Towards Flexible and Automated Testing in Production Systems Engineering Projects

Dietmar Winkler
Christian-Doppler-Laboratory for Security and Quality Improvement in the Production System Lifecycle
Technische Universität Wien
Vienna, Austria
Dietmar.Winkler@tuwien.ac.at

Kristof Meixner
Christian-Doppler-Laboratory for Security and Quality Improvement in the Production System Lifecycle
Technische Universität Wien
Vienna, Austria
Kristof.Meixner@tuwien.ac.at

Stefan Biffl
Institute of Information Systems Engineering, Information & Software Engineering
Technische Universität Wien
Vienna, Austria
Stefan.Biffl@tuwien.ac.at

***Abstract* — Automated and systematic testing of automation systems (AS) and production systems (PS) require an integrated testing tool chain for test case development, execution and reporting. In practice, the test automation tool chain cannot be fully automated because of missing links between different tools used in the test automation process. Closing these gaps typically require (high) human effort. Furthermore, domain and software testing expertise is often bundled by one (expensive) engineer who is responsible for the application domain (reflected in use cases and test cases) and software tests (software test code). This paper presents a flexible Testing Automation Framework (TAF) that enables the configuration of test processes involving different tools and various layers for test automation and enables separated roles for the application domain and software tests. We build on best-practice test automation from Software Engineering and design a test automation process for the automation systems domain. We demonstrate the feasibility with a use case, derived from production systems automation, with selected tools covering all test automation layers. First results showed the feasibility of the framework in the evaluation use case making test processes more flexible and automated. Although the successful implementation of the TAF can support the efficient configuration and execution of test processes, there is additional effort for preparing the flexible and automated tool chain.**

Keywords—software and system testing, test automation framework, automation systems, production systems, test configuration, feasibility study.

I. INTRODUCTION

The growing share of software components in automation and production systems makes engineering projects more complex and risky and increases the need for systematic (and automated) software and systems testing [2]. Software testing refers to executing a pre-defined set of test cases with the intent to identify defects [17], i.e., identifying deviations of the software code and the specification and/or requirements. In Software Engineering, testing and test automation are common best-practices in a continuous integration and test environment [4]. Based on a *software code repository*, where developers commit their software code, *(automated) build processes* compile and build related releases and initiate *test processes*. Results of test processes are test reports for further usage, e.g., for defect repair and decision making. In the Automation Systems (AS) or Production System (PS) domain, these type of automated (testing) processes are still rare, often because of the involvement of complex hardware and/or missing tools that can support software tests. Although the testing of complex hardware might require real-time testing (out of scope of this paper) the configuration of test cases is still challenging.

In previous works we presented a framework for automated testing of AS based on test cases that could be derived from UML models [5][6][20] and on Keyword Driven Tests approaches [7]. However, human effort is required for test case definition and/or for model construction that can be used for test case generation. Furthermore, integrated tool chains within such frameworks strongly rely on selected tools that need to be configured and maintained – all changes, e.g., caused by tools that should be replaced or updated, require considerable effort.

Although in Software Engineering, test automation and test automation tool chains are established approaches and widely used in practice, in AS and PS, test automation is often not fully automated, because test automation tool chains include a set of limitations:

- *Need for specialized experts,* who have sufficiently high expertise in the application domain, i.e., in AS and PS engineering from a hardware perspective AND in the software engineering domain for software and systems testing, where software code is needed. Typically these experts are rare in an industry context and, thus, expensive. Thus, a goal is to provide a test automation framework that is capable of separating these roles for more efficient and effective testing processes.
- *Need for configuration capabilities:* Testing tool chains are often based on wired connections between defined and selected tools (on various test automation layers [5], e.g., for test management, test case design, execution, and reporting), with limited flexibility regarding changes, updates, and extensions. Changes can include exchanging selected tools in the tool chain or methods – typical requirements in project consortia where different project participants apply individual and customized tools or tool chains in their individual contexts.

The financial support by the Austrian Federal Ministry for Digital, Business and Enterprise and the National Foundation for Research, Technology and Development is gratefully acknowledged.

- *Gaps in the integrated tool chain.* Less flexible integrated tool chains can often lead to gaps in the testing tool chain that require human experts to bridge these gaps with considerable human effort. For instance if no specific tool is available for test case definition, test cases have to be written manually. Thus, mechanisms are required to support automation engineers to bridge these gaps.

To overcome these limitations, the goal of this paper is to design a flexible and configurable Test Automation Framework (TAF) that is capable of (a) decoupling test specific roles; (b) enable a comprehensive and integrated test automation tool chain that is efficiently configurable by domain experts and automation engineers; and (c) efficiently identify and overcome gaps in the test automation tool chain to enable efficient and effective test processes in ASE and PSE. We design a test process definition for test automation in the ASE and PSE domain and present a feasibility study for selected best-practice tools, derived from Software Engineering, applied in the ASE and PSE domain.

The remainder of this paper is structured as follows: Section II presents related work on automation systems and software test automation. Section III introduces research questions. We present the real-world use case in Section IV and the test automation framework / test automation process in Section V and present a feasibility study in Section VI. Section VII includes the discussion of the results. Finally, Section VIII concludes and identifies future work.

II. Background and Related Work

This section summarizes related work on automation systems and software test automation.

