The Future of Testing

Improving Levels of Automation and Test Coverage in Telecommunications

QiTASC
the magic of testing

Authors:
Leora Courtney-Wolfman
Michael Zehender

Contacts:
leora.courtney@qitasc.com
michael.zehender@qitasc.com

Place, Date:
Vienna/Austria, March 2019
FN 269602z

Abstract

Automated testing is an increasingly dominant topic in telecommunications and QA. Despite test automation's increased prominence, it remains heavily underutilized across all industries. In terms of telecommunications, this can be partly explained by the unique set of challenges involved in testing its infrastructure, which relate to the scale, scope and fragmentation of the industry. Furthermore, a lack of comprehensive research, terminology and technical literature about test automation in telecommunications exacerbates the difficulties in defining and achieving best practices.

In order to bridge the gap between corporate testing needs and potential automation solutions, this white paper identifies five challenges that act as barriers to automated testing in telecommunications:

1. Different test tools for different testing phases
2. Devices, models and OS versions
3. Complicated software with a steep learning curve
4. Proper test coverage, which refers to how much functionality is actually tested
5. Level of automation, which refers to the amount of test-related activities that are automated.

The white paper then discusses potential solutions to these challenges, as outlined in scholarly and technical literature. For each of the five topics, a brief discussion of intaQt, as well as QiTASC's suite of automation solutions, is included at the end of the section to illustrate how stakeholders can establish a sustainable test framework. These examples demonstrate how using a testing framework that ensures reusability across projects, an incremental approach to automation and tools that extend automation beyond text case execution itself provide excellent opportunities for improved levels of test automation and test coverage.

About QiTASC

QiTASC's „key task" as a Quality Improving Tools And Services Company is to provide you with the technology and know-how to quickly improve test coverage in your test projects and get your products to the market sooner.

Our test automation product, intaQt is at the heart of this mission, and can be complemented with our suite of test automation products to scale up and down as necessary. QiTASC also provides managed testing if you prefer to hand the testing over to our experts.

Table of Contents

Introduction

A high premium is placed on quality assurance and testing to ensure mobile networks and services are defect-free and that customers can enjoy uninterrupted, efficient service—but at what cost and complexity? A commonly-cited figure suggests that during the past ten years, testing has consumed an average of 25%-40% of telecommunications firms' budgets (Veselov & Vsevolod, 2010; Tuteja & Dubey, 2012; World Quality Report 2015-2016 & 2017-2018). Yet, as companies compete for customers in the face of shrinking profit margins and tighter deadlines, it is critical that they find ways to reduce expenditures while maintaining—and improving—the quality of their products and services. Improving testing efficiency through automation is one of the main keys to achieve goals such as finding ways to reduce expenditures while maintaining and improving the quality of their products and services.

Achieving these goals is critical for companies as they compete for customers in the face of shrinking profit margins. Furthermore, using the *same* testing tools across all testing phases is an efficient, cost-effective approach (Pinola et al. 2013). Therefore, using a testing solution that can be flexibly used in all phases—from development, to field acceptance and to live environments— and that should be given high priority when choosing test automation software.

While hundreds of test automation tools exist for mobile application testing, far less exist for testing telecommunications networks and their infrastructure. Consequently, there are few resources available to help stakeholders make informed choices about automated testing. Therefore, this white paper addresses some of those knowledge gaps. In order to handle constant change and compounding complexities, incrementally automating QA activities using a single, holistic suite of test software is an achievable goal that in the long run shifts resources away from repetitive, cost- and time-intensive manual testing activities, while decreasing time-to-market.

This white paper addresses five challenges facing test automation in the telecommunications industry: **different testing tools and phases, too many devices and operating systems, complicated software, test coverage and degree of automation.** Within the context of existing research and technical reports, the white paper presents approaches to mitigate these challenges and additionally demonstrates the role that QiTASC's test automation software plays in achieving effective test projects that produce market-ready results.

Target Audience

This white paper is directed towards telecommunications decision makers involved in choosing automated testing solutions, consultants for such decision makers and technical product managers.

The Market for Automated Testing in Telecommunications

The telecommunications industry has seen a steady adoption of increasingly sophisticated automated testing tools over the past 15 years, which can be used at various stages of the development cycle and for different purposes. Arguably scale, complexity and the costs associated with telecommunications networks pose some of the largest barriers to QA testing in this field for *both* manual and automated testing. However, as available information is limited regarding test solutions—both in terms of research materials and online documentation aimed at non-technical experts. This lack of information is compounded by inconsistent terminology used to describe testing activities and approaches, making it difficult to search for and find useful resources.

Despite the lack of conclusive information, the general findings from QA and automation indicate the trends in the industry's use of automated testing. The 2017-2018 World Quality Report (WQR), which is the most comprehensive account of figures and outlooks for QA and testing, reports a shift towards customer-driven testing, noting that test scenarios should reflect "consumer usage patterns." This testing, however, remains largely manual and the report continues that "automation is currently under-exploited in QA and testing", adding that the level of automation across all industries surveyed is only about 16 % (p.8). This percentage refers to testing activities such as test design, execution, data creation and analysis. The report urges stakeholders to invest in and improve their levels of automation, as this is the only way to manage the testing scale and coverage required within a fast-paced, global business environment.

Although concerns exist about human jobs being replaced by computers, automation appears to benefit both the organizations that adopt it and their employees. Ben-Ner and Urtasun (2010) state occupations that have been traditionally complex (as opposed to routine, low-complexity labor) are associated with a greater, more intense „adoption of computer-based technologies (CBT)" by employees. They suggest that CBT enables higher productivity by automating the basic, routine activities found in high-skilled occupations, while providing employees with more information output as well as more time and opportunities to interpret this output and expand their technical expertise. In other words, automation complements a tester's responsibilities and skills as they relate to problem-solving and technical know-how rather than substituting it.

The WQR 2017-2018 finds that a majority of their respondents state that test automation helps them better detect defects, allows for greater test case reusability and decreases the length of the test cycle. This leads to improved time-to-market, reduced QA spending, fewer defects after releases and better test coverage. To better understand the potential benefits of automated testing, the following sections present some of the key challenges that telecommunications firms face when confronted with automated testing.

Current Challenges in Test Automation

Improved levels of automation should be a key priority for QA in telecommunications, as argued for in both recent WQR reports and scientific research. Pinola et al. (2013) state that, „for software-intensive systems such as modern telecommunication equipment, software testing is required throughout the development cycle of the product." Nevertheless, there is no straightforward, „one-size-fits-all" approach to the automated testing of telecommunications infrastructure. In order to better understand how automated testing supports high-quality telecommunications infrastructure, this white paper introduces five challenges that interfere with or prevent effective testing.

These challenges are well documented in WQR reports and research articles:

1. Proper test coverage
2. Level of automation
3. Different testing tools for different testing phases
4. Complicated software with a difficult learning curve
5. Too many device models and OS versions

Challenge No. 1

Proper Test Coverage

Test coverage refers to what percent of a product's functionality is assessed via a test project. The importance of test coverage is that it is linked to reliability as well as the probability of finding errors. Unlike the previously-mentioned challenges such as device usage and software know-how, test coverage presents fundamental threats to the quality and findings of all testing activities. Galindo et al. (2016) note „it is difficult for developers, regardless of development team size or proficiency, to test their software products on all or even most platform configurations before release".

Metrics used to assess test coverage include *code coverage*, which looks at how many lines of code are covered in test cases, and *data-oriented coverage*, where databases of test input (for example, behaviors, actions) and outputs (for example, desired outcomes) determines the conditions of a test case. Additionally, *keyword-driven* testing links keywords to a testable action or function, for example, `phone call` or `download data`.

When developing test suites, **features**, or **Feature Files**, are commonly used to represent each unique test case. These features describe sets of behaviors and expected outcomes for a given product (or groups of products) and may be grouped into a hierarchical, tree-like model (Galindo et al. 2016). However, as products and their functionality increase

in scope, the potential Feature Files required to maintain a good level of test coverage appears to grow exponentially, meaning achieving optimal coverage becomes more complex.

