Conference Journal 2019

Interesting Insights into Professional Practice

Papers of Lecturers at the Software Quality Days

2019 software quality days

EXPERIENCE THE VALUE OF QUALITY

Impressum

Publisher / Herausgeber

Software Quality Experts GmbH
Gewerbepark Urfahr 8
4040 Linz

https://www.software-quality-days.com/
info@software-quality-days.com
+43 5 0657-0

Published by Software Quality Days, January 2019

Content

KANN SOFTWARE UNETHISCH SEIN? **4**
DER BEITRAG, DEN SOFTWARE ENGINEERING ZU ETHISCHER SOFTWARE LEISTEN KANN
Dr. Andrea Herrmann

CLOUD AGNOSTIC CONTINUOUS QUALITY ASSURANCE **8**
A BRIEF GUIDELINE FOR QUALITY ASSURANCE
Andreja Sambolec

AUFBAU EINER CD-PIPELINE FÜR EINE PLATTFORM APPLIKATION ALS DEVOPS ANSATZ: EIN ERFAHRUNGSBERICHT **12**
AUF WELCHE PROBLEME STÖßT MAN BEI EINEM SCHRITTWEISEN AUFBAU EINER MODERNEN CONTINUOUS-DELIVERY-PIPELINE UND GLEICHZEITIGER UMSTELLUNG AUF AGILE SOFTWAREENTWICKLUNG? WAS SIND UNSERE ANSÄTZE? MIT WELCHEN TOOLS BZW. PRAKTIKEN HABEN WIR DIE UMSETZUNG VOLLZOGEN UND WIE HAT DAS TEAM REAGIERT? DAS LANGFRISTIGE ZIEL DABEI IST, DAS DEVOPS-KONZEPT SO GUT ALS MÖGLICH ZU VERWIRKLICHEN UND DIE MAßNAHMEN DAFÜR NACHHALTIG ZU VERBESSERN UND ZU ERWEITERN.
Dipl.-Ing. Patrick Koch, Dipl.-Ing. Michael Mitter

SAG'S MIT EINEM LÄCHELN **23**
EMOTIONSBASIERTE QUALITÄTSSICHERUNG
Simon André Scherr

DISCOVER QUALITY REQUIREMENTS **30**
... WITH THE MINI-QAW, A SHORT AND FUN WORKSHOP FOR AGILE TEAMS
Thijmen de Gooijer, Michael Keeling, Will Chaparro

HOW SOFTWARE ENGINEERS READ YOUR QUALITY PROCESSES **35**
OR: HOW TO WRITE BETTER SOFTWARE QUALITY PROCESSESQUALITY PROCESSES ARE NECESSARY IN TODAY'S INDUSTRIAL ENVIRONMENTS TO ENSURE DELIVERY OF HIGH QUALITY PRODUCTS IN AN EFFICIENT WAY. HOWEVER, WRITING AND DEPLOYING SUCH PROCESSES IS A NON-TRIVIAL ACTIVITY.
Johannes Loinig

THE IOT-TESTWARE **39**
FUNCTIONAL AND NON-FUNCTIONAL TESTING FOR THE IOT
Axel Rennoch, Alexander Kaiser

Kann Software unethisch sein?

Der Beitrag, den Software Engineering zu ethischer Software leisten kann

Software kann als Werkzeug genutzt werden und ist dann genauso ethisch neutral wie ein Stück Papier. Das ändert sich, sobald Software Informationen filtert oder sogar Entscheidungen trifft. Analog zu Privacy by Design sollte auch Ethics by Design Prinzip ethische Kriterien beim Software Engineering von Anfang an mit berücksichtigen.

Software als Werkzeug ist ethisch neutral

Ethik ist die Lehre vom guten oder bösen Handeln. Darüber, was nun konkret gut oder böse ist, lässt sich Bücher füllen. Im Großen und Ganzen gibt es jedoch einen gesellschaftlichen Konsens darüber, der in Gesetzen, Höflichkeitsregeln und Hausordnungen codifiziert ist. Doch das soll uns hier weniger beschäftigen als die Frage, wie ethisch Software ist und wie Software Engineering dazu beitragen kann, ethische Software zu entwickeln.

Ob Sie einen Brief von Hand auf Papier schreiben oder auf einer Tastatur in einen Texteditor eintippen, ändert nichts an dessen Inhalt und daran, welche ethischen Folgen dieser Inhalt haben wird. Nur die Form ist eine andere. Man könnte über das Detail diskutieren, ob handgeschriebene Nachrichten persönlicher und respektvoller sind als getippte, aber das ist wohl Geschmackssache und abhängig davon, was momentan als üblich gilt. Der Texteditor als solcher ist jedoch nicht unethisch, sondern nur ein reines Werkzeug.

Auch dass Informationen digital vorliegen, ist nicht ethisch oder unethisch an sich, sondern bringt eben das eine oder andere Risiko mit sich. Die bessere Verfügbarkeit von Daten wird erkauft durch ein erhöhtes Risiko von unautorisierten Zugriffen und Kopien, von Datenverlust oder Datendiebstahl. Wer Daten digital verwaltet, ist ethisch und auch per Gesetz dazu verpflichtet, diese angemessen sicher aufzubewahren. Grundsätzlich scheint es in unserer Gesellschaft aber den praktischen Konsens zu geben, dass wir der Bequemlichkeit halber gewisse erhöhte Risiken akzeptieren

Informationsaufbereitung ist ethisch gesehen kritisch

Entscheidungen zu treffen ist meistens ethisch relevant, weil dabei Verantwortung übernommen wird. Dies gilt insbesondere dann, wenn Menschen von der Entscheidung betroffen sind. Heutzutage treffen Profis zahlreiche solcher Entscheidungen auf der Grundlage von digital vorliegenden Informationen: Kann dieser Kreditantrag genehmigt werden oder nicht? Sind die Laborergebnisse besorgniserregend? Sind die Grenzwerte überschritten? Besteht Handlungsbedarf?

