Dealing with Challenges of Automating Test Execution

Architecture proposal for automated testing control system based on integration of testing tools

Valery Safronau and Vitalina Turlo
Software Testing Automation Department
Applied Systems Ltd.
Minsk, Belarus
safer@appsys.net, turlo@appsys.net

***Abstract*—If implemented correctly, automated software testing maybe an efficient way to circumvent time and resource shortages and ensure faster time to market for new products. Our experience and survey data show that the execution of automated tests is often accompanied by a number of time-consuming and routine operations that are performed manually, e.g., operating virtual machines, setup and cleanup of the environment, test launch, logging defects, etc. These menial chores can be automated with the help of simple command files or by developing an automated testing control solution. In the long run, the latter is a more efficient approach. The paper focuses on the challenges that companies face in attempting to build such a solution, and provides practical recommendations on implementation. Finally, we provide an architecture proposal for the system for automated testing based on testing tools integration, define its features and describe the interactions between its components.**

Keywords-Desktop application testing; survey; industrial experience; integration of testing tools; automated testing control solution.

I. INTRODUCTION

High quality, timely testing is crucial to the development of a reliable software product. By the same token, running regression tests on every released (stable) build is critical, especially in the case of continuously developed complex systems with extensive functionality. However, due to the shortage of resources, regression testing is often being neglected, and its significant lack or incompleteness is one of the greatest problems in software development quality assurance.

In order to solve this issue, Automated Testing (AT) is used. The fact is that implementing AT can be a great challenge in its own right, as it requires well-tuned software development and testing processes as well as clearly organized communication flows. “One of the primary reasons software testing tool implementations fail is because there is little or no testing process in place before the tools are purchased [1].”

In many instances the expression “automated testing” is misleading, as the testing process is still being controlled by a test engineer, especially where desktop applications are concerned. According to the online surveys conducted by Applied Systems Ltd. via the SurveyMonkey.com service, a tester has to manually fulfill some or all of the operations to execute automated tests on a new product build, such as configuring the testing environment, starting/shutting down Virtual Machines (VMs), launching tests, submitting bugs to a tracking system, closing fixed bugs, generating reports, and so on [2][3]. In addition to being very time-consuming, manual operations drastically increase the probability of human error. For these reasons, our goal is to enable the unmanned execution of the full AT cycle by completely automating these routine operations.

In this paper we describe a new, efficient approach to controlling automated software testing that meets the aforementioned challenges. The solution is based on the integration of testing tools. It has been applied in practice, and has proven useful in the automated testing of desktop applications, ensuring non-stop execution of tests while eliminating menial and boring tasks from the work of testers. One of the most obvious benefits of this solution is guaranteed regression testing of each new build.

The present work focuses on the realization of unmanned execution of automated tests – from environment set-up and test launch to defect tracking and report generation – but not on design and development of automated test scripts. We assume that test automation engineers know how to create tests that are reliable, maintainable and data-driven, while complying with the principles of test case independence, absence of redundant code, and scalability.

The findings of this paper are based on more than five years of practical experience in the automated testing of desktop software, as well as the results of two IT community surveys with a pool of more than 300 respondents.

Section 2 gives an overview of key previous work in the field of automated testing. In Section 3 we examine the evolution of automated testing and suggest a new classification of test automation levels emphasizing the amount of manual routine operations in the AT process. Section 4 is dedicated to exploring the main challenges inherent in building an Automated Testing Control System (ATCS). In Section 5 we propose the working archetype of such an ATCS with a detailed description of its main features and components. The Conclusion section summarizes the paper’s findings and outlines the field of research for future work.

The insights of the present work will be useful to Test Automation Engineers, Heads of Testing and QA departments, and those practitioners who wish to develop an in-house solution for automated testing control.

II. RELATED WORK

As automated software testing gains popularity, the body of literature on the subject has been growing steadily in recent years. They provide test engineers with the theoretical and practical base necessary for a successful implementation of automated tests [1][4][5][6]. Authors with extensive professional experience in the industry guide the reader through the decision whether to automate tests, help to navigate through a plethora of testing tools to select the best fit ones, and give advice on building robust and documented testing processes [1]. The works also offer guidance on test planning, design, development, execution, and evaluation [4][6].

