curiosity SOFTWARE IRELAND

# Principles of Continuous Testing

## A Practitioner's Guide

*05-12-2018*

curiosity SOFTWARE IRELAND

© Curiosity Software Ireland Ltd. 2018

vip

# Principles of Continuous Testing

## A Practitioner's Guide

### Table of Contents

*Principles of Continuous Testing* ........ 4

*New approach, same barriers?* ........ 6

A question of method: mini-Waterfalls undermine continuity ........ 6

People, process, technologies: practical barriers to Continuous Testing ........ 7

*"Shift left" with Model-Driven Development* ........ 11

*"Shift right" to automate and optimise TestDev* ........ 14

Automated Test Case Design ........ 15

"Just in Time" Test Data ........ 17

On Demand Environments ........ 19

Complete Test Automation ........ 20

*Robotic Process Automation manages the TestOps* ........ 22

*Accurate development and rigorous testing in-sprint* ........ 26

*Continuous Testing: an ongoing process* ........ 27

*Find out more* ........ 28

*End Notes* ........ 28

 © Curiosity Software Ireland Ltd. 2018 

Continuous Testing has become an ideal for organisations striving to build and deliver ever-more complex systems, in ever-shorter iterations. This eBook considers the core principles of Continuous Testing, and what it means in practice to fulfil them. It begins by considering what Continuous Testing is in its essence, and common barriers to its adoption. An alternative approach to designing, building, and testing systems is then set out, arguing that it both fulfils the core principles of Continuous Testing, and can remove perennial barriers to its adoption.

curiosity SOFTWARE IRELAND © Curiosity Software Ireland Ltd. 2018 vip

# Principles of Continuous Testing

Rigorously testing fast-changing, fast-growing systems within ever-shorter iterations demands that every moment is spent testing new functionality. Any time maintaining existing assets will quickly become unsustainable, as the growing complexity of systems will soon mean that there are more assets than can be updated in-sprint. Meanwhile, attempting to formulate and execute new tests only after maintenance, at the end of a cycle, will only lead to a mountain of technical debt, as testing rolls over constantly to the next iteration.

Quality Assurance must instead commence the moment design and development does, to avoid large portions of the released system going untested and being left exposed to costly defects. This leads us to a Shift Left approach, the first principle of Continuous Testing:

1. **Shift Left, Shift Right**. Testers must act as critical modellers, working from day one to build quality into testable systems. This not only avoids the higher remediation costs associated with detecting bugs late, but can enable a greater degree of subsequent automation as QA moves "right" through the delivery cycle.
2. **Parallelism.** There must accordingly be an emphasis on parallelism, with test teams working simultaneously with one another, as well as with developers and business analysts. Only then can the procedural constraints and miscommunications associated with a siloed approach be avoided.
3. **"Don't Repeat Yourself" (DRY**).[i] There must further be an emphasis on re-usability, so that the effort of previous iterations is seamlessly leveraged in future sprints. This is required if QA is required to focus only on newly developed or updated functionality.
4. **Complete automation**. Continuous Testing further requires the removal of bottlenecks associated with manual effort. This demands a near zero touch approach to moving from system designs to an up-to-date set of test cases, environments, automation logic, and data. QA then becomes an automated comparison of the intended system, housed in accurate designs, to the developed system, housed in


© Curiosity Software Ireland Ltd. 2018

code. This end-to-end test automation requires the automation of TestDev tasks like test creation and execution, as well as the automation of TestOps tasks through Robotic Process Automation.

5. **Continuous Feedback**. The complete automation of TestOps and TestDev task requires information flowing seamlessly across the whole delivery lifecycle. This also facilitates the Shift Left, Shift Right approach necessary to automate test maintenance: introducing a traceability from QA assets back "left" to the design from which they are derived means that test results can be fed directly into the design; at the same time, any changes made to the design reverberate "right", being mirrored automatically in the test assets. A complete information flow between technologies further maximises observability and the amount of both machine data and testing metadata that can be learned from in QA. This maximises confidence in test results, while enabling targeted and evidentially informed testing. Collecting as much testing metadata as possible in QA will also be a prerequisite to any form of artificially intelligent testing.

These five principles indicate broadly the nature of Continuous Testing. The challenge comes in their implementation, especially as few QA teams can simply halt their cycles to rebuild their processes, tools, and teams from the ground up; they instead must work iteratively to improve the existing best practice, while continuing to test for upcoming releases. Some of the barriers to Continuous Testing will now be considered, before setting out some practical steps for fulfilling the five principles now defined.


© Curiosity Software Ireland Ltd. 2018

# New approach, same barriers?

It has now been nearly five decades since Winston W. Royce described the sequential, Waterfall Model, arguing in 1970 for the lack of feedback inherent in it.[ii] The arrival of *The Agile Manifesto*[iii] in 2001 validated many of Royce's concerns, and several principles of delivery have since sprung up. Some, such as Continuous Delivery and DevOps, are cognate to the principles of Continuous Testing just described; and yet, barriers to achieving this ideal remain.

## A QUESTION OF METHOD: MINI-WATERFALLS UNDERMINE CONTINUITY

A major reason is that "Agility" at most organisations does not mean swiftly moving complete, minimally viable, and rigorously tested systems through short delivery cycles. Nor does it mean fail-fast experimentation, where we can learn from mistakes made early and continually improve our understanding of the system being created. It instead means mini-Waterfalls, where Business Analysts design systems, developers code them, and QA tests them – all in that order. The difference is they try and squeeze it into six weeks, rather than 18 months.

These mini-Waterfalls remain in stark contrast to the principles set out above. They are inherently siloed, and there is likewise little parallelism. Teams must instead wait for the previous stage to be completed, and for the information they need to fulfil their siloed role to be passed on. Testing new functionality therefore starts late, in contrast to a Shift Left approach. When information is finally moved on, it is furthermore typically in a format that must be manually converted during the next stage. This limits automation and produces defects as uncertainty and miscommunication inevitably creep in.

Finally, there is a lack of either re-usability or feedback. As information has been translated manually from one stage to the next, there is little inherent traceability across the delivery pipeline. A change made to the system design or a new user story must in turn be reflected manually in code and tests. This complex maintenance pushes QA further back, so that defects are detected late, where they require far more time and expense to fix. It is then


© Curiosity Software Ireland Ltd. 2018

equally difficult to update code and the design when defects are discovered, as the lack of traceability renders it difficult to pinpoint the exact origin of the problem in tests or code,

## PEOPLE, PROCESS, TECHNOLOGIES: PRACTICAL BARRIERS TO CONTINUOUS TESTING

These methodological barriers to Continuous Testing are reflected in the practices adopted at each stage of the delivery lifecyle. A "mini-Waterfall" at a given organisation might look as follows:

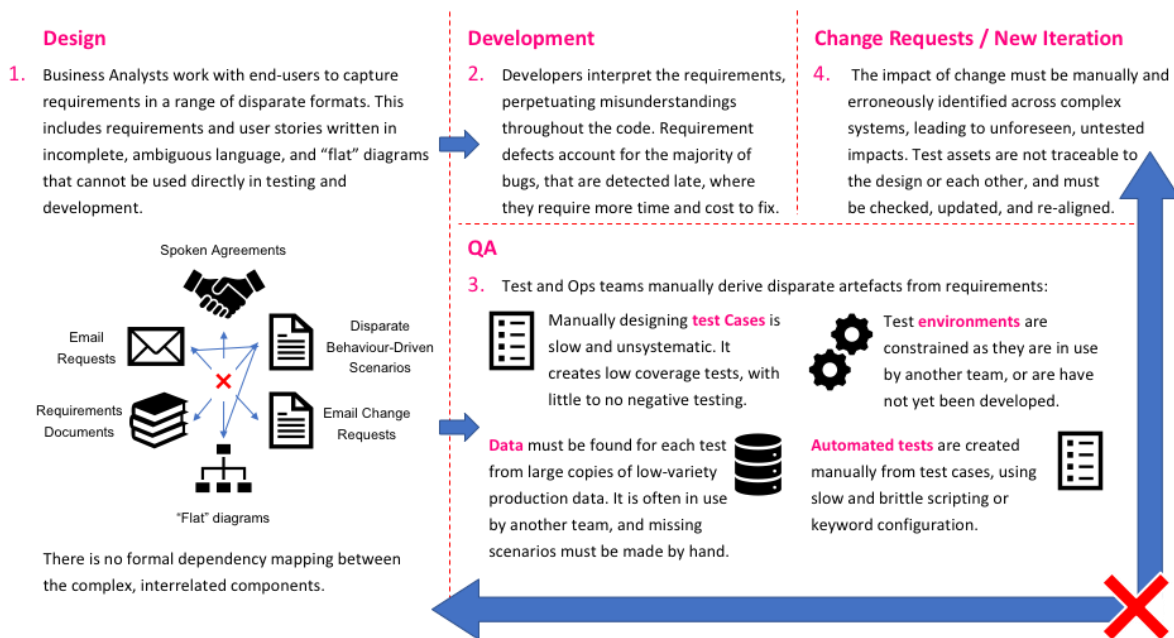


**Figure 1:** *The linear stages of mini-Waterfalls.*

Each of these siloed stages leaks quality and increases the over-reliance on manual process. This is the direct result not only of how the design, test, and development assets are created, but particularly how they are *stored*:

1. **Requirements** are gathered from end-users and converted into a range of unconnected documents and diagrams. This includes written user stories, behaviour-driven scenarios, and requirements documents. They are usually written in ambiguous natural language, far removed from the system logic. There are sometimes additional diagrams, from hand-drawn models to "flat" business process models. There is rarely formal dependency mapping or traceability between the complex, interrelated components set out in these disparate files and formats.


© Curiosity Software Ireland Ltd. 2018

2. **Code** is derived from the imprecise, incomplete requirements. Design defects and ambiguities account for the majority of defects produced in development,[iv] and the majority of defect remediation cost.[v] The lack of dependency mapping passes responsible to developers to envisage how vastly complex systems should fit together, but no one person can have complete knowledge of such systems. This throws up integration errors, while missing logic in the system design leaves many decisions to developer assumption and imagination; when their decision deviates from the user's desired functionality, additional defects arise.
3. **Test cases** must likewise be derived manually from "flat", imprecise system designs that render automated techniques impossible. Testers instead attempt to string together the disparate files in their minds, manually creating test cases to exercise the system logic. This is slow and repetitious, formulating the same test steps in overlapping test cases and inputting them one-by-one into Application Lifecycle Management tools. It is further unsystematic, leading to low coverage tests in spite of the wasteful overtesting. A simple system will contain more paths through its logic than any one person can imagine in their heads, and research suggests than in 2018 66% of companies still struggle "merely deciding what to test."[vi] Manually created test cases therefore focus repeatedly on the obvious, "happy path" scenarios, neglecting the negative paths and unexpected results most likely to cause system collapse. Combine this with the logic absent in the requirements from which tests are derived, and it is rare to find manually created test cases that test more than a fifth of the logic contained in the system under test.
4. **Test data** must additionally be moved to test environments, and then found or created for each specific test case. This manual effort creates further bottlenecks, while the use of production data in less-secure QA environments risks costly non-compliance. Complex data masking and transfers leave test teams waiting for data that is almost always out-of-date by the time Ops teams have provisioned it. Additional delays arise as test teams compete for the same copies of data, creating cross-team constraints and undermining parallelism. Testers waste further time searching through the unwieldy data sets for the exact data combinations needed to


© Curiosity Software Ireland Ltd. 2018

execute their tests; however, production data is low-variety, drawn from expected user behaviour, and by definition contains only past scenarios. It does not contain the data needed to test new functionality or negative scenarios, and testing is pushed further back as QA teams must create complex data scenarios by hand.

5. **Automated test execution** is necessary to execute the number of tests required to rigorously test complex systems within short iterations. However, the time saved automating test execution is frequently outweighed by the manual effort it introduces. This is typically either complex, manual scripting, or fiddly keyword configuration. As a result, rates of functional test automation remain low, no higher than 30% at around one third of organisations, and less than 50% for the majority of teams.[vii] Meanwhile, 67% of testers and developers reported in a 2018 survey that attempts to adopt Continuous Testing have expanded total test execution time.[viii] Automating execution further does nothing to improve the rigour of the test cases being automated. As automation pioneer, Dorothy Graham, and Mark Fewster comment, "it is the quality of the tests that determines whether or not bugs are found, and this has very little, if anything, to do with automation."[ix]
6. **QA environments** create further bottlenecks as testers wait for them to be provisioned by Ops teams. They must then compete for the limited number that become available. Interdependent components might similarly be unfinished, or might be relied upon my critical live systems. Cloning complex distributed systems is resource-intensive, costly, and time-consuming, while manual stubbing for virtualisation is laborious and inaccurate. The result is that testers lack the environments and tests in which to execute tests within any given iteration.
7. **Manual maintenance** is the greatest barrier to Continuous Testing. When a change request or new user story arrives, testers and developers must first identify its impact across vast and complex systems. The lack of formal dependency mapping renders automated analysis impossible, and some upstream and downstream impacts are inevitably overlooked. The unforeseen impact of apparently innocuous or slight changes in turn throw up integration errors, and numerous high-profile outages point to the damage these can create. Then there's the time associated with


© Curiosity Software Ireland Ltd. 2018

reflecting the impact of a change in the disparate QA assets. The manual derivation of test cases, data, and automated tests undermines traceability to the system design, while the test artefacts are further not traceable to one another. The mountain of existing test assets must therefore be separately checked and updated as the system changes, repeating much of the effort of creating them in the first place. This is slow and laborious, and maintaining a valid regression pack can quickly consume whole iterations. Alternatively, invalid tests can be allowed to simply pile up, but then defects are thrown up due to invalid tests and not genuine bugs in the code.

These commonplace practices are antithetical to the principles of Continuous Testing set out in section one. They introduce siloes and manual effort at each stage, while pushing testing ever-further back and undermining re-usability. We now consider practical steps that can be taken to fulfil the above principles, creating full traceability and parallelism across an automated delivery pipeline.


© Curiosity Software Ireland Ltd. 2018

# "Shift left" with Model-Driven Development

Ambiguous, incomplete designs not only introduce the majority of defects to software development, they undermine collaboration between the Business and IT, while preventing automation further "right". Requirements must instead be housed in a format that is complete, unambiguous, and accessible to both BAs and engineering teams. That way, they can be leveraged *directly* in development and testing, without the bottlenecks and miscommunication associated with their manual translation across siloes.

Model-Driven Development offers one way to do this, aligning the BAs and engineering teams who can work in parallel from the same models. Flowchart modelling in particular has the advantage of being already familiar to requirements gatherers. Many BAs already use formats like Business Process Modelling Notation (BPMN), which can likewise be used to visually break a system's logic down into core cause-and-effect statements. Flowcharts therefore remain easy-to-use for those without engineering backgrounds, yet provide the formality needed to eliminate ambiguity and drive automated testing directly.

