Advanced Software Engineering

21

Developing and testing model-based software securely and efficiently

Prof. Dr. Ina Schieferdecker · Dr. Tom Ritter
Fraunhofer Institute for Open Communication Systems FOKUS

Summary

Software rules them all! In every industry now, software plays a dominant role in technical and business innovations, in improving functional safety, and also for increasing convenience. Nevertheless, software is not always designed, (re) developed, and/or secured with the necessary professionalism, and there are unnecessary interruptions in the development, maintenance, and operating chains that adversely affect reliable, secure, powerful, and trustworthy systems. Current surveys such as the annual World Quality Report put it bluntly, directly correlated with the now well-known failures of large-scale, important and/or safety-critical infrastructures caused by software. It is thus high time that software development be left to the experts and that space be created for the use of current methods and technologies. The present article sheds light on current and future software engineering approaches that can also and especially be found in the Fraunhofer portfolio.

21.1 Introduction

Let us start with the technological changes brought about by the digital transformation which, in the eyes of many people, represent revolutionary innovations for our society. Buildings, cars, trains, factories, and most of the objects in our everyday lives are either already, or will soon be, connected with our future gigabit society via the ubiquitous availability of the digital infrastructure [1]. This will change information sharing, communication, and interaction in every field of life and work,

Reimund Neugebauer, *Digital Transformation*
https://doi.org/10.1007/978-3-662-58134-6_21

be it in healthcare, transport, trade, or manufacturing. There are several terms used to describe this convergence of technologies and domains driven by digital networking: the Internet of Things, smart cities, smart grid, smart production, Industry 4.0, smart buildings, the Internet of systems engineering, cyber-physical systems, or the Internet of Everything. Notwithstanding the different aims and areas of application, the fundamental concept behind all of these terms is the all-encompassing information sharing between technical systems – digital networking:

Digital networking is the term used to refer to the continuous and consistent linking of the physical and digital world. This includes digital recording, reproduction, and modelling of the physical world as well as the networking of the resulting information. This enables real-time and semi-automated observation, analysis, and control of the physical world.

Digital networking facilitates seamless sharing of information between the digital representations of people, things, systems, processes, and organizations and develops a global network of networks – an *inter-net* – that goes far beyond the vision of the original Internet. But this new form of network is no longer a matter of networking for its own sake. Instead, individual data points are combined into information in order to develop globally networked and networkable knowledge and utilize this both for increasing understanding as well as for the management of monotonous or safety critical processes.

In light of this digital networking, the central role of software continues to increase. Digital reproductions – the structures, data, and behavioral models of things, systems, and processes in the physical world – are all realized via software. But so are also all of the algorithms with which these digital reproductions are visualized, interpreted, and reprocessed, as well as all of the functions and services of the infrastructures and systems such as servers and (end) devices in the network of networks. Until recently the essential characteristics of the infrastructures and systems were defined by the characteristics of the hardware, and it was largely a matter of software and hardware co-design. Now the hardware is moving into the background due to generic hardware platforms and components and is being defined by software or even virtualized from the user's point of view. Current technical developments here are software defined networks including network slices, or cloud services such as Infrastructure as a Service, Platform as a Service, or Software as a Service.

In addition, these software-based systems today significantly influence critical infrastructures such as electricity, water, or emergency care: they are an integral part of the systems such that both the software contained or used as well as the infrastructures themselves become so-called critical infrastructure. Here, we are using the term "software-based system" as an overarching term for the kinds of systems whose functionality, performance, security, and quality is largely defined by soft-

ware. These include networked and non-networked control systems such as control units in automobiles and airplanes, systems for connected and autonomous driving, and systems of systems such as the extension of the automobile into the backbone infrastructure of the OEMs. But also systems (of systems) in telecommunications networks, IT, industrial automation, and medical technologies are understood by this term.

Software-based systems today are often distributed and connected, are subject to real-time demands (soft or hard), are openly integrated into the environment via their interfaces, interact with other software-based systems, and use learning or autonomous functionalities to master complexity. Independently of whether we are now in a fourth revolution or in the second wave of the third revolution with digitization, the ongoing convergence of technologies and the integration of systems and processes is brought about and supported via software. New developments such as those in augmented reality, fabbing, robotics, data analysis, and artificial intelligence, too, place increasing demands on the reliability and security of software-based systems.