Grundsätzlich bietet Software die Möglichkeit, umfangreichere, aktuellere und besser aufbereitete Informationen als Entscheidungsgrundlage vorliegen zu haben, als jemals zuvor in der Geschichte der Menschheit. Das ist gut. Allerdings gibt es auf diesem Planeten mehr Informationen, als ein Mensch intellektuell verarbeiten kann. Dabei hilft Software beim Filtern, Aufbereiten, Vergleichen, vor allem auch bei dem Unterscheiden zwischen relevanter Information und solcher, die getrost ignoriert werden kann. Da diese Hilfsmittel bestimmen, welche Informationen uns zur Verfügung stehen und damit die Entscheidung beeinflussen, sind sie ethisch relevant.

Was wäre, wenn durch das Fehlen wichtiger Informationen eine falsche Entscheidung getroffen würde? Man spricht gerne von Filterblasen. Stellen Sie sich vor, eine Suchmaschine würde Ihnen von allen zwei Millionen Suchergebnissen zu Ihrer Frage nur zehn anzeigen und von den anderen erfahren Sie nicht, dass sie überhaupt existieren. Das wäre ein ethisches Problem. Selbst wenn die meisten Benutzer sich tatsächlich mit den ersten zehn Treffern schon zufrieden geben, weil sie die gesuchte Antwort schon gefunden haben, macht es trotzdem einen Unterschied, ob die restliche Information zugänglich ist oder nicht.

Jeder, der einen Suchalgorithmus programmiert, übernimmt also ethische Verantwortung. Nun können wir aber die Verantwortung für die gefundene Informationsauswahl nicht allein auf den Architekten der Suchfunktion abwälzen. Welche Informationen in einer bestimmten Situation nötig ist, kann der ja gar nicht vorhersehen. Darum muss auch der Benutzer seinen Teil der Verantwortung tragen: Er muss kompetent mit Suchbegriffen umgehen können, die Regeln der Informationsfilterung seiner Werkzeug kennen (z. B.: Was kann man auf einer Ultraschallaufnahme erkennen und was wird nicht angezeigt?), verschiedene Informationsquellen anzapfen, sich vielleicht sogar selbst einen Vorrat an häufig benötigten Informationen anlegen. Wer sein gesamtes Weltbild ergoogelt oder sich auf die Nachrichten auf einer einzigen Webseite oder in einer einzigen Zeitung verlässt, der ist selbst

schuld, wenn er einseitig informiert bleibt. Die Digitalisierung hat definitiv den Zugang zu Informationen erleichtert und die müssen wir nutzen.

Wenn Software Entscheidungen trifft

Wenn Experten aufgrund von vorgefilterter Information Entscheidungen treffen, können sie die Fehler der Suchfunktion eventuell noch ausgleichen. Sie können anders suchen, weitere Quellen prüfen, das gefundene Wissen durch ihre Erfahrung und durch Daten aus ihrer eigenen Datenbank ergänzen. Richtig kritisch wird die Situation jedoch, wenn die Software selbständig entscheidet. Schon wenn sie Entscheidungen vorschlägt, stellt sich die Frage, welche Informationen ihr zugrunde liegen, welche Kriterien sie verwendet, wie diese gewichtet werden, ob diese überhaupt in der aktuellen Situation passen. Ist der Entscheidungsalgorithmus intransparent, kann er leider nicht geprüft werden.

Betrachten wir als simples Beispiel die automatische Rechtschreibprüfung versus Rechtschreibkorrektur. Die Rechtschreibprüfung unterschlängelt potenzielle Tippfehler rot, gibt dem Benutzer also einen Hinweis oder schlägt sogar eine Verbesserung vor. Die Entscheidung über eine Korrektur trifft der Benutzer. Es ist ja durchaus möglich, dass er ein Wort aus künstlerischen Gründen anders schreiben möchte, einen neuen Begriff erfunden hat oder Wörter aus einer Fremdsprache entlehnt.

Bei der automatischen Rechtschreibkorrektur dagegen entscheidet der Texteditor selbsttätig, ein Wort zu korrigieren, auch ohne ausdrückliche Benachrichtigung. Damit übernimmt der Computer die Verantwortung für die Rechtschreibung. In diesem Beispiel kann der Autor die erfolgte Manipulation beim Korrekturlesen immerhin entdecken und korrigieren. Das ist bei vielen Anwendungen nicht mehr der Fall. Vieles, was Software automatisch tut, könnten die Benutzer nicht mehr von Hand tun. Umso wichtiger ist die dann Korrektheit des Algorithmus!

Ethics by Design

Die Europäische Datenschutzgrundverordnung verlangt, dass beim Software Engineering von Software, die personenbezogene Daten verarbeiten wird, bereits Datenschutzaspekte berücksichtigt werden. Dieses Prinzip nennt sich Privacy by Design. Dies ist konsistent mit der bekannten Tatsache, dass sich gerade Qualitätseigenschaften von Software wie Sicherheit, Performanz und dergleichen nur schwer im Nachhinein nachrüsten lassen, sehr wohl aber systematisch hinein designen. Man muss nur von Anfang, ab der Anforderungsermittlung, diesen Aspekt berücksichtigen. Bei Privacy by Design lauten die Fragen: Welche zu verarbeitenden Daten sind personenbezogen oder personenbeziehbar? Welche Verarbeitung durch wen soll erlaubt oder nicht erlaubt sein?