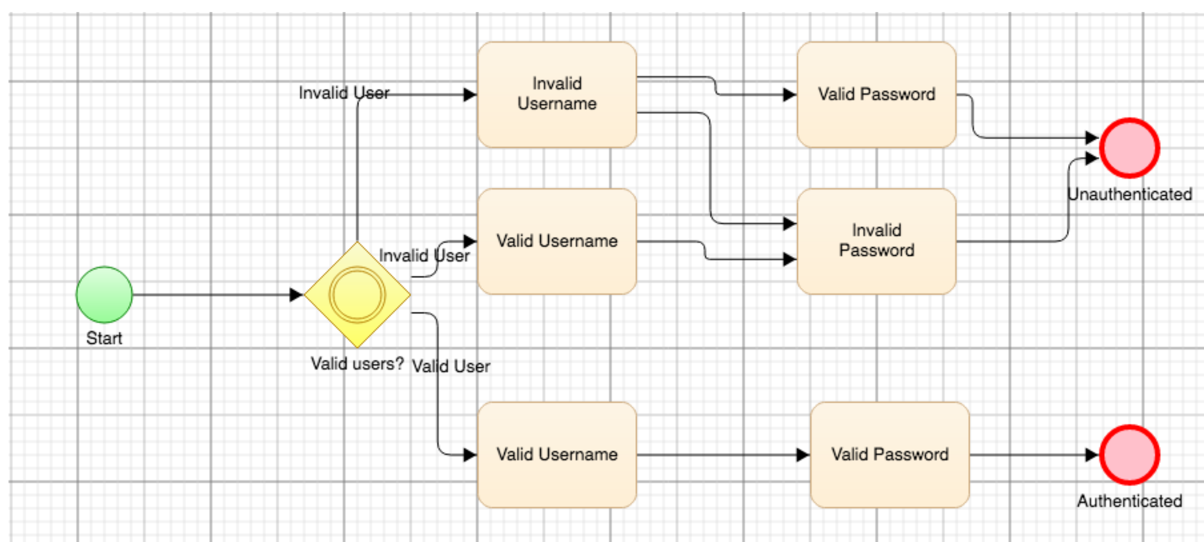


**Figure 2:** *A simplified model of a log-in screen. BPMN-style modelling is already familiar to requirements gatherers, and offers a way to unambiguously break a system down into its core cause-and-effect logic.*


© Curiosity Software Ireland Ltd. 2018

Such formal modelling during the design phase is further a form of Shift Left testing, doing the hard thinking earlier to eliminate the defects associated with ambiguous and incomplete requirements. Missing logic is more easily spotted in visual models, while automated algorithms can identify decision points with missing arrows. The ability to embed re-usable subprocesses within master flowcharts also enables formal dependency mapping of interrelated processes. The same models can then be provisioned to developers, providing clear and concise specifications of the system logic, with full mapping between the interdependent components:

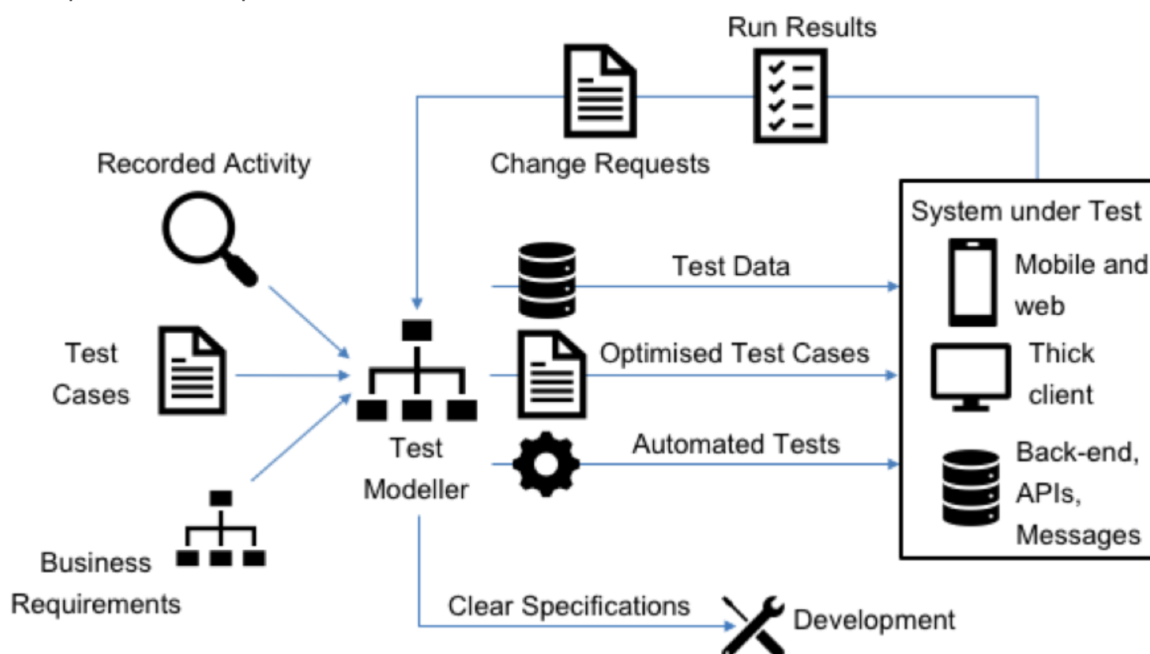


**Figure 3**: Formal, flowchart models can be rapidly built from existing tests and designs and recorded, and can drive testing and development directly.

A common objection to formal modelling is that it is not possible to completely model systems within short iterations. Complete requirements are instead seen as an outdated artefact of Waterfall approach, to be replaced by a constant stream of user stories and change requests. Methodologies like Behaviour-Driven Developments then appear to offer a quicker way to move from discrete statements of desired functionality to developing and testing them.

 © Curiosity Software Ireland Ltd. 2018 

Fortunately, this is not a mandatory choice: it is possible to have clear, unambiguous specifications, and still respond to change requests and new requirements in-sprint. Firstly, there are a range of accelerators that automate or increase the agility of formal modelling. The VIP Test Modeller provides a range of importers to convert written user stories and existing test cases automatically to formal models. This includes Gherkin scenarios for Behaviour-Driven Development:

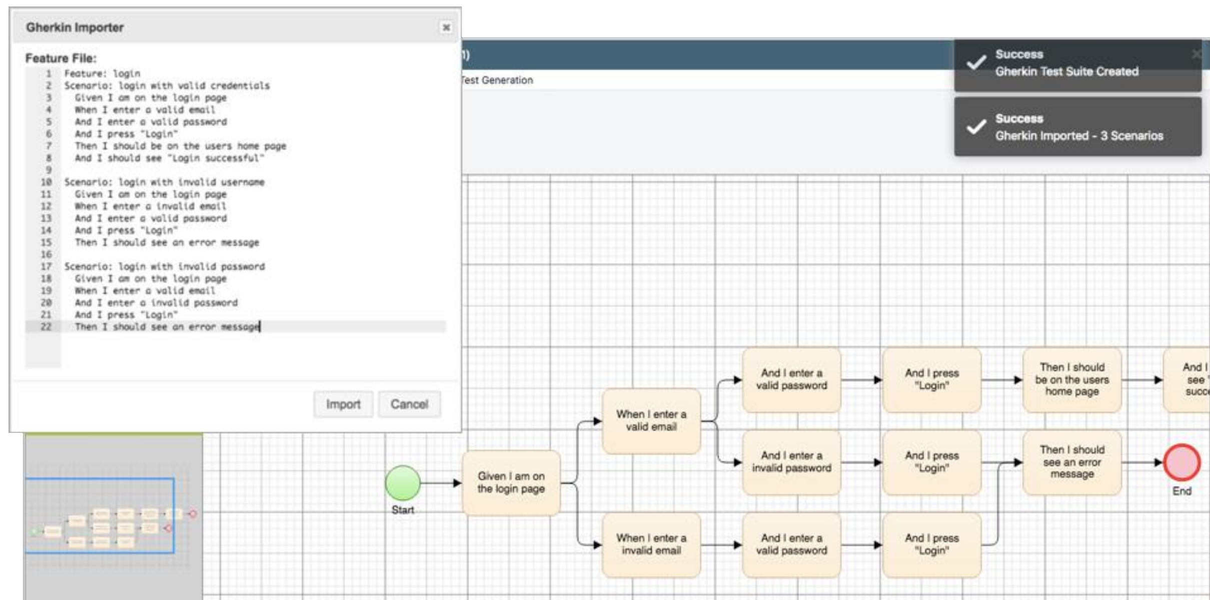


**Figure 4:** *Copying and pasting Gherkin Scenarios directly to formal models in The VIP Test Modeller.*

The VIP Test Modeller further provides a UI Recorder to convert automated or manual activity directly to flowchart models, working to reverse-engineer complete models from existing systems rapidly and pay off technical debt.

