Advanced Software Engineering

Modell-basiert Software effizient und sicher entwickeln und prüfen

Ina Schieferdecker, Tom Ritter

Abstrakt

Software rules them all! In jeder Branche spielt Software mittlerweile eine dominante Rolle für Innovationen technischer oder geschäftlicher Art, für die Verbesserung der funktionalen Sicherheit oder aber für die Erhöhung des Komforts. Gleichsam wird Software nicht immer mit der nötigen Professionalität entworfen, (weiter-)entwickelt und/oder abgesichert – und gibt es unnötige Brüche in den Entwicklungs-, Wartungs- und Betriebsketten, die zuverlässigen, sicheren, performanten und vertrauenswürdigen Systemen zuwiderlaufen. Aktuelle Umfragen wie im jährlichen World Quality Report sprechen hier eine deutliche Sprache, welche direkt mit den bekannt gewordenen Software-verursachten Ausfällen großer, wichtiger und / oder sicherheitskritischer Infrastrukturen korreliert. So ist es höchste Zeit, Software-Entwicklung den Experten zu überlassen und den Raum für die Nutzung aktueller Methoden und Technologien zuzulassen. Dieser Artikel wirft einen Blick auf aktuelle und zukünftige Software-Engineering-Ansätze, die Sie auch und insbesondere im Fraunhofer-Portfolio finden.

Einleitung

Lassen Sie uns mit den technologischen Veränderungen der digitalen Transformation beginnen, die in den Augen vieler Menschen revolutionäre Veränderungen unserer Gesellschaft darstellen: Gebäude, Autos, Züge, Fabriken und die meisten Dinge unseres Alltags sind bereits oder werden in naher Zukunft mittels der überall verfügbaren digitalen Infrastruktur unserer zukünftigen Gigabit-Gesellschaft verbunden sein [1]. Dies wird den Informationsaustauch sowie die Kommunikation und Interaktion in allen Lebens- und Arbeitsbereichen verändern – sei es im Gesundheitswesen, in Verkehr, Handel oder Produktion. Für diese durch die digitale Vernetzung getriebene Technologie- und Domänen-Konvergenz, gibt es viele Begriffe: Internet of Things, Smart Cities, Smart Grid, Smart Production, Industrie 4.0, Smart Buildings, Internet of Systems Engineering, Cyber-Physical Systems oder Internet of Everything. Trotz unterschiedlicher Zielrichtungen und Anwendungsbereiche liegt all diesen Begriffen als Basiskonzept ein allumfassender Austausch von Informationen zwischen technischen Systemen zu Grunde – eben die »Digitale Vernetzung«:

> Mit Digitaler Vernetzung bezeichnen wir die durchgehende und durchgängige Verknüpfung der physischen Welt mit der digitalen Welt. Dazu gehören die digitale Erfassung, Abbildung und Modellierung der physischen Welt, sowie die Vernetzung der daraus entstehenden Informationen. Diese ermöglicht die zeitnahe und teilautomatisierte Beobachtung, Auswertung und Steuerung der physischen Welt.

Die digitale Vernetzung ermöglicht einen nahtlosen Informationsaustauch zwischen den digitalen Abbildern von Personen, Dingen, Systemen, Prozessen sowie Organisationen und baut ein weltweites

Netz von Netzen – ein Inter-Net – auf, das weit über die Vision des ursprünglichen Internets hinausgeht. Bei dieser neuen Form der Vernetzung geht es aber nicht mehr nur um das Vernetzen an sich. Vielmehr werden einzelne Daten zu Informationen zusammengefasst, um weltweit vernetztes und vernetzbares Wissen aufzubauen und dieses sowohl für ein wachsendes Verständnis als auch die Steuerung monotoner oder sicherheitskritischer Abläufe zu nutzen.

Mit Blick auf diese digitale Vernetzung nimmt die zentrale Rolle der Software weiter zu. Nicht nur werden die digitalen Abbilder, also die Strukturen, Daten und Verhaltensmodelle der Dinge, Systeme und Prozesse der physischen Welt, mittels Software realisiert, sondern ebenso alle Algorithmen mit denen diese digitalen Abbilder visualisiert, interpretiert und weiterverarbeitet werden, als auch alle Funktionen und Dienste der Infrastrukturen und Systeme wie Server und (End-)Geräte im Netz der Netze. Während noch vor nicht allzu langer Zeit die wesentlichen Eigenschaften der Infrastrukturen und Systeme durch die Eigenschaften der Hardware definiert wurden und es im Wesentlichen um Software- und Hardware-Codesign ging, tritt die Hardware aufgrund generischer Hardwareplattformen und -komponenten in den Hintergrund und wird durch Software definiert oder gar aus Nutzersicht virtualisiert. Aktuelle technische Entwicklungen sind hierzu Software-Defined Networks inklusive Network Slices oder Cloud-Dienste wie Infrastructure as a Service, Platform as a Service oder Software as a Service.

Zudem beeinflussen diese softwarebasierten Systeme heutige kritische Infrastrukturen wie die Strom-, Wasser- oder Notfallversorgung wesentlich: sie sind integraler Bestandteil der Systeme, so dass die enthaltene bzw. genutzte Software als auch die Infrastrukturen selbst zur sogenannten kritischen Infrastruktur werden. „Softwarebasiertes System“ nutze ich als Oberbegriff für derartige Systeme, die maßgeblich durch Software in ihrer Funktionalität, Leistungsfähigkeit, Sicherheit und Qualität bestimmt werden. Dazu gehören vernetzte und nicht vernetzte Steuerungssysteme, wie z. B. Steuergeräte in Automobilen, Flugzeugen, Systeme für das vernetzte und autonome Fahren und Systeme von Systemen, wie z. B. die Verlängerung des Automobils in die Backbone-Infrastruktur der OEMs. Aber auch Systeme (von Systemen) in Telekommunikationsnetzen, der IT, Industrieautomatisierung und Medizintechnik werden darunter verstanden.