For a constructive discussion on which tests cases should be automated and guidelines for assessment of return on investment, see the work by Dustin and Garrett in [6] and [7], as well as Chapter 2 in Mosley and Posey in [1].

In *Automated Software Testing* Dustin, Rashka and Paul introduce the concept of Automated Test Life-Cycle Methodology (ATLM), "which is a structured methodology geared toward assuring successful implementation of automated testing [4]." They identify five phases of ATLM, namely:

1) Decision to automate test.
2) Automated test tool acquisition.
3) Introduction of automated testing to a new project and its optimization.
4) Test planning, design, development and execution.
5) Test evaluation.

Mosley and Posey argue that ATLM is an "artificial construct" that is not very useful for practitioners. They argue against the idea of a software testing life cycle, and claim that the result of the implementation of test automation depends on the quality of the processes already in place in the organization [1]. Despite certain differences in their approach to testing, Dustin and Mosley both promote a deliberate, well-reasoned preparation for test automation, including in-depth studies of test requirements, setting realistic expectations and planning for automated testing.

Our contribution to the existing knowledge on the topic consists in proposing an architecture design for automated testing control system, which is based on the integration of testing tools. We focus on how to realize completely unmanned test execution.

III. EVOLUTION OF AUTOMATED SOFTWARE TESTING

"Automated software testing" is a controversial expression employed by software companies regardless of the test automation level they have achieved.

Attempts to classify the levels of maturity of automated testing are not new. For instance, Dustin et al. correlate the four levels of automated testing described by Krause to the Software Testing Maturity Model (TMM) [4][8][9].

At the initial TMM level testing is not separated from debugging. It corresponds to "accidental automation," automated testing that is nonexistent or carried out on an ad-hoc or experimental basis. Test automation is not supported by process, planning and management activities; scripts are not reusable or maintainable.

At the second, Phase Definition level, testing and debugging are separated, and "incidental automation" occurs. At this phase automated scripts are adapted, but not reusable, and there are no defined processes.

The integration phase corresponds to a level of maturity where testing no longer follows coding, but is integrated into the software life cycle. At this stage, automated testing is referred to as "intentional automation." The process is well documented and well-managed; scripts' reusability and maintainability are at the core of test design and development.

At the fourth TMM level, testing is a measured and quantified process. Defects are tracked and assigned a severity level. In automated testing, this stage is called "advanced automation" and is supplemented with post-release defect tracking. The test team is an integral part of product development, which ensures that bugs are found as early as possible.

The classification of automated testing maturity levels that we suggest below does not conflict with the TMM model. However, we focus on a different criterion, which is the number of operations that are still being performed manually during automatic test execution. In addition, we emphasize such factors as organizational needs and project length and requirements.

In this section, we will define three stages of testing automation evolution as we view it and provide their principal characteristics (see Fig. 1).

A. *Infancy Stage*

This phase is marked by the emergence of scripts and automated tests. The scripts usually perform frequent, routine functions necessary to prepare the product for testing, e.g., the copying of product installation and configuration files to the testing PC and basic system setup. The scripts can also be used to verify particular product functionality. Along with the scripts, the automated tests created with special test automation tools (e.g., Visual Studio, HP QTP) appear in the testing process of an organization.

The main characteristics of this stage are:

- Lack of arranged test storage (generally, the tests are stored on the tester's PC and used solely by him or her, i.e., they are not reusable or adapted to any changes of tested interfaces).
- Need for systemized test launch.

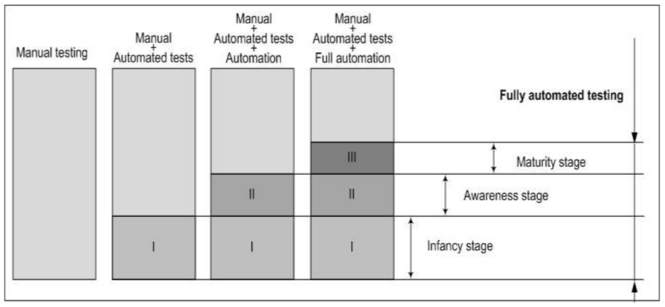


Figure 1. Stages of testing evolution.

