A Parallel Execution Approach for Efficient Regression Testing in the OutSystems Test Infrastructure

Filipe Miguel Meireles Fonseca
Instituto Superior Técnico
filipe.fonseca@ist.utl.pt
November 2009

Abstract
As software complexity grows so does the size of test suites and scripts. When that occurs, problems emerge for developers in the form of increased effort to manage the test process and longer execution time of test suites. The last is especially problematic as the recurring costs of performing the same tests on a regular basis (like regression testing) throughout the product’s lifetime may become unbearable. Being an integral part of software development, testing needs to evolve and break free from the conventional methods.

This project presents a strategy which relies on parallel execution of test suites, allocating test cases across multiple processes on a single or on distributed machines to achieve a scalable solution capable of higher performance and increased reliability.

But distributed testing urges to deploy and maintain replicated computing resources for every system configuration supported by the products. Manual administration of such operations is typically time-consuming and error-prone. For this reason it was also considered a mechanism of automated systems provisioning to create, configure, and integrate new testing machines seamlessly into the system with minimal human intervention, allowing for repeatability and scalability.

The work is framed in the existing test infrastructure of OutSystems.

Keywords: regression testing, parallelization, systems provisioning, virtualization.

1 Introduction
To test means to compare an actual result to a standard [3]. In software development, to verify that no defects exist in a program it must be tested with all possible input data—both valid and invalid. But because the complexity of software is generally intractable, complete testing is simply unfeasible: the domain of program input is too large, there are too many possible input paths, and many of the design and specification issues are difficult to test [4].

A further difficulty has to do with the dynamic nature of programs. After a software module experiences changes, regression testing is recommended to ensure that functionalities previously working correctly still behave as expected. This is usually performed by running some or all of the test cases created for previous versions. While the positive influence on software quality is clear, the expenses of doing regression testing are often prohibitive [11].
1.1 Goal

OutSystems\(^1\) offers a fully integrated platform for the delivery and management of web business applications, using agile methodologies. The development of such tool demands a fine-tuned continuous software testing process.

There is already an implemented and well-established test infrastructure. Yet, it's consensual among R&D developers that the major drawback is related to the poor test execution performance. The most expensive task is regression testing which is performed on a daily basis when code is committed to promote confidence that changes are correct and haven't adversely affected other portions of the program [6]. Using the retest-all approach for such tests leaves the average duration of a run on a single machine at eighteen hours.

The main target of this project is to propose and implement solutions to significantly decrease execution time of regression test suites in the existing test infrastructure of OutSystems.

1.2 Requirements

In order to solve the issues stated above and achieve the goal of this project, the solution(s) must respect the following requirements:

1. **Take into account the growth in the amount of tests.** Over time, the increasing size of test suites is unavoidable and therefore a scalable solution capable of maintaining the performance under this condition is crucial.

2. **Should be economic.** Take advantage of the available resources by neither acquiring extra hardware nor expensive commercial software;

3. **Transparent to users.** Any complexity added to the test process must be hidden from the end-user whenever possible.

4. **Extend the existing test infrastructure.** It would be out of scope to fully redesign the rooted infrastructure and related processes; for that reason, all work must augment and enhance the current systems and not rebuild them.

5. **Developed using OutSystems platform.** The whole test infrastructure is already powered by OutSystems flagship product, and it's utterly desirable to keep it that way.

2 Related Work

Several methods exist to reduce the cost of regression testing.

**Test selection** assumes that running all test cases is wasteful and picks a representative subset of the whole test suite [6, 11]. While this certainly reduces execution time, it does so at the cost of also reducing fault detection effectiveness.

**Test prioritization** techniques schedule test cases in an execution order according to some criterion. The purpose of this prioritization is to increase the probability of the suite to detect faults at earlier testing stages [7, 2, 11]. Obviously, in the end the execution time will remain the same.

\(^1\)http://www.outsystems.com
Test distribution aims to reduce time costs by operating on the principle that large workloads can be divided into smaller ones, which are then executed concurrently over a set of machines [5, 1, 9]. The cost of allocating and managing additional computational resources is the biggest downside.

Not being mutually incompatible, the perfect solution would be to combine the three approaches to take advantage of each one’s benefits. But because that would require an effort beyond the available time for this project, the focus will be on test distribution as it is the only that alone conforms with all the requirements stated in Section 1.2.

3 Architecture and Implementation

Before presenting the solution developed to tackle regression testing efficiency problems, this section first overviews both hardware and test infrastructures in place at OutSystems prior to the start of this project.

3.1 Hardware infrastructure

Like most companies, OutSystems owns and maintains a modest data center used for internal and external operations. Despite currently having only three servers, virtualization [10] is the key to run dozens of machines simultaneously.