21.2 Software and software engineering

Let us examine things in greater depth. According to the IEEE Standard for Configuration Management in Systems and Software Engineering (IEEE 828-2012 [1]), software is defined as "computer programs, procedures and possibly associated documentation and data pertaining to the operation of a computer system". It includes programmed algorithms, data capturing or representing status and/or context, and a wide range of descriptive, explanatory, and also specifying documents (see Fig. 21.1).

A look at current market indicators reveals the omnipresence of software: according to a 2016 Gartner study, global IT expenditures of $3.5 billion were expected in 2017. Software is thus the fastest-growing area, at $357 billion or 6% [4]. Bitkom, as well, supports this view [5]: according to its own survey of 503 companies with 20 or more staff, every third company in Germany is developing its own software. Among large organizations with 500 or more staff, the proportion rises as high as to 64%. According to this survey, already every fourth company in Germany employs software developers, and an additional 15% say they want to hire additional software specialists for digital transformation.

Nevertheless, 50 years after the software crisis was explicitly addressed in 1968, and after numerous approaches and new methods in both software and quality engineering, the development and operation of software-based connected systems is still not smooth [8]. The term "software engineering" was initially introduced by

F. L. Bauer as a provocation: “the whole trouble comes from the fact that there is so much tinkering with software. It is not made in a clean fabrication process, which it should be. What we need, is software engineering.” The authors Fitzgerald and Stol identify various gaps in the development, maintenance, and distribution of software-based systems that can be closed via methods of continuous development, testing, and rollout.

Studies on breakdowns and challenges in the Internet of Things (IoT) complete our view here: according to self-reports by German companies, four in five of them have an “availability and data security gap” in IT services [9]. Servers in Germany, for example, stand idle for an average of 45 minutes during an outage. The estimated direct costs of these kinds of IT failures rose by 26% in 2016 to $21.8 million, versus $16 million in 2015. And these figures do not include the impacts that cannot be precisely quantified such as reduced customer confidence or reputational damage to the brand.

The top two challenges connected to IoT are security in particular IT security and data protection, as well as functional safety and interoperability of the software-defined protocol and service stacks [10].

In keeping with this, the latest 2016–17 edition of the World Quality Report [3] shows that there is a change in the primary goals of those responsible for quality assurance and testing that is accompanying the ongoing pervasion of the physical world by the digital world with the Internet of Things. The change picks up the increasing risk of breakdown and the criticality of software-based connected systems from the perspective of business and security. Thus, increasing attention is given to quality and security by design, and the value of quality assurance and testing is being raised in spite of, or indeed due to, the increasing utilization of agile and DevOps methods. Thus, with the complexity of software-based connected systems, expenditures for the realization, utilization, and management of (increasingly virtualized) test environments are also increasing. Even though extensive cost savings are equally possible in this area through automation, the necessity of making quality assurance and testing even more effective at all levels remains.

21.3 Selected characteristics of software

Before turning to current approaches to developing software-based connected systems, let us first take a look at the characteristics of software. Software should be understood as a technical product that must be systematically developed using software engineering. Software is characterized by its functionality and additional qualitative features such as reliability, usability, efficiency, serviceability, compati-

bility, and portability [12]. Against the backdrop of current developments and revelations, aspects of ethics as well as of guarantees and liability must also supplement the dimensions of software quality.

For a long time, software was considered to be free from all of the imponderables inherent to other technical products, and in this way was seen as the ideal technical product [11]. A key backdrop to this is the fact that algorithms, programming concepts and languages, and thus any computability is traced to the Turing computability (the Turing thesis). According to Church's thesis, computability here incorporates precisely those functions which can be calculated intuitively by us. Thus, while non-computable problems such as the halting problem elude algorithmics (and thus software), for each intuitively computable function there is an algorithm with limited computing complexity that can be realized via software. Here, the balance between function, algorithm, and software is the responsibility of various phases and methods of software engineering such as specification, design, and implementation as well as verification and validation. If alongside this understanding of intuitive computability, software now sounds like a product that is simple to produce, this is by no means the case. What began with the software crisis still holds true today. Herbert Weber reiterated this in 1992, "the so-called software crisis has thus far not yet produced the necessary level of suffering required to overcome it" [13]. Also Jochen Ludewig in 2013 formulated it as, "the requirements of software engineering have thus not yet been met" [11]. The particular characteristics of software are also part of the reason for this.