Three types of challenges to test coverage can be identified from the literature:

1. **Practical challenges**

   Practical challenges, for example, limitations to how much can be reasonably tested given a specific amount of time and resources. The WQR 2017-2018 describes a need for high test coverage combined with time-to-market demands in the high-tech sector. This implies a need for a growing volume of tests, covering more features, in a shorter amount of time (Yoo & Harman 2010).

2. **Informational challenges**

   Informational challenges, for example, limitations to being able to define optimal test coverage based on employee or industry knowledge. Although increased test coverage is one of test automation's biggest strengths, the WQR 2017-2018 describes the „insufficient ability to *define* the right test coverage and depth" as a source of inefficient testing.

3. **Technical challenges**

   Technical challenges, e.g., technical barriers that restrict what types of testing activities, including creating test sets, can be done and/or how. Even with automated testing, it is currently impossible to define and execute tests for every possible scenario given a complex set of variables such as configurations, use cases and networks (Galindo et al. 2016).

Solutions: Test Prioritization and Test Pruning

Although automation helps avoid test scope reduction, some reduction is often required whether because of time, budget or practical limitations. Morgado and Paiva (2016) stress that „[u]sually tests can not be exhaustive and, thus, it is necessary to select which tests to perform or to select a subset of the overall behavior to test". Two methods that lead to a manageable number of test cases involve *test pruning* and *test prioritization*.

Test pruning refers to methods that reduce the size of the entire test suite by eliminating unnecessary tests, while maintaining appropriate test coverage. On the other hand, test prioritization orders tests, such that the „most valuable tests [are executed first] to ensure that critical software components are tested" (Galindo et al. 2016). Modern test pruning and test prioritization are typically based upon combinatorial algebra that tests for combinations of parameters, allowing for the further categorization/grouping of tests. Following this, further methods may be applied to certain tests, such as merging/grouping, ordering and/or discarding them.

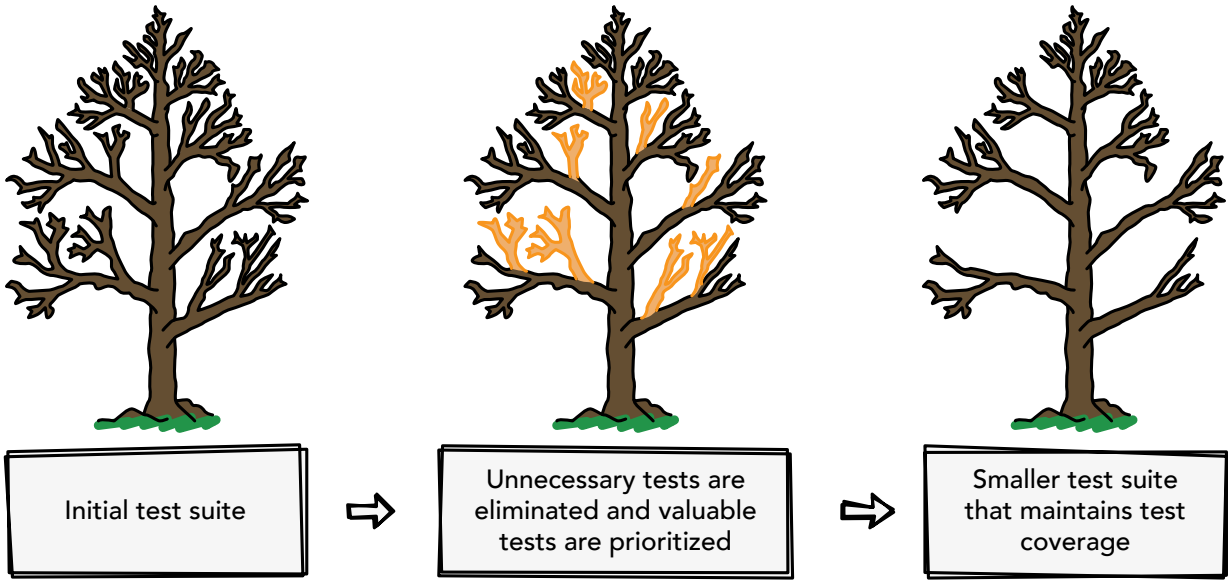


Figure 1 Test pruning

Galindo et al. (2016) describe an automated test pruning solution that defines the costs and values of a set of tests, and derives a subset of test cases that should be tested, as well as a subset of test cases that can be eliminated or „pruned"—either because the test is not relevant to the test scope, is already covered by another test or is simply too expensive or resource-intensive. Likewise, Choi et al. (2013) present a technique that „checks [the] equivalence between two model states" and prunes by „aggressively merging" tests with equivalent states together. This type of pruning can also self-correct in the event that the merged test cases no longer match. Meanwhile, Fierens et al. (2010) describe using machine learning where pruning criteria develops a probability tree, and comment on the implications of over-pruning or under-pruning. Regardless of the pruning approach, it is best achieved using automated methods because of the volume of data that must be sorted.

Regarding **test prioritization**, Srikanth et al. (2005) developed a quantitative tool that prioritizes according to four criteria, on a scale of 1 to 10:

1. First, „customer-assigned priority", which refers to the fact that a large amount of software functions are never or rarely used by the customer, and therefore should have a lower testing priority than functions that are frequently used.
2. Second, „requirement volatility", which measures „how many times a requirement has been changed during the development cycle".
3. Third, „development-perceived implementation complexity" addresses the complexity of implementing a requirement.
4. Fourth, „fault proneness" prioritizes features that are known to experience [frequent] failures. On the other hand, a single criterion may be used.

Biswas et al. (2011) describe an approach that prioritizes according to test cases with a „higher fault-detection capability". They add that this type of prioritization is useful because it enables the development team to make bug fixes sooner, which is especially valuable given the pressure for fast times-to-market as well as the uncertainties and contingencies that may arise during a test project.

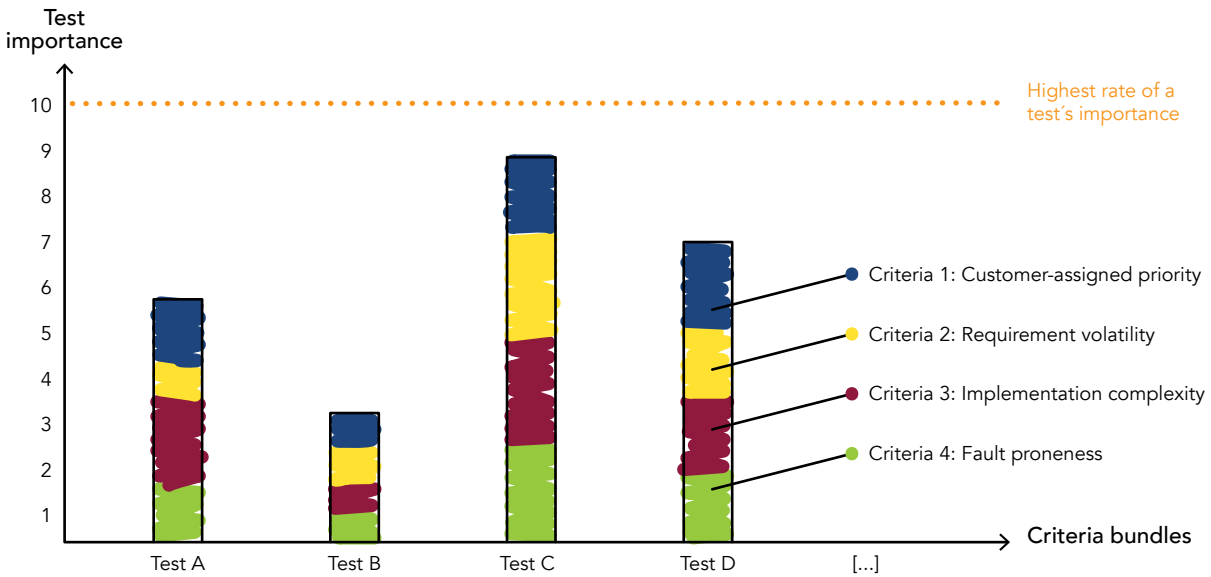


Figure 2 Test prioritization

Using intaQt to Maximize Test Coverage

intaQt contains several types of features that promote good test coverage. First, intaQt's custom languages enable creating functions and models that can be accessed by all Feature Files. Thus, a single function that contains shared functionality, such as accessing a customer database to retrieve customer information and make adjustments to the account, can be called from multiple Feature Files that also contain different parameters.