To understand to what extent the available servers would support additional load, the hardware specifications and usage statistics are summarized in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>SRV001</th>
<th>SRV002</th>
<th>SRV003</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>8 × 2327GHz</td>
<td>8 × 2327GHz</td>
<td>8 × 2327GHz</td>
</tr>
<tr>
<td>RAM Memory</td>
<td>32GB</td>
<td>32GB</td>
<td>32GB</td>
</tr>
<tr>
<td>Total Disk Space</td>
<td>2410GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R&amp;D Disk Space</td>
<td>900GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virtual Machines</td>
<td>28</td>
<td>43</td>
<td>54</td>
</tr>
<tr>
<td>Average CPU</td>
<td>22%</td>
<td>25%</td>
<td>32%</td>
</tr>
<tr>
<td>Average Memory Usage</td>
<td>61%</td>
<td>61%</td>
<td>59%</td>
</tr>
<tr>
<td>Free Disk Space</td>
<td>252GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Free R&amp;D Disk Space</td>
<td>154GB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As is patent by these numbers there’s a surplus of hardware capacity which can be harvested to improve testing efficiency. Notwithstanding this evidence, overloading the servers may lead to non-linear service degradation, meaning that smart resources management is desirable, particularly for memory usage and disk space.

3.2 Former test infrastructure

To answer the specific testing challenges of its all-in-one agile development platform, the R&D team at OutSystems built (some years ago) a custom solution powered by its own technology. The never-ending improvements resulted in a complex system that works closely with several other services to handle nearly all aspects of the testing process. The infrastructure is schematized in Figure 1 and comprises the following major components:
Figure 1: The high-level test infrastructure

- **Dashboard**, a central application for storage and management of tests, instances, and test results.
- **Instance**, an abstract concept which represents a certain system configuration for testing a certain product branch.
- **Regression**, an application installed on a dedicated (virtual) machine to fetch and execute a set of general automated tests associated with an instance.
- **Build Server**, a dedicated machine to perform software builds from the platform source code.
- **SVN**, a version control system where all source code resides.

For long this implementation fitted the R&D team requirements, but the continuous growth of test suites size led to unbearable execution times.

### 3.3 Newly developed test infrastructure

A new architecture was developed which relies on parallel execution of test suites, allocating test cases across multiple processes on a single or on distributed machines, to achieve a scalable solution capable of higher performance and increased reliability. The implemented solution is depicted in Figure 2 and includes three significant enhancements, described in detail next.

#### 3.3.1 Multiple Regressions

The surplus hardware capacity together with the limitation of a single Regression per Instance confirmed the necessity of refactoring the former infrastructure to enable multiple distributed Regressions to process test cases simultaneously for a same Instance.
The work focused on making widespread adjustments to the data model and subroutines logic of the Dashboard application in order to introduce the Regression concept, formerly masked by the Instance due to their one to one mapping.

### 3.3.2 Multithreading

Although the use of multiple Regressions undeniably represents a step forward to make the most of the available infrastructure, it was found that each individual Regression could not avail all computational power of its underlying machine. This provided a leeway to reduce testing duration using computational power that would otherwise be wasted.

The original Regression application had just a single thread of execution: sequentially running test cases until the test suite terminates. Since the testing process is predominantly I/O bound, a thread spends most of its time waiting for network, file or database I/O and not actually performing useful computation. In view of this evidence, a second solution was implemented for Regressions to support multiple testing threads. Within each thread, test cases are run sequentially, but the threads themselves may be executed in parallel.

The challenge on this matter was the implementation of thread exclusion, as the OutSystems platform does not provide any “classic” exclusion primitives built-in. The solution found was via database abstractions, which manage concurrency through such mechanisms as transactions. A transaction is a collection of operations that form a single logical unit of work and have the following properties: atomicity, consistency, isolation and durability (ACID). Taken as a whole, the transaction-management component of a database system ensures the integrity of the data, concealing the issues of concurrency and fault tolerance [8].

### 3.3.3 Regression Factory application

Upon overcoming the challenge of parallel tests execution, it’s tempting to assume the job is done. But all things considered, we must not forget that a distributed
testing environment calls for administration of replicated computing resources for every system configuration supported by the products. For this reason it was also considered a mechanism of automated systems provisioning to create, configure, and integrate new testing machines seamlessly into the system with minimal human intervention, allowing for repeatability and scalability.

Unluckily, the feasibility of the idea ran into hurdles right upon the start. Turns out that the bootstrap of a fresh, autonomous OutSystems-ready machine is bound to some manual configuration steps, being the most problematic a license approval from the support team and a database clone from the help-desk team, both puzzling to workaround due to technical and political aspects. The practical alternative is to make use of the OutSystems farm setup feature in which additional machines (named Front-End’s) work subordinate to a master server (named Controller), sharing its license and database. The major downside is that the Front-End’s depend on the Controller to perform some heavy CPU tasks, thus ceiling the achievable performance gains. Figures 3 and 4 clarify the distinctions between the two setup possibilities.