First and foremost, software is immaterial, such that all of the practical values for material products do not apply or are only transferable in a limited sense. Thus, software is not manufactured but "only" developed. Software can be copied practically without cost, with the original and the copy being completely identical and impossible to distinguish. This leads, among other things, to nearly unlimited possibilities for reusing software in new and typically unforeseen contexts.

On the one hand, using software does not wear it out. On the other hand, the utilization context and execution environment of software are constantly evolving such that untarnished software does in fact age technologically and indeed logically and thus must be continually redeveloped and modernized. This leads to maintenance cycles for software that, instead of restoring the product to its original state, generate new and sometimes ill-fitting, i.e. erroneous, conditions.

Software errors thus do not arise from wear and tear to the product but are built into it. Or errors develop in tandem with the software's unplanned use outside of its technical boundary conditions. This is one way that secure software can be operated insecurely, for example.

In addition, the days of rather manageable software in closed, static, and local use contexts for mainly input and output functionalities are long gone. Software is

largely understood as a system built on distributed components with open interfaces. The components of these can be realized in various technologies and by various manufacturers, and with configurations and contexts that may change dynamically. These may further incorporate third-party systems flexibly by means of service orchestrations and various kinds of interface and network access, which must be able to serve various usage scenarios and load profiles. Actions and reactions cannot be described by consistent functions.

Our understanding of intuitive computability is being challenged daily by new concepts such as data-driven, autonomous, self-learning, or self-repairing software. In doing so, software is increasingly using heuristics for its decision-making in order to efficiently arrive at practicable solutions, even in the case of NP-complete problems. The bottom line is that software-based connected systems, with all of the elementary decision-making they incorporate, are highly complex – the most complex technical systems that have yet been created. In this process potential difficulties arise, simply due to the sheer size of software packages. Current assessments of selected open-source software packages, for example, reveal relationships between software complexity, "code smells", which are indicators of potential software defects, and software vulnerabilities, the software's weak points with respect to IT security. This relationship may not be directly causal but is nevertheless identifiable and worthy of further investigation [14].

21.4 Model-based methods and tools

In what follows, we illustrate selected model-based methods and tools for the efficient development of reliable and secure software that are the result of current R&D studies at Fraunhofer FOKUS.

Models have a long tradition in software development. They originally served the specification of software and its formal verification of correctness. In the meantime they are commonly used as abstract, technology-independent bearers of information for all aspects of software development and quality assurance [15]. They thus serve to mediate information between software tools and to provide abstractions for capturing complex relationships. One example of this, in the context of risk analysis and assessment, or of systematic measurement and visualization of software characteristics, and also of software test automation, is via model-based testing or test automation. As Whittle, Hutchinson, and Rouncefield argue in [16], the particular added value of model-driven software development (Model-Driven Engineering, MDE) is the specification of the architectures, interfaces, and components of software. Architecture is also used by FOKUS as the foundation for docu-

mentation, functionality, interoperability, and security in the methods and tools introduced in what follows.

Process automation

Modern software development processes often use teams at various sites for individual components and to integrate commercial third-party components or open source software. Here, a wide range of software tools are used, with various individuals participating in different roles, whether actively or passively. The central problems here are the lack of consistency of the artifacts created in the development process, the shortage of automation, and the lack of interoperability between the tools.

ModelBus® is an open-source framework for tool integration in software and system development and closes the gap between proprietary data formats and software tool programming interfaces [17]. It automates the execution of tedious and error-prone development and quality assurance tasks such as consistency assurance across the entire development process. To do this, the framework uses service orchestrations of tools in keeping with SOA (service-oriented architecture) and ESB (enterprise service bus) principles.

The software tools of a process landscape are connected to the bus by the provision of ModelBus® adaptors. Adaptors are available for connecting IBM Rational Doors, Eclipse and Papyrus, Sparx Enterprise Architect, Microsoft Office, IBM Rational Software Architect, IBM Rational Rhapsody, or MathWorks Matlab Simulink.

21.5 Risk assessment and automated security tests

Safety-critical software-based systems are subject to careful risk analysis and evaluation according to the ISO Risk Management Standard [18] in order to capture and minimize risks. For complex systems, however, this risk management may be very time-consuming and difficult. While the subjective assessment of experienced experts may be an acceptable method of risk analysis on a small scale, with increasing size and complexity other approaches such as risk-based testing [21] need to be chosen.