Softwarebasierte Systeme sind heutzutage oftmals verteilt und vernetzt, unterliegen Echtzeitanforderungen (weichen bzw. harten), sind offen über ihre Schnittstellen in die Umgebung eingebunden, stehen mit anderen softwarebasierten Systemen in Interaktion und nutzen lernende oder autonome Funktionalitäten zur Beherrschung der Komplexität. Unabhängig davon, ob wir uns mit der Digitalisierung nun in einer vierten Revolution oder in der zweiten Welle der dritten Revolution befinden: die fortschreitende Konvergenz von Technologien und die Integration von Systemen und Prozessen wird über Software vermittelt und getragen. Auch neue Entwicklungen wie für Augmented Reality, Fabbing, Robotik, Datenanalytik und Künstliche Intelligenz stellen zunehmende Anforderungen an die Zuverlässigkeit und Sicherheit softwarebasierter Systeme.

Software und Software Engineering

Lassen Sie uns etwas tiefer einsteigen: Nach dem IEEE Standard for Configuration Management in Systems and Software Engineering (IEEE 828-2012 [1]) ist Software definiert als „computer programs, procedures and possibly associated documentation and data pertaining to the operation of a computer system“. Sie umfasst programmierte Algorithmik, den Zustand und / oder Kontext

erfassende bzw. repräsentierende Daten und verschiedenste beschreibende, erläuternde als auch spezifizierende Dokumente, siehe Abb. 1.

Ein Blick auf die aktuellen Marktzahlen zeigt die Omnipräsenz von Software: Laut Gartner 2016, werden 2017 weltweit IT-Ausgaben von 3,5 Billionen US$ erwartet. Dabei ist Software mit 357 Mrd. US$ der mit 6% am stärksten wachsende Bereich [4]. Auch der Bitkom stützt diese Aussagen [5]: Laut seiner Umfrage bei 503 Unternehmen ab 20 Mitarbeiter*inne*n entwickelt jedes dritte Unternehmen in Deutschland eigene Software. Unter den großen Unternehmen mit 500 oder mehr Mitarbeiter*inne*n sind dies sogar 64 Prozent. Schon jetzt beschäftigt laut dieser Umfrage jedes vierte Unternehmen in Deutschland Software-Entwickler und weitere 15 Prozent geben an, dass sie für die digitale Transformation zusätzlich Software-Spezialisten einstellen wollen.

Jedoch, die Entwicklung und der Betrieb softwarebasierter, vernetzter Systeme erfolgt auch seit bald 50 Jahren nach der expliziten Adressierung der Software-Krise 1968 [6] und mannigfaltigen Ansätzen und neuen Methoden zum Software Engineering als auch Quality Engineering nach wie vor nicht reibungsfrei [8]. F.L. Bauer führte den Begriff des Software Engineerings ursprünglich als Provokation ein: “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.“ Die Autoren Fitzgerald und Stol benennen diverse Lücken bei Entwicklung, Wartung und Vertrieb softwarebasierter Systeme, die durch Methoden kontinuierlichen Entwickelns, Prüfens und Ausrollens geschlossen werden können.

Komplettiert wird diese Sicht durch Studien zu Ausfällen und Herausforderungen im Internet der Dinge (das Internet of Things, kurz IoT): Nach eigenen Angaben von deutschen Unternehmen existiert bei vier von fünf eine „Verfügbarkeits- und Datensicherungslücke“ von IT-Services [9]. So steht in Deutschland ein Server bei einem ungeplanten Ausfall durchschnittlich 45 Minuten still. Die geschätzten direkten Kosten für solche IT-Ausfälle stiegen 2016 um 36 Prozent auf 21,8 Mio US$ gegenüber 16 Mio US$ im Jahr 2015. Dabei enthalten diese Zahlen nicht die Auswirkungen, die sich nicht genau beziffern lassen wie sinkendes Kundenvertrauen oder Schaden für die Marke.

Die beiden mit IoT verbundenenTop-Herausforderungen sind Sicherheit, insbesondere IT-Sicherheit und Datenschutz als auch funktionale Sicherheit, und Interoperabilität der durch Software definierten Protokoll und Service Stacks [10].

In seiner jüngsten Ausgabe zeigt passend dazu der World Quality Report 2016-2017 [3] einen mit der weiteren Durchdringung der physischen mit der digitalen Welt entlang des Internets der Dinge einhergehenden Wandel in den vorrangigen Zielen der Qualitätssicherungs- und Prüf-Verantwortlichen. Er greift das zunehmende Ausfallrisiko und die Kritikaliät softwarebasierter, vernetzter Systeme aus Geschäfts- und Sicherheitsperspektive auf. So wird zunehmend auf Qualität und Sicherheit by Design geachtet und werden Qualitätssicherung und Prüfung in ihrer Wertigkeit angehoben, trotz oder besser wegen der steigenden Nutzung von agilen und DevOps-Methoden. So steigen mit der Komplexität der softwarebasierten, vernetzten Systeme auch die Aufwendungen für die Realisierung, die Anwendung und das Management (zunehmend virtualisierter) Testumgebungen. Obwohl ebenso durch Automatisierung in diesem Bereich umfangreiche Kostenersparnisse möglich sind, bleibt die Notwendigkeit bestehen, Qualitätssicherung und Prüfung auf allen Ebenen weiterhin effizienter zu machen.

Ausgewählte Eigenschaften von Software

Bevor wir in aktuelle Ansätze für die Entwicklung von softwarebasierten vernetzten Systemen diskutieren, lassen Sie uns einen Blick auf die Eigenschaften von Software werfen. Software ist als technisches Produkt zu verstehen, das mittels Software Engineering systematisch entwickelt werden muss. Software wird durch seine Funktionalität und weitere Qualitätsmerkmale wie Zuverlässigkeit, Benutzbarkeit, Effizienz, Wartbarkeit, Sicherheit, Kompatibilität und Portabilität charakterisiert [12]. Vor dem Hintergrund aktueller Entwicklungen und Enthüllungen müssen Aspekte der Ethik als auch der Gewährleistung und Haftung die Dimensionen der Software-Qualität ergänzen.