- Shortage of documented procedures and common practices for interpreting test results and creating reports.

The infancy stage lacks a systematic approach to the integration of AT into the software quality assurance process.

At this point the tests are often unstable and their results cannot be reliable. However, they may free up a certain amount of resources by helping testing specialists fulfill the most routine tasks.

Typically, organizations that dwell at this phase have short-term projects, and thus lack the opportunity to upgrade to a higher level of test automation. These include small companies that have no testing process as such, as well as firms that are just beginning to use automated tests.

B. *Awareness Stage*

In this phase the majority of activities are automated using, for example, batch files: launching tests, starting and shutting down VMs, copying the necessary configuration files to the testing PC and other operations.

The following features are typical of the second stage:

- Improvement of test quality.
- Arrangement of centralized storage for tests and libraries of functions (tests become reusable).
- Tests are launched automatically upon the issue of each new product build.
- Naming rules for automated tests take effect.
- Guidelines for processing test results (submitting and closing bugs) are elaborated.

At this stage, which is the most widespread among companies, we may trace particular signs of automated testing. A typical company representing this phase is a developer of middle- and long-term software projects, which has well-established testing processes and realizes the need for regular regression testing.

C. *Maturity Stage*

This is the most advanced level of automated software testing, where it is seamlessly integrated into the company's testing processes. As Mark Fewster and Dorothy Graham put it, "A mature test automation regime will allow testing at the "touch of a button" with tests run overnight when machines would otherwise be idle [5]."

We characterize this stage as "full testing automation". By that we mean that **all** the operations related to test execution are done automatically, without the participation of a test engineer. These include:

- Starting and shutting down Virtual Machines (VMs) in cases using virtualization during testing.
- Configuring the testing environment.
- Queuing builds for testing according to their priority.
- Execution of tests upon successful build compilation.
- Submission of defects to the Bug Tracking System (BTS).
- Closing fixed bugs in the BTS (optional).
- Generation of a unified report on all passed tests.

Obviously, all of these elements should be automated to the extent that it is cost and time efficient [1]. Generally, such an advanced level of automation is attained by companies developing complex software products with extensive functionality. They are engaged in middle- and long-term projects and have to meet the challenges of missing or incomplete regression testing, and the effort of achieving the advanced level is worthwhile for them.

As we proceed, we will assume that introducing automated tests is a decided matter, its economical feasibility is proven, and a company's goal is to achieve the maturity stage where tests are executed automatically, i.e., without the interference of a test engineer. Many publications discuss the criteria according to which tests should be automated. They also provide techniques for evaluating return on investment of test automation [6][7]. These particular topics are beyond the scope of this paper.

This paper focuses on achieving the advanced level of automation by means of a special automated testing control solution. Below, we describe the main challenges of developing an ATCS and propose the software architecture of such a system.

Along with the above-listed functionality, the AT control solution provides a common User Interface (UI) that enables the user to fully parameterize test execution and customize all related tools (virtualization server, BTS, automated scripts) according to the project requirements.

At the maturity stage, the experience of earlier attempts at testing automation is taken into account, and special attention is paid to the scalability and expandable architecture of the ATCS itself.

IV. MAIN CHALLENGES OF BUILDING AN AUTOMATED TESTING CONTROL SOLUTION

In order to remove manual activities from automated test execution, test engineers have a choice: whether to develop special configuration and command files, or attempt to create a special AT control program with a front-end interface that would send relevant instructions to the testing tools [5]. We focus on the latter, as this approach is more thoughtful and sustainable.

This section will cover the most common challenges that software companies are contending with while building automated testing solutions. These statements are based on our own professional experience, the experience of our colleagues and the results of our industry research.