An additional opportunity for more objective analysis is provided by the use of security tests in line with ISO/IEC/IEEE "Software and systems engineering – Software testing" (ISO 29119-1, [19]). A further option is to first have experts carry out a high-level assessment of the risks based on experience and literature. In order to make this initial risk assessment more accurate, security tests can be employed at

precisely the point where the first high-level risk picture shows the greatest vulnerabilities. The objective test results may then be used to enhance, refine, or correct the risk picture thus far. However, this method first becomes economically applicable with appropriate tool support.

RACOMAT is a risk-management tool developed by Fraunhofer FOKUS, which in particular combines risk assessment with security tests [20]. Here, security testing can be directly incorporated into event simulations that RACOMAT uses to calculate risks. RACOMAT facilitates extensive automation of risk modelling through to security testing. Existing databases such as those of known threat scenarios are used by RACOMAT to ensure a high degree of reuse and avoid errors.

At the same time, RACOMAT supports component-based compositional risk assessment. Easy-to-understand risk graphs are used to model and visualize an image of the risk situation. Common techniques such as fault tree analysis (FTA), event tree analysis (ETA), and conducting security risk analysis (CORAS) may be used in combination for the risk analysis in order to be able to benefit from the various strengths of the individual techniques. Starting with an overall budget for risk assessment, RACOMAT calculates how much expenditure is reasonable for security testing in order to improve the quality of the risk picture by reducing vul-

Fig. 21.1 Risk analysis and security testing with RACOMAT (Fraunhofer FOKUS)

nerabilities. The tool offers recommendations on how these means should be used. To do this, RACOMAT identifies relevant tests and places them in order of priority.

21.6 Software mapping and visualization

Software-based systems are becoming ever more complex due to their increasing functions and their high security, availability, stability, and usability requirements. In order for this not to lead to losses of quality and so that structural problems can be identified early on, quality assurance must commence right at the beginning of the development process. A model-driven development process where models are key to the quality of the software-based system is well suited to this. Up to now, however, quality criteria for this were neither defined nor established. In future, model characteristics and their quality requirements need to be identified, and additionally mechanisms found with which their properties and quality can be determined.

Metrino is a tool that checks and ensures the quality of models [22]. It may be used in combination with ModelBus® but can also be employed independently. With the aid of Metrino, metrics for domain-specific models can be generated, independently defined, and managed. The metrics produced can be used for all models that accord with the meta-model used as the basis for development. Metrino thus analyzes and verifies properties such as the complexity, size, and description of software artifacts. In addition, the tool offers various possibilities for checking the computational results of the metrics and representing them graphically – for ex-

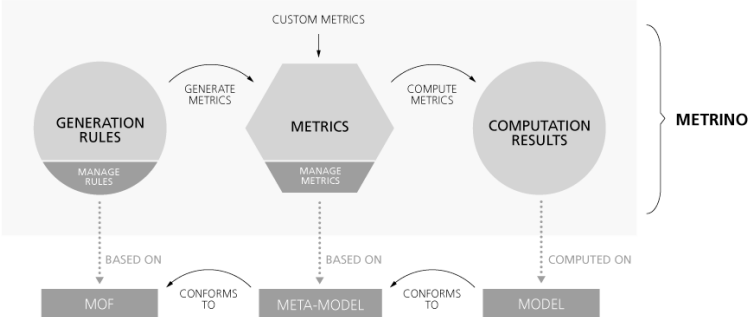

Fig. 21.2 Model-based software measurement and visualization with Metrino (Fraunhofer FOKUS)

ample in a table or spider chart. Since Metrino saves the results from several evaluations, results from different time periods can also be analyzed and compared with one another. This is the only way that optimal quality of the final complex software-based system can be guaranteed.

Metrino is based on the Structured Metrics Metamodel (SMM) developed by the Object Management Group (OMG) and can be used both for models in the Unified Modeling Language (UML) as well as for domain-specific modelling languages (DSLs). On top of that, Metrino's field of application includes specialized, tool-specific languages and dialects.

Whether for designing embedded systems or for software in general, Metrino can be used in the widest variety of different domains. The tool can manage metrics and apply them equally to (model) artifacts or also to the complete development chain, including traceability and test coverage.