Lange galt Software als frei von allen Unwägbarkeiten, die anderen technischen Produkten anhaften, und in diesem Sinne als das ideale technische Produkt [11]. Ein wesentlicher Hintergrund hierbei ist, dass Algorithmen, Programmierkonzepte und -sprachen und damit jede Berechenbarkeit auf die Turing-Berechenbarkeit zurückgeführt wird (die Turing These). Nach der Church'schen These umfasst hierbei Berechenbarkeit genau die für uns intuitiv berechenbaren Funktionen. Während sich also nicht-berechenbare Probleme wie das Halteproblem der Algorithmik und damit der Software entziehen, findet sich für jede intuitiv berechenbare Funktion ein Algorithmus mit begrenzter Berechnungskomplexität, der durch Software realisiert werden kann. Hierbei obliegt der Abgleich zwischen Funktion, Algorithmus und Software verschiedenen Phasen und Methoden des Software Engineerings wie Spezifikation, Entwurf und Implementierung als auch Verifikation und Validierung.

Wenn nun entlang des Verständnisses der intuitiven Berechenbarkeit Software als einfach herzustellendes Produkt klingt, ist sie es mitnichten. Was mit der Software-Krise begann, von Herbert Weber 1992 wiederholt als „Die sogenannte Software-Krise hat bisher noch nicht den nötigen Leidensdruck erzeugt, der notwendig ist, um sie zu überwinden" [13] und noch 2013 von Jochen Ludewig 2013 als „Der Anspruch des Software Engineerings ist also noch nicht eingelöst." [11] formuliert wurde, gilt bis heute. Das liegt auch an den besonderen Eigenschaften von Software:

Zuvorderst ist Software immateriell, so dass alle Erfahrungswerte zu (materiellen) Produkten nicht gelten oder nur bedingt übertragbar sind. So wird Software nicht gefertigt, sondern „nur" entwickelt. Software kann zu nahe Null-Kosten kopiert werden, wobei Original und Kopie völlig gleich und nicht unterscheidbar sind. Das führt u.a. zu bald unbegrenzten Möglichkeiten, Software auch in neuen, typischerweise nicht vorhergesehenen Kontexten wiederzuverwenden.

Einerseits verschleißt die Benutzung von Software diese nicht. Andererseits entwickeln sich der Nutzungskontext und die Ausführungsumgebung von Software kontinuierlich, so dass die nicht verschleißende Software doch technologisch als auch logisch veraltet; und so kontinuierlich weiterentwickelt und modernisiert werden muss. Hierbei kommt es zu Wartungszyklen für die Software, die eben nicht den alten Zustand des Produkts wieder herstellen, sondern neue und unter Umständen nicht passende oder – kurz gesagt – fehlerhafte Zustände erzeugen.

So entstehen Fehler der Software nicht durch Abnutzung des Produkts, sondern werden in sie eingebaut. Oder aber Fehler entstehen entlang einer ungeplanten Nutzung der Software, die außerhalb der technischen Randbedingungen stattfindet. So kann beispielsweise eine sichere Software unsicher betrieben werden.

Zudem sind die Zeiten für eher beherrschbare Software in geschlossenen, statischen und lokalen Nutzungskontexten für im Wesentlichen Ein-/Ausgabefunktionalitäten lange vorbei. Software wird

größtenteils als aus verteilten Komponenten aufgebautes und bezüglich seiner Schnittstellen offenes System verstanden, dessen Komponenten in verschiedenen Technologien und von verschiedenen Herstellern realisiert sein können, dessen Konfigurationen und Kontexte sich dynamisch ändern können, das Dritt-Systeme über Dienstorchestrierungen und verschiedene Schnittstellen- und Netzzugänge flexibel einbinden kann und das verschiedene Nutzungsszenarien und Lastprofile bedienen können muss. Die Aktionen und Reaktionen lassen sich nicht durch stetige Funktionen beschreiben.

Unser Verständnis der intuitiven Berechenbarkeit wird tagtäglich durch neue Konzepte wie beispielsweise für datengetriebene, autonome, selbstlernende oder selbstreparierende Software herausgefordert. Dabei nutzt Software zunehmend Heuristiken für ihre Entscheidungen, um auch bei NP-vollständigen Problemen effizient zu praktikablen Lösungen zu kommen. Unterm Strich sind Softwarebasierte vernetzte Systeme entlang der enthaltenen Elementarentscheidungen sehr komplex und die komplexesten technischen Systeme, die bislang geschaffen worden sind. Dabei birgt bereits die schiere Größe von Software-Paketen potentiell Schwierigkeiten. So zeigen beispielsweise aktuelle Messungen an ausgewählten Open Source Software-Paketen Relationen zwischen Software-Komplexität, Code Smells, die Indikatoren für mögliche Software-Defekte sind, und Software Vulnerabilities, die Schwachstellen der Software bezüglich IT-Sicherheit, die zwar nicht direkt kausal, aber erkennbar und weiter zu untersuchen sind [14].

Modellbasierte Methoden und Werkzeuge

Im Folgenden erläutern wir ausgewählte modellbasierte Methoden und Werkzeuge zur effizienten Entwicklung zuverlässiger und sicherer Software, die Ergebnis der aktuellen F&E-Arbeiten bei Fraunhofer FOKUS sind.

Modelle haben in der Software-Entwicklung eine lange Tradition. Sie dienten ursprünglich der Spezifikation von Software und ihrer formalen Verifikation der Korrektheit. Mittlerweile werden sie allgemein als abstrakte, technologieunabhängige Träger von Informationen für alle Aspekte in der Software-Entwicklung und Qualitätssicherung genutzt [15]. So dienen sie als Informationsmittler zwischen Software-Werkzeugen, als Abstraktionsmittel zur Erfassung komplexer Zusammenhänge wie bei der Risikoanalyse und -bewertung, der systematischen Vermessung und Visualisierung von Software-Eigenschaften als auch der Automatisierung von Software-Prüfungen, beispielsweise durch Modell-basiertes Testen oder Testautomatisierung. Wie Whittle, Hutchinson und Rouncefield in [16] argumentieren, ist der besondere Mehrwert modellgetriebener Software-Entwicklung (Model-Driven Engineering, MDE) die Fixierung der Architekturen, Schnittstellen und Komponenten einer Software. Die Architektur als Fundament für Dokumentation, Funktionalität, Interoperabilität und Sicherheit wird auch von den im Folgenden vorgestellten Methoden und Werkzeugen von FOKUS genutzt.