For example, the first Feature File may involve a subscriber making a phone call while roaming in a foreign country, and another file may entail the subscriber making a phone call from their home country.

Second, as described in `Different Testing Tools for Different Testing Phases`, intaQt has been developed specifically so that test cases can be reused across different testing phases. This functionality greatly reduces the need to write different Feature Files and models to account for changes to the system under test, the hardware being used and the stage of development. intaQt's combination of flexibility and reusability supports the efficient expansion of test coverage within a project. These same characteristics also play a strong role in eliminating the need for different testing tools during different testing phases and, as discussed below, also facilitate improved levels of automation.

Challenge No. 2

Level of Automation

What QA activities should be automated? Which should be reserved — at least temporarily — for manual testing? Level of automation in this white paper refers to the share of testing activities that are performed by machines instead of manually by a human. This includes test case development (for example,, writing tests cases and creating a project structure), test case execution and test case analysis. At one extreme, a „big bang approach" entails going from no automation to complete—or as high a level of automation as possible—in a single step.

Conversely, an incremental approach integrates certain features or test modules in multiple phases, typically from a bottom-up or a top-down approach, which takes a hierarchical view of the software modules or products being tested. Top-down testing involves testing at the highest level, the main or core module, and branching out until testing reaches the lowest level. In other words, covers minor feature with few interdependencies. On the other hand, a bottom-up approach tests the lowest modules first (Galin, 2004).

The WQR 2018 notes that the big bang approach can lead to disappointing results when the return on investment does not meet stakeholder expectations (2017 p. 30). One reason is that this approach makes it difficult to identify sources of failure, since everything has been integrated into the test scope at once (Galin, 2004; do Como Marchado et al. 2012). Galin (2004) adds that determining the source of errors and correcting them is an „onerous task"- and costly too. Correcting a fault, within the context of a big bang approach, requires „consideration of the possible effects of the correction on several modules at one and the same time." This poses uncertainty for budget planning and test-fix-release scheduling (ibid.).

While incremental testing promotes more efficient error detection, greater predictability and requires fewer resources to correct problems over the course of a test project, it is more costly at the outset and may require creating many more custom models and configurations, as well as spending more time preparing and structuring a project.

Solution: A Focused, Incremental Approach

An incremental approach involves gradually automating test activities until a desired level of automation is reached. This approach is almost always recommended over a big bang approach: Incremental automation enables better bug detection and correction, allows users to gradually become familiar with new processes and software and spreads risk associated with a major change in testing strategy. As the tests themselves become fully automated, other areas of a project can be automated too, including test case development, reporting and data analysis. Much of the literature regarding level of automation focuses primarily on the execution of test cases themselves, with little attention given to other testing activities such as test creation and data analysis. However, WQR reports and recent research have noted that greater attention needs to be paid to the lack of automation during test case creation.

Regarding test execution, The WQR 2018 finds that organizations „introducing automation in discrete chunks achieve better results than those" who try to immediately introduce full automation across an entire company's QA workflow. Galin (2004) states the two main advantages of incremental testing are that, first, having smaller modules promotes a higher level of error detection. Second, these errors are simpler and less resource-intensive to correct because they are isolated from the rest of the product being tested. These benefits stand in contrast to the „relatively low rate of big bang error identification" (Galin 2004). Similarly, Thomas (2006) states that in a top-down incremental approach, „once a level has completed its testing, the tester knows that any problems that appear in the future are more than likely caused by newly added Units, this decreases the scope of places to search once Bugs arise".

In rare circumstances, the big bang approach may be a logical approach. For example, if testing a „very small and simple" product, aggregating all quality control efforts into a single module may be practical and low-risk (Galin 2004; Thomas, 2006). However, Galin (2004) asserts that „it is generally accepted that incremental testing should be preferred despite its disadvantages", while Thomas (2006) adds that the big bang approach should not be applied to larger programs. Despite the limited use cases for a big bang approach, a study by Konka (2011) found that a big bang approach in small and simple projects could lead to future problems: While the approach initially worked, as the project grew in size and complexity, the test script became more difficult to maintain (p. 21). Therefore, the limited use cases for big bang testing, its „relatively low rate of error identification" (Galin, 2004), and the risks this approach poses for future projects makes it unsuitable for core network testing.

When considering what to automate and when, Bartley (n.d.) recommends five overlapping criteria:

1. Test environment complexity
2. Level of testing
3. How often the test needs to be run
4. Ease of automating the pass/fail criteria
5. Test stability and repeatability

As a test environment become more complex, more work is required to maintain it. For example, it might be more efficient to manually maintain a test environment if it contains a large amount of software and hardware prerequisites and interdependent configurations. Bartley adds that automating infrequently-run tests may not produce a strong return on investment compared to those that are run much more often. This could be due to the changes to the test environment that occur between infrequent test runs, software updates, or the need to manually check or update configurations between test runs. Regarding pass/fail criteria, greater complexity and maintenance is associated with the increasing difficulty of predicting a test case's outcome. Finally, unstable tests are more costly and difficult to maintain, canceling out benefits from automating them. Bartley describes how in some cases, manual tests with the problematic characteristics described above should be automated later on in a project after the software itself has become more stable.

For most of the above scenarios, test cases that are not initially suitable for automation may become easier to automate as a project progresses: As key features and configurations increase in their stability and become easier to maintain and predict, newly-integrated or soon-to-be-integrated modules should decrease in their complexity and uncertainty. This is because many of characteristics or dependencies of the newly-introduced modules have already been tested, corrected and integrated into the test environment. Bartley's five criteria of what and when to automate highlights the value of taking an incremental approach to automation, and illustrates the need for carefully evaluating what should be automated and when.

Combining QiTASC Products to Achieve High Levels of Automation

QiTASC provides a suite of tools that encourage incremental automation—not only in test execution, but also in device (phone) management, reporting and bug tracking. While **intaQt**'s flexible configurations and reusable modules encourage the incremental automation of test cases themselves, QiTASC's automated reporting service, **conQlude**, automates the collection and interpretation of the massive amount of data output generated by upwards of thousands of test executions per day. **conQlude** is also compatible with most project management systems (PMS), which gives users additional

flexibility with post-hoc data analysis. In terms of device management, QiTASC's **sQedule** automates phone acquisition, optimizing the availability of phones and matching them with available test cases. Finally, QiTASC's command line interface, **intaQt Client**, enables test projects to be executed via continuous integration services such as TeamCity and Jenkins.

Before testing with QiTASC's products, users may consider conducting an exploratory manual testing phase to ensure product and project know-how on the tester side and to identify areas that may be difficult to automate or maintain. Next, the test team develops intaQt test cases that cover a small cross-section of functionality, which allows new users to become familiar with intaQt as they apply their project know-how to the test. As the project grows and new features are integrated into the test environment, the level of automation and test coverage continues to increase.