21.7 Model-based testing

The quality of products is the decisive factor of being accepted on the market. In markets with security-related products in particular, such as medicine, transportation, or automation sectors, for example, quality assurance is thus accorded high priority. In these sectors, quality is equally decisive for product authorization. Quality assurance budgets, however, are limited. It is thus important to managers and engineers that the available resources are utilized efficiently. Often, manual testing methods are still being used, even if only a comparatively small number of tests can be prepared and conducted in this way, and they are additionally highly prone to error. The efficiency of manual testing methods is thus limited, and rising costs are unavoidable. Model-based test generation and execution offers a valuable alternative: the use of models from which test cases can be automatically derived offers enormous potential for increasing test quality at lower costs. In addition, case studies and practical uses have shown that when model-based testing techniques are introduced necessary investment costs in technology and training pay off quickly [24].

Fokus!MBT thus offers an integrated test modelling environment that leads the user through the Fokus!MBT methodology and thus simplifies the creation and use of the underlying test model [25]. A test model contains test-relevant, structural, behavior- and method-specific information that conserves the tester's knowledge in a machine processable fashion. In this way, it can be adapted or evaluated at any time, say for the generation of additional test-specific artifacts. Additional benefits of the test model are the visualization and documentation of the test specification.

Fokus!MBT uses the UML Testing Profile (UTP), specified by the Object Management Group and developed with significant contributions from FOKUS, as its modeling notation. UTP is a test-specific extension of the Unified Modeling Language (UML) common in industry. This allows testers to use the same language concepts as the system architects and requirements engineers, thus preventing communication issues and encouraging mutual understanding.

Fokus!MBT is based on the flexible Eclipse RCP platform, the Eclipse Modeling Framework (EMF), and Eclipse Papyrus. As a UTP-based modeling environment, it has all of the UML diagrams available as well as additional test-specific diagrams. Alongside the diagrams, Fokus!MBT relies on a proprietary editor framework for describing and editing the test model. The graphical editor user interfaces can be specifically optimized for the needs or abilities of the user in question. In doing so, if necessary, it is possible to completely abstract from UML/UTP, allowing specialists unfamiliar with IT to quickly produce a model-based test specification. This is also supported by the provision of context-specific actions that lead the user through the Fokus!MBT methodology. In this way, methodically incorrect actions or actions, which are inappropriate for the context in question are not even enabled. Based upon this foundation, Fokus!MBT integrates automated modeling rules that guarantee adherence to guidelines, in particular modelling or naming conventions, both after and during working on the test model. These preventative quality assurance mechanisms

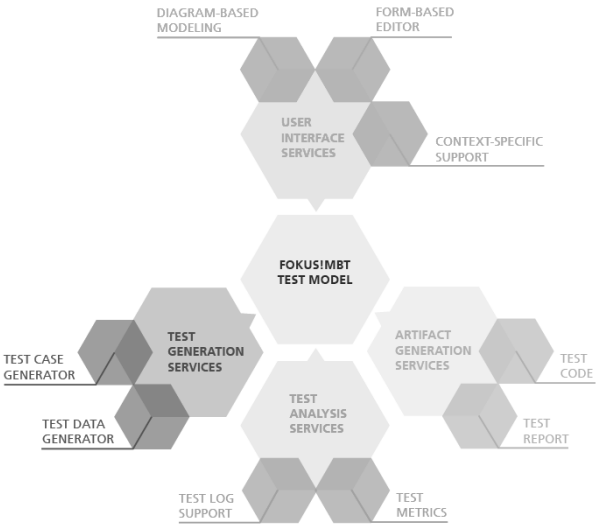

Fig. 21.3 Model-based testing with Fokus!MBT (Fraunhofer FOKUS)

distinguish Fokus!MBT from other UML tools, accelerate model generation, and minimize costly review sessions.

The fundamental goal of all test activities is validating the system to be tested vis-à-vis its requirements. Consistent and uninterrupted traceability, in particular between requirements and test cases, is indispensable here. Fokus!MBT goes one step further and also incorporates the test execution results into the requirements traceability within the test model. In this way, a traceability network is created between requirement, test case, test script, and test execution result, thus making the status of the relevant requirements or the test progress immediately assessable. The visualization of the test execution results additionally facilitates the analysis, processing and assessment of the test case execution process. The test model thus contains all of the relevant information to estimate the quality of the system tested, and support management in their decision-making related to the system's release.