Prozessautomatisierung

Heutige Softwareentwicklungsprozesse nutzen oftmals Teams an verschiedenen Standorten für einzelne Komponenten und die Integration von kommerziellen Dritt-Komponenten oder Open Source Software. Dabei kommen verschiedenste Software-Werkzeuge zum Einsatz und verschiedene Personen in unterschiedlichen Rollen – ob aktiv oder passiv – nehmen teil. Zentrale Probleme sind hierbei die fehlende Konsistenz der Artefakte, die im Entwicklungsprozess entstehen, die mangelnde Automatisierung und die fehlende Interoperabilität zwischen den Werkzeugen.

ModelBus® ist ein Open Source Framework für die Werkzeugintegration in der Software- und Systementwicklung und schließt die Lücke zwischen proprietären Datenformaten und Programmierschnittstellen von Software-Werkzeugen [17]. Es automatisiert die Ausführung mühsamer und fehleranfälliger Entwicklungs- und Qualitätssicherungsaufgaben, wie zum Beispiel die Konsistenzsicherung über den gesamten Entwicklungsprozess. Dabei benutzt das Framework Dienstorchestrierungen von Werkzeugen nach SOA (Service-Oriented Architecture)- und ESB (Enterprise Service Bus)-Prinzipien.

Die Software-Werkzeuge einer Prozesslandschaft werden durch die Bereitstellung von ModelBus® Adaptoren an den Bus angeschlossen. Für die Anbindung von IBM Rational Doors, Eclipse und Papyrus, Sparx Enterprise Architect, Microsoft Office, IBM Rational Software Architect, IBM Rational Rhapsody oder MathWork Matlab Simulink stehen Adaptoren zur Verfügung.

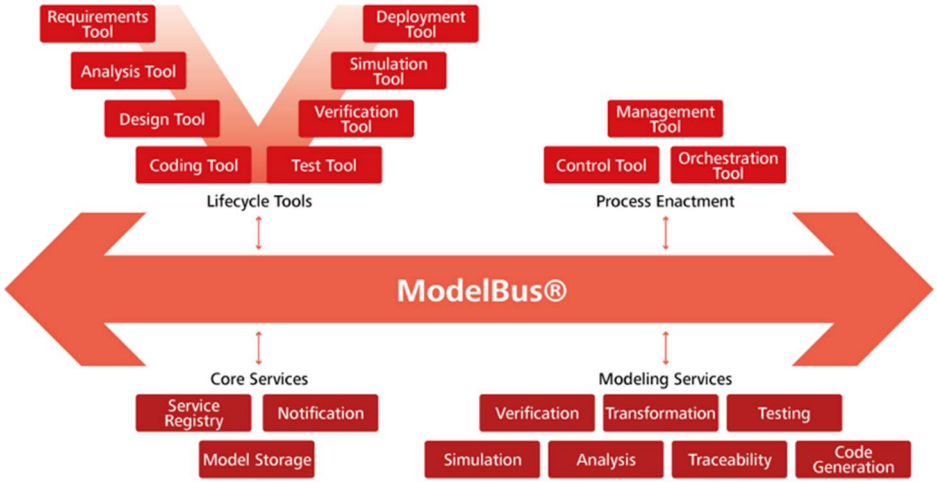

Abb. 4: Prozessautomatisierung mit ModelBus

Risikobewertung und automatisierte Sicherheitstests

Sicherheitskritische softwarebasierte Systeme werden nach dem ISO-Standard „Risk management“ (ISO 31000, [18]) einer sorgfältigen Risikoanalyse und -bewertung unterworfen, um die Risiken zu erfassen und zu minimieren. Für komplexe Systeme kann dieses Risiko Management jedoch sehr aufwendig und schwierig sein. Während die subjektive Einschätzung erfahrener Experten im Kleinen eine akzeptable Methode zur Risikoanalyse sein kann, müssen bei zunehmender Größe und Komplexität andere Ansätze beispielsweise Risikobasiertes Testen [21] gewählt werden.

Eine weitere Möglichkeit zu einer objektiveren Analyse besteht im Einsatz von Sicherheitstests entsprechend ISO/IEC/IEEE „Software and systems engineering - Software testing“ (ISO 29119-1, [19]). Dazu bietet es sich an, zunächst von Experten eine High-Level-Abschätzung der Risiken basierend auf Erfahrung und Literatur durchführen zu lassen. Um dieses initiale Risiko-Assessment genauer zu machen, können Sicherheitstests genau dort eingesetzt werden, wo das erste High-Level-Risikobild die größten Unsicherheiten aufweist. Die objektiven Testergebnisse können dann benutzt werden, um das bisherige Risikobild zu erweitern, zu verfeinern oder zu korrigieren. Wirtschaftlich anwendbar wird diese Methode jedoch erst mit angemessener Tool-Unterstützung.

RACOMAT ist ein von Fraunhofer FOKUS entwickeltes Werkzeug für das Risiko-Management, welches insbesondere das Risiko Assessment mit Sicherheitstests kombiniert [20]. Das Sicherheitstesten kann

dabei unmittelbar in Ereignis-Simulationen eingebunden werden, welche RACOMAT zur Berechnung von Risiken nutzt. RACOMAT ermöglicht eine weitgehende Automatisierung von der Risikomodellierung bis hin zum Sicherheitstesten. Bestehende Datenbanken etwa von bekannten Bedrohungsszenarien werden von RACOMAT genutzt, um ein hohes Maß an Wiederverwendung sicherzustellen und Fehler zu vermeiden.