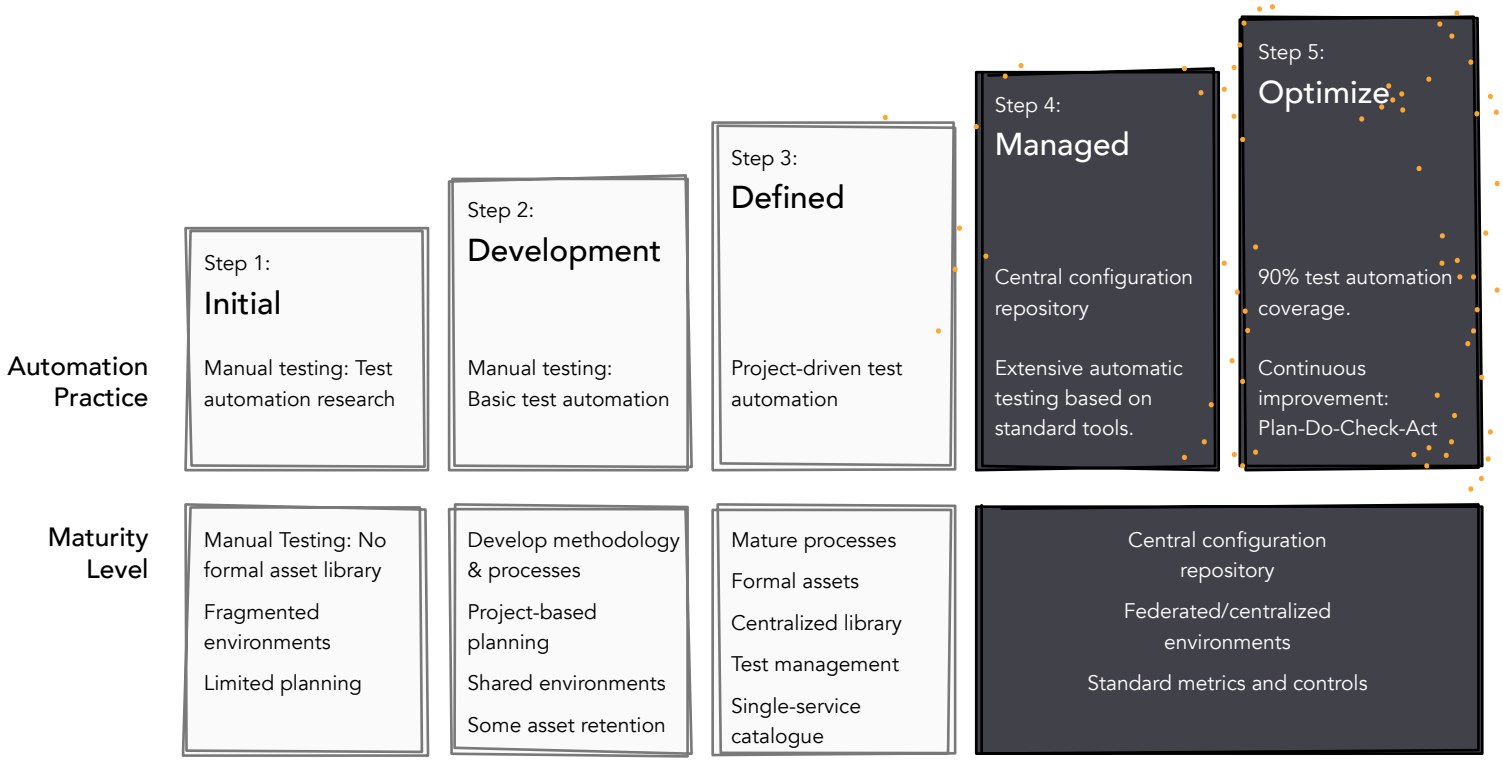


Figure 3 Improved levels of automation over time

To further promote incremental automation, **intaQt Studio** provides an integrated Git client, which is a version control tool that tracks changes made to test cases and other artifacts within a project. This lets users within a project maintain and access changed files, and even revert back to previous versions if needed. Additionally, **intaQt**'s Built-ins facilitate the automation of external functionality, not normally included within the test scope, including protocols such as HTTP, SCP and SSH, languages such as SQL and XML, formats such as CSV and JSON, as well as other important utilities including date and

time handling, data manipulation and multimedia recognition. For even greater levels of automation, **intaQt Client** allows tests projects to be run as *builds* via the continuous integration servers TeamCity and Jenkins, leaving the user with more time to analyze test output and correct errors. Finally, even test analysis activities can be automated via QiTASC's **conQlude** reporting service, which collects all test metadata from a project's text executions and sorts it into a centralized database for interpretation or further data manipulation.

Reaching a high level of automation with **intaQt** along with QiTASC's suite of automation and productivity tools is entirely achievable for telecommunications test projects. **intaQt**'s robust, reusable framework supports an incremental approach to automation by allowing different modules to be added and expanded upon as a project progresses. **intaQt**'s Built-ins further encourage automation by providing functionality for a wide range of requirements, including backend systems, data manipulation and different file systems. By integrating version control tools, continuous integration compatibility with **intaQt Client**, the **conQlude** reporting service and **sQedule**'s intelligent resource management, QiTASC delivers a holistic test solution that helps stakeholders maintain high quality services while finding and correcting errors faster and reducing the time-to-market.

Challenge No. 3

Different Testing Tools for Different Testing Phases

A lack of unified network testing tools—products or processes that can be used across development cycles and with different combinations of real and simulated devices—poses challenges to the efficiency and cost-effectiveness of test automation. Pinola et al. (2013) note that telecommunication systems require testing through *all phases* of the product's development. This typically involves using real and simulated devices in different configurations and at different stages of the testing process, resulting in multiple test modules to account for the variation in devices. The authors add that this is a costly problem, because of a lack of flexible testing tools that can re-used across the development cycle.

Because of this lack of unified tools, testing telecommunications infrastructure may involve using several applications, each requiring users to develop, configure, write and execute the same test cases multiple times, while being proficient with each testing application. Furthermore, test results from different applications may be incommensurate, thus, they cannot be shared or directly compared with each other, meaning additional steps must be taken to synchronize results.

Solution: A Wrap-Around Approach with Compound Steps and Simulated Devices

A recent approach to network testing involves what has been described as *wrap-around* methodology. This means developing a „flexible environment architecture that wraps around the testing target and can be configured to support different testing needs throughout the testing life cycle" (Pinola et al. 2013). A wrap-around approach can be accomplished with a flexible testing framework that enables a gradual transition from simulated to real devices. Such frameworks must support the integration of real phones and external simulators, e.g., those that generate SS7 traffic such as CAP and INAP. Additionally, the architecture must have its own internal simulators that can recreate the way a device acts and communicates with external events.

Using intaQt Across Multiple Test Phases

intaQt's Compound Steps allow users to access the same type of wrap-around functionality proposed in the research of Pinola et al. Compound Steps contain multiple criteria including actions and characteristics about voice calls, SMS transfers and data downloads. For example, a Voice Call Compound Step includes all events and actions that occur from when the caller makes a call until the call has ended. intaQt provides default values for all the criteria, however these values may be explicitly specified within Compound Steps by using Step Details.

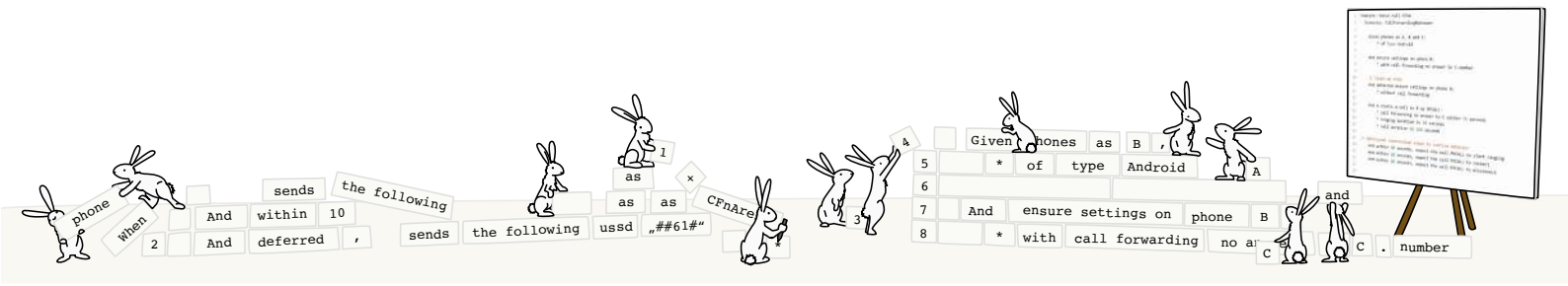
Example Voice Call Compound Step

```
Feature: MyCall
  Scenario: Compound Call

    Given phones as A and B:
      * of type Android

    And A starts a call to B as MYCALL:
      * detect incoming call within 10 seconds
      * callee does not answer
      * ringing duration is 30 seconds
      * caller ends the call

    And expect the call MYCALL to start ringing

    Then verify !A.isConnected()
    Then verify !B.isConnected()
```

To enable testing at different phases, a configuration switch lets the user switch between real phones and simulated devices, depending on the use case. This eliminates the need for writing multiple test cases depending on what type of device is used. A potential use case could look like:

- At the development stage, an entire call flow involves simulated devices.
- During field acceptance, a combination of real and simulated phones is used, depending on which network components are integrated into the test module.
- In field acceptance, only real phones are used and the simulation switch is deactivated.