Analog dazu schlage ich vor, dass im Sinne eines „Ethics by Design“-Prinzips während der Anforderungsanalyse an eine Software bereits ethische Aspekte berücksichtigt werden. Ein Mindestmaß an ethischem Niveau wird bei den meisten Anwendungen durch Gesetzeskonformität erreicht. Die Einhaltung der Datenschutzgesetze ist ein Teilaspekt von Ethik. Ethische Software ist grundsätzlich gesetzeskonform, doch das genügt nicht. Ethik geht über gesetzliche Regelungen hinaus. Die oben diskutierten Aspekte rund um die Informationsbereitstellung und Entscheidungsfindung müssten dabei meiner Meinung nach einen Schwerpunkt bilden:

Wenn ein Mensch auf der Grundlage digital bereitgestellter Informationen Entscheidungen trifft, muss sichergestellt sein, dass diese Informationen dafür geeignet sind, genau diese Entscheidung bestmöglich unterstützt wird. Auch wenn nicht alle potenziell relevanten Informationen gleichzeitig angezeigt werden können oder sollen, sollte es doch für den Benutzer ersichtlich sein, dass und wo noch weitere Informationen vorliegen, wie die aktuelle Auswahl getroffen wurde und welche Aspekte gegebenenfalls nicht angezeigt werden.

Trifft die Software selbständig Entscheidungen, dann müssen diese nachvollziehbar sein und durch einen Menschen veränderbar. Zu allererst muss jedoch geprüft werden, ob es überhaupt ethisch vertretbar ist, dass diese Entscheidung automatisiert wird. Wie hoch ist die Wahrscheinlichkeit für eine Fehlentscheidung und welcher Schaden kann dabei schlimmstenfalls entstehen? Wird dieses Risiko angemessen ausgeglichen? Wenn beispielsweise das selbstfahrende Auto oder ein medizinisches Diagnosesystem mehr Leben rettet als es zerstört, könnte dies ein Argument sein. Bei strengerer Betrachtung dagegen könnte man sich auf den Standpunkt stellen, dass Maschinen grundsätzlich keine Menschen töten dürfen. Hier prallen zwei klassische Ethik-Systeme aufeinander: Der Utilitarismus strebt das maximale Gesamtwohl an und betrachtet hier das durchschnittliche Risiko, das sich durch Automatisierung in vielen Bereichen sicher optimieren lässt. Dies ist insbesondere darum der Fall, weil auch Menschen zahlreiche Fehler unterlaufen. Die deontologische Prinzipienethik jedoch würde eine Software ablehnen, die potenziell jemals einen Menschen tötet. Welche der beiden Schulen sich durchsetzen sollte, wage ich nicht zu entscheiden.

Zum Glück gibt es nicht nur die Vollautomatisierung oder die gute alte Handarbeit durch fehleranfällige Menschen. Eine Kombination von Mensch und Maschine ergibt in vielen Fällen das optimale Team: Die Maschine stellt die Informationen zur Verfügung und bereitet sie auf, der Mensch entscheidet.

Diese ethischen Fragen sind definitiv noch nicht zu Ende diskutiert und müssen insbesondere fallweise entschieden werden. „Ethics by Design“ als Software Engineering Prinzip einzuführen wäre ein erster Schritt, dass Ethik überhaupt ein Thema bei der Entwicklung von Software würde, und zwar von Anfang an. Stakeholder und deren Bedürfnisse zu analysieren, gehört ohnehin zur Anforderungsanalyse.

Weiterer Lesestoff zum Thema

Die Fachgruppe Rechtsinformatik der Gesellschaft für Informatik e.V. erstellte neulich erst im Auftrag des Sachverständigenrats für Verbraucherfragen eine Studie zum Thema „Technische und rechtliche Betrachtungen algorithmischer Entscheidungsverfahren“. Dort werden unter anderem die ethische Zertifizierung von Algorithmen und die Einrichtung einer staatlichen Stelle für algorithmische Entscheidungen empfohlen.

Bibliographie

Gesellschaft für Informatik (2018). Technische und rechtliche Betrachtungen algorithmischer Entscheidungsverfahren. Studien und Gutachten im Auftrag des Sachverständigenrats für Verbraucherfragen. Berlin: Sachverständigenrat für Verbraucherfragen. ISSN: 23658436, verfügbar unter: http://www.svr-verbraucherfragen.de/wp-content/uploads/GI_Studie_Algorithmenregulierung.pdf