A. Automation Systems Engineering

In Automation Systems Engineering (ASE) and Production Systems Engineering (PSE) projects, over the last decades, an increasing share of functionality has shifted from mechanics or electrics to into software [3][19]. Among others, there is a growing importance of testing in ASE and PSE projects [18]. Ramler *et al.* [15] report on practical receipts and lessons learned in context of automated testing of industrial automation systems with focus on test driven and data driven testing. Furthermore, concepts of *Capture & Replay* can help to better create and execute test cases in the industrial automation domain [14] in a test automation context.

However, the application of best practices rely on a comprehensive test automation tool chain that covers different testing process steps, i.e., (1) Test Management, (2) Test Case Definition and Test Data Generation, (3) Test Execution, (4) Testing the *System under Test* (SuT), i.e., a physical system and/or a simulation, and (5) Test Reporting and Follow-up. Follow-up refer to defect repair and decision making based on test results, e.g., repeating a test runs. Fig 1a presents a basic testing processes according to Spillner *et al.* [17].

In the test automation process, key stakeholder roles include: *Test Managers* who are responsible for test planning, resource allocation, and decision making based on test reports; *Domain Experts* are typically responsible for deriving test cases for testing the SuT because they are aware of the technical requirements of the system; *Software Test Experts* are capable of writing software code as a pre-condition for test execution; and *Test Automation Engineers* prepare the Test Automation Framework (TAF). Typically, in industry practice several roles are embodied within one person, e.g., the domain expert has to define test cases (he knows what to test), write the test code (he knows how to test), and execute the test code manually. Thus, domain experts have to be an expert in several disciplines and often suffer from limited resources. Thus, a main challenge is to decouple test automation roles for better resource distribution and allocation *(C1: Role Concept)*.

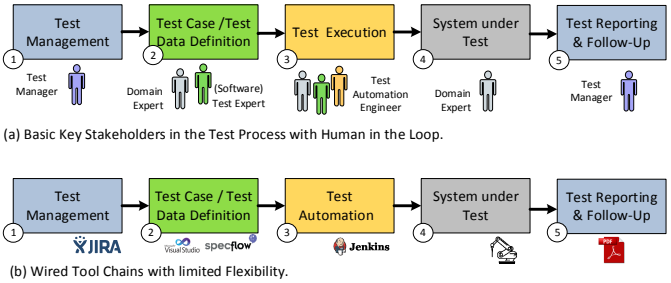

Fig. 1. Test Process Steps and Key Stakeholders: (a) human-supported tool chains and (b) wired tool chains with limited flexibility.

Beyond the need for experts, tool chains are often loosely or even not connected to each other and require a considerable human effort to bridge gaps in the test automation tool chain, e.g., for designing test cases or for executing test cases (Fig 1a). Thus, a second challenge focus on bridging gaps in the test automation tool chain for test process improvement *(C2: Human-in-the-Loop)*. If tool chains are available, they are often wired with limited flexibility regarding changes or extensions. Fig. 1b presents an example of a wired tool chain including *Jira*[1] for test management, *specflow*[2], a behaviour driven test case definition approach for test case derivation [22] and *Visual Studio*[3] for test code construction in C#, and *Jenkins*[4], a platform for automating build and test processes. Finally, test reports are generated, e.g., as PDFs. The SuT can be a physical component or a simulation, e.g., *Virtual or Digital Twins* [16] in context of virtual commissioning [9]. However, wired tool chains often lack in flexibility regarding the maintainability of the tool chain or the exchange of tools, e.g., in context of project consortia where different project partners apply different tool sets, or extending the capabilities of existing tool chains. Based on these limitations the third challenge focuses on overcoming limitations of wired tool chains to enable a flexible configuration of test automation tool chain settings *(C3: Flexible TAF)*.

For testing proposes, there is a need to create software test code based on requirements and the expected functional behaviour of the automation system. Several approaches address the generation of test cases based on models [21], e.g., UML models, for testing automation or production systems

[1] *Jira*: www.atlassian.com/software/jira
[2] *Specflow*: specflow.org/
[3] *Visual Studio*: www.visualstudio.com
[4] *Jenkins*: jenkins.io

[5][6][8][10] or based on CAEX [13]. Based on state-based (e.g., state charts) or behaviour-based models (e.g., sequence charts), test cases can be derived automatically. Anand *et al.* report on an orchestrated survey of methodologies for automated software test generation [1]. However, the resulting test cases need to be embedded within a testing framework, such as [15]. Therefore, a forth challenge is to embed software test code and/or test code generation approaches in the test automation tool chain *(C4: Test Code Management)*.

B. Software Test Automation

In Business Informatics the *Continuous Integration and Test Approach* [4] is an established approach in agile software development [11], where test runs are typically embedded within a software build process and, thus, represents the foundation for test automation.

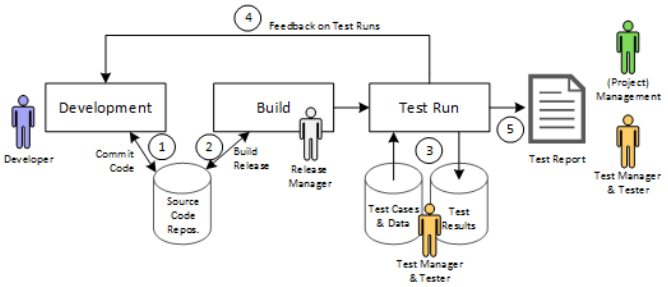

Fig. 2. Simplified Continous Integration and Test Process, derived from Business Informatics.