Secondly, the ability to automate the creation of maintenance of rigorous QA artefacts means that formal modelling can increase the ability to respond to fast-changing user needs, whereas disparate and unconnected scenarios and user stories can simply introduce further defects to test and development. The automation made possible in this "Shift Left, Shift Right" approach will be described in the next sections, turning now to the ways in which Model-Based Testing can automate and optimise TestDev tasks.


© Curiosity Software Ireland Ltd. 2018

# "Shift right" to automate and optimise TestDev

The mathematical precision of the flowchart models enables a "Shift Right" approach, having already improved the design through Shift Left QA in the design phase.

Formal models deliver value *directly* after the design phase, without requiring wholesale translation into new files and formats. Developers can code directly from the design, while testers can overlay additional data and logic to generate optimised testing assets automatically. This eliminates manual QA effort, while optimising testing to exercise the system logic housed in requirements systematically. All test assets are further aligned and traceable to a single asset, the model, meaning that updating the flowchart automatically maintains tests.

QA in turn becomes an automated comparison of how the system should work, housed in the design, to the system that has been developed in the code. The same designs from which code has been built are used to generate and maintain the assets needed to rigorously test fast-changing system: test cases, data, virtual services, and automated tests.

curiosity SOFTWARE IRELAND © Curiosity Software Ireland Ltd. 2018 vip

## Automated Test Case Design

QA teams can generate optimised test cases directly from the flowchart models, by virtue of their mathematical precision. The flowcharts serve as directed graphs, where the paths from their start to end nodes are equivalent to test cases. Automated graphical analysis can therefore identify the test cases contained in the model automatically, much like how a GPS identifies possible routes through a city street map:

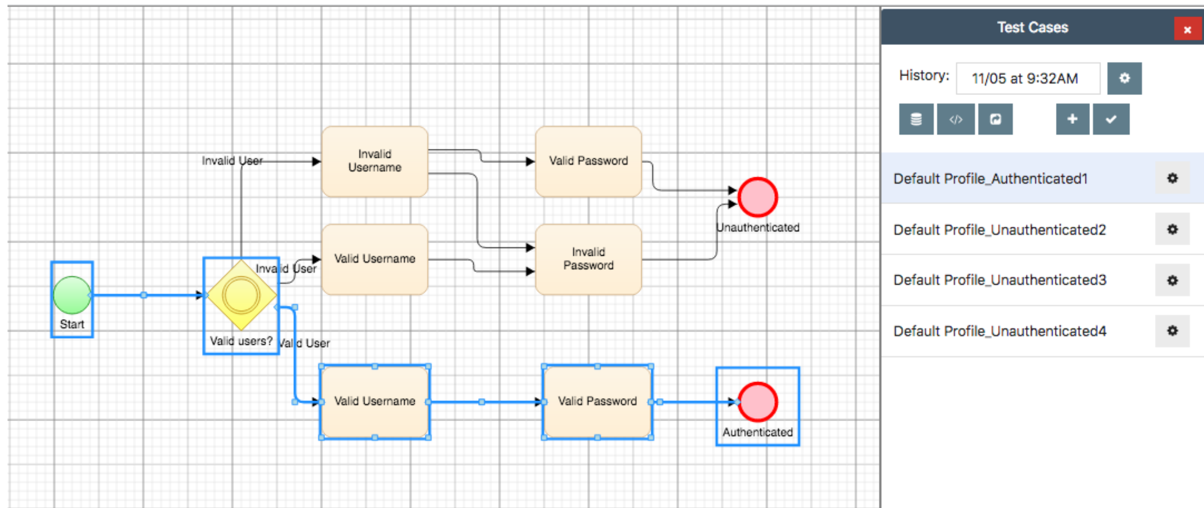


**Figure 5:** *Paths through the flowchart models are equivalent to test cases, that can be identified automatically.*

Automated test case generation not only eliminates slow and repetitive manual test creation, it also helps to ensure the rigour of testing. Automated coverage algorithms can generate test cases that exercise every path in the system model exhaustively. This tests all the logic housed in the system designs, with testing growing more rigorous as designs become more complete.

Coverage algorithms can similarly generate a set of optimised test cases to focus testing on new or critical functionality. This provides a reliable, risk-based approach to reducing the number of tests when there is not time to execute the high volume of tests associated with even simple systems. The VIP Test Modeller offers granular coverage-driven test generation, varying coverage by feature or subprocess. Test cases can target higher risk, higher visibility functionality, while exercising the surrounding logic as much or as in the remaining time.


© Curiosity Software Ireland Ltd. 2018

This Risk-Based approach generates the smallest set of test cases required to rigorously test important functionality, reducing test execution time without introducing negative risk. Test coverage is maximised where it matters, while wasteful over testing is avoided: the coverage algorithms will generate the smallest set of tests needed to exercise every important node, consolidating overlapping test steps into fewer test cases. Model-Based Testing therefore provides a reliable, requirements driven approach to identifying what to test.

The automated test design also maximises observability and the accuracy of feedback created by testing, by virtue of the link created between test and requirement logic. The test steps are equivalent to the logic gates and nodes in the cause-and-effect models. When tests fail, root cause analysis can therefore pinpoint the moment of failure in the nodes shared by several failing tests. A more exhaustive set of tests can further be generated to focus on the logic associated with failed tests, to maximise visibility and avoid the creation of false test results by false positives.

With Model-Based Testing, QA teams know that they got the right result, for the right reason. This maximises testing confidence, while defect reports highlight to developers *exactly* where in the system design a defect has most likely occurred, working to remediate defects as quickly and accurately as possible.


© Curiosity Software Ireland Ltd. 2018

## "Just in Time" Test Data

Testers can further specify test data at the model level, automatically generating data at the same time as test cases are created. This enables on demand, "just in time" access to compliant data with which to execute every possible test, avoiding the delays and impaired quality associated with slowly moving low-variety production data to QA environments.