Autorin

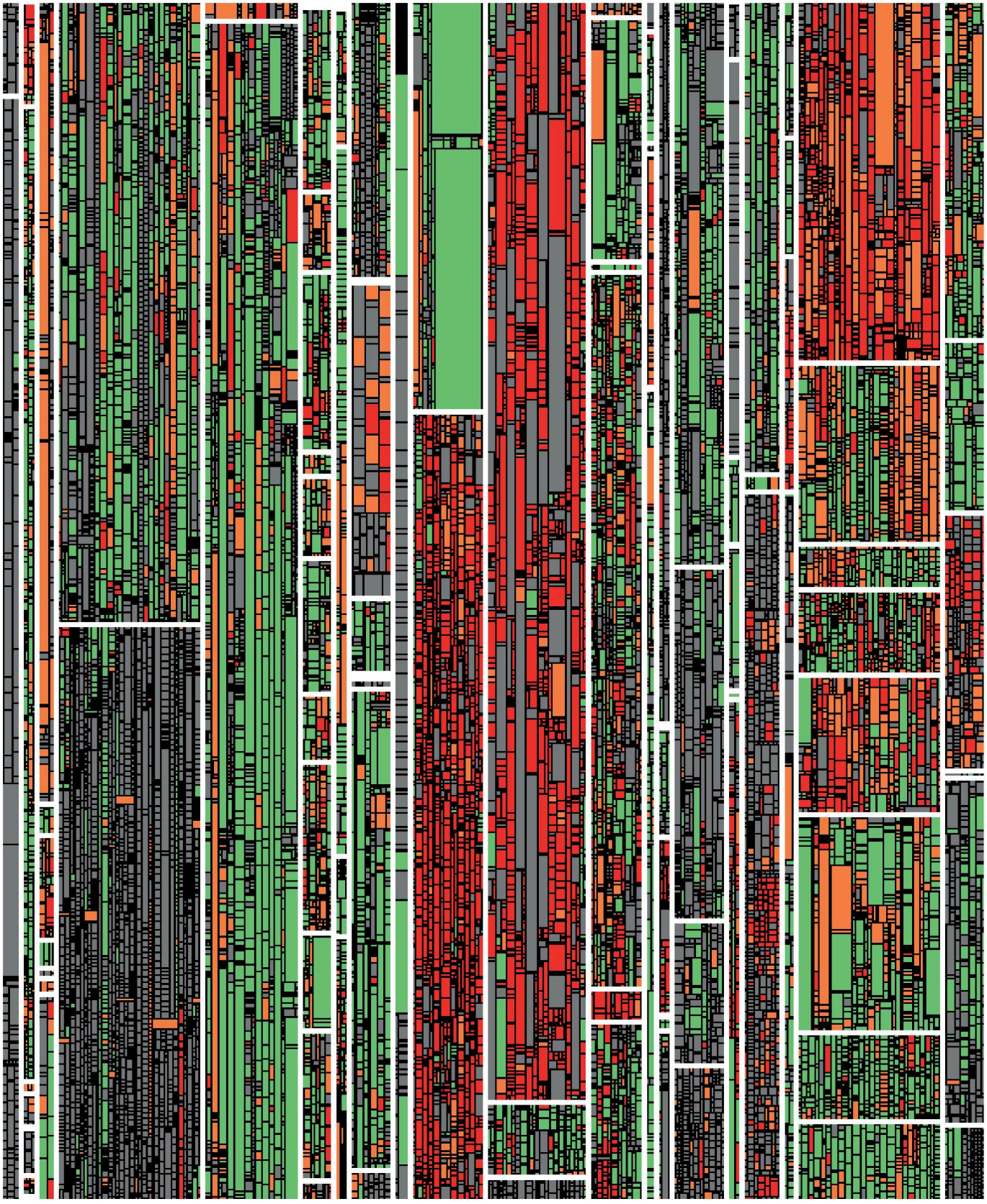
Dr. Andrea Herrmann

Freiberufliche Trainerin und Beraterin in Software Engineering, mehr als 20 Jahre Berufserfahrung in IT-Projekten, Forschung und Schulungen, drei Gastprofessuren.

www.herrmann-ehrlich.de

herrmann@herrmann-ehrlich.de

Haben wir die entscheidenden Änderungen eigentlich getestet?



www.testgap.io

Cloud agnostic continuous quality assurance

A brief guideline for quality assurance

Scaling up any software development project usually brings lots of hidden obstacles that can affect the throughput of development teams and even delay or completely derail projects. One technical aspect of any software project that has big influence on the velocity of development teams is code quality. Defining code quality is very difficult and team dependent, but it is a very important starting point. A good definition should focus on readability and long term maintainability of code and help to avoid implementation of processes and tools that would be redundant or an overhead. The main focus of this article is to provide provide examples of cloud agnostic tools and techniques that can be implemented on any cloud and on premise data-center as support for developers to maintain code quality.

Introduction

Cloud computing is aggressively affecting ways data centers are designed. With the trend of moving infrastructure into the could, developers have to change their tool-set to match new requirements.

The tools that we want to include are Open Source, because they offer the highest level of flexibility for developers and the lowest entry point expenses. Additionally, they have to support the cloud deployment by supporting containerized run-time environments.

Code review process

Most software teams face issues when it comes to protecting code quality in normal development processes. There are many techniques that can help to overcome common pitfalls. Most important of them is the code review.

A code review itself partially tries to reproduce advantages of pair programming, while enabling better scalability in the implementation. Pair programming is a technique where two developers are working on a single workstation. One developer is the so-called “driver” and is responsible for writing the actual code, while the second one is the “navigator” and is responsible for observing and guiding the “driver”.

Advantages of the pair programming are obvious, such as knowledge sharing and control of the code that is being produced. But there is also a hidden advantage of “navigator” having a completely different mindset than the “driver”. The “navigator” can observe the bigger picture and the bigger context of code being written. The problem with pair programming is scaling it up in big teams and it is even impossible to implement it on a geographically distributed teams.

In case of pull requests, the “driver” is a developer submitting code for review and the “navigator” is a person or a group of people doing the code review. During the review process, the advantages are mostly the same as in pair programming but since the entire process in done via a digital platform, it has even a couple of more.

The entire process of the pull request contains information about the exact changes being done on the code. Most of the tools allow commenting in each changed line individually, adding comments and even staring entire discussion threads for the code that is being submitted. Most valuable of all is that this process stays as a documentation within the system. That means that in the future anybody can refer to it as part of knowledge gathering or even bug fixing and patching.