Fig. 2 presents a typical and simplified continuous integration and test process: (1) *Developers* commit newly constructed/generated software code into a code repository; (2) *Release Managers* initiate a build process (on purpose or on a regular basis, e.g., daily or nightly builds) including test runs and reporting; (3) *Test managers* and *Testers* prepare test scenarios, test cases and test data for test execution; (4) *Developers* receive feedback, e.g., defect reports and/or test reports, on the results of the test runs with focus on their committed piece of code; (5) finally, a test report is generated, used by different stakeholders for analysis and decision making. For instance, *Test Managers* and *Testers* can use test reports for providing evidence on the results of executed test runs, *Project Managers* can use test reports for analyzing the quality of the software code and for assessing the status of the project and *Business Managers* can use test reports to assess the overall progress of the project, e.g., based on coverage analysis.

The main goal of test automation in the automation systems domain is making automated tests more robust and maintainable [15]. *Data-Driven Tests* focus on the separation of test data and parametrized test scripts. Based on a *Test Automation Framework* (TAF) test scripts can be executed with a set of different data sets to identify defects more efficient and effective. *Keyword-Driven Tests* focuses on a separation of the low level interaction with the SuT (close to the physical system or simulation) from the testing logic, i.e., test cases and test scenarios. This separation supports the involvement of domain experts and users (who are familiar with the SuT) but might have low experience in test code construction [7]. *Behavioural-Driven Tests* [22] focus on test case design based on real-world use cases. Domain experts formulate test cases based on natural language in a *Gherkin Domain Specific Language (Gherkin DSL)* by following a standardized input sequence [12], i.e.,

```
GIVEN (x), WHEN (y), THEN (z).
```

While `<GIVEN>` describes pre-conditions and the test environment, `<WHEN>` focuses on the action under test, and `<THEN>` describes the expected outcomes of the test. Based on this notation, domain experts could use their natural language for defining the test cases without knowing details on the implemented test cases.

Although these concepts can enable the definition and execution of test cases in the automation systems and production systems domain, there is still the need for more flexible test processes and configurable testing tool chains that support test automation efficiently and effective.

III. Research Questions

Based on related work and discussions with industry partners, we identified a set of research issues:

RQ1. What are the needs for automated testing in the automation systems domain? More complex and software-intensive systems in Automation Systems Engineering (ASE) and Production Systems Engineering (PSE) require adapted test approaches, i.e., processes and frameworks. Furthermore, specialized testers should be unburdened by separating domain knowledge and test expertise. Finally, test processes need to be more flexible, supported by integrated and configurable tool chains in the test automation domain. Thus, RQ1 focuses on identifying requirements for a flexible test automation framework and testing process.

RQ2. How can a test automation concept be designed to support flexible and systematic testing? To overcome *hard-wired* test automation tool chains, a more flexible and configurable approach is needed from a tooling and process perspective. Thus, RQ2 focuses on the design of a test automation framework (TAF) that enables flexibility and configurability of tool chain settings and support systematic testing in ASE and PSE projects.

RQ3. What are the benefits and limitations of a test process based on a flexible test automation framework? RQ3 focuses on evaluating the test automation framework to identify benefits and limitations based on a real world use case and prototype implementation.

IV. Use Case Description

The main goals of this paper include (a) the identification of critical requirements of a flexible test automation framework (TAF) based on a typical scenario in the automation systems and production systems domain, (b) the definition of a flexible test automation framework that is capable of addressing and handling these requirements, and (c) the conceptual evaluation of the framework based on a real world use case.

Fig 3. illustrates the context of the evaluation, a simulation of a robot arm, a typical component in context of a production system in the automation system domain. Typical functional requirements of the robot arm include: (a) checking the current status and position of the robot arm, (b) moving the robot arm to the initial state or to a defined position, (c) grapping an item, moving the item to a defined position, and (d) releasing the item at the designated position.

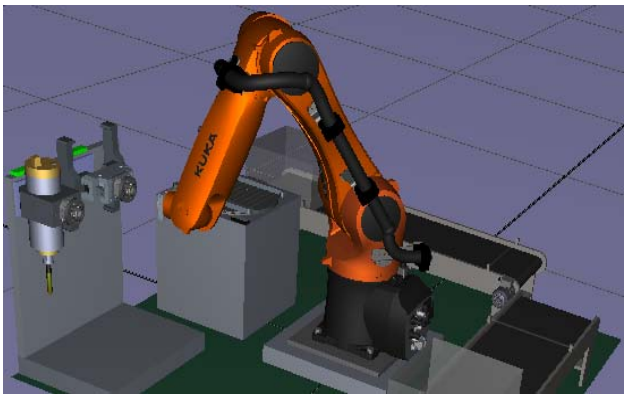

Fig. 3. Use Case: Robot Arm as Part of a Production System (Simulation).

In this paper we focus on the design and evaluation of the flexible test automation framework and its application without considering timing and real-time behaviour of the SuT, i.e., the real-world robot or a simulation. The authors are aware of timing and real-time constraints that need to be addressed. However, in context of this paper, these type of requirements are out of scope.

V. Flexible Test Automation Framework

This section summarizes critical requirements for flexible test automation and proposes the flexible test automation framework embedded within a test process, derived from Software Engineering.