Synthetic data generation offers the means to create rich, realistic data to execute every test, without the legislative risks associated with production data. The VIP Test Modeller provides over 500 combinable data functions, to quickly create all the data required for optimal test coverage. This includes the negative scenarios and outliers not available in production data, as well as data combinations needed to test new functionality:

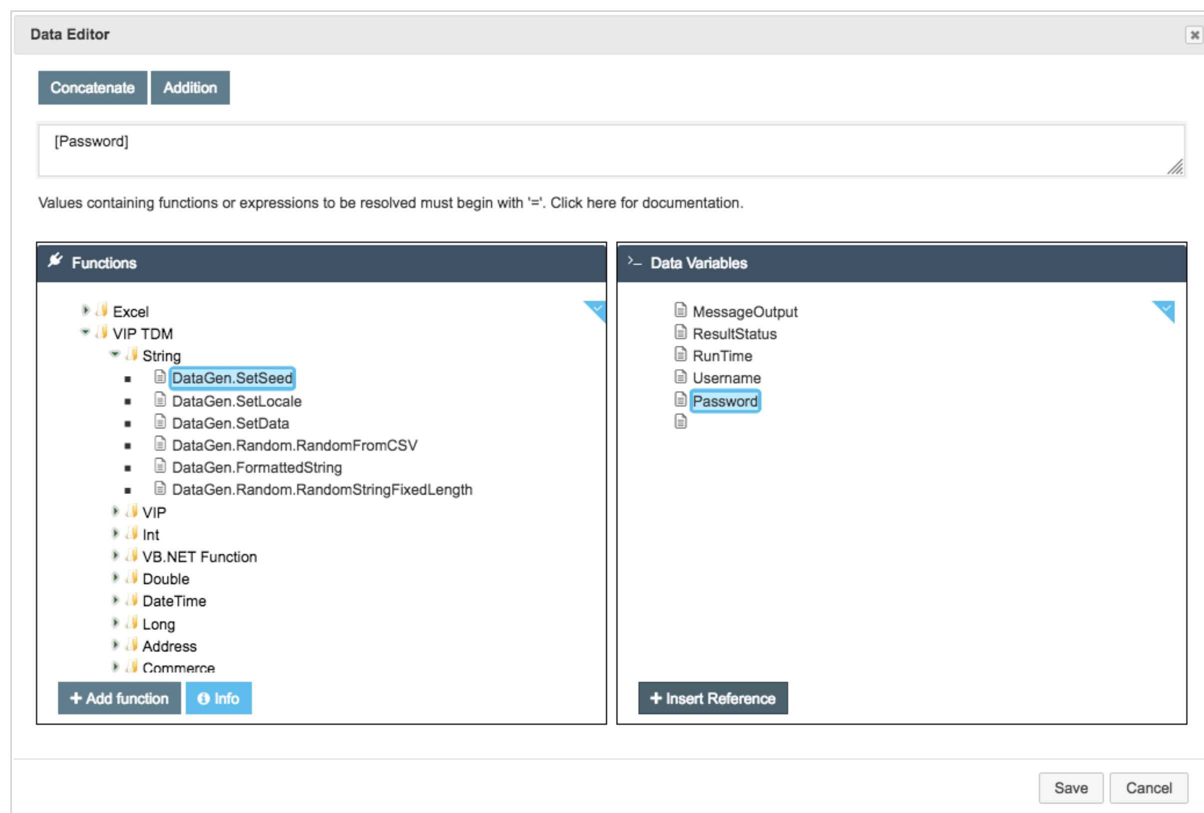


**Figure 6:** *500+ easy-to-use data generation functions create data variables needed to execute every test step.*


© Curiosity Software Ireland Ltd. 2018

The dynamic data functions can be defined directly in the nodes of the flowchart models, or can be figured in Excel for data-driven automation frameworks to consume:

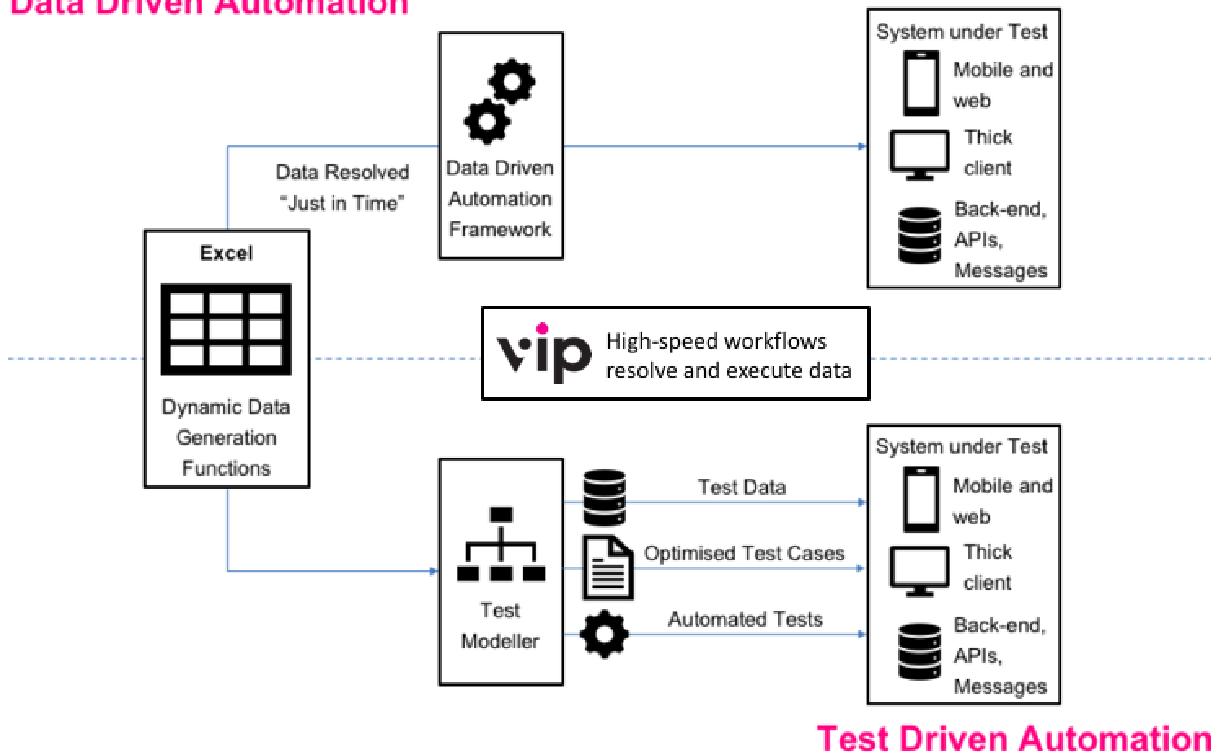


**Figure 7:** *The "just in time" data can be consumed directly by data-driven automation frameworks from Excel, or can be overlaid at the model level for Test Driven Automation.*

The functions are resolved "just in time" as tests are created or executed, and can further reference a range of connected databases and files. This draws on the most up-to-date system data during test execution, using automated combinatorial techniques to combine the values into new data sets for the latest test cases. Dynamic, "just in time" data resolution renders test data up-to-date by definition, providing QA with all the data they need for rigorous testing, when and where they need it.

 © Curiosity Software Ireland Ltd. 2018 

## On Demand Environments

The environments in which to execute the rigorous test cases and data can similarly be spun up from data defined at the model level.

With accurate and automated service or message virtualisation, this does not require complex stubbing, nor resource-intensive copies of complex systems. Instead, a service definition is automatically profiled and parsed, and converted into a dependency model of the Request and Responses associated with the service or API. The same sort of automated combinatorial techniques used to generate test cases and data are then applied, generating rich virtual data that contains every combination of Request and Response:

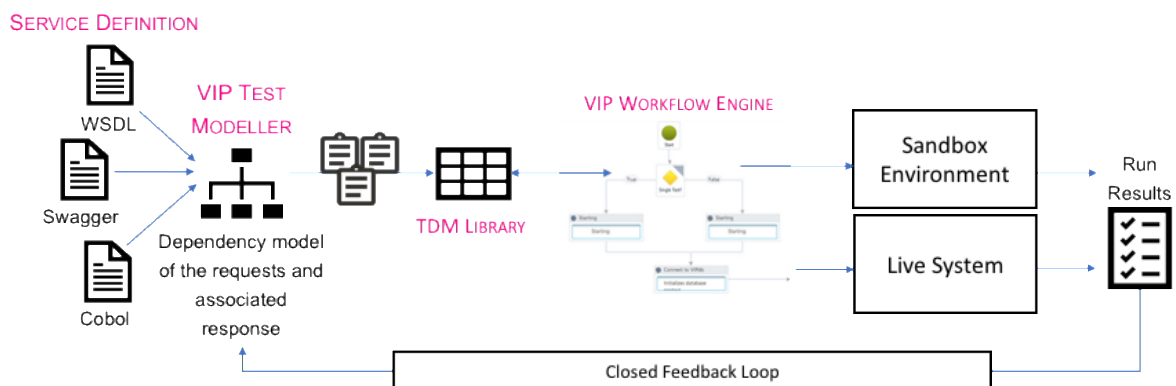


**Figure 8:** Comprehensive virtual data is quickly generated from parsed service definitions. It is moved automatically to live systems or sandbox environments.

