing Manager, as well as Product Manager, Project Manager and Developer, before becoming a Senior Director of QA & Testing and PMO. Yaron is the former President of ISTQB® (International Testing Qualifications Board) and is also the President and founder of the ITCB® (Israeli Testing Certification Board). He is a member of IQAMF (Israeli QA Managers Forum) and in SIGiST Israel.

Miroslav Bures received his Ph.D. at Czech Technical University in Prague, Faculty of Electrical Engineering, where he currently leads the Software Testing IntelLigent Lab (STILL). His research interests are Model-based Testing (path-based and combinatorial testing algorithms), employment of artificial intelligence in testing techniques, efficient test automation and quality assurance methods for the Internet of Things solutions, reflecting specifics of this technology. In these areas, he also leads several R&D and experimental projects. He is a member of Czech chapter of the ACM, CaSTB, ISTQB Academia workgroup and participates in many activities in the professional testing community.

Shlomo Mark, professor, currently the dean of the faculty of engineering at SCE Shamoon College of Engineering - Ashdod campus is a professor of software engineering, and the Head of the NMCRC – the Negev Monte Carlo Research Center at SCE. His main research interests are Quality in software engineering, Agility and Agile life cycle, scientific computing and software Life Cycle for Scientific software product, Computational Modeling for Physical, Environmental and Medical Application.

Received: July 1, 2017; Accepted: April 11, 2018.