A. Requirements for Flexible Test Automation

Based on related work, identified challenges (see Section II.A), and discussions with industry partners, we identified a set of critical requirements needed for flexible and configurable test automation. Table I presents a summary of Challenges (Cx) and identified critical requirements (Rx) and the relationship between them.

R1. Clear Role Definition. Challenge *C1* refers to the bundled roles of key stakeholders, e.g., domain experts, software test experts, and automation engineers who are responsible for test case definition and test case execution. Due to bundled roles, responsibilities, limited resources are hard to manage. Separated roles for domain experts and software test experts aim at better supporting individual and role-specific tasks, distribute the work load, and improves resource management.

R2. Configurable Tool Chains. To overcome limitations of strictly defined and wired test automation tool chains (or individually connected isolated tools), a flexible configuration approach can help to support modular and configurable tool chains across all steps of the test automation process. *R2* refers to Challenge *C2* and *C3* to support flexibility and maintainability in the test automation tool chain. Furthermore, changes in the test automation tool chain have to be addressed to support changing tools or methods along the test process, e.g., when new industry partners (with different test automation tool chains) should be included in the project. Finally, test code management (*C4*) should be addressed, e.g., new test code should be added by manually creating or generating software tests [1].

R3. Extensibility of Tool Chains. Challenge *C2* refers to the need to bridge gaps in the tool chain that need to be managed by humans with considerable effort. For instance, if no test automation platform is available, test cases have to be executed manually by experts; if no structured (and tool supported) approach is available for test case definition, human experts need to elicit appropriate test cases from requirements or the expected behaviour of the SuT manually. A flexible test automation framework (*C3*) should be capable of supporting test management more efficient and effective.

R4. SuT Management. In automation and production systems projects often two variants of the system are available, i.e., (a) the physical system including real hardware components or (b) *Digital Twins* [16] for simulation, testing and training purposes. Thus, the test automation framework has to be flexible to be capable of addressing both system types, e.g., during the commissioning phase. *R4* is connected to Challenge *C3*.

TABLE I. Overview on Requirements and Challenges.

Requirements \ Challenge	C1	C2	C3	C4
	Role Concept	Human-in-the-Loop	Flexible TAF	Test Code Management
R1: Clear Role Definition	X			X
R2: Configurable Tool-Chains		X	X	X
R3: Extensibility of Tool Chains		X	X	
R4: SuT Management			X	
R5: Method and Tool Support		X	X	X
R6: Test Code Management	X			X

R5. Efficient Tool and Methods Support. In industry practice, a wide range of different vendor specific tool solutions are available that could address one or more process steps of the test automation tool chain. For instance, *Jira* or *Polarion*[5] could be used for test management; *Keyword-Driven* or *Behaviour-Driven* test approaches could be used as test case definition approach. However, based on the project type or the testing context, a wide range of different methods and tools are available that could be used within the testing framework. It remains to the test manager or project manager to select the best-practice approaches in a given testing context. Nevertheless, the test automation framework should be capable of introducing different methods or tools in the existing test automation framework. This requirement is connected to *C2*, *C3*, and *C4*.

[5] *Polarion*: https://polarion.plm.automation.siemens.com/

R6. Efficient Test Code Management. Even if test scenarios and test cases are available, software test code need to be implemented to execute test cases automation-supported in the testing environment. Often the domain expert (*C1*) takes the additional role of a software test expert who implements the software code. However, a flexible test framework should decouple these roles by introducing interfaces for identifying the testing needs (based on scenarios) as input for software test construction and for providing / linking newly introduced software test code to related scenarios. Thus, the test automation framework should be capable of managing test code in an effective and efficient way (*C4*).

B. Flexible Test Automation Framework

Needs and requirements for a flexible Test Automation Framework (TAF) in the automation systems or production system domain represent the foundation for the proposed framework to address identified challenges.

Fig. 4 presents the proposed *Test Automation Framework* (TAF) covering four different layers of the test automation framework: *System under Test (SuT)* (Layer 1), *Test Automation* (Layer 2), *Test Case Definition* (Layer 3), and *Test Management* (Layer 4). Each layer can support several specialized tools (related to the designated layers) which could be available within the TAF application. Note that individual tools on each layer are illustrated by using boxes. For example, on Layer 4 two tools are available for Test Management, i.e., *Jira* and *Polarion*. Note that the selection of the tools are based on discussion with industry partners with focus on a concrete engineering project.

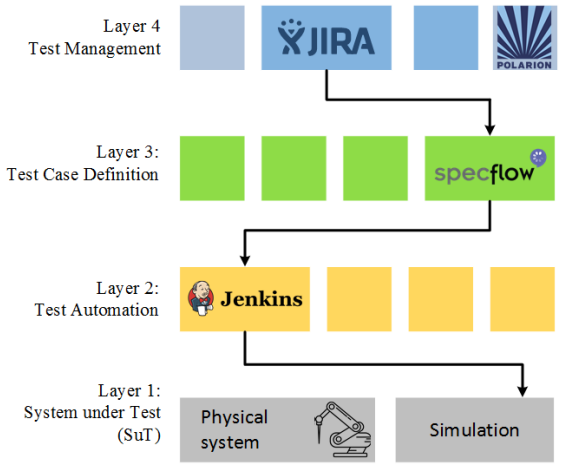

Fig. 4. Proposed Flexible Test Automation Framework (TAF).