With the view of studying certain problems of automated test execution, we conducted two online surveys. The first survey took place from May 10 to May 30, 2011 among the Russian-speaking IT community from all over the world. The link to the survey was placed at one of the most popular IT-specialized resource sites, Habrahabr.ru. The questionnaire consisted of 10 questions and gathered answers from 292 respondents [2]. The second survey was run among members of testing related groups on professional networking site LinkedIn.com from May 25, 2011 to June 4, 2011, and received 34 responses [3]. It was comprised of the same questions as the first survey and included an additional question (see Section IV-C below).

The objective of our surveys was to show that despite the use of automated tests there are manual routine operations a tester typically performs to have them executed.

A. *Incomplete Automation*

"When you start implementing automated tests, you will find that you are running the (supposedly automated) tests manually. Automating some part of test execution does not immediately give automatic testing [5]."

To assess the level of testing automation in their organizations, we asked a question in our survey: "During the automated testing, what are the operations that you still have to perform manually?" This particular question received 247 answers, and 45 respondents skipped it [2].

According to the survey, only 12.6% of respondents claim that in their organization all the operations related to AT are executed automatically, i.e., without the interference of an operator. As illustrated in Fig. 2, the most widespread tasks that a tester has to perform manually are submitting and closing bugs in the BTS (60.3% and 55.1%, respectively), launching automated tests (52.2%) and creating reports (44.5%). As the question allowed multiple answers, the total percentage exceeds 100 % [2].

On the one hand, these operations are monotonous and have a lower added value than, for example, the creation of new test cases – an alternative to investing the tester's time. On the other hand, they are time-consuming. For instance, in the case of data-driven testing, where the value of each particular output is important, a new bug must be submitted each time the test criteria are violated. On average, an experienced tester submits a defect into the bug-tracking system, including completing the assigned fields, in slightly more than a minute, and closing a fixed bug takes about 15 seconds[1]. Multiplied by the number of defects the tester has to process, the amount of wasted time may be considerable.

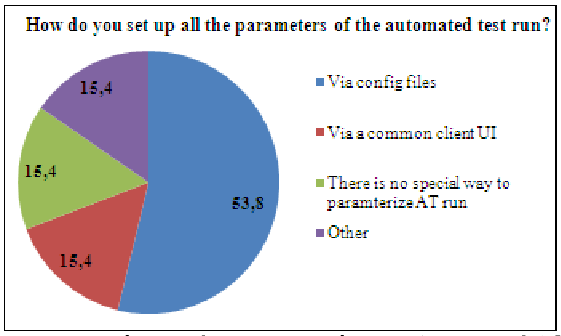


Figure 3. Configuring the parameters of AT run (survey results) [3].

[1] Measurements were done with the following properties:
1. Bug Tracking System (BTS): Microsoft Team Foundation Server (MS TFS), Mantis, Bugzilla.
2. Experimenters: 2 Testers (both 4 years of experience).
During the experiment 10 bugs were created with the following required fields:
MS TFS: Title, AssignTo, Iteration, Area, Tester, FoundIn, Severity.
Mantis: Category, Summary, Description, Platform, OS, Severity.
Bugzilla: Component, Version, Summary, Description, Severity, Assignee.
Opening BTS is also measured.

Another example is the set-up of the testing environment, which is carried out manually by 25.5% of our respondents [2]. The tester has to place a specific file into a specific directory before the automated test run. These actions are time-consuming, difficult to document and can be easily missed, resulting in flawed test results [5].

B. *System Scalability and Expandability*

In our interviews with peers, we found that oftentimes when a company develops a system for controlling automated testing, it focuses on the tools currently used without providing for system expandability. As a matter of fact, the solution being built for specific tools has important shortcomings. For instance, when the organization upgrades to a new version of the bug tracking system, or wants to add virtualization servers to the test lab, or introduces new types of automated tests created using a different framework, system integration and customization efforts will have a significant cost.

The outcome is the same when the crucial factor of system scalability is not taken into account. As product functionality increases over time, the number of automated tests increases as well, and there is a need for rational distribution of virtualization resources. The extension of the virtualization capabilities results in the rise of efforts to maintain the test automation system, and to manage a number of additional elements.

Therefore, such features as scalability and expandability have to be realized in the testing control solution's architecture in order to maximize its performance through the software life cycle.