There are many tools having integrated pull request mechanisms (like GitHub or BitBucket). But since the focus on this article is cloud independent tools, we are going to focus on Gerrit as an open source implementation of code collaboration tool.

Gerrit was first introduced by Google for the Android development requirements. Later on, it was forked and continued being developed as an open source project. Gerrit integrates into development pipelines by offering a solution for the following requirements:

- Git repository management
- Code review platform

Gerrit has some advantages over other tools mentioned before. One of the most important is that it is open sourced and freely available for modification and use. Beyond that it offers API’s for the entire life-cycle of pull requests (Example on Image 1) and allows for an easy integration with other systems via a powerful plugin system and a variety of plugins available to be used.

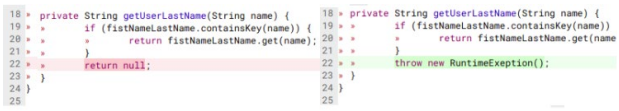


Image 1: Code review

With all properties mentioned, Gerrit fits perfectly into a cloud independent solution for code collaboration and hosting. It can easily be setup on the premises data center. We can do the transition to the cloud simply by using the replication plugin. Also, it can be used as a cloud native solution.

SonarQube

In the software development field, there is a lots of experience gathered as best practice, which helps to avoid most common pitfalls of the programming language that is used. While code review helps in implementing most of those practices, implementation still requires lot of attention and time from the "navigator".

Luckily there are tools available that have gathered these practices and can offer them as automated code analysis solutions. The most prominent solution available on the market, especially in the Java development world, is the SonarQube.

SonarQube is an open source platform which is providing automatic code quality analysis. It can be integrated into the software development pipeline and used as a tooling for developers, to whom it then offers instant information about possible problems in the written code.

Integration into the development pipeline fits into the code review process. We can provide information to "navigator" whether the submitted code is breaking some of the rules being checked by the SonarQube.

This is just a small use case of the platform. SonarQube gathers data about code coverage of test execution, also it gathers the historical information about all analyzes and offers the statistical analyzes of that data.

Rules offered by the platform are quite extensive and configurable. Same as with other tools mentioned here, there is a plugin system that is offering extensions to the platform. Coming together with vibrant community, finding the appropriate solution for the project specific need is quite easy. Example of SonrQube report is shown on Image 2.

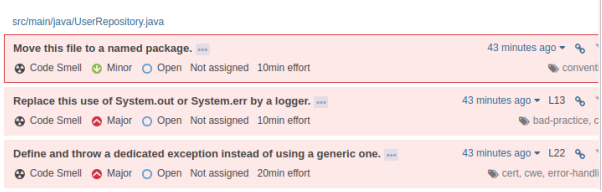


Image 2: SonarQube report

Being feature rich, open source and extendable solution, SonarQube lends on our list of cloud independent solutions. It can be easily configured to run in the data-center or in the cloud.

Jenkins integration with SonarQube and Gerrit

In our entire story up to now we were talking about tools and their integration. The missing peace of the puzzle in this story was the glue holding them all together. This is perfect spot to be filled in by the Continuous integration tool.

Current market of the Continuous integration platforms is quite powerful and offering lots of well integrated and easy to use solutions. In our case we are going to focus on one of the most important open source solutions for continuous integration, Jenkins.

Jenkins started as a fork of the Hudson project, originally developed by the Sun Micro-systems. It is developed as a open source solution for the continuous integration and delivery. Jenkins is based on the plugins which allow integration of almost all types of projects into the continuous integration pipeline.

In our setup, Jenkins is the glue for all off the other tools. When there is a Pull request coming to the Gerrit, a Jenkins job is triggered to perform the SonarQube analysis and report back the result. Beyond that, Jenkins will also report any problems about the build and reject the pull request if it does not satisfy some basic requirements. This way, developers get quick feedback about their code and some time is saved for the "navigator" before even starting the review of the code. Example of Jenkins pipeline on Image 3.

Image 3: Jenkins pipeline

Jenkins satisfies all of our criteria for being cloud independent tool and also can be considered as one of the most important tools in any development pipeline. Setting up Jenkins is easy and deployment on any cloud environment is supported by pre-built Docker containers.

Test coverage benefits

Beyond the lots of metrics offered by the SonarQube, during analyzes we want to mention one of them as being most controversial. Code coverage measures which lines of code were executed during the test execution.

This can on the one hand offer quick feedback to the developer about absence of tests and on the other hand, most importantly, it can indicate branches which may have

been missed. The missed branches can give insights about possible bugs in the code. Code coverage offers developer certain level of trust into the code.

There is also a big downside of this measurement technique. Eli Goldratt wrote: “Tell me how you measure me and I will tell you how I will behave”. If developer’s code is measured in quality by the code coverage, he will most likely write tests to have high code coverage and by this all of the advantages of this metric will be lost.

Code coverage needs to be natural part of the development process and should be carefully observed not to provide the misinformation about code quality. The report can be integrated into the Gerrit review process by using Jenkins for the test execution and storing coverage report into the SonarQube.

Even thought this chapter is not about any tool, it gives clear picture about how integrated development process and tools can add value for “driver” and “navigator”.

Automatic code refactoring

One additional tools that can be added to CI pipeline is tools for automatic code improvements. Unfortunately open source offers are not existing so we will be making exception in this example and refer to proprietary tool jSparrow.

jSparrow is rule based automatic code improvement tool for Java which can be integrated into your continuous integration pipeline. Current implementation offers only integration into Github pull request process as GitHub app version.