The flexible configuration of (selected) tools is based on connections between different layers (e.g., between *Jira* and *Specflow*) based on defined and general interfaces that hold relevant information from the source tool (e.g., for test management) required by the destination tool (e.g., for test case definition). From an architecture point of view, related information is based on *common concepts* [21] as a simple and straight-forward approach for connecting different disciplines and tools. Note that the identification of common concepts and the implementation of general purpose interfaces needs expertise from the software, application, and knowledge engineering domain and require some effort for implementation [2]. In addition, the application of the framework requires some initial effort for configuration, i.e., for setting up the tool chain. Tool chain setup and configuration is typically executed by the test manager and/or the test automation engineer.

VI. Prototype and Conceptual Evaluation

Based on the flexible Test Automation Framework (TAF) and on Software Engineering (SE) Best-Practices, we have implemented a prototype with selected tools on all four test automation layers. Section VI.A presents the basic workflow based on the use case, and Section VI.B-VI.D present screenshots of the prototype implementation.

A. Testing Process based on the Test Automation Framework

Fig. 5 presents the implemented test process and prototype for testing the robot arm. The use case has been described in Section IV.

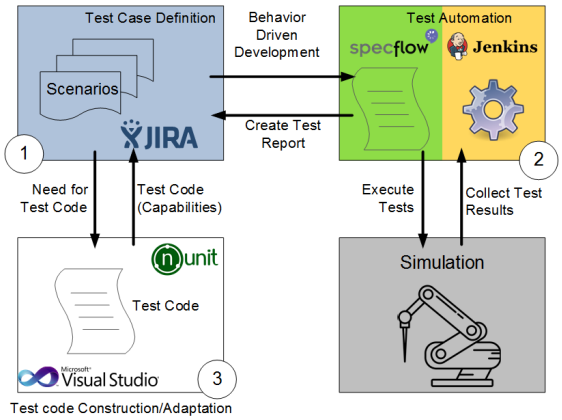

Fig. 5. Testing Process Prototype based on the Test Automation Concept.

Test Case Definition (Fig 5., Step 1): We selected *Jira*, a well-established issue tracking application in Software Engineering context and implemented a *Plug-In* that is able to configure test cases and scenarios based on the *Gherkin* language for *Behaviour Driven Testing*. Existing software test code, derived from a code repository such as *GitHub*[6], is analysed by the *Jira Plug-In* and available for selecting appropriate test case fragments. Note that the *Gherkin* language enables domain experts (who are not familiar with software test code) to design test cases and test scenarios based on the expected behaviour of the SuT (where the domain experts are familiar with). These language aspects, i.e., `GIVEN`, `WHEN`, `THEN` statements, are automatically linked to the software test code via *Specflow*.

Test Automation (Fig 5., Step 2): In a first step, *Specflow* uses defined test cases, derived from the previous test case definition step, links individual language aspects to existing software code and builds an executable piece of software for test case execution. The second step focuses on the test case execution with *Jenkins* that (a) initiates and executes the test

[6] *GitHub*: github.com

run on the SuT, on a simulation in this case and (b) collects test results from executed test runs. These test results are passed to the test management component, i.e,. *Jira*, to be visualized in the *Jira Plug-in*. Thus, domain experts or test managers receive fast feedback on their designed test cases in the *Test Management Software*.

Test Code Construction and Adaptation (Fig 5., Step 3). An open question is how the test code is generated and/or constructed and made available for test case definition in Step 1. A *Software Test Expert* is responsible for test case construction and/or generation and for maintaining the test code. He receives requests from *Test Managers, Domain Experts* and *Automation Systems Experts* (from Step 1) by following the *Gherkin* notation, develops test code, in *NUnit* in the prototype implementation, and provides test code to the *Test Manager*, *Domain Expert* and *Automation Systems Experts*. Note that providing additional test cases is not in the scope of this paper.

B. Prototype: Definition of Test Scenarios and Test Cases

In the conceptual evaluation we focus on a *Gripper Tool Test*, as part of the robot arm testing project. Thus, we start with the *Jira Plug-In* to configure the test case:

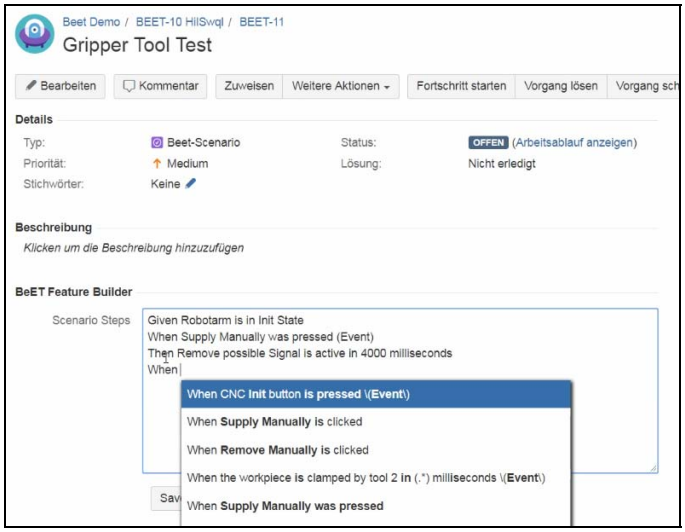

Fig. 6. Jira Plug-In Prototype for Test Case Definition based on Behaviour-Driven Testing with the Gherkin Language.