By using these Compound Steps, only one test case is required for each use case:
The same test case can be concurrently executed on the developer's machine, in the production system or at any phase in between. The only difference is whether or not the configuration switch that tells intaQt to use real or simulated devices is activated.



Challenge No. 4

Complicated Software with a Steep Learning Curve

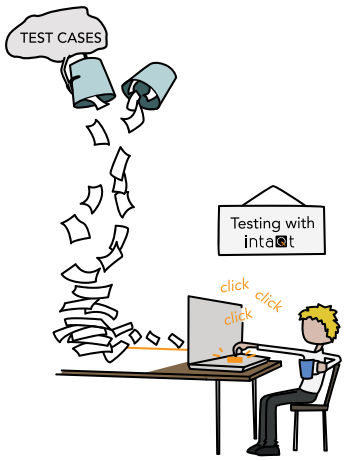
Several interacting information problems impede the understanding of *which* software to use, learning new automation software as well as training or hiring employees to execute automated test projects. Aside from the challenges of learning new software, the WQR 2017-2018 (2017) finds the „lack of specialist in-house knowledge of the depth and range of automation techniques" as another factor explaining the low amount of automated testing across industries.

Test automation is seen as requiring a considerable time investment in order to realize its benefits. One concern raised by mobile app developers is that automation tools are complicated to work with and require a lot of time to learn because of poor and limited documentation as well as a lack of training materials (Kochar et al. 2015). Furthermore, test automation tools often require the user to have language-specific programming skills (Zhu et al. 2008). This creates additional challenges and costs, whether by upgrading existing testers' skills, hiring new employees with the expertise to efficiently use the new software—or both.

Solution: Leverage Employee Skill Variation and Knowledge Transfer

Software testing entails multiple levels of expertise and skillsets. The software testing firm Abstracta describes different career levels of testers (Toledo, 2015), such as:

- **Junior/entry-level testers,**
  who do *basic testing activities* such as executing tests and reporting bugs while gaining exposure to simple test case design and QA issues;
- **Mid-level testers,**
  who have the same responsibilities as a junior tester, but are more actively involved in designing test cases and QA activities;
- **Senior testers,**
  who have years of testing experience and often have specialized knowledge about certain testing applications and industry-specific issues;



- **Test automators/technical testers,**
  who have programming knowledge that enables them to work on backend aspects of test case design, performance and integration with external systems;
- **Test managers,**
  who lead test teams and focus on employee distribution to ensure that testers are matched with the testing tasks that they are most suited for.

The benefit of having testers with varied levels of expertise is that it enables efficient resource allocation and knowledge transfer: Senior testers and those with programming experience can much more easily abstract a new automation tool, and become proficient with components requiring, for example, scripting knowledge, industry knowledge and how the software integrates with the system under test or additional external components. Likewise, these experienced testers are critical for passing on knowledge and training more junior employees.

A diversified employee structure also facilitates cost-effective, focused training: For example, after adopting new testing software or learning about new features, senior testers and test managers gain in-depth knowledge about the tools and how they relate

back to their team's projects. While training, such as on-site or classroom-style learning takes place, junior testing employees can continue working on existing testing activities while senior employees upgrade their skills. With the newly gained knowledge they have gained from the training sessions, the senior testers and managers pass on the knowledge to junior and mid-level team members before returning to their more technically-complex testing activities.

Finally, as cited earlier, Ben-Ner and Urtasun (2010) found in their own research, as well as in previous research, that occupations involving complexity, problem solving and variety have seen positive, complementary professional benefits from automation. Test automation increases employee efficiency and enables to manage their time more wisely, allowing them to devote their time to problem solving, analysis and on-the-job learning (pp. 25-26).

intaQt Studio Productivity Features for Beginners and Experts

Maintaining a test team with a variation in skills and experience promotes the efficient distribution of activities while also allowing new users to improve their technical skills. Furthermore, testers with advanced skills can lead their teams while managing more complex tasks that require scripting knowledge. However, because skill variation on its own is not enough, **intaQt**, as well as **intaQt Studio**, contain an extensive line of Built-ins, which are features that help speed up the learning process for beginners and experts alike, while eliminating the need for testers to deal with complicated backend systems and languages.

intaQt Feature Files use a natural-sounding frontend language, which allows users without any programming knowledge to gain proficiency in writing, executing and troubleshooting test cases. Learning **intaQt** is further facilitated by QiTASC's integrated development environment, intaQt Studio, which contains productivity features including auto-completion, error inspections and refactoring. These features simplify creating, adapting and managing test cases, while helping the user work independently and effectively.

The example below shows a Feature File, where **intaQt Studio**'s autocompletion helps the user write a step that reads a thermometer from a Smart Home mobile app. The `Custom Step` label indicates that this step was created by a user using **intaQt**'s custom language.