The complete virtual data can be used to provide environments for rigorous functional testing, or individual combinations can be consumed by virtualisation tools for quick and accurate unit testing.

The same approach facilitates automated API testing, without the need to manually define complex tests and expected results from poorly documented, fast-changing services. The high-speed VIP workflow engine can exercise Requests generated from The VIP Test Modeller against a live system or sandbox environment, rigorously testing every route to a set of virtual end-points. The actual Responses are then compared to expected Responses that have been defined systematically form the model, automatically generating accurate run results.


© Curiosity Software Ireland Ltd. 2018

## Complete Test Automation

By this point, developers have been provided with clear and complete specifications from which to code, and QA have created the test cases, data, and environments needed to validate the developed system. All test assets have been generated directly from the model, keeping them aligned as they are generated from logical definitions of the latest system.

The same flowcharts can be used to generate automated tests, automatically comparing the developed system to the requirements models. The VIP Test Modeller offers a simple, fill-in-the-blanks approach to defining automation logic at the model level, with drop-down menus of available actions and data:

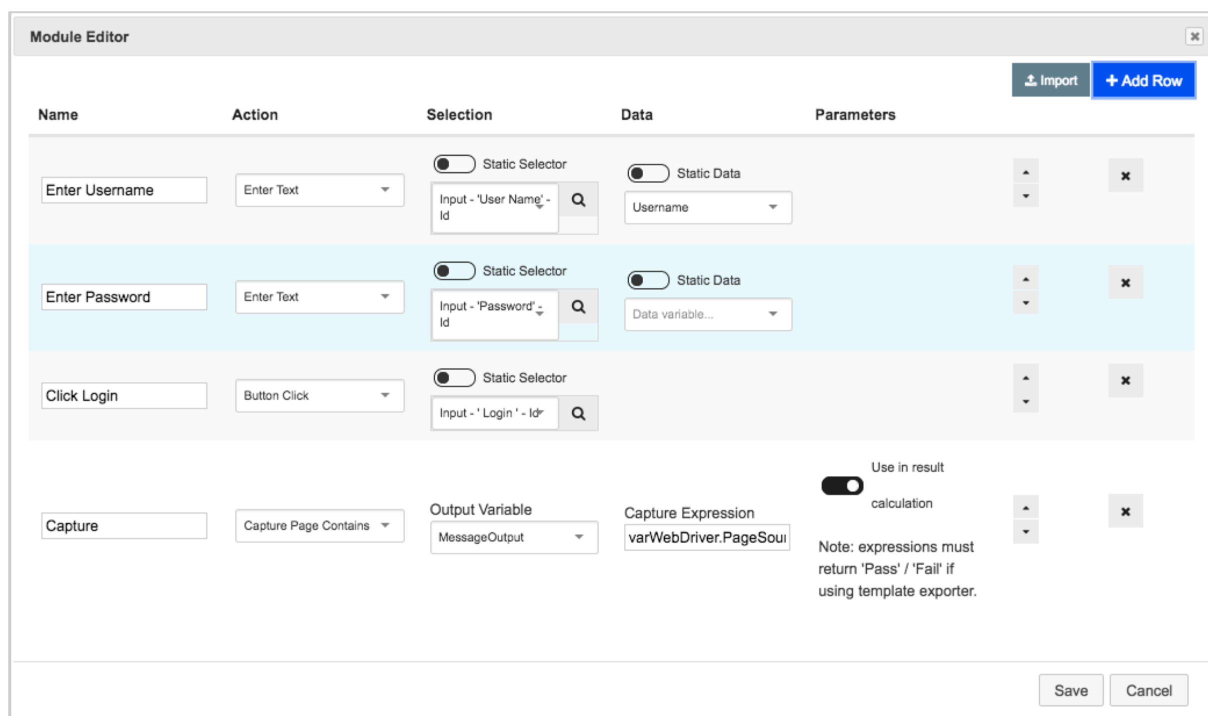


**Figure 9:** A scriptless, fill-in-the-blanks approach to defining automated test actions.

These actions are equivalent to the test steps in the automatically generated test cases, and are ordered to create automation modules. Each action has static or dynamic data associated with it, executing each test case automatically with the test data defined at the model level.


© Curiosity Software Ireland Ltd. 2018

A range of accelerators simplify the process, making test execution automation possible without slow and technical scripting. An object scanner automatically generates page object models from UIs, complete with the data and message activity needed to exercise the scraped elements. The UI Recorder similarly records all the information needed for automated testing, using the keyword, drop-down approach shown above to assign the scraped elements to models.

Manual testers and experienced automation engineers alike can in turn move from the system under test to automated testing in minutes, adopting a "low code" approach to eliminate the delays associated with manual scripting or fiddly keyword configuration. The automated testing further goes far beyond scriptless techniques like record and playback; by assigning re-usable automation logic to individual nodes in the model, the recorded activity and scraped logic can be combined and recombined to maximise test coverage.

The low code approach also maximises re-usability, a core principle of Continuous Testing. A central repository provides access to modelled components that already have automation logic, data, and test cases associated with them. QA teams can drag-and-drop the common functionality to new, master flowcharts, quickly assembling the system components into end-to-end models. Testing in turn becomes faster the more it is performed, working from the efforts of previous iterations to test new functionality rigorously.

Automated tests generated in The VIP Test Modeller additionally enable a high degree of granularity and observability, in spite of the simplicity of their creation. Mid-test assertions validate that the system is performing in accordance with the expected result *at every point*. This makes sure that the right end result has been reached for the right reason, in contrast to automated tests that have checks only at the end. Not only does this avoid the invalid test results created by false positives, it creates a trail of bread crumbs through the system logic when tests fail. Root cause analysis can then be applied, to trace this trail to the exact point in the system that is producing the automated test failure.

curiosity SOFTWARE IRELAND © Curiosity Software Ireland Ltd. 2018 vip

# Robotic Process Automation manages the TestOps

The Model-Based approach now set out automates and optimises test engineering or "TestDev" tasks: tasks that develop artefacts for testing new system logic. Test teams using this approach can work from accurate system designs to generate optimised test cases, data, virtual services, and automated scripts.

However, automating TestDev alone is not enough to achieve the end-to-end automation needed for Continuous Testing. It does nothing to remove the repetitious QA processes that surround test asset creation and execution. These rule-based processes generally focus on organisational practices and internal communication. They involve inputting testing metadata across a range of chat, email, and Application Lifecycle Tools. Typical "TestOps" tasks include:

1. Writing teams on chat tools like Slack as tests run or errors are found;
2. Emailing run results and test metrics to managers;
3. Inputting test steps into ALM tooling and updating test results;
4. Adding bug reports to JIRA;
5. Maintaining to do lists in shared sprintboards.

These are all invaluable tasks that are paramount to good cross-team collaboration and project management. However, they are largely auxiliary to the bread-and-butter of executing high quality tests against a system.

It is in this world of operations that Robotic Process Automation, or RPA, initially arose. RPA refers to the use of non-invasive bots that perform rule-based tasks otherwise performed by a human. This typically involves workflows that act in the background to rapidly execute repeatable tasks. Test automation and RPA harness similar technologies, and RPA is a growing trend in QA, as test teams increasingly attempt to automate tasks surrounding test execution.