Dabei wird von RACOMAT ein komponentenbasiertes, kompositionales Risiko Assessment unterstützt. Es werden intuitiv verständliche Risikographen verwendet, um ein Bild der Risikolage zu modellieren und zu visualisieren. Für die Risikoanalyse können bekannte Verfahren wie Fault Tree Analysis (FTA), Event Tree Analysis (ETA) und Conducting Security Risk Analysis (CORAS) in Kombination eingesetzt werden, um von den verschiedenen Stärken der einzelnen Verfahren profitieren zu können. Ausgehend von einem Gesamtbudget für das Risiko Assessment berechnet RACOMAT, wie viel Aufwand für das Sicherheitstesten sinnvoll ist, um die Qualität des Risikobildes durch Reduktion von Unsicherheiten zu verbessern. Das Werkzeug gibt Empfehlungen, wie diese Mittel eingesetzt werden sollen. Dafür identifiziert RACOMAT relevante Tests und priorisiert diese.

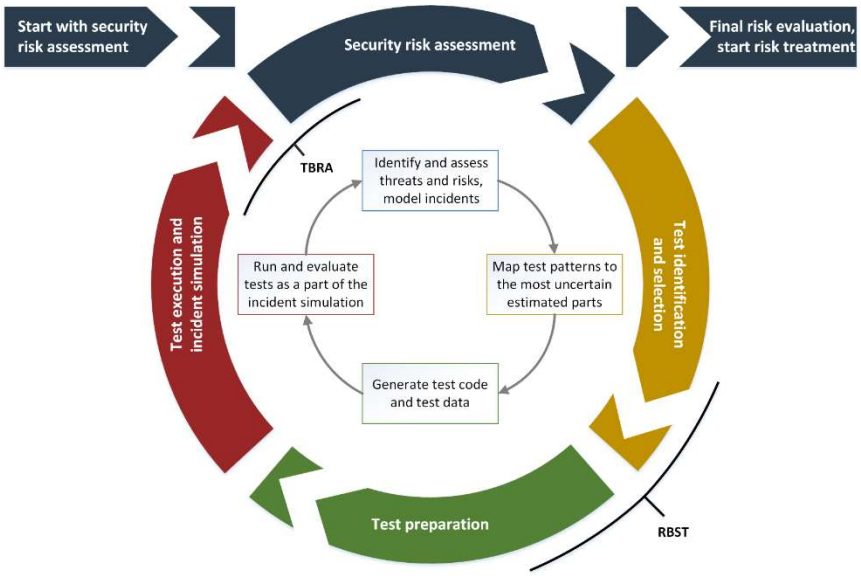

Abb. 4: Risikoanalyse und Sicherheitstests mit RACOMAT

Software-Vermessung und Visualisierung

Softwarebasierte Systeme werden durch immer mehr Funktionen sowie hohe Sicherheits-, Verfügbarkeits-, Stabilitäts- und Benutzbarkeitsanforderungen immer komplexer. Damit es dadurch nicht zu Qualitätseinbußen kommt und strukturelle Probleme frühzeitig erkannt werden können, muss bereits am Anfang des Entwicklungsprozesses mit der Qualitätssicherung begonnen werden. Hierfür eignet sich ein modellgetriebener Entwicklungsprozess, in dem Modelle von zentraler Bedeutung für die Qualität des softwarebasierten Systems sind. Bisher wurden für diese jedoch weder Qualitätskriterien definiert noch etabliert. Zukünftig müssen Modelleigenschaften und deren Qualitätsanforderungen identifiziert und zusätzlich Mechanismen gefunden werden, mit denen sich ihre Eigenschaften und Qualität ermitteln lassen.

Metrino ist ein Werkzeug, das die Qualität von Modellen prüft und sicherstellt [22]. Es kann in Kombination mit ModelBus® genutzt, aber auch unabhängig eingesetzt werden. Mithilfe von Metrino

lassen sich Metriken für domänenspezifische Modelle generieren, selbständig definieren und verwalten. Die erstellten Metriken können auf alle Modelle angewendet werden, die mit dem als Entwicklungsbasis genutzten Meta-Modell übereinstimmen. So analysiert und verifiziert Metrino Eigenschaften wie Komplexität, Größe und Beschreibung von Software-Artefakten. Zusätzlich bietet das Werkzeug verschiedene Möglichkeiten, um die rechnerischen Ergebnisse der Metriken zu überprüfen und grafisch darzustellen – etwa in einer Tabelle oder in einem Netzdiagramm. Indem Metrino Ergebnisse mehrerer Evaluationen speichert, können zudem Ergebnisse aus unterschiedlichen Zeiträumen analysiert und miteinander verglichen werden. Nur so kann eine optimale Qualität des endgültigen, (komplexen) softwarebasierten Systems gewährleistet werden.

Metrino basiert auf dem Structured Meta-Model (SMM), das von der Object Management Group (OMG) entwickelt wurde, und kann sowohl für Modelle der Unified Modeling Language (UML) als auch für domänenspezifische Modellierungssprachen (DSLs) genutzt werden. Der Anwendungsbereich von Metrino schließt außerdem spezielle, werkzeugspezifische Sprachen und Dialekte mit ein.

Egal ob beim Entwerfen eingebetteter Systeme oder bei Software im Allgemeinen: Metrino ist in unterschiedlichsten Domänen anwendbar. Das Werkzeug kann Metriken verwalten und sie gleichermaßen auf (Modell-)Artefakte oder auch auf die komplette Entwicklungskette anwenden, was die Rückverfolgbarkeit und Abdeckungsanalyse miteinschließt.