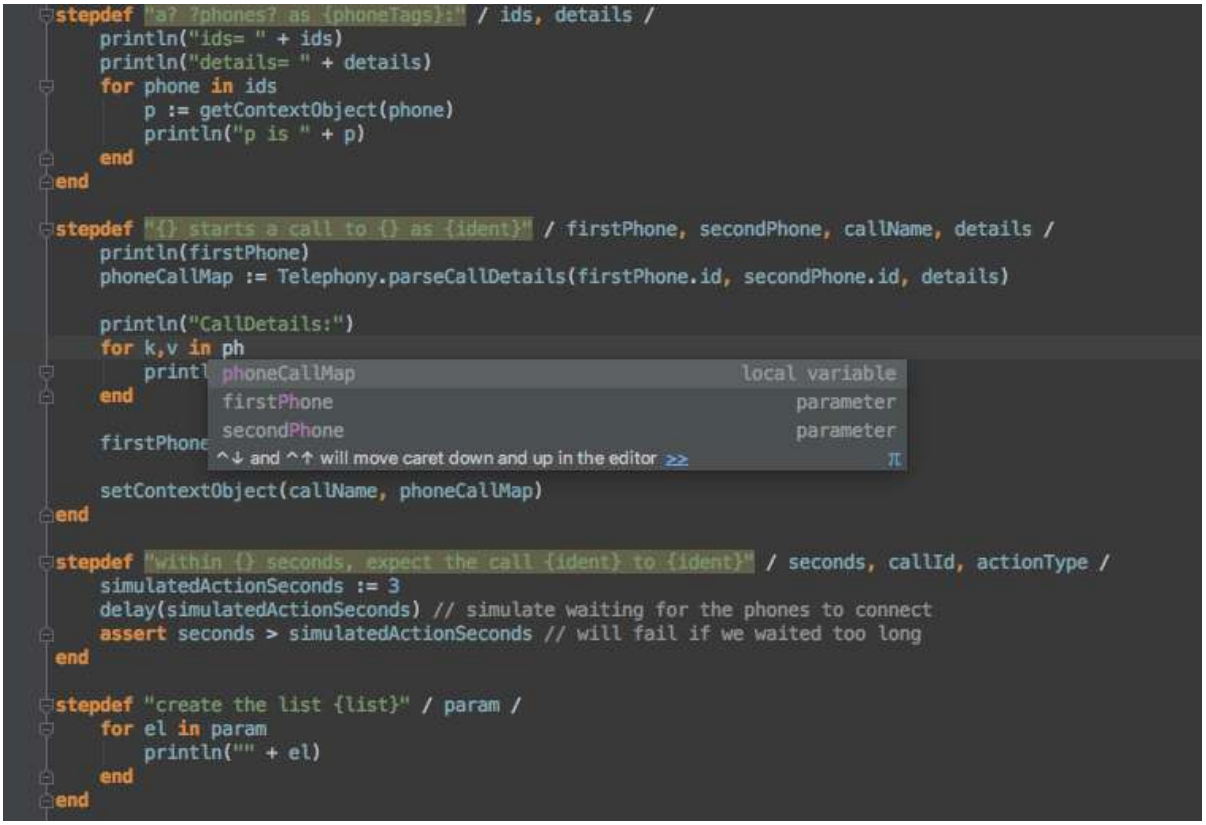


Figure 4 Auto-completion in intaQt Studio

For experienced testers with scripting knowledge, intaQt's custom UI Steps and Steps languages enable users to create models and functions that perform use case-specific activities and can even interact with external hardware/software „behind the scenes". Often, users incorporate one of intaQt's many Built-ins to enhance the complexity of their tests: intaQt Built-ins provide out-of-the-box functionality that are available to integrate, such as database connections, XML matching, programming language-specific content generation and utility functions. These Built-ins apply to many contexts and enable writing elaborate test cases that extend beyond intaQt itself. Furthermore, intaQt Built-ins limit the need to code in multiple applications or learn about different programming languages, protocols and file formats.

The example below shows the custom Stepdef, or Step Definition accessed by the `read temperature` step shown in the Feature File above. In this case, an experienced user would write the custom script within a Stepdef, which integrates external services into the test case. The Stepdef uses two of intaQt's Built-ins: the File Built-in and the Image Built-in, which allow intaQt to access a file containing an image of the thermostat, open the image and examine the temperature reading on the thermostat.

```
stepdef "within {} seconds, expect the call {ident} to {ident}" / seconds, callId, actionType /
    simulatedActionSeconds := 3
    delay(simulatedActionSeconds) // simulate waiting for the phones to connect
    assert seconds > simulatedActionSeconds // will fail if we waited too long
end

stepdef "create the list {list}" / param /
    for el in param
        println("" + el)
    end
end
```

Figure 5 Custom intaQt stepdef implementation

From the perspective of a junior tester who wrote the `read temperature…` step in the Feature File example, their user experience does not change at all: They can write the `read temperature…` step into the Feature File without considering what behaviors happen in the background, while their senior colleague takes care of complex backend activities.

intaQt Studio's integrated development environment promote teamwork and knowledge transfer amongst employees with different levels of expertise. This is done by linking Feature Files, which are easy for users with no programming experience to master, to custom Stepdefs that define highly complex test cases and integrate with external interfaces. Incorporating additional Built-ins into test cases further promotes efficient test management by including out-of-the-box functionality for multiple interfaces, languages and protocols so that testers do not need to spend much time dealing with additional technical challenges.

As a best practice, it is recommended that a knowledgeable intaQt tester should accompany any test *trial run* to ensure that common automation mistakes are not made. Alternately, a customer's first project can be created and executed by professionals who then train others to use intaQt and understand what different test execution outcomes mean.

Challenge No. 5

Managing Devices, Models and OS Versions

What is the optimal number of devices that must be tested to be confident that a product works properly? Does that amount stay the same or change over time? Although this issue has not been prominently discussed within the context of telecommunications networks, several studies on mobile application testing have attempted to address it. One case study on mobile application testing describes on in August 2012, nearly 4000 models Android devices, representing 599 brads and multiple operating systems, had downloaded OpenSignal's app (Villas-Boas, 2015). It would be cumbersome to even attempt to test 10% of these models, whether using a manual or automated approach.

In Galindo et al. (2016), the authors discuss mobile app testing and describe how less than 25 Android devices are actually „officially certified" as compatible with Skype, thus only 25 models have fulfilled test requirements to be considered supported by Skype. Nevertheless, the application remains popular and is accepted as being compatible with Android Devices. The authors also add that it would be too costly to acquire and test every available Android model. Furthermore, the different configurations required for each model would cause the cost of testing to outweigh potential benefits. In such cases where many models should be tested, simulated devices are often a solution. However, each simulated phone model may also require different configurations. As a result, the costs and efforts relative to the benefits of testing act as a natural limit to the number of devices that can currently be tested.

In addition to ensuring the appropriate number of models and OS versions, determining the correct amount of phone units themselves can also pose a problem for testing. For example, a pool of 100 mobile phones might be available for a project with 2000 test cases. If there are 20 users executing test cases, each of which require two phones that are configured to have the same behavior and characteristics, then there will always be a surplus of 60 phones. Therefore having 100 devices would be a poor use of resources.

On the other hand, if 500 test cases require a phone to be configured so that it behaves as though it is being used in another country and with randomly selected test cases, how many phones should be configured to be domestic vs. international? As additional variables like subscriber characteristics, are added to the test case, or the amount of phones needed in a test case, allocating phones becomes a greater challenge and the risk of test failure grows.

Solution - Focus on Network Functionality and Resources, Not the Devices

It is a misconception that aiming to test as many models as possible is critical to testing telecommunications infrastructure. When the goal is testing network functionality, the importance is in the network's behavior rather than the device models. In other words, the user is not testing the device: The device is a means to test how the network functions. In the case of Android phones, their models all behave quite similarly in a network. Therefore, an emphasis should be placed on ensuring there is a large enough pool of devices to ensure resource availability when executing multiple test cases.

A whitepaper discussing mobile app testing suggests that using 30 different device models provides test coverage for about 80% of all potential devices. For example, having 8-16 Android devices that represent different makes and OS versions along with 4-8 Apple devices (Orasi, 2012). With that suggestion in mind, mobile app testing *does* require a greater diversity of models compared to network testing because of app-

specific considerations such as screen size, input types and OS version that all affect the way a user interacts with an application. On the other hand, these same issues are not present when testing core networks: Screen size does not affect whether or not a call goes through.

In this context, the amount of device models required to test telecommunications infrastructure is subjective. More importantly, projects should have enough devices (or „units") to satisfy their resource requirements. Regarding the pool of devices, different approaches vary to maintain device availability, such as flexible customer configurations, non-randomized test case assignment and scheduling. However, these are time-consuming activities that may require a significant amount of labor, such as developing predictive models.

QiTASC's Intelligent Resource Management

QiTASC provides several features that allow users to optimize device availability for test cases. First, QiTASC's **sQedule** is an intelligent scheduler that evaluates and allocates phones, allowing for multiple test cases to be run in parallel by **sQedule** Agents. **sQedule** Agents use a special mode of **intaQt** that requests test cases from the **sQedule** Server. A list of all test cases are passed to the **sQedule** process, and **sQedule** will always try to execute as many test cases as possible according to the test cases' properties and available phones. Finally, **sQedule** is especially useful for managing test cases that are extremely long or must be run outside of employee working hours.

In addition to **sQedule**'s scheduling capabilities, **intaQt** provides the **intaQt** Phone Service, which allows for managing remotely-connected phones, while **intaQt Studio** includes a Phone Plugin that shows all available phones -- remote or locally-attached -- for a given project.

The **intaQt** Phone Service enables scenarios such as roaming test cases, where the customer is located outside of their local calling area. Additionally, it means that testers can access phones that are plugged into machines other than the computer on which they are running **intaQt**. Additionally, SIM card management, which may be used in combination with the **intaQt** Phone Service, further enables robust phone management.

When a tester uses the Phone Plugin, they can access information about all phones in a project as well as the phones' properties, including hardware information and „customer" information such as the phone number and other account details. The Phone Plugin also lets the user view the phone's behavior in real time. Furthermore, it allows a user to interact with a phone from within **intaQt Studio** and manually perform actions if required, such as opening applications, selecting dialog boxes and copying/pasting text.