© Curiosity Software Ireland Ltd. 2018

The goal of Robotic Process Automation then is to automate the broader tasks that facilitate high-speed, high coverage testing, with bots executing the repetitious heavy-lifting otherwise performed by test teams. More than simply mimicking *what* test teams do, effective RPA needs to perform the tasks across the technologies in use at an organization. This introduces the concept of "DevOps++", where existing technologies are connected by non-invasive workflows to keep data aligned across the whole delivery pipeline:

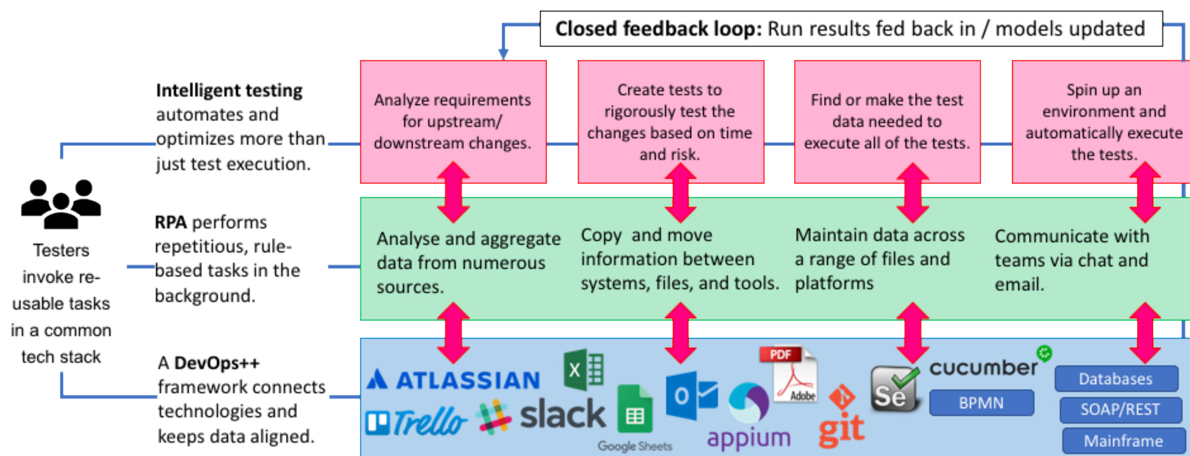


**Figure 10:** Robotic Process Automation mimics the tasks QA teams perform, across existing technologies.

The VIP Test Modeller integrates with VIP to combine optimised test automation with high-speed RPA. VIP is a fully connected, high-speed workflow engine that offers a comprehensive range of out-of-the-box connectors and API support. As tests are maintained in The VIP Test Modeller, VIP will therefore keep test metadata up-to-date across technologies, inputting test cases, data, and virtual services into Application Lifecycle Management tools and QA environments. VIP further executes automated tests generated in The VIP Test Modeller across distributed environments, updating run results in requisite fields across tools and providing email and chat updates.

VIP's connectivity enables it to maintain data across multiple platforms simultaneously. For instance, if test cases are stored in HP ALM but run results in JIRA, VIP will simultaneously store the latest test case information in HP ALM fields, but update the run results in the relevant JIRA tickets as tests are executed. This replaces the need for testers to repetitiously copy and edit data across numerous tools, allowing them to focus on developing test assets to test new and evolving logic.


© Curiosity Software Ireland Ltd. 2018

VIP similarly automates the internal communications that are necessary for a functioning test team, but can create repetitious processes. Email reporting and alerts can be set up, to notify managers or test teams of progress, while task boards can be automatically updated to assign tasks as tests are created and executed.

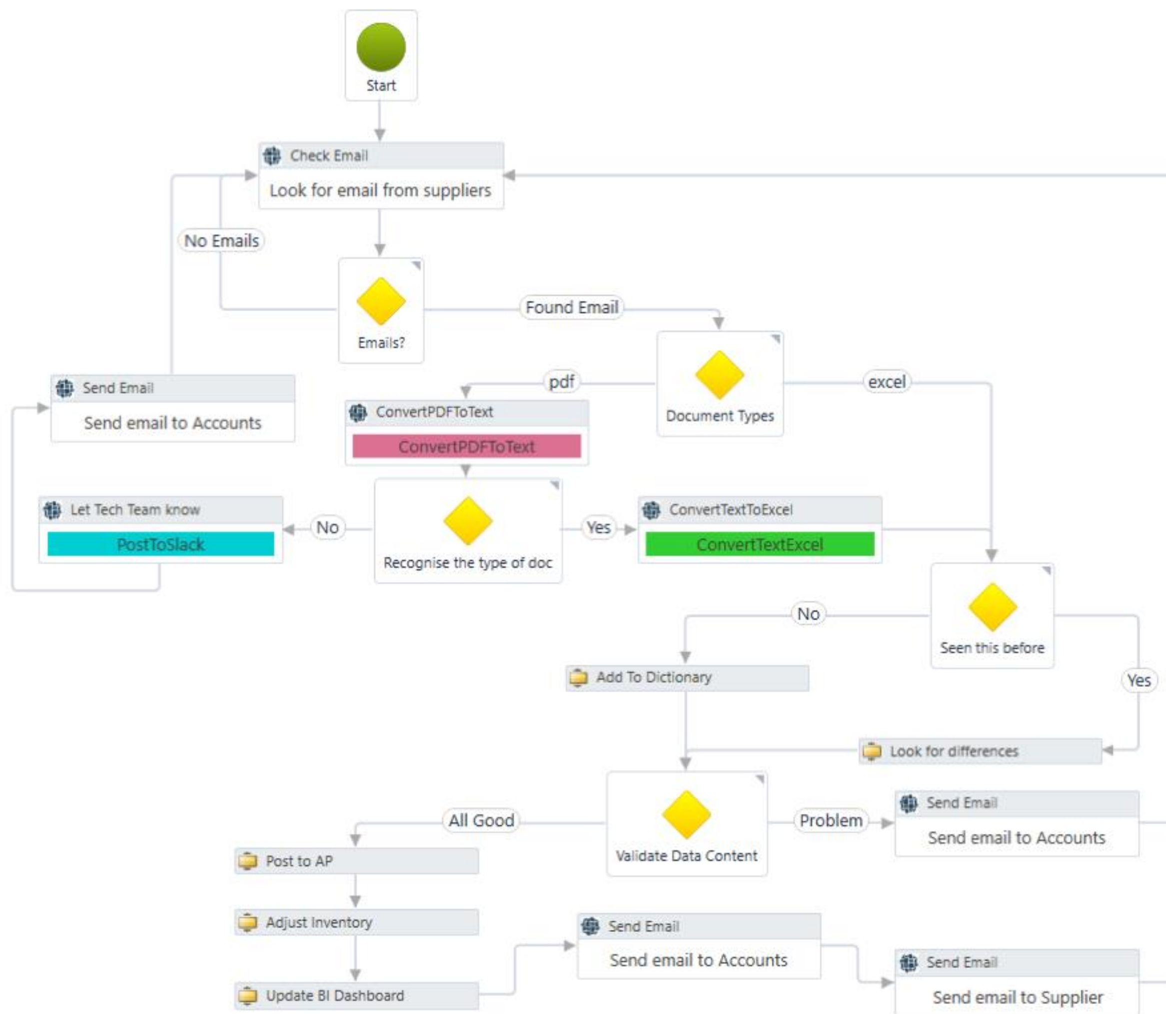


**Figure 11:** Re-usable, out-of-the-box workflows create high-performance bots that perform rule-based tasks.

The combination of RPA and end-to-end TestDev automation allows QA Teams more time to focus on testing new functionality at the start of each iteration. RPA handles the TestOps tasks that facilitate testing, while the TestDev tasks they perform are optimized and automated. Testers can produce higher quality test assets faster, invoking re-usable workflows from a common technology like Slack to perform TestOps tasks for them.