C. *Absence of an Easy-to-use Control Tool (User Interface)*

In the majority of cases there is no single client interface for control and adjustment of the AT process, which negatively affects the overall performance. The settings of test execution are parameterized by means of config files (see Fig. 3) [3]. More often than not, the code of the configuration files is not subject to validation, resulting in an increase in human errors.

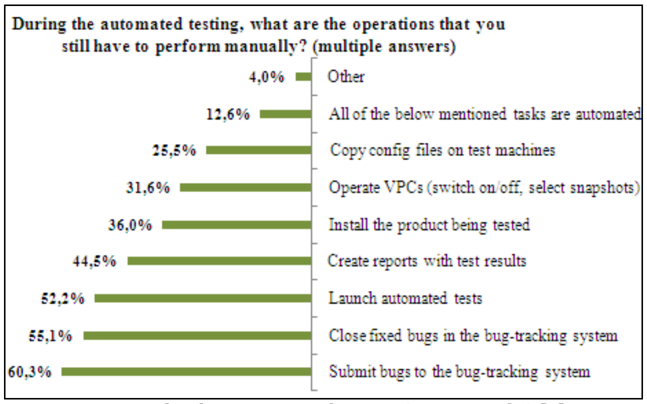


Figure 2. Manual tasks in automated testing (survey results) [2].

It is recommended to develop a front-end providing a “user interface that is independent of the automation tool used [5]. A common UI that enables the set-up of the automated test run without writing a single line of code augments the efficiency of software quality assurance. It helps reduce the learning curve, as the test engineer works with the single UI instead of interacting with several tools.

D. *Lack of Uniform Cumulative Reports*

In general, each test automation framework, such as HP QTP or IBM Rational Robot, generates reports in its own native format. In our practice we experienced firsthand the situation when a stakeholder (manager, customer) was not able to view test results because the corresponding tool was not installed on his/her computer. Even if any of these frameworks possesses an export feature, they have to handle a stack of separate lengthy records.

A testing automation solution should provide for uniform cumulative reports, meaning that a single summary report is based on the results of a batch of test cases and is presented in a structured and easily readable form. Furthermore, up-to-date results should be available and accessible at any moments of test execution, and the storage of reports history should be enabled.

E. *Uninterrupted Operation*

An automated testing process resembles a conveyor belt. At the entry point, we have new builds of the system under test, and at the output, found defects and reports. To ensure the continuity of operation and system stability, it is necessary to develop mechanisms preventing system hang-up. For instance, if a test has an error, it will be run endlessly, keeping a virtual machine’s resources busy and preventing the execution of other queued tests. Therefore, it is useful to implement the “timeout kill” algorithm to ensure the system’s fault tolerance.

F. *Insufficient or Lacking System log*

Deficient system logging hampers the debugging process, which makes an ATCS non-transparent and its activities hardly traceable. Therefore, when developing a system for automated testing control, it is crucial to enable the logging of all system components, including the events of automated tests, virtual machines, defect management system, reports, etc. These measures help minimize the time needed for debugging and increase the efficiency of software quality assurance and validation.

V. Architecture Proposal of Automated Testing Control Solution

In this section, we describe an efficient approach to bring automated test execution to the highest maturity level. We present a working archetype of the software system that controls automated tests and is independent of testing tools used. The proposed solution eliminates routine manual operations from the test execution process.

A. *Integration of Testing Tools*

The approach we recommend consists of building a coherent and comprehensive software solution which independently controls all operations related to AT – from launching tests and operating virtual machines to submitting bugs and generating reports. The solution, as Fig. 4 suggests, is based on the integration of software tools engaged in AT, namely the file server, the build machine, the versioning control system, the virtualization server, the bug tracking system, and automated tests themselves.

In order to develop such an integrator, first one needs to analyze the tester’s interaction with all the above-mentioned tools. The second step is to examine the APIs (Application Programming Interfaces) of each tool. The final stage is the development of a solution that integrates all these software tools under a common UI, via which the tester can easily and quickly adjust the automated testing control system according to the requirements and processes established in the organization.