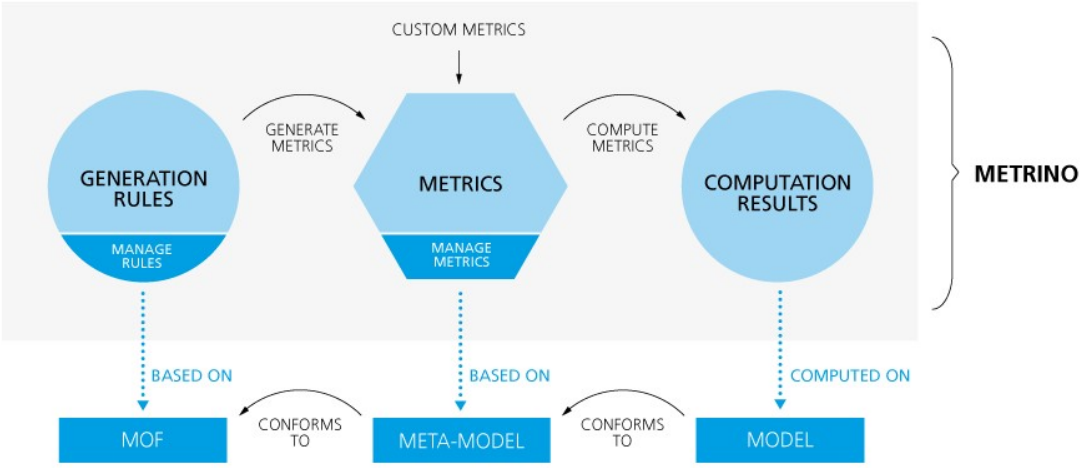

Abb. 4: Modell-basierte Software-Vermessung und Visualisierung mit Metrino

Modell-basiertes Testen

Die Qualität von Produkten entscheidet maßgeblich über ihre Akzeptanz im Markt und insbesondere in Märkten mit sicherheitsrelevanten Produkten, wie z. B. der Medizin-, Transport- oder Automatisierungsbranche, wird der Qualitätssicherung daher eine hohe Priorität eingeräumt. In diesen Branchen ist Qualität ebenso entscheidend für die Zulassung von Produkten. Dennoch sind die Budgets für die Qualitätssicherung begrenzt. Für Manager und Ingenieure ist es daher wichtig, dass die vorhandenen Mittel effizient eingesetzt werden. Noch finden oftmals manuelle Testmethoden Anwendung, auch wenn so nur eine vergleichsweise geringe Anzahl von Tests erzeugt und durchgeführt werden kann und dies zudem sehr fehleranfällig ist. So ist die Effizienz manueller

Testmethoden begrenzt und steigende Kosten sind unvermeidbar. Eine wertvolle Alternative bietet die modellbasierte Testgenerierung und -ausführung: Die Verwendung von Modellen, aus denen automatisch Testfälle abgeleitet werden können, bietet ein enormes Potenzial zur Steigerung der Testqualität bei niedrigeren Kosten. Darüber hinaus hat sich in Fallstudien und bei Praxiseinsätzen gezeigt, dass sich die bei der Einführung modellbasierter Testverfahren nötigen Investitionskosten in Technik und Schulungen bereits nach kurzer Zeit amortisieren [24].

So bietet Fokus!MBT eine integrierte Testmodellierungsumgebung, die den Benutzer zielführend entlang der Fokus!MBT-Methodik leitet und so die Erstellung und Nutzung des zugrundeliegenden Testmodells vereinfacht [25]. Ein Testmodell beinhaltet testrelevante strukturelle, verhaltensspezifische und methodikspezifische Informationen, die das Wissen des Testers maschinenverarbeitbar konservieren. Dadurch lässt es sich jederzeit anpassen oder auswerten – etwa zur Generierung weiterer testspezifischer Artefakte. Weitere Vorteile des Testmodells sind die Visualisierung und Dokumentation der Testspezifikation. Als Modellierungsnotation verwendet Fokus!MBT das von der Object Management Group spezifizierte und von FOKUS maßgeblich mitentwickelte UML Testing Profile (UTP), eine testspezifische Erweiterung zur in der Industrie weit verbreiteten Unified Modeling Language (UML). Dies ermöglicht den Testern, die gleichen Sprachkonzepte der Systemarchitekten und Anforderungsingenieure zu verwenden, beugt so Kommunikationsproblemen vor und fördert das gegenseitige Verständnis.

Fokus!MBT basiert auf der flexiblen Eclipse RCP-Plattform, dem Eclipse Modeling Framework (EMF) sowie Eclipse Papyrus. Als UTP-basierte Modellierungsumgebung verfügt es über alle Diagramme der UML, sowie zusätzliche testspezifische Diagramme. Neben den Diagrammen setzt Fokus!MBT auf ein proprietäres Editor-Framework zur Darstellung und Bearbeitung des Testmodells. Die grafischen Editor-Oberflächen lassen sich gezielt für die jeweiligen Bedürfnisse bzw. Kenntnisse der Benutzer optimieren. Dabei wird, wenn nötig, gänzlich von UML/UTP abstrahiert, was es auch IT-fremden Fachexperten ermöglicht, modellbasierte Testspezifikation in kurzer Zeit zu erstellen. Dies wird zudem durch die Bereitstellung kontextspezifischer Aktionen unterstützt, die den Benutzer entlang der Fokus!MBT-Methodik leiten. So werden methodisch inkorrekte oder im jeweiligen Kontext nicht zielführende Aktionen gar nicht erst ermöglicht. Darauf aufbauend integriert Fokus!MBT automatisierte Modellierungsregeln, welche die Einhaltung von Richtlinien – insbesondere Modellierungs- oder Namenskonventionen – nach und während der Arbeiten am Testmodell garantieren. Diese präventiven Qualitätssicherungsmechanismen unterscheiden Fokus!MBT von anderen UML-Werkzeugen, beschleunigen die Modellerstellung und minimieren kostspielige Reviewsitzungen.

Die Validierung des zu testenden Systems hinsichtlich seiner Anforderungen ist das wesentliche Ziel aller Testaktivitäten. Die konsequente und lückenlose Nachverfolgbarkeit, insbesondere zwischen Anforderungen und Testfällen, ist dabei unverzichtbar. Fokus!MBT geht einen Schritt weiter und bezieht zudem die Testausführungsergebnisse in die Anforderungsnachverfolgbarkeit innerhalb des Testmodells mit ein. Dadurch entsteht ein durchgängiges Nachverfolgbarkeitsnetzwerk zwischen Anforderung, Testfall, Testskript und Testausführungsergebnis, wodurch der Status der jeweiligen Anforderungen oder der Testfortschritt unmittelbar berechenbar werden. Die Visualisierung der Testausführungsergebnisse ermöglicht darüber hinaus, den Ablauf der Testfallausführung zu analysieren, aufzubereiten und auszuwerten. Somit beinhaltet das Testmodell alle relevanten Informationen, um die Qualität des getesteten Systems abzuschätzen und das Management bei seiner Entscheidungsfindung über die Freigabe des Systems zu unterstützen.