Example Test Case with intaQt Studio Phone Plugin

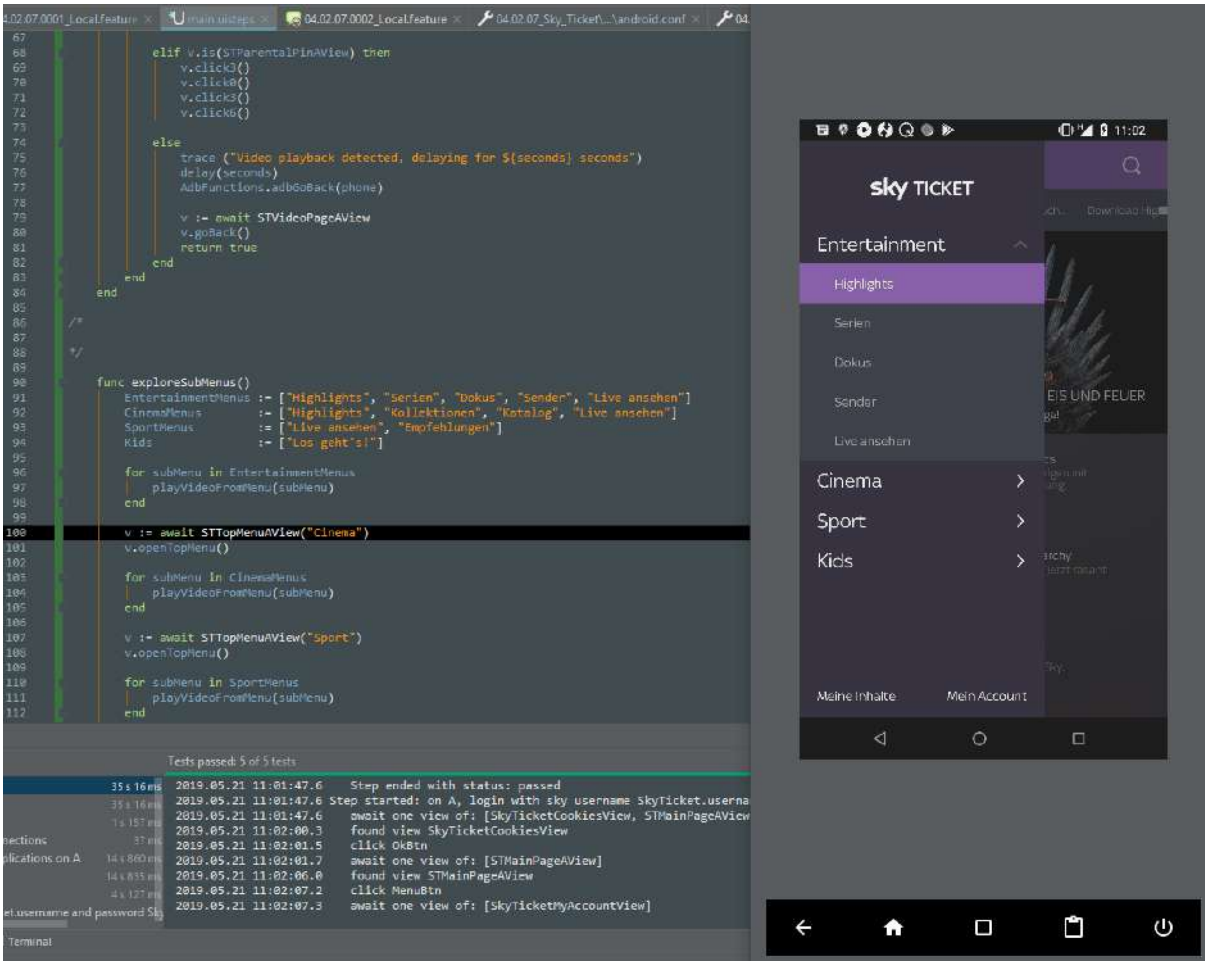


*Figure 6 **intaQt Studio** Phone Plugin streaming a remote Apptest execution*

Finally, **intaQt** facilitates the use of virtual phones that can be accessed with a simple configuration switch. Test cases that do not require real devices, such as tests run during a project's development stage, may use virtual phones, while test cases run in a field production environment may only use real phones.

QiTASC's combination of device management tools ensures that tests are scheduled, executed, and that resources—real or simulated—are allocated in the most efficient way possible. Furthermore, **sQedule** allows projects to use a combination of testers and **sQedule** Agents, depending on resource availability, test requirements and test phase.

Discussion

The need for multiple test tools for different activities, combined with practical challenges to achieving optimal test coverage with a high level of automation, all strongly reflect the fragmented nature of telecommunications software as well as the differentiation between its components, features and additional hardware requirements.

Many automated testing solutions are only able to cover certain aspects of testing, such as simulate some devices or can automating certain actions. A high level of automation implies not only test execution, but also other activities such as test case development and results analysis. These activities, too, are limited by testing tools that can only automate one aspect of a test project, especially when the tools cannot integrate with other interfaces. As a result, full test coverage is difficult to attain when a single solution lacks the capabilities to automate all (or most) requirements. At the same time, difficult-to-learn or cumbersome software and a lack of expert knowledge within test teams create additional barriers to achieving effective test automation strategies and realizing the ongoing benefits of automated testing.

Given these interdependent challenges, it is understandable that the WQR 2017-2018 reports low levels of automation across industries, including telecommunications. However, as testing tools become more sophisticated and industry know-how expands, it is expected that the use of automation in QA activities will continue to increase. When investigating test automation solutions, developing a proof of concept with potential vendors allows customers to identify test requirements and pain points, and additionally provide an opportunity for exploratory testing activities that may reveal further QA needs or gaps in knowledge.

At QiTASC, proofs of concept generally lasts for several weeks, involving automating a small subsection of an anticipated test project. This enables QiTASC to understand customer's needs, their system under test, and additional products that may further be beneficial towards the project such as the **conQlude** Reporting Service or **sQedule** Intelligent Resource Management.

intaQt's comprehensive and customizable capabilities make it extremely well-suited for testing intelligent networks and core network functionality, as it can cover and automate most activities and scenarios that would be executed by a real-world mobile customer. However, as described above, just because something can be automated, it does not mean that we should switch to automate everything at once. The "big bang" approach risks creating too much overhead at the beginning of a project. A better approach involves incrementally increasing test coverage and levels of automation as new modules are integrated into a project. This encourages users to become confident with software so that they can independently develop increasingly-complex tests, while sharing knowledge amongst members of the test team as a project progresses.

At the beginning of a test project, initial test case development, execution and analysis will inevitably use up a significant amount of time, where troubleshooting and technical support from QiTASC may be required. However, once a project has passed the stage of initial exploratory automation, intaQt facilitates the rapid expansion of test coverage and levels of automation within the context of a sustainable test framework.

First, the intaQt custom languages support reusability, meaning that existing functions and models can be easily applied to and accessed from any test case within a project. This is especially useful for projects with a lot of backend integration, e.g., external HTTP, SSH and/or XML. Second, regarding backend systems, intaQt Built-ins allow users to integrate such backend integrations into its test suites. This eliminates the need for using multiple, additional, automation tools. Finally, intaQt's configurations, like its custom languages, are flexible enough that they allow users to instantly switch between technical requirements, such as simulated or real devices, subscriber properties and security settings without changing anything in the test case itself.

Conclusion

With the right combination of technology and project management, QiTASC helps customers overcome the challenges described in this white paper. By incrementally increasing levels of test automation via **intaQt**, test teams increase their technical know-how, and are able to find and solve errors faster and spend more time focusing on product development instead of menial QA tasks. Combining **intaQt** with QiTASC's additional automation products for managing and scheduling devices as well as reporting enhances levels of automation, test coverage and productivity. While improving high levels of automation should be a goal for intelligent network testing, certain tasks are best left to manual testing. These include extremely complex applications or features, tests that are rarely run and tests that are expected to be unreliable or inconsistent. Nevertheless, in order to maintain high product quality while ensuring swift times-to-market, firms have no choice but to increase their use of test automation tools across all QA activities.