The first step (Fig 5., Step 1) includes the definition of the test scenario and test case, executed by the *Test Manager*, *Domain Expert*, or *Automation Systems Expert* based on a "natural language" test case respectively designed by using the *Gherkin* language. Fig 6. presents the screenshot of the *Jira* application and the *Jira Plug-In* for test case definition, where a *Test Case Issue* is defined. The *Jira Plug-In* BeET (Behaviour Enhanced Testing) includes a component for configuring the test scenario, i.e., a sequence of `GIVEN - WHEN – THEN` statements. Note that individual and available software test cases have been made available beforehand by the *Software Test Expert* for selection and mapping to the defined test sequence. In the example the statement *"Robotarm is in Init State"* has been selected as pre-condition (`GIVEN`) for the test case, followed by a sequence of `WHEN`, `THEN` statements. See Fig 6 for a test case example based on the *Gherkin* language. Output of this process step is a *Specflow* configuration file that is created and made available in the test environment, i.e., a dedicated input file for the testing framework, supported by *Jenkins*.

C. Prototype: Mapping of Test Code to Specflow

The second process step (Fig 5., Step 2) uses the *specflow* configuration file and links all aspects of the *Gherkin* language to existing software test cases. Fig 7 presents a screenshot of the software test code that is linked to the *specflow* configuration file entry related to *"Robotarm is Init State"*.

```
[Given(@"System is in initial State")]
public void GivenSystemIsInInitState()
{
    resetAllToZero();
    double axis1 = 0, axis2 = 0, axis3 = 0, axis4 = 0, axis5 = 0, axis6 = 0, cncInit = 0;

    int res = CheckState(axis1, axis2, axis3, axis4, axis5, axis6, cncInit);

    if (res != 0)
        Assert.IsTrue(AssertIntEqual(0, 1, "Idelmode?"), "AssertionException: the system is not in idle state");

    AssertIntEqual(0, CheckState(axis1, axis2, axis3, axis4, axis5, axis6, cncInit), "Idle State");
}

[When(@"the robotArm is moved to (.*) on (.*)")]
public void robotArmMovesbyOnAxis(double value, String axis)
{
    double temp = OPCClient.Read(axis);
    double oldValue = temp;
    double steps = 50;
    double offset = (temp - value) / steps;
    for (int i = 0; i < steps; i++)
    {
        temp = temp - offset;
```

Fig. 7. Related Test Source Code derived from the Code Repository.

Note that the output of this process step is a binary file that holds the executable source code for *"Gripper Tool Test"* scenario. Also note that missing software test code cannot be selected by the Test Manager and will not appear in the binary. However, missing software test code initiates a request to the *Software Test Expert* for providing the related test case. For notification purposes we used the concept of *Jira Issues* for informing related engineers to extend/maintain the software test code repository.

D. Prototype: Automated Execution based on the Simulation

Following the *Continuous Integration and Test* approach (see Section II.B and [4] *Jenkins* is used to build and test the SuT, i.e., the robot arm simulation in the evaluation use case. Fig 8. presents a snapshot of the stage view of the *Jenkins build and test execution process*, including time consumption and the current state of the test process. Note that often test managers are not interested in the progress of the test process but on the results. Thus, a progress bar is not included in the test management view (i.e., in *Jira*).

Beyond test progress monitoring, *Jenkins* collects test results and delivers individual results back to the *Test Case Issue* in *Jira*. Fig 9. Presents the summarized results of the executed test case. In the example the *"GripperToolTest"* failed as a regression test in the continuous integration and test framework. Note the *Jira Plug-In* provides an overview on the test results (with focus on individual *Test Case Issues* in *Jira*. More detailed information (enabled via Double-Click on the results) is available in the *Jenkins* Test Report.

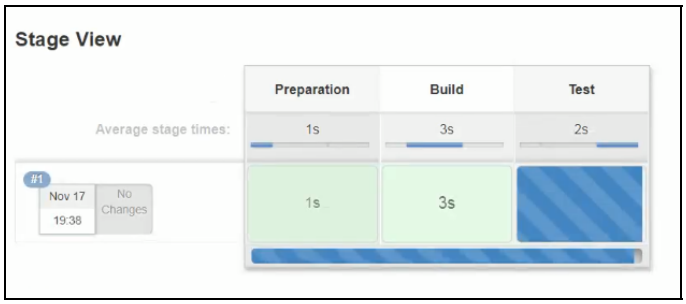

Fig. 8. Progress of the Test Automation Process (*Jenkins* View).

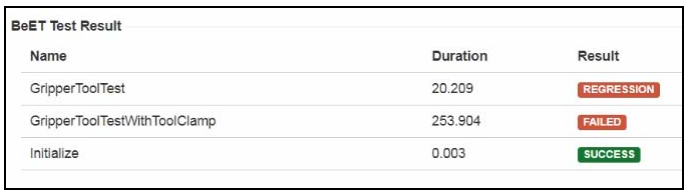

Fig. 9. Summarized Results of Executed Test Cases in Jira.

Note that individual test runs can be started via the *Jira Plug-In* manually or by configuration in *Jira* and/or *Jenkins* on a regular basis, e.g., during daily or nightly builds.