On every pull request tool makes fixes and pushes them back to the pull request which can then be reviewed further and merged manually, or by skipping manual review process, merged automatically.

jSparrow is focused on fixing lots of issues detected by SonarQube, and more, which makes is valuable since improvement can be measured and can help save time on fixing annoying issues, specially on legacy projects.

Conclusion

By supporting development life cycle with tools, we can greatly contribute to the overall code quality. Additionally we can also reduce the effort of scaling up the software development teams even to multiple locations with the information flow and processes being preserved.

Another important conclusion we can make is that an open source solutions are mature enough and feature rich enough to support the entire development life-cycle of the product. They are as well ready to be deployed on the wide range of the environments, like on the premise data centers and the cloud.

Author

Andreja Sambolec

Software developer working at Splendit IT-Consulting GmbH.

Java developer, with strong Eclipse plugin development background. Focused on technical knowledge and organizational skills to improve organization and optimize development processes.

andreja.sambolec@splendit.at

Automatic Java Refactoring

Fits in every infrastructure!

eclipse maven

Enhance
Maintainability and Readability

Clean Code

`jSparrow` establishes robust coding standards based on best practices.

Reduce
Code Smells within Minutes

Productivity

Within a few minutes `jSparrow` removes code smells and bugs automatically.

Minimize
Technical Debt and Maintenance Work

Code Quality

`jSparrow` increases code quality and keeps modern Java standards.

Don't lose time with tedious and repetitive work
- let jSparrow do it!

splendit
IT-CONSULTING GMBH
www.splendit.at
www.jSparrow.eu

Aufbau einer CD-Pipeline für eine Plattform Applikation als DevOps Ansatz: Ein Erfahrungsbericht

Auf welche Probleme stößt man bei einem schrittweisen Aufbau einer modernen Continuous-Delivery-Pipeline und gleichzeitiger Umstellung auf Agile Softwareentwicklung? Was sind unsere Ansätze? Mit welchen Tools bzw. Praktiken haben wir die Umsetzung vollzogen und wie hat das Team reagiert? Das langfristige Ziel dabei ist, das DevOps-Konzept so gut als möglich zu verwirklichen und die Maßnahmen dafür nachhaltig zu verbessern und zu erweitern.

Der DevOps-Gedanke ist zurzeit allgegenwärtig und fördert das Bewusstsein, jede Komponente zu optimieren, die einen Einfluss darauf hat, Software schneller, qualitativ hochwertig und stabil auszuliefern. Ebenso gilt es, Mechanismen zu installieren, welche das Feedback der Kunden bzw. Benutzer nachhaltig in den Prozess einfließen lassen. Wie geht man allerdings behutsam vor, um die Basis dafür in Form einer Continuous-Delivery-Pipeline zu schaffen? Dies unter dem Umstand, dass der Ausgangspunkt dafür Prozesse und Mechanismen aus dem Wasserfallmodell sind? Dieser Artikel soll als Erfahrungsbericht dienen, um Problemstellungen vor und während der Umsetzung aufzuzeigen. Des Weiteren soll er erläutern, welche Ansätze zum Erfolg geführt haben.

Einführung

Als Ausgangspunkt für diesen Bericht dienen zwei neu gegründete Scrum-Teams. Beide Teams entwickelten sich stark über die Jahre. Das Software-Produkt ist eine seit über 20 Jahren bestehende Applikation zur Daten-Visualisierung und -bearbeitung, hauptsächlich bestehend aus nativem Code (C++). Für heute selbstverständliche Konzepte wie Continuous-Integration, Test-Automation, Virtualisierung, etc. haben erst umgesetzt werden müssen. Ebenso war bezüglich der zu nutzenden Tools der Einsatz von einem Version-Control-System oder eines Continuous-Integration-Servers noch nicht vorhanden. Das Arbeiten nach agilen Methoden hatte sich ebenso erst zu etablieren. Gleichzeitig wurde der Einsatz der Applikation als Plattform Produkt immer stärker gefördert und eine neue Benutzeroberfläche mit managed Code war in Planung.

Für die Umsetzung einer effizienten und stabilen Continuous-Delivery-Pipeline haben also mehrere Aspekte betrachtet werden müssen.

- **Neue Entwicklungsteams und neue Arbeitsweisen**

Das jeweilige Teamgefüge befand sich hinsichtlich der agilen Methodenumsetzung in einer frühen Entwicklungsstufe: Teamplayer vs. Einzelkämpfer, Prozessversteher vs. Totalverweigerer, Junior vs. Senior.

Die Arbeitsweisen mussten sich aufgrund des Wechsels vom etablierten Wasserfallmodell zu agiler Entwicklung (SAFe) ändern.

- **Neupositionierung und Erweiterung der Applikation**

Die Applikation begann sich strategisch als Plattform Produkt in dem Unternehmen auszurichten. Eine neue Benutzeroberfläche sollte mittels managed Code implementiert werden mit der Prämisse, alle Funktionalitäten der bestehenden GUI weiterhin zu unterstützen. Engines

für Grafik, Mathematik und Datenverwaltung wurden auch aufgrund von Kompatibilitätsgründen nur minimal geändert und verblieben zur Gänze im nativem Code.

- **Design und Implementierung der Build- und Testautomatisierung**

Der Aufbau einer modernen Infrastruktur mit Automatismen wurde gestartet. Die Funktionsweise des vorhandenen, selbstentwickelten Testing-Tools („TestManager“) musste weiterhin gewährleistet und in die Automatisierung integriert werden. Design und Implementierung einer neuen Testumgebung wurden vorangetrieben.