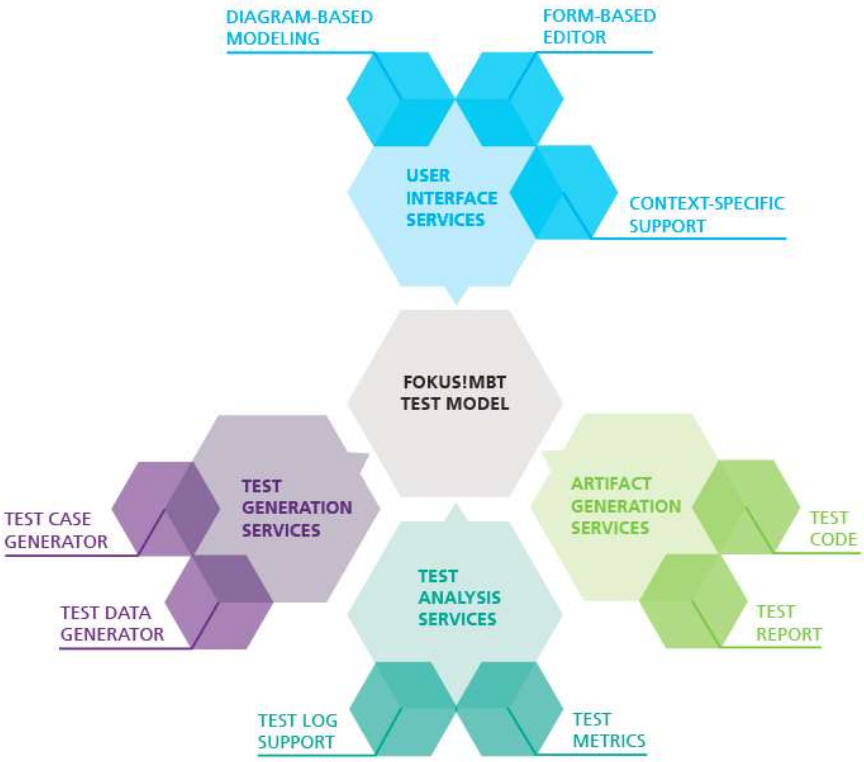

Abb. 2: Modell-basiertes Testen mit Fokus!MBT

Testautomatisierung

Analytische Methoden und insbesondere dynamische Prüfungsansätze sind ein zentrales und oft auch exklusives Instrument, um die Qualität gesamter Systeme zu überprüfen. Software-Tests benötigen dabei alle typischen Elemente der Software-Entwicklung, denn Tests sind selbst softwarebasierte Systeme und müssen deshalb genauso entwickelt, konstruiert, geprüft, validiert und ausgeführt werden. Testsysteme besitzen darüber hinaus die Fähigkeit zu kontrollieren, zu stimulieren, zu beobachten und das System unter Test zu bewerten. Obwohl Standardentwicklung und Programmierungstechniken meist auch für Tests anwendbar sind, sind spezifische Lösungen für die Entwicklung eines Testsystems wichtig, um auf dessen Besonderheiten Rücksicht nehmen zu können. Dieser Ansatz trieb die Entwicklung und Standardisierung von spezialisierten Testspezifikations- und Testimplementierungssprachen voran.

Einer der ursprünglichen Gründe für die Entwicklung der Tree and Tabular Combined Notation (TTCN) war die präzise Konformitätsdefinition für Protokolle von Telekommunikations-Komponenten gemäß ihrer Spezifikationen. Testspezifikationen wurden eingesetzt, um Testprozeduren objektiv zu definieren und das Equipment auf regelmäßiger Basis zu bewerten, zu vergleichen und zu zertifizieren. So wurde die automatisierte Ausführung auch für TTCN überaus wichtig.

Über die Jahre wuchs die Bedeutung von TTCN und verschiedene Pilotprojekte demonstrierten eine erfolgreiche Anwendbarkeit außerhalb der Telekommunikation. Mit der Annäherung der Telekommunikations- und der Informationstechnologiebranche wurde die direkte Anwendbarkeit von TTCN auch für Entwickler aus anderen Branchen sichtbar. Diese Trends sowie die Charakteristika jüngerer IT- und Telekommunikationstechnologien stellten auch neue Anforderungen an TTCN: Ergebnis ist TTCN-3 (Testing and Test Control Notation Version 3, [27]).

TTCN-3 ist eine standardisierte und moderne Testspezifikations- und Testimplementierungssprache, die vom European Telecommunication Standards Insitute (ETSI) entwickelt wurde. Fraunhofer FOKUS war maßgeblich an der Entwicklung von TTCN-3 beteiligt und ist verantwortlich für verschiedene Elemente der Sprachdefinition, inklusive Part 1 (Konzepte und Kernsprachen), Part 5 (Laufzeit-Schnittstellen) und Part 6 (Testkontroll-Schnittstellen) und TTCN-3 Werkzeuge und Testlösungen [28][29].

Mit Hilfe von TTCN-3 können Tests textuell oder grafisch entwickelt und automatisiert zur Ausführung gebracht werden. Im Gegensatz zu vielen (Test-)Modellierungssprachen umfasst TTCN-3 nicht nur eine Sprache für die Spezifikation von Tests, sondern ebenso eine Architektur und Ausführungsschnittstellen für TTCN-3 basierte Testsysteme. Aktuell nutzt FOKUS beispielsweise TTCN-3 für die Entwicklung der Eclipse IoT-Testware zur Prüfung und Absicherung von IoT-Komponenten und Lösungen [30].