21.8 Test automation

Analytical methods and dynamic testing approaches in particular are a central and often also exclusive instrument for verifying the quality of entire systems. Software tests thereby require all of the typical elements of software development, because tests themselves are software-based systems and must thus be developed, built, tested, validated, and executed in exactly the same way. In addition to that, test systems possess the ability to control, stimulate, observe and to assess the system being tested. Although standard development and programming techniques are generally also applicable for tests, specific solutions for the development of a test system are important in order to be able to take its unique features into account. This approach expedited the development and standardization of specialized test specification and test implementation languages.

One of the original reasons for the development of Tree and Tabular Combined Notation (TTCN) was the precise conformity definition for telecommunications component protocols according to their specification. Test specifications were utilized to define test procedures objectively and assess, compare, and certify the equipment on a regular basis. Thus, the automated execution became exceptionally important for TTCN, too.

Over the years, the significance of TTCN grew and various pilot projects demonstrated a successful applicability beyond telecommunications. With the convergence of telecommunications and information technology sectors, the direct applicability of TTCN became obvious to developers from other sectors. These trends, along with the characteristics of more recent IT and telecommunications technolo-

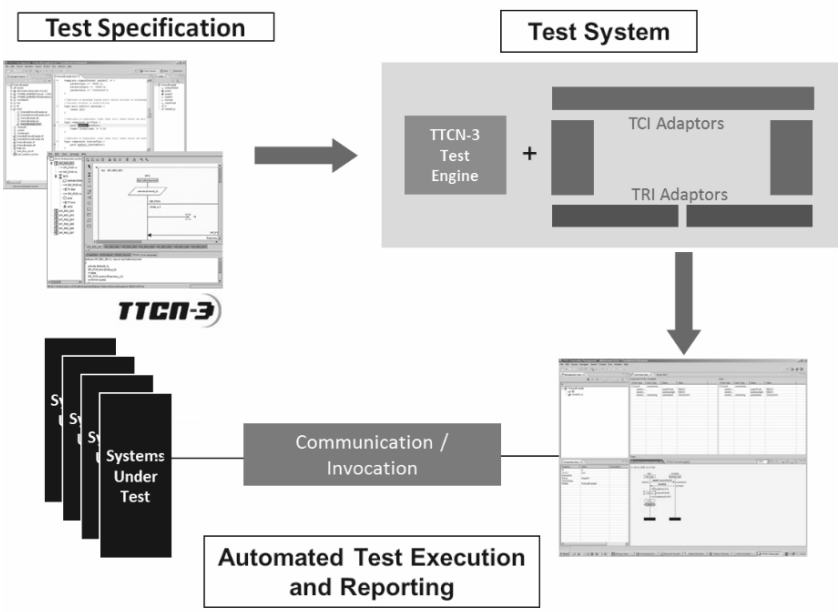

Fig. 21.4 Test automation with TTCN-3 (Fraunhofer FOKUS)

gies also placed new requirements on TTCN: the result is TTCN-3 (Testing and Test Control Notation Version 3, [27]).

TTCN-3 is a standardized and modern test specification and test implementation language developed by the European Telecommunication Standards Institute (ETSI). Fraunhofer FOKUS played a key role in TTCN-3's development and is responsible for various elements of the language definition, including Part 1 (concepts and core languages), Part 5 (runtime interfaces), and Part 6 (test control interfaces), as well as TTCN-3 tools and test solutions [28][29]. With the aid of TTCN-3, tests can be developed textually or graphically, and execution can be automated. In contrast to many (test) modeling languages, TTCN-3 comprises not only a language for test specification but also an architecture and execution interfaces for TTCN-3-based test systems. Currently, FOKUS uses TTCN-3 for developing the Eclipse IoT-testware for testing and securing IoT components and solutions, for example [30].

21.9 Additional approaches

It is not possible to introduce all of our methods, technologies, and tools here. Our publications (see also [31]) contain further information on

- Security-oriented architectures,
- Testing and certifying functional security,
- Model-based re-engineering,
- Model-based documentation,
- Model-based porting to the cloud, or
- Model-based fuzz tests.

21.10 Professional development offerings

It is not enough to simply develop new methods, technologies, and tools. These also need to be distributed and supported during their introduction to projects and pilots.