VII. Discussion

Based on the ongoing need for efficient and flexible tests in automation and production systems engineering projects we have identified a set of challenges and requirements for a flexible test automation framework (RQ1), proposed a TAF that could address these needs (RQ2) and conceptually evaluated the proposed framework in a real world use case based on simulating the behaviour of a robot arm (RQ3).

RQ1. What are the needs for automated testing in the automation systems domain? Based on related work and industry partner needs we identified four main challenges (Cx) in test automation for automation and production systems and identified six critical requirements (Rx). Main results include (a) modified role concepts (C1) that is needed to separate roles for test case definition, i.e., test managers, domain experts, automation systems engineers, or non-software experts should be able to define test cases based on the expected outcome of the SuT and monitor the actual outcome of the test run, while software test experts (C4) provide and manage testing source code; and (b) requirements for a flexible TAF that support overcoming human-in-the-loop activities (C2) and configuration and extension activities of the TAF itself (C3). Section V.A. and Table I summarizes the challenges and derived requirements.

RQ2. How can a test automation concept be designed to support flexible and systematic testing? In Section V.B we presented the proposed Test Automation Framework (TAF) to enable flexible configuration of the testing framework on four different layers along the testing process. Main contributions include the flexibility to configure the TAF according to individual project needs, e.g., if a defined tool set is available, or to bridge human-in-the-loop activities if no comprehensive tool chain is available. The prototype implementation of the TAF focuses on a real world use case to show the feasibility of the approach based on selected tools, derived from Software Engineering, applied to the automation systems domain. We applied the behaviour-driven test approach by using the *Gherkin* language to support domain experts and non-software test experts in designing, and executing test cases. The separation of roles, i.e., domain experts and software test experts allows to focus on individual tasks for test case definition (domain expert) and software test construction (software test expert). The *Jira* Plug-In supports the overall test process on important test automation layers.

RQ3. What are the benefits and limitations of a test process based on a flexible TAF? To identify strengths and limitations of the TAF and the test process we conceptually evaluated requirements in context of the proposed approach and traditional manual and wired test automation tool chains. Table II presents the overview of the conceptual evaluation.

TABLE II. Comparison of Manual Testing, Wired Tool Chains, and the Flexible Test Automation Framework wrt. Requirements (- less supported, o neutral support, + well supported)

Requirements	Test Automation Process Variants		
	Manual Testing	Wired Tool Chain	Flexible Tool Chain
R1: Clear Role Definition	-	O	+
R2: Configurable Tool-Chains	O	-	+
R3: Extensibility of Tool Chains	O	-	+
R4: SuT Management	O	-	+
R5: Method and Tool Support	O	O	O
R6: Test Code Management	O	O	+

TABLE III. Comparison of Manual Testing, Wired Tool Chains, and the Flexible Test Automation Framework wrt. Setup and Application Requirements (- less supported, o neutral support, + well supported)

Requirements	Test Automation Process Variants		
	Manual Testing	Wired Tool Chain	Flexible Tool Chain
R7: Initial Implemenation Effort	O	-	-
R8: Maintaining the test approach	O	-	+
R8: Single Application	O	O	+
R9: Frequent Test Runs	-	O	+

For manual test automation, the flexibility of the test approach strongly depends on individual engineers and their capabilities. Therefore, the assessment is set to neutral support. The flexibility of wired tool chains is very limited. Thus, changing individual aspects of the testing approach requires a considerable high effort. The flexible Tool Chain (TAF) supports almost all requirements well. However, the introduction of new methods and tools might require initial

effort for implementation. Therefore, we included four additional requirements in the evaluation (see Table III) concerning (a) setup and maintenance the test automation concept; (b) single application of test runs; and (c) frequent test runs as best-practice in test automation.

While the initial implementation (R7) effort is considerable high for tool chains, maintenance (R8) include a high effort for wired tool chains but less effort for the flexible TAF. While manual test configurations have strong weaknesses regarding frequent test runs (R8 and R9), wired tool chains and flexible tool chains perform similar in a defined setting. Concerning the flexible configuration of test cases, the proposed process approach outperforms wired tool chains.

VIII. Conclusion and Future Work

In this paper we presented challenges and requirements for flexible test automation in the automation systems and production systems domain. Furthermore, we proposed a flexible test automation framework (TAF), presented a pilot implementation of the TAF with selected tools, and conceptually evaluated strength and limitations of the approach. First results showed the feasibility of the testing approach in the evaluation use case to improve test automation processes in the automation systems domain. Although the implementation of the TAF supports efficient configuration and execution of test processes, additional effort is required to initially setup the test automation framework.

Limitations and future work. This paper focuses on a flexible test automation framework and a conceptual evaluation. Thus, the selected tool set is limited and need to be extended as future work. Main goal is to provide a tool box on every layer of the test automation framework to enable efficient and effective configuration capabilities for application in the automation systems domain. The evaluation use case focuses on a real-world but small-scale component, a robot arm, that might have a limited complexity. Future work will include large-scale evaluations in industry settings, involving simulations and physical systems to investigate strength and limitations of the approach. Finally, future work will also focus on (a) improving the framework concepts, (b) extending the TAF tool box towards a variety of tools and methods relevant in the test automation process of ASE and PSE, and (c) include test case generation approaches as foundation for deriving test cases from models.

Acknowledgment

The financial support by the Austrian Federal Ministry for Digital, Business and Enterprise and the National Foundation for Research, Technology and Development is gratefully acknowledged.