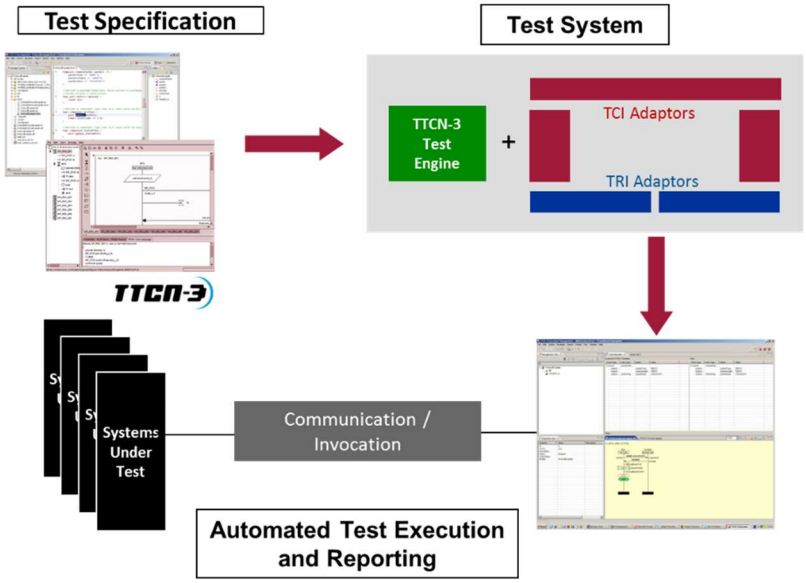

Abb. 2: Testautomatisierung mit TTCN-3

Weitere Ansätze

Nicht alle unsere Methoden, Technologien und Werkzeuge können hier vorgestellt werden. In unseren Publikationen (siehe auch [31]) finden sie weitere Informationen zu

- Sicherheitsorientierten Architekturen,
- Prüfen und Zertifizieren funktionaler Sicherheit,
- Modell-basiertem Re-Engineering,
- Modell-basierter Dokumentation,
- Modell-basierter Portierung in die Cloud oder
- Modell-basierten Fuzz Tests.

Weiterbildungsangebote

Es genügt nicht allein, neue Methoden, Technologien und Werkzeuge zu entwickeln. Diese müssen ebenso vermittelt werden und in Projekten und Piloten bei ihrer Einführung begleitet werden.

So engagiert sich Fraunhofer FOKUS seit langem bei der beruflichen Weiterbildung. Das Institut hat in Kooperation mit dem ASQF (Arbeitskreis Software-Qualität und Fortbildung [32]) und in Kooperation mit dem GTB (German Testing Board [33]) und dem ISTQB (International Testing Qualifications Board [34]) die folgenden Weiterbildungsschemata initiiert und / oder maßgeblich mitentwickelt:

- 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

Zudem bildet Fraunhofer FOKUS mit der Hochschule für Technik und Wirtschaft und der Technischen Hochschule Brandenburg ein Konsortium im Lernlabor Cybersicherheit [35] mit Weiterbildungsmodulen zu

- Secure Software Engineering
- Security Testing
- Quality Management & Product Certification
- Sichere E-Government-Lösungen
- Sichere Public-Safety-Lösungen

Diese und weitere - wie Angebote zu Semantic Business Rule Engines oder Offenen Verwaltungsdaten - sind zudem über die FOKUS-Akademie [36] erreichbar.

Ausblick

Software-Entwicklung und Qualitätssicherung befinden sich im Spannungsfeld von zunehmender Komplexität, der Forderung nach qualitativ hochwertiger, sicherer und zuverlässiger Software und dem gleichzeitigen ökonomischen Druck auf kurze Entwicklungszyklen und schnelle Produkteinführungszeiten.

Modell-basierte Methoden, Technologien und Werkzeuge adressieren die daraus resultierenden Herausforderungen und unterstützen insbesondere moderne, agile Entwicklungs- und Absicherungsansätze. Kontinuierliche Entwicklung, Integration, Prüfung und Inbetriebnahme profitieren in besonderem Maße von modell-basierten Ansätzen, da sie ein starkes Fundament für Automatisierung bilden und mit ihrer Unabhängigkeit von konkreten Software-Technologien auch zukünftige Technologieentwicklungen unterstützen können.

Es sind weitere Fortschritte in der modell-basierten Entwicklung zu erwarten beziehungsweise bereits Teil der aktuellen Forschung. Während die eigentliche Integration sowie die Testausführung bereits nahezu vollständig automatisiert durchgeführt werden, ist die Analyse und Behebung von Fehlern immer noch eine größtenteils manuelle Tätigkeit, was aufwendig und selbst wiederum fehleranfällig ist und so zu immensen Verzögerungen und Kosten führen kann. Selbstreparierende Software unter Anleihe der vielfältigen Software-Komponenten in Open Source Software, mittels Mustererkennung und Analyse durch Deep Learning-Methoden und Reparatur und Bewertung mittels evolutionärer Software Engineering-Ansätze würde einen weiteren Schritt in der Automatisierung bringen. So könnte Software nicht nur selbstlernend, sondern selbstreparierend werden.

Aber bis dahin gilt es

- Software Engineering als Ingenieursdisziplin zu verstehen und es den Experten zu überlassen, komplexe softwarebasierte, vernetzte Systeme zu entwickeln und abzusichern
- Software Engineering selber als Forschungs- und Entwicklungsbereich weiter auszubauen und unsichere, manuelle Schritte der Software-Entwicklung und Absicherung zu automatisieren
- Zu den Software-Plattformen Monitoring- und Prüfumgebungen für alle Ebenen einer digitalisierten Anwendungslandschaft aufzusetzen, die über Virtualisierungsmethoden effizient verwaltet werden können.
- Sicherheit, Interoperabilität und Benutzungsfreundlichkeit nehmen einen zunehmenden Stellenwert in der Qualität softwarebasierter vernetzter Systeme ein und erfordern Priorität beim Entwurf, der Entwicklung und Absicherung.

Referenzen

[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.
[33] GTB: German Testing Board, http://www.german-testing-board.info/, besucht am 25.7.2017.
[34] ISTQB: International Software Testing Qualifications Board, http://www.istqb.org/, besucht am 25.7.2017.
[35]Fraunhofer: Lernlabor Cybersicherheit, https://www.academy.fraunhofer.de/de/weiterbildung/information-kommunikation/cybersicherheit.html, besucht am 25.7.2017.

[36]FOKUS: FOKUS-Akademie, https://www.fokus.fraunhofer.de/de/fokus/akademie, besucht am 25.7.2017.
[37]FOKUS: System Quality Center, https://www.fokus.fraunhofer.de/sqc, besucht am 25.7.2017.