In other words, instead of customizing and configuring each tool separately (virtualization server, BTS, automated tests, etc.), the tester will be able to adjust all settings via a single easy-to-use UI.

The prototype of the described automated testing control system was developed and successfully deployed by Applied Systems Ltd. The program architecture consists of three modules:

1) *Automated Test Manager (ATManager)*

ATManager is a complex service that controls the whole AT process and assures communication among all elements in the system. It plays a central part in the functioning of the test automation solution and works using the algorithms described below.

When a new project is created, the tester (operator) presets the ATCS for verifying a specific build branch: adds tests into the system, groups them into test runs, etc. Once this is done, ATManager takes over and probes every new build in automatic mode.

1. ATManager monitors the state of the build machine via its API. If the new build is completed successfully, ATManager is notified and starts the testing procedure. Each build can have several test runs configured to verify it. Different test runs can be executed simultaneously on different machines.
2. ATManager finds an appropriate test machine (VM or physical PC). Each test run has a set of virtual machines on which they can be executed.

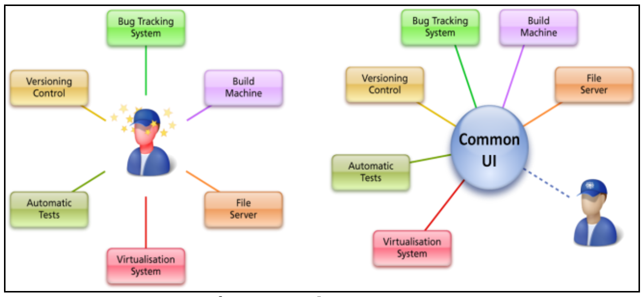


Figure 4. Integration of testing tools.

3. ATManager starts a VM (in the case of using virtualization during testing) via the API of the corresponding virtualization server. It chooses appropriate machines from the least busy virtualization server.
4. ATLauncher is initialized.
5. ATManager deploys build binaries and automated tests from the file server or version control system on the VM. Then it configures the environment on the test machine. ATLauncher launches tests.

…*Automated tests are executed*…

6. ATManager sends test results to the File Server.
7. ATManager submits/updates defects to the BTS, closes fixed bugs if these actions were allowed by the tester. ATManager service fills in the required fields in the BTS (e.g., Title, Tester, Product, Assigned to, etc.) using its API (see Fig. 5 for illustration).

2) Agent for Launching Automated Tests (ATLauncher)

ATLauncher is a console application installed on each testing machine. It is a small service that does not impact the performance of the host system.

The main functions of the module are:

- presetting the testing environment (i.e., copy configuration files, install the software under test);
- launching various types of tests (using the APIs of frameworks in which they were developed);
- processing and converting the test results, etc.

The module architecture must be expandable and allow the addition of new features.

ATLauncher starts working after ATManager has started a VM (in the case of using virtualization) and copied all required files, including automated test scripts and config files. The XML file created by the control module ATManager contains the description of tests and usage instructions. As soon as the tests are finished, ATLauncher creates a special results file to notify ATManager about the completion of its task.

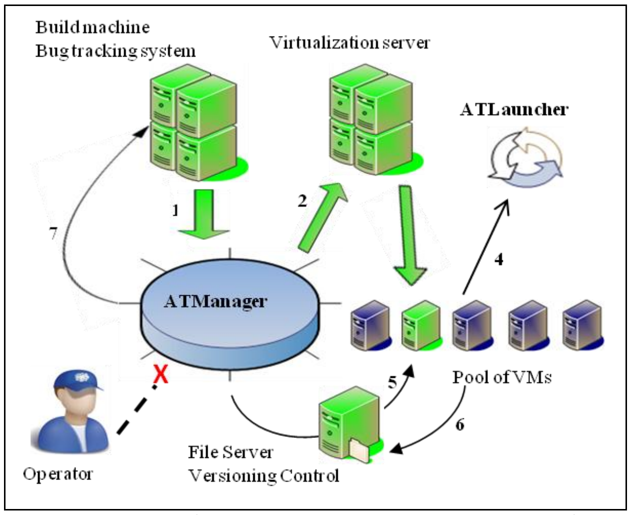


Figure 5. Scheme of communication between the components of the ATCS.