To speak to someone at QiTASC about its suite of automation tools, or to arrange for a demo, please contact

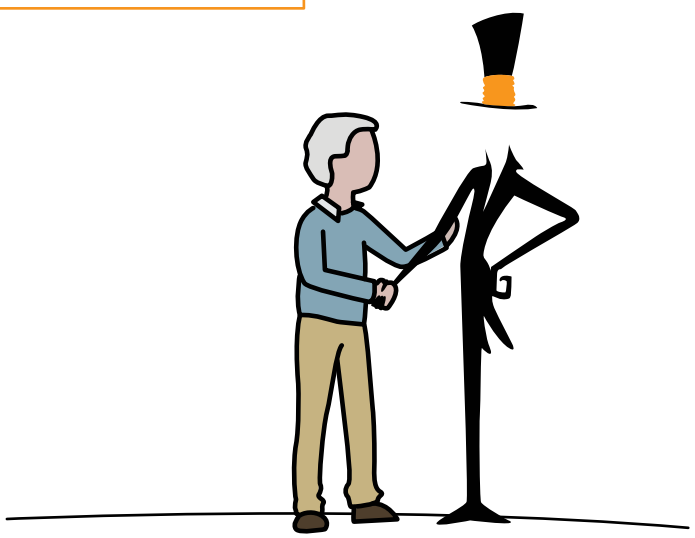
Can Davutoglu
can.davutoglu@qitasc.com

or

Sales Team
sales@qitasc.com

To learn more about QiTASC's range of test automation solutions and use cases, please visit our website at

https://www.qitasc.com



Co-Author information

Leora Courtney-Wolfman

Lead Technical Writer

Leora Courtney-Wolfman has been leading QiTASC's documentation team since 2016, where she oversees all documentation workflows including manuals, tutorials and newsroom content. Prior to joining QiTASC, Leora worked as a scientific editor.

Leora holds an MSc in economics from the Vienna University of Economics and Business and a BA from the University of British Columbia.

Michael Zehender

CEO Research & Development

Michael Zehender is a passionate engineer who leads the QiTASC development team's efforts to create the best test automation tools available. He focuses on an integral, holistic approach to test automation that also helps QiTASC's customers to maintain a high level of quality and support that they expect and depend on.

Michael holds a BSc in Computer Sciences from the Technical University of Vienna.

References

„Android Fragmentation Visualized - OpenSignal". OpenSignal. August 2012. https://opensignal.com/reports/fragmentation.php.

Bartley, Mike, PhD. *Achieving Business Benefits through Automated Software Testing.* Article https://www.bcs.org/upload/pdf/test-automation-mbartley.pdf.

Ben-Ner, Avner, and Ainhoa Urtasun. „Computerization and skill bifurcation: the role of task complexity in creating skill gains and losses". *ILR Review* 66, no. 1 (2013): 225-267.

Biswas, Swarnendu, Rajib Mall, Manoranjan Satpathy, and Srihari Sukumaran. „Regression test selection techniques: A survey". *Informatica* 35, no. 3 (2011).

do Carmo Machado, Ivan, Paulo Anselmo da Mota Silveira Neto, and Eduardo Santana de Almeida. „Towards an integration testing approach for software product lines". In *Information Reuse and Integration (IRI), 2012 IEEE 13th International Conference* on, pp. 616-623. IEEE, 2012.

Fierens, Daan, Jan Ramon, Hendrik Blockeel, and Maurice Bruynooghe. „A comparison of pruning criteria for probability trees". *Machine Learning* 78, no. 1-2 (2010): 251.

Galin, Daniel. *Software quality assurance: from theory to implementation*. Pearson Education India, 2004.

Galindo, José A., Hamilton Turner, David Benavides, and Jules White. „Testing variability-intensive systems using automated analysis: an application to Android". *Software Quality Journal* 24, no. 2 (2016): 365-405.

Gao, Jerry, K. Manjula, P. Roopa, E. Sumalatha, Xiaoying Bai, Wei-Tek Tsai, and Tadahiro Uehara. „A cloud-based TaaS infrastructure with tools for SaaS validation, performance and scalability evaluation". In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pp. 464-471. IEEE, 2012.

Gao, Jerry, Xiaoying Bai, Wei-Tek Tsai, and Tadahiro Uehara. „Mobile application testing: a tutorial". *Computer* 47, no. 2 (2014): 46-55.

Kochar, Pavneet Singh, Ferdian Thung, Nachiappan Nagappan, Thomas Zimmermann, and David Lo. „Understanding the test automation culture of app developers". In *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*, pp. 1-10. IEEE, 2015.

Kong, Liang, Hong Zhu, and Bin Zhou. „Automated testing EJB components based on algebraic specifications". In *null*, pp. 717-722. IEEE, 2007.

Konka, Bharat Bhushan. „A case study on Software Testing Methods and Tools". (2012).

Malaiya, Yashwant K., Michael Naixin Li, James M. Bieman, Rick Karcich, and Bob Skibbe. „The relationship between test coverage and reliability". In *ISSRE*, pp. 186-195. 1994.

Morgado, Inês Coimbra, and Ana CR Paiva. „Impact of execution modes on finding Android failures". *Procedia Computer Science* 83 (2016): 284-291.

„Mobile Application Testing: Rationale and Best Practices for Cloud-Based Automated Testing". Orasi. 2012. https://www.orasi.com/wp-content/uploads/2016/08/Orasi_Mobile_Testing_WP.pdf.

Mobile Application Testing Rationale and Best Practices for Cloud-Based Automated Testing. 2012. White paper.

Pinola, Jarno, Juho Perälä, Petri Jurmu, Marcos Katz, Seppo Salonen, Jonne Piisilä, Jouko Sankala, and Pekka Tuuttila. „A systematic and flexible approach for testing future mobile networks by exploiting a wrap-around testing methodology". *IEEE Communications Magazine* 51, no. 3 (2013): 160-167.

Srikanth, Hema, Laurie Williams, and Jason Osborne. „System test case prioritization of new and regression test cases". *In Empirical Software Engineering*, 2005. *2005 International Symposium on*, pp. 10-pp. IEEE, 2005.

Thomas, Gareth. „Object Orientated Integration Testing." (2006).

Toledo, Federico. „The Career Path of a Software Tester: An Infographic". *Abstracta* (blog), 2015. Accessed August 16, 2018. https://abstracta.us/blog/software-testing/career-path-software-tester-infograhic/.

Thomas, Gareth. „Object Orientated Integration Testing". (2006).

Tuteja, Maneela, and Gaurav Dubey. „A research study on importance of testing and quality assurance in software development life cycle (SDLC) models." *International Journal of Soft Computing and Engineering (IJSCE)* 2, no. 3 (2012): 251-257.

Veselov, Alexey, and Vsevolod Kotlyarov. „Testing automation of projects in telecommunication domain." In *Proceedings of the Spring/Summer Young Researchers' Colloquium on Software Engineering*, no. 4. Федеральное государственное бюджетное учреждение науки Институт системного программирования Российской академии наук, 2010.

Villas-Boas, Andreas. „One of the Biggest Problems with Android Keeps Getting Worse." *Business Insider*, August 5, 2015. https://www.businessinsider.com/android-fragmentation-graph-from-opensignal-2015-8?IR=T.

World Quality Report 2015-2016. Report. 2015. https://www.capgemini.com/resources/world-quality-report-2015-16/.

World Quality Report 2017-2018. Report. 2017. https://www.capgemini.com/resources/world-quality-report-2017-18/.

Yoo, Shin, and Mark Harman. „Regression testing minimization, selection and prioritization: a survey". *Software Testing, Verification and Reliability* 22, no. 2 (2012): 67-120.

Zhu, Xiaochun, Bo Zhou, Juefeng Li, and Qiu Gao. „A test automation solution on GUI functional test". In *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, pp. 1413-1418. IEEE, 2008.

"Quality is never an accident.
It is always the result of intelligent effort."

(John Ruskin)

Get in touch!

Contact us

+43 1 810 21 73
info@qitasc.com

Visit us at

Diefenbachgasse 53
1150 Vienna / Austria