© Curiosity Software Ireland Ltd. 2018

The greater degree of connectivity between tools further aligns teams and technologies closely, avoiding siloes and the delays associated with them. Connecting existing technologies maximises their value and remove the need to manually convert information from one technology to the next. This not only reduces the manual effort required of testers, but improves the flow of information between systems.

DevOps++ accordingly enables a greater amount of metadata to be collected during testing, using this information in future to inform testing decisions. This drives up confidence in test results and enables evidentially informed QA, where new tests can be generated dynamically based on test results and history. Collecting sufficient metadata will further be a prerequisite to any form of artificially intelligent testing. This might include directly harnessing test results to auto-generate a new set of test cases that focus in on possible root causes of test failures, for instance.

curiosity SOFTWARE IRELAND © Curiosity Software Ireland Ltd. 2018 vip

# Accurate development and rigorous testing in-sprint

The "Shift Left, Shift Right" approach now set out enables test and development teams to react quickly to fast-changing systems. The traceability introduced by Model-Driven approaches automates maintenance of existing test assets, while the upstream and downstream impact of a change is identifiable from the formal models. QA and development teams can therefore focus on developing new functionality, updating only the parts of systems and test environments that have been impacted by a change.

Firstly, developers know exactly what needs to be updated when new user stories or change requests are incorporated into existing models at the start of an iteration. The full dependency mapping made possible by formally modelling the system makes it far easier to spot the impact across interdependent components reliably; a change made to one model will be reflected automatically in all master flows in which the updated functionality is a subprocess. Development teams can then focus on implementing the upstream and downstream impact of a change request on the existing system, working to avoid the defects and system outages associated with unforeseen consequences.

The automation of test maintenance meanwhile allows QA teams to validate any changes made to the system rigorously, during the same iteration as the change has been made. Generating automated tests, data, and virtual services from a single model creates a single, central QA artefact. Update the flowchart, and you maintain all your test assets in one fell swoop. New and rigorous test packs can be executed as systems evolve, quickly and rigorously testing any changes made to a system before it is released.


© Curiosity Software Ireland Ltd. 2018

# Continuous Testing: an ongoing process

Technologies and best practices can make Continuous Testing a reality today. A Model-Driven approach to designing, development, and testing systems not only fulfils the core principles of Continuous Testing, but it can eliminate the practical barriers to its adoption.

Formal modelling during the design phase facilitates close alignment between engineering teams and requirements gatherers, all of whom work from the same flowcharts. This collapses the once linear stages of "mini-Waterfalls", and the delays these siloes create. It further enables "Shift Left" testing during the design phase, working to eradicate requirements defects early and design systems that are both higher quality and testable.

Test and development teams can further work in parallel from the same models, adopting a "Shift Right" approach where rigorous test artefacts are derived directly from fast-changing systems. QA then becomes a largely automated comparison of the developed system to the user's desired functionality, accurately captured in the flowchart models that drive testing. Meanwhile, re-usability is maximised and the elimination of manual test maintenance further frees test teams to focus on new or critical functionality.

Robotic Process Automation can complete the automation of otherwise slow and manual processes, while connecting existing technologies maximises the flow of information between systems and the amount of feedback gathered. The greater degree of machine data and testing metadata collected in turn enables evidence-based testing, working from testing history to focus QA on new, critical, and high-risk functionality.

This approach to Continuous Testing is an ongoing process of uncovering more information about the system under test, feeding it back into the flowchart model with each iteration. The more complete the models become through such experimentation, the more rigorous the testing. Meanwhile, observability is maximised, allowing ever-more granular insights into the system to be harnessed, building better tests and better systems with each new iteration.


© Curiosity Software Ireland Ltd. 2018

# Find out more

Please visit the Curiosity website to discover more about practical solutions to adopting Continuous Testing. A range of hands-on demos are available on Youtube, if you would like to see these approaches working in practice.

If you would like to find out more, or arrange a demo, please do not hesitate to get in touch on info@curiosity.software, or using the details below.

## Stay up-to-date

Keep up-to-date with the latest Curiosity resources.

Follow us for news and resources.

Stay up-to-date with blogs and videos.

## Get in touch

Info@curiosity.software

Curiosity Software Ireland

Unit 6, The Mill

Bray

Co. Wicklow

Ireland

+353 1254 4350 (Ireland)

+1 914 297 7512 (USA)

TWITTER, TWEET, RETWEET and the Twitter logo are trademarks of Twitter, Inc. or its affiliates.


© Curiosity Software Ireland Ltd. 2018

# End Notes

[i] Margaret Rouse (2018), "DRY Principle", retrieved from https://whatis.techtarget.com/definition/DRY-principle on 05-12-18.
[ii] Winston W. Royce (1970), "Managing the Development of Large Software Systems", *Technical Papers of Western Electronic Show and Convention (WesCon)* (Los Angeles, USA).
[iii] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas (2001), "The Agile Manifesto". retrieved from https://agilemanifesto.org/ on 05-12-18.
[iv] Bender RBT (2009), *Requirements Based Testing Process Overview*, pp. 2, 16. Retrieved from http://benderrbt.com/Bender-Requirements%20Based%20Testing%20Process%20Overview.pdf on 05-12-18; Soren Lausen and Otto Vinter (2001), "Preventing Requirement Defects: An Experiment in ProcessImprovement", in *Requirements Engineering* (2001, 6:37-50), p. 38. Retrieved from http://www.itu.dk/people/slauesen/Papers/PrevDefectsREJ.pdf on 05-12-18; P Mohan, A Udaya Shankar, K JayaSriDevi (2012), "Quality Flaws: Issues and Challenges in Software Development", in Computer Engineering and Intelligent Systems (2012, 3:12:40-48), p. 44. Retrieved from www.iiste.org/Journals/index.php/CEIS/article/viewFile/3533/3581 on 05-12-18.
[v] P Mohan, A Udaya Shankar, K JayaSriDevi (2012), "Quality Flaws: Issues and Challenges in Software Development", in Computer Engineering and Intelligent Systems (2012, 3:12:40-48), p. 45. Retrieved from www.iiste.org/Journals/index.php/CEIS/article/viewFile/3533/3581 on 05-12-18; Bender RBT (2009), *Requirements Based Testing Process Overview*, p. 2. Retrieved from http://benderrbt.com/Bender-Requirements%20Based%20Testing%20Process%20Overview.pdf on 05-12-18.
[vi] Vanson Bourne and Panaya (2018), survey of over 300 IT decision makers in the UK and US. Cited from Islam Soliman (2018), "AI & automation vs humans: the future of software testing?", *DevOpsOnline* (16-11-18). Retrieved from http://www.devopsonline.co.uk/14159-2-ai-and-automation-vs-human-testers/ on 05-12-18.
[vii] *Ibid.*
[viii] KMS Technology (2018), survey of 135 developers and testers. Cited from Islam Soliman (2018), "Continuous Testing Survey Results Released", *Software Testing News* (04-12-18). Retrived from https://www.softwaretestingnews.co.uk/continuous-testing-survey-results-released/ on 05-12-18.
[ix] Dorothy Graham and Mark Fewster (2009), "That's No Reason to Automate", in *Better Software* (July/August 2009), p. 33. Retrieved from http://www.dorothygraham.co.uk/downloads/generalPdfs/NoReasonAut.pdf on 05-12-18.


© Curiosity Software Ireland Ltd. 2018