3) Control Panel

The user interface is represented by the control panel. It is a client application which facilitates the interaction between the tester and the ATCS, providing the tools necessary to configure and manage the test run for an application under test.

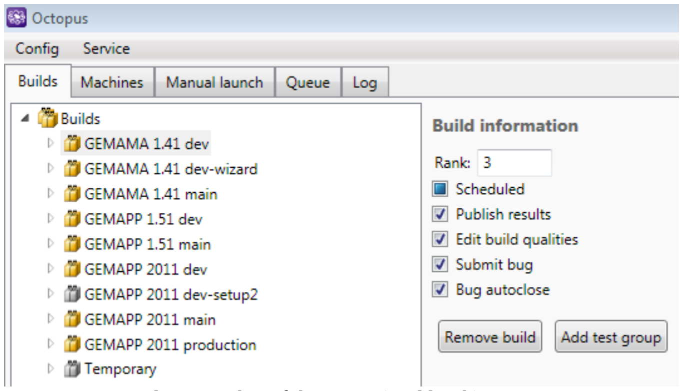
The UI allows the user to:

1. Specify the tests to be run on each build, assign priority to the build branch, schedule test launch on event (issue of a new build) or on schedule; choose defect tracking options (Fig. 6).
2. Allocate the sets of valid machines for each test run, assign tests for execution on a particular real or VMs and their snapshots.
3. Manually launch tests on a specific build, interrupt test execution.
4. View ATManager's logs.
5. Monitor the testing queue in real-time (Fig. 7).

In the time of ever-increasing mobility, it is useful to provide access to the control panel from the desktop as well as a web interface.

B. Distribution of Functionality Among Components

While creating a solution to control an automated testing cycle it is necessary to distribute the functionality of your future system among its components.



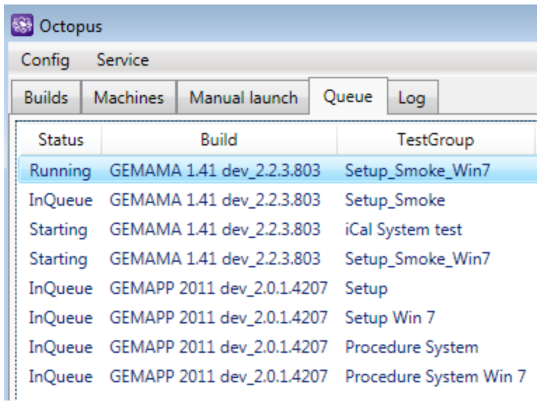
Builds
- GEMAMA 1.41 dev
- GEMAMA 1.41 dev-wizard
- GEMAMA 1.41 main
- GEMAPP 1.51 dev
- GEMAPP 1.51 main
- GEMAPP 2011 dev
- GEMAPP 2011 dev-setup2
- GEMAPP 2011 main
- GEMAPP 2011 production
- Temporary

Build information

Rank: 3

☐ Scheduled
✓ Publish results
✓ Edit build qualities
✓ Submit bug
✓ Bug autoclose

Figure 7. Sample screenshot of the ATCS(Builds tab).



Status	Build	TestGroup
Running	GEMAMA 1.41 dev_2.2.3.803	Setup_Smoke_Win7
InQueue	GEMAMA 1.41 dev_2.2.3.803	Setup_Smoke
Starting	GEMAMA 1.41 dev_2.2.3.803	iCal System test
Starting	GEMAMA 1.41 dev_2.2.3.803	Setup_Smoke_Win7
InQueue	GEMAPP 2011 dev_2.0.1.4207	Setup
InQueue	GEMAPP 2011 dev_2.0.1.4207	Setup Win 7
InQueue	GEMAPP 2011 dev_2.0.1.4207	Procedure System
InQueue	GEMAPP 2011 dev_2.0.1.4207	Procedure System Win 7

Figure 7. Sample screenshot of the ATCS(Queue tab).