![Figure 3: Autonomous setup architecture on OutSystems platform](image)

![Figure 4: Farm setup architecture on OutSystems platform](image)

Accepted this shortcoming and bearing in mind the available options in this domain it was considered more viable to build a new application from scratch. Architecturally it comprises two separate components:

**Virtualization module** to enable programatic interaction with the existing virtualization platform, providing several operations convenient to the provisioning process.

**Provisioning module** to define the complete provisioning workflow, based upon the operations made available by the virtualization module. In essence, its functionality boils down to the provision and discard actions; the former manages the process of automatically introducing new *Regression* machines and the latter reverts that very same process.
The application key features are:

- Strikes a balance in the provisioning process by leveraging both configuration automation and cloning. An existing standard virtual machine is first cloned to establish a minimal OS install and then all necessary applications and configurations are layered on top to make the machine fully functional.

- Uses a pool of initialized virtual machines to shorten the provisioning times. Because cloning a virtual machine is a rather long operation (taking about 15 minutes to clone the current standard template).

- Automatically upgrades/downgrades the hardware settings of Controller machines, to better handle the additional load caused by the farm setup and multithreading.

4 Results

The premise of this project stemmed from a realization that longer execution time of regression test suites is a growing bottleneck in software development. The solutions implemented aimed to fill such gap in the context of the existing OutSystems test infrastructure. This section presents the results of the performed work and corresponding evaluation.

4.1 Quantitative Analysis

Figure 5 presents the results collected with multiple Regressions in a autonomous setup (each machine has a different database and Controller). The introduction of two additional Regressions reduced the total test duration to 6:48h which represents a 62% improvement.

![Figure 5: Test time with multiple Regressions in a autonomous setup](image)

Figure 6 presents the results collected with multiple Regressions in a farm setup (all machines share the same database and Controller). The results are less satisfactory since the Controller is an inevitable constriction point. The introduction of
two additional Regressions reduced the total test duration to 11:14h which represents a 36% improvement.

Figure 6: Test time with multiple Regressions in a farm setup

Figure 7 presents the results collected with multiple threads in a single Regression. As expected, the improvements are behind of those achievable with multiple machines and there’s a maximum threshold from which the performance starts to decline; in this particular scenario the optimal value seems to be four threads. Even so, any speedup is welcome since the costs are practically zero—using computational power that would otherwise be wasted.

Figure 7: Test time of multiple threads in a single Regression.

Having seen the individual figures of each solution is somewhat easy to foresee the outcome of combining them. The overall top results recorded for each setup were:
• A duration of 4:51h with three autonomous Regressions running 4 threads each, which represents a 73% improvement.

• A duration of 9:19h with two farm Regressions running 4 threads each, which represents a 47% improvement.

4.2 Qualitative Analysis

Overall, when assessing the final project outcome up against each requirement listed in Section 1.2, the balance is quite positive as all the items have been successfully accomplished. The choice of parallelizing the execution of test suites was manifestly correct as the obtained improvements would hardly be matched from test selection and/or prioritization methodologies.

As one would expect, there are also negative aspects. Together with the speedup, parallelization strategies brought along an undesirable increase of failed test cases (a test case fails when the output is different from expected or when an unpredicted exception is raised). A brief inspection of the situation revealed it happens mostly due to:

• Misconfiguration of test cases, presenting specific machine dependencies, that eventually fail if not run on the original machine.

• Heavyweight test cases that, when executed in parallel with others, usually end up raising a timeout exception, after exceeding the time limits established by the OutSystems platform. Two solutions exist to overcome this issue, the test cases can be split, making them more lightweight, or tagged as critical, to have exclusive access to the execution environment.

This is a nuisance for testers who need to track and correct the problematic tests cases over time, fortunately only accounting for 3.3% of the entire test suite.

Regarding the Regression Factory application, there are no actual setbacks, but the limitation to provisioning only Regressions in a farm setup is a little disconcerting as it stands in the way to achieve a truly automatically scalable testing environment.

It remains only to mention that all the project developments have been deployed in the production environment for a while now and no major issues were reported.

5 Conclusion

Advances in technology and development tools have enabled the creation of complex software but also hampered testing: long execution time of test suites top the recurrent problems. The former OutSystems test infrastructure was patently afflicted by this situation since its native design was not performance oriented.

Because testing is an integral part of software development, this project focused on solving its efficiency problems, describing the conceptual foundation and architecture for a solution that parallelizes the execution of regression test suites. Such strategy aims to reduce time costs by operating on the principle that large workloads can be divided into smaller ones, which are then solved concurrently.

The key feature of the proposed solution is the ability to create, configure and launch new test machines on demand, over a virtualized infrastructure, responding dynamically to meet test suites execution deadlines.

This approach is believed to have responded to OutSystems requirements, improving the development process and therefore the product offered.
References