Fraunhofer FOKUS has thus for a long time been involved in professional development. The institute has initiated and/or played a key role in developing the following professional development schemes in cooperation with the ASQF (Arbeitskreis Software-Qualität und -Fortbildung – "Software Quality and Training Working Group") [32], GTB (German Testing Board) [33], and the ISTQB (International Testing Qualifications Board [34]):

- GTB Certified TTCN-3 Tester
- ISTQB Certified Tester Foundation Level – Model-Based Testing
- ISTQB Certified Tester Advanced Level – Test Automation Engineer
- ASQF/GTB Quality Engineering for the IoT

Further, Fraunhofer FOKUS, together with HTW Berlin and Brandenburg University of Applied Sciences, is also forming a consortium, the Cybersecurity Training Lab [35], with training modules on

- Secure software engineering
- Security testing
- Quality management & product certification
- Secure e-government solutions
- Secure public safety solutions

This and other offerings, such as on semantic business rule engines or open government data, are also available via the FOKUS-Akademie [36].

21.11 Outlook

Software development and quality assurance are both subject to the competing requirements of increasing complexity, the demand for high-quality, secure, and reliable software, and the simultaneous economic pressure for short development cycles and fast product introduction times.

Model-based methods, technologies, and tools address the resulting challenges, and in particular support modern agile development and validation approaches. Continuous development, integration, testing and commissioning benefit from model-based approaches to a particular degree. This is because they form a strong foundation for automation and can also support future technology developments due to their independence from specific software technologies.

Additional progress in model-based development is to be expected or indeed forms part of current research. Whereas actual integration and test execution are already conducted in a nearly entirely automated fashion, the analysis and correction of errors remains a largely manual task, one that is time-consuming and itself subject to error and can thus lead to immense delays and costs. Self-repairing software would be an additional step towards greater automation, borrowing from the diverse software components in open source software using pattern recognition and analysis through deep learning methods, and repair and assessment using evolutionary software engineering approaches. In this way, software could become not only self-learning but also self-repairing.

Nevertheless, until then it is important to

- Understand software engineering as an engineering discipline and leave it to experts to develop and to ensure their quality including safety and/or security,
- Continue to develop software engineering itself as a field of research and development and automate insecure manual steps in software development and validation,
- Consider beginning with draft monitoring and testing environments for all levels of a digitized application landscape that can be efficiently managed via virtualization methods for software platforms,
- Consider that security, interoperability, and usability are gaining increasingly in importance in the quality of software-based connected systems and demand priority during design, development, and validation.

Sources and literature

[1] Henrik Czernomoriez, et al.: Netzinfrastrukturen für die Gigabitgesellschaft, Fraunhofer FOKUS, 2016.
[2] IEEE: IEEE Standard for Configuration Management in Systems and Software Engineering, IEEE, 828-2012, https://standards.ieee.org/findstds/standard/828-2012.html, besucht am 15.7.2017.
[3] World Quality Report 2016-2017: 8th Edition – Digital Transformation, http://www.worldqualityreport.com, besucht am 15.7.2017.
[4] Gartner 2016: Gartner Says Global IT Spending to Reach $3.5 Trillion in 2017, http://www.gartner.com/newsroom/id/3482917, besucht am 15.7.2017.
[5] Bitkom Research 2017: Jedes dritte Unternehmen entwickelt eigene Software, https://www.bitkom.org/Presse/Presseinformation/Jetzt-wird-Fernsehen-richtig-teuer.html, besucht am 15.7.2017.
[6] NATO Software Engineering Conference, 1968: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF, besucht am 21.7.2017.
[7] Friedrich L. Bauer: Software Engineering – wie es begann. Informatik Spektrum, 1993, 16, 259-260.
[8] Brian Fitzgerald, Klaas-Jan Stol, Continuous software engineering: A roadmap and agenda, Journal of Systems and Software, Volume 123, 2017, Pages 176-189, ISSN 0164-1212, http://dx.doi.org/10.1016/j.jss.2015.06.063, besucht am 21.7.2017.
[9] VEEAM: 2017 Availability report, https://go.veeam.com/2017-availability-report-de, besucht am 21.7.2017.
[10] Eclipse: IoT Developer Trends 2017 Edition, https://ianskerrett.wordpress.com/2017/04/19/iot-developer-trends-2017-edition/, besucht am 21.7.2017.
[11] Jochen Ludewig und Horst Lichter: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken. 3., korrigierte Auflage, April 2013, dpunkt.verlag, ISBN: 978-3-86490-092-1.
[12] ISO/IEC: Systems and software engineering -Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models, ISO/IEC 25010:2011, https://www.iso.org/standard/35733.html, besucht am 22.7.2017.
[13] Herbert Weber: Die Software-Krise und ihre Macher, 1. Auflage, 1992, Springer-Verlag Berlin Heidelberg, DOI 10.1007/978-3-642-95676-8.
[14] Barry Boehm, Xavier Franch, Sunita Chulani und Pooyan Behnamghader: Conflicts and Synergies Among Security, Reliability, and Other Quality Requirements. QRS () 2017 Panel, http://bitly.com/qrs_panel, besucht am 22.7.2017.
[15] Aitor Aldazabal, et al. „Automated model driven development processes.“ Proceedings of the ECMDA workshop on Model Driven Tool and Process Integration. 2008.
[16] Jon Whittle, John Hutchinson, and Mark Rouncefield. „The state of practice in model-driven engineering.“ IEEE software 31.3 (2014): 79-85.
[17] Christian Hein, Tom Ritter und Michael Wagner: Model-Driven Tool Integration with ModelBus. In Proceedings of the 1st International Workshop on Future Trends of Model-Driven Development – Volume 1: FTMDD, 35-39, 2009, Milan, Italy.
[18] ISO: Risk management, ISO 31000-2009, https://www.iso.org/iso-31000-risk-management.html, besucht am 22.7.2017.