In Table 1, we suggest possible options to distribute basic functionality among ATM, ATL and the client UI. To coordinate the work of all these components, one needs to develop a large set of algorithms and solutions.

VI. Conclusion and Future Work

One of the most serious problems facing software development companies today is the lack of resources for regular and comprehensive regression testing.

The most obvious and popular solution is implementing automated testing. However, despite the abundance of tools for testing automation, this endeavor presents many challenges, especially in the case of testing desktop applications. In the first place, the word "automated" does not mean, as one might be led to believe, that operations are handled without human interaction. In fact, in the process of automated test execution – *Configure the testing environment → Start the virtual test machine → Launch tests → Execute tests → Submit bugs to the bug-tracking system → Generate test reports* – only a few operations, besides the test execution itself, are automated. In addition to the fact that "non-automated" activities are time-consuming and inefficient, they also leave room for human error.

To meet these challenges, some companies developing complex software are trying to create a solution that would control the whole AT process from A to Z without the participation of an operator. Only 12.6% succeed [2].

TABLE 1. DISTRIBUTION OF BASIC FUNCTIONALITY AMONG COMPONENTS OF ATCS

Functionality	**Module**		***Control via common UI***
	AT-Manager	***AT-Launcher***	
1. Operate virtual machines (VM) - start/shut down VM - add VM and snapshots to the system - group VMs	+	-	+
2. Launch automated tests	-	+	+
3. Submit/close bugs in the BTS	+	-	+
4. Generate a uniform cumulative report	+	-	+
5. Convert results into a single easy-to-interpret format	+	-	-
6. Copy tests, config files, product setup files to the testing machine	+	-	+
7. Install the tested product	-	+	+
8. Log all system events	+	+	-
9. Abort text execution	+	+	+

In this paper, we have described an efficient and innovative approach to automating test execution based on the integration of all testing tools under a common UI. We also provided practical advice on how to develop such an AT control solution, proposed the system architecture, defined the key functionality of its components and schematized the communication among them.

In the future, we plan to assess the costs and benefits of implementing an AT control solution into a company's QA management system.

Acknowledgment

We would like to express gratitude to Ivan Bachtin, who did a wonderful job developing the prototype of the automated testing control system we refer to in this paper, and to Sergei Usovich, who provided financial support and shared our enthusiasm. We also thank our friends and colleagues who shared their experience and insights and contributed to this work, namely Maks Shatokhin, Alexander Abramov, and Dmitry Romanovich.

In fact, a few hundred people contributed to the progress of this paper in different ways, and we are especially thankful to the respondents to our online surveys, which provided a solid support to our statements and arguments.

References

[1] D. Mosley, B. Posey, "Just Enough Test Automation," Prentice Hall, 2002, pp. 12-14.

[2] Online survey "Problems of automated desktop software testing" by Applied Systems Ltd. via Habrahabr.ru, https://www.surveymonkey.com/s/automated_testing_problems 30.05.2011.

[3] Online survey "Challenges of automated software testing" by Applied Systems Ltd. via LinkedIn.com, https://www.surveymonkey.com/s/automated_testing, 30.05.2011.

[4] E. Dustin, J. Rashka, and J. Paul, "Automated Software Testing: Introduction, Management, and Performance," Addison-Wesley Professional, 1999, pp. 38- 45.

[5] M. Fewster, D. Graham, "Software Test Automation: Effective use of test execution tools," Addison-Wesley Professional, 1999, pages: 3, 62, 246, 329.

[6] E. Dustin, T. Garrett, "Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality," Addison-Wesley Professional, 2009, pp. 192-204.

[7] T. Garrett, "Useful Automated Software Testing Metrics," http://idtus.com/img/UsefulAutomatedTestingMetrics.pdf, 21.07.2011.

[8] M. Krause, "A Maturity Model for Automated Software Testing," Medical Device and Diagnostic Industry Magazine, December 1994.

[9] I. Burnstein, T. Suwanassart, and C. Carlson, "The Development of a Testing Maturity Model," Proc. Ninth International Quality Week Conference, San Francisco: The Software Research Institute, 1996.