- **Aufreißen von etablierten Strukturen und Prozessen**

Dies betrifft die unterschiedlichen Phasen einer Continuous-Delivery-Pipeline, in unserem Beispiel vor allem die Deployment-Automation. Es waren keinerlei Automatismen vorhanden. Das Deployment und die finalen Tests wurden seit jeher manuell von Kollegen einer dezidierten Testing-Abteilung durchgeführt.

In diesem Artikel wird der Fokus auf die Implementierung von Automatismen und Tools mit den jeweilig auftretenden Problemen gelegt, sowie Best-Practice-Ansätze empfohlen. Ebenso finden alle Aspekte Erwähnung, welche Einfluss darauf gehabt haben. Diese werden anhand des jeweiligen Meilensteins der Entwicklung der Pipeline erläutert.

Definition einer Continuous-Delivery-Pipeline und Beschreibung der Struktur des Artikels

„There is no such thing as The Standard Pipeline, but a typical CD pipeline will include the following stages: build automation and continuous integration; test automation; and deployment automation.” [1]

Abbildung 1: Phasen einer Continuous-Delivery-Pipeline

Abbildung 1 visualisiert die 3 Phasen der Continuous-Delivery-Pipeline. Für jede einzelne Phase werden die Ausgangssituation, die betreffenden Problemstellungen, die praktische Umsetzung und ein Ausblick bezüglich Erweiterungen erläutert.

In dem Buch „Continuous Delivery“ von Jez Humble und David Farely wird eine Deployment-Pipeline mit folgenden Stufen definiert:

„
- The commit stage
- Automated acceptance test stages
- Manual test stages
- Release stage

“
[3]

Im Vergleich zur Definition der Pipeline, die wir als Referenz verwenden, fällt auf, dass sie keine manuelle Teststufe enthält. Dies bedeutet nicht, dass wir kein manuelles Testen anwenden. Wir möchten in diesem Artikel allerdings den Fokus auf Automatisierung und Infrastruktur legen.

Build-Automation und Continuous-Integration

- **Ausgangssituation**

Aus historischen Gründen lag die Verantwortung der Erstellung der Builds von der Applikation bei einem anderen Entwicklungsteam. Das Anwerfen eines neuen Builds erfolgte bei Bedarf und war nur einem bestimmten Personenkreis gestattet. Die Erstellung wurde mittels Aufrufen einer Vielzahl an Batch Skripts (großteils Perl basiert) auf einer dezidierten Server-Maschine umgesetzt. Der Build-Prozess enthielt immer zwei Applikationen, da die Kollegen vom Projektteam ebenso an einem Produkt arbeiteten, welches unserem ähnlich war bezüglich der Funktionalität und der grafischen Benutzeroberfläche. Es bestand eine große gemeinsame Codebasis, daraus resultierte natürlich eine starke Koppelung zwischen zahlreichen Komponenten.

- **Problemstellung**

1. Die Erstellung der Builds lag nicht in der Verantwortung unserer Teams.
2. Das Builden konnte nicht automatisiert durchgeführt werden, nur bestimmte autorisierte Personen waren dazu berechtigt.
3. Kein Einsatz eines Continuous-Integrations-Servers, welcher die Builds mit den entsprechenden Commits transparent machte.
4. Kein Einsatz eines modernen Version-Control-Systems, welches es ermöglicht, in Verbindung mit einem Continuous-Integration-Server Codes Änderungen von Sub-Branches regelmäßig in den Master-Branch überzuführen.

5. Keine Möglichkeit der Build-Parallelisierung und -Queueing.
6. Ein fehlgeschlagenes Build bedeutete mühsame Analysetätigkeiten (u.a. ein großes Logfile für alle Build-Steps).
7. Kein Einsatz von Units Tests zur ersten Prüfung während der kontinuierlichen Integration.
8. Es bestand eine starke Koppelung/Abhängigkeit mit einer anderen Applikation.
9. Mit der Einführung von Sprints und Erstellung von zu buildenden Sub-Branches waren schlagartig effektivere Automatismen notwendig. Dies konnte die existierende Infrastruktur nicht abdecken.

- **Umsetzung**

Aufgrund bereits verfügbarer Lizenzen innerhalb des Unternehmens wurde eine Neuimplementierung mittels Jetbrains TeamCity angestrebt.

Die Projektstruktur wurde von Grund auf neu erstellt: Grob skizziert umfasst diese das Builden der Git-Sourcen für 32 und 64 Bit mit einem abschließenden Upload auf ein Artefakt-Repository (Artifactory). Die Buildschritte bestehen aus klassischen Windows Eingabeaufforderungen, Powershell-Skripts, ebenso das Ausführen von MSBuild zum Builden der Solutions. Variablen werden entsprechend global mittels Build-Parameter gesetzt.

Um Continuous-Integration und den Automatismus eines Nightly-Builds bezugnehmend auf die Ressourcen zu realisieren, waren zumindest immer drei dezidierte Build-Agents den entsprechenden Konfigurationen zugewiesen (derzeit sind es vier).

Nightly-Builds werden nachts automatisch getriggert, die dabei erzeugten Install-Bundles, Nuget-Packages und Plattform-Libraries werden entsprechend versioniert im Artifactory veröffentlicht.

Die Nightly-Builds, die neben dem Bauen der Applikation auch alle Unit-Tests ausführen, können jederzeit manuell gestartet werden.

Continuous-Integration erfolgt nicht nur auf der Ebene der Master-Branches, sondern auch auf jener der jeweiligen Feature-Branches, und das für zumindest eine Bitness. Die Install-Bundles können direkt per Aufsuchen der gewünschten Build-Nummer im Web-Interface von TeamCity heruntergeladen und danach installiert werden. Unit-Tests werden an dieser Stelle ebenfalls automatisiert ausgeführt bevor die Bundles komprimiert zur Verfügung stehen.