[19] ISO/IEC/IEEE: Software and systems engineering – Software testing – Part 1: Concepts and definitions. ISO/IEC/IEEE 29119-1:2013, https://www.iso.org/standard/45142.html, besucht am 22.7.2017.
[20] Johannes Viehmann und Frank Werner. „Risk assessment and security testing of large scale networked systems with RACOMAT." International Workshop on Risk Assessment and Risk-driven Testing. Springer, 2015.
[21] Michael Felderer, Marc-Florian Wendland und Ina Schieferdecker. „Risk-based testing." International Symposium On Leveraging Applications of Formal Methods, Verification and Validation. Springer, Berlin, Heidelberg, 2014.
[22] Christian Hein, et al. „Generation of formal model metrics for MOF-based domain specific languages." Electronic Communications of the EASST 24 (2010).
[23] Marc-Florian Wendland, et al. „Model-based testing in legacy software modernization: An experience report." Proceedings of the 2013 International Workshop on Joining AcadeMiA and Industry Contributions to testing Automation. ACM, 2013.
[24] Ina Schieferdecker. „Model-based testing." IEEE software 29.1 (2012): 14.
[25] Marc-Florian Wendland, Andreas Hoffmann, and Ina Schieferdecker. 2013. Fokus!MBT: a multi-paradigmatic test modeling environment. In Proceedings of the workshop on ACadeMics Tooling with Eclipse (ACME ,13), Davide Di Ruscio, Dimitris S. Kolovos, Louis Rose, and Samir Al-Hilank (Eds.). ACM, New York, NY, USA, Article 3, 10 pages. DOI: https://doi.org/10.1145/2491279.2491282
[26] ETSI: TTCN-3 – Testing and Test Control Notation, Standard Series ES 201 873-1 ff.
[27] Jens Grabowski, et al. „An introduction to the testing and test control notation (TTCN-3)." Computer Networks 42.3 (2003): 375-403.
[28] Ina Schieferdecker und Theofanis Vassiliou-Gioles. „Realizing distributed TTCN-3 test systems with TCI." Testing of Communicating Systems (2003): 609-609.
[29] Juergen Grossmann, Diana Serbanescu und Ina Schieferdecker. „Testing embedded real time systems with TTCN-3." Software Testing Verification and Validation, 2009. ICST'09. International Conference on. IEEE, 2009.
[30] Ina Schieferdecker, et al. IoT-Testware – an Eclipse Project, Keynote, Proc. of the 2017 IEEE International Conference on Software Quality, Reliability & Security, 2017.
[31] FOKUS: System Quality Center, https://www.fokus.fraunhofer.de/sqc, besucht am 22.7.2017.
[32] ASQF: Arbeitskreis Software-Qualität und Fortbildung (ASQF), http://www.asqf.de/, besucht am 25.7.2017.