References

[1] Anand S., Burke E.K., Chen T.Y., Clark J., Cohen M.B., Grieskamp W., Harman M., Harrold M.J., McMinn P.: “An orchestrated survey of methodologies for automated software test generation”, In: Journal of Systems and Softare (JSS), 86(8), pp. 1978-2001, Elsevier, 2013.

[2] Biffl S., Sabou M. (Eds.): “Semantic Web Technolgies for Intelligent Engineering Applications”, Springer, 2016.

[3] Broy M.: “The ‘Grand Challenge’ in Informatics: Engineering Software-Intensive Systems”, In: IEEE Computer, 39(10) pp. 72–80, 2006.

[4] Duvall P.M., Matyas S., Glover A.: “Continuous Integration: Improving Software Quality and Reducing Risk”, Addison-Wesley, 2007.

[5] Hametner R., Winkler D., Östreicher T., Surnic N., Biffl S.: “Selecting UML Models for Test-Driven Development along the Automation Systems Engineering Process“, In Proc. of the 2010 IEEE Int. Conf. on Emerging Techn. and Factory Automation (ETFA), WIP, Spain, 2010.

[6] Hametner R., Kormann B., Vogel-Heuser B., Winkler D., Zoitl A.: “Test Case Generation Approach for Industrial Automation Systems”, In Proc. of the 5th Int. Conf. on Automation, Robotics and Applications (ICARA), New Zealand, 2011.

[7] Hametner R., Winkler D., Zoitl A.: “Agile Testing Concepts Based on Keyword-Driven Testing for Industrial Automation Systems”, In: Proc. of the 38th Annual Conf. of the IEEE Industrial Electronics Society (IECON), Canada, 2012.

[8] Hussain T., Frey G.: “UML-Based Development Process for IEC 61499 with Automatic Test Case Generation”, In: Proc. of the 2006 IEEE Conf. on Emerging Techn. and Factory Automation (ETFA), Czech Republic, 2006.

[9] Hoffmann P., Schumann R., Maksoud T.M., Premier G.C.: “Virtual Commissioning of Manufacturing Systems A Review and New Approaches for Simplification”, In: Proceedings of the 24th European Conf. on Modelling and Simulation (ECMS), pp. 175-181, 2010.

[10] Iyenghar P., Pulvermüller E., Weterkamp C.: “Towards Model-Based Test Automation for Embedded Systems using UML and UTP“, In: Proc. of the 2011 IEEE Conf. on Emerging Techn. and Factory Automation (ETFA), France, 2011.

[11] Martin R.C.: “Agile Software Development, Principles, Patterns, and Practices”, Pearson Education, 2013.

[12] Micallef M., Colombo C.: “Lessons learnt from using DSLs for automated software testing” In: Proc. of the 8th Int. Conf. on Software Testing, Verification and Validation Wsh (ICSTW), pp.1-6, IEEE, 2015.

[13] Petola J., Sierla S., Aarnio P., Koskinen K.: “Industrial Evaluation of functional Model-Based Testing for process control applications using CAEX“, In: Proceedings of the 2013 IEEE Conf. on Emerging Techn. and Factory Automation (ETFA), Italy, 2013.

[14] Prahofer H., Schatz R., Wirth C., Mossenböck H.: “Deterministic Replay Debugging of IEC 61131-3 SoftPLC Programs”, In: Proc. of the 8th Int. Conf. on Industrial Informatics (INDIN), 2010.

[15] Ramler R., Putschögl W., Winkler D.: “Automated Testing of Industrial Automation Software: Practical Receipts and Lessons Learned”, In: Proc. of the 1st Wsh on Modern Software Engineering Methods for Industrial Automation (MoSEMInA), in conf. with the 36th ICSE conference, India, 2014.

[16] Rosen R., von Wichert G., Lo G., Bettenhausen K.D.: “About the Importance of Autonomy and Digital Twins for the Future of Manufacturing“, In: IFAC-PapersOnLine, 48(3), pp. 567-572, 2015.

[17] Spillner A., Linz T., Schaefer H.: “Software Testing Foundations: A Study Guide fort he Certified Tester Exam”, Rocky Nook, 4th ed., 2014.

[18] Vogel-Heuser B., Fay A., Schäfer I., Tichy M.: „Evolution of Software in Automated Production Systems: Challenges and Research Directions”, In: Journal of Systems and Software, 110, pp. 54-84, 2015.

[19] Vyatkin V.: “Software Engineering in Industrial Automation: State-of-the-Art Review” In: IEEE Trans. on Ind. Inf. 9(3), pp. 1234-1249, 2013.

[20] Winkler D., Hametner R., Östreicher T., Biffl S.: “A Framework for Automated Testing of Automation Systems“, In: Proc. of the 2010 IEEE Int. Conf. on Emerging Techn. and Factory Automation (ETFA), WIP, Spain, 2010.

[21] Winkler D., Sabou M., Biffl S.: “Improving Quality Assurance in Multi-Disciplinary Engineering Environments with Semantic Technologies” In: L.D. Kounis “Quality Control and Assurance – An Ancient Greek Term ReMastered”, Book Chapter, Chapter 8, pp. 177-200, 2017.

[22] Wynne M., Hellesoy A., Tooke S.: “The cucumber book: behaviour-driven development for testers and developers”, Pragmatic Bookshelf, 2017.