- **Ableitungen und Empfehlungen**

Die Einführung von Continuous-Integration funktioniert nicht nur durch eine Tool-Umstellung. Auch das gesamte Entwicklungsteam muss dafür bereit sein, einen Benefit in den neuen Methoden zu sehen. („Habe den Master gebrochen, was nun?“)

Das modulare Konzept von TeamCity erlaubt eine einfache Erweiterung bzw. Skalierbarkeit um zusätzliche Komponenten. Hier ist ein Verwenden von Build-Configuration-Templates und -Parameter unumgänglich.

Die Integration von Issue-Trackern (z.B. Jira) ermöglicht automatisches Erkennen von Commit-Messages (in unserem Fall Git) und erstellt direkte Links zum Issue selbst. Dies ermöglicht es auch Personen außerhalb des Entwicklungsteams wie z.B. Customer Service Kollegen eine einfache Nachvollziehbarkeit. („Welcher Content ist denn im Build enthalten, ist der Fix wirklich im Master oder nur im Feature-Branch?“).

Das Aufsetzen eines neuen Build-Agents scheint nur im ersten Moment ein einfaches Prozedere zu sein. Oftmals liegt hier der Hund im Detail begraben. Als sinnvoll hat sich hier das Erstellen eines Cookbooks (ein „HowTo“ mit allen notwendigen Schritten) etabliert oder auch einfach das Klonen einer bereits bestehenden Instanz, auf welcher der Build-Agent-Dienst läuft, herausgestellt.

Test Automation

- **Ausgangssituation**

Der Testprozess war nicht im Entwicklungsteam verankert, ein andere Abteilung hatte die Verantwortung dafür. Das Testing-Tool („TestManager“) selbst war eine in C# umgesetzte Eigenentwicklung und speziell an die Bedürfnisse der Applikation angepasst. Änderungen im Code der Applikation hatten einen direkten Einfluss auf das Testing-Tool selbst. Die Ergebnisse der einzelnen Testläufe wurden mittels Mails oder telefonischer Nachfrage kommuniziert. Es war notwendig, das Testing-Tool manuell per eigener Benutzeroberfläche zu bedienen. Die Tests wurden an physikalischen Maschinen durchgeführt unter Verwendung von Wechselfestplatten.

Diese Integrationstests, durchgeführt mit dem TestManager, waren damals die einzigen automatisierten Tests, allerdings mit einer hohen Abdeckung der Funktionalitäten der Applikation.

- **Problemstellung**

1. Im Team herrschte wenig Bewusstsein für das Testen bzw. über den Einsatz des Testing-Tools („Prozess schreibt zwingend Testen vor, dann machen wir es eben.“)
2. Der Entwickler des Testing-Tools wechselte in eine andere Abteilung, das Know-how über das Tool ging zu einem Großteil verloren.

   Es wies eine starke Koppelung an die Applikation auf, war nicht vollautomatisiert verwendbar und bezüglich Architektur nicht ausgelegt für Erweiterungen.
3. Die Testresultate waren nicht transparent für die Teams. Dadurch war auch kein Mehrwert

für die Entwicklung selbst erkennbar („Mein lokaler Build ist eh fehlerfrei!“).

4. Es bestand keinerlei Ausfallsicherheit bei den Test-Maschinen. Bei einem Stromausfall gingen sämtlich Resultate und Konfigurationen verloren.
5. Aufgrund des Gebrauchs von Wechselfestplatten und des nicht möglichen vollautomatisierten Einsatzes des Testing-Tools waren zahlreiche manuelle Tätigkeiten notwendig. Angefangen von dem manuellen Einsatz von Wechselfestplatten, dem Neustart des Rechners, der Installation der Software sowie des Starts eines Testlaufs.
6. Die Tests wurden nicht parallel durchgeführt, schon gar nicht für unterschiedliche Versionen der Applikation. Das rein sequentielle Ausführen war sehr zeitintensiv.
7. Es gab keine transparente Testhistorie bez. Testausführung (Zeitpunkt der Testausführung, Testergebnisse, Betriebssystem). Eine Ableitung von Trends bezüglich Softwarequalität war unmöglich.
8. Die Testausführung erfolgte nicht in regelmäßigen Abständen. Dadurch wurden etwaige Fehler aufgrund neuer Commits mitunter recht spät identifiziert.
9. Für den gesamten Testprozess gab es keine offiziell verantwortlichen Personen.
10. Das Test Resultat war nicht verlässlich, eine Reihe von Flaky-Tests war immer im Resultat enthalten.
11. Die Tests waren in ihrer Wartung sehr aufwändig. Ein Ersetzen der Tests war nicht möglich, da sie Funktionalitäten prüften, die über Jahrzehnte entwickelt worden waren. Wir benötigten Gewissheit über den großen Anteil an Legacy-Codes.
12. Wir konnten die separaten Branches nicht mit den Integrationstests prüfen, sondern erst nach dem Merge in den Master.
13. Es gab keine funktionalen, automatisierten Tests für die grafische Benutzeroberfläche

- **Umsetzung**

Beibehaltung der aktuellen Version des Testing-Tools und Integration einer API zur Automatisierung mittels TeamCity

Der erste Ansatz war, Bestehendes in Form des aktuellen Test-Tools weiterzuverwenden. Es wurde aber eine Automatisierung angestrebt, sowie die Notwendigkeit gesehen, die Testresultate transparent zu machen. Daher wurde eine API in das Testing-Tool eingeschleust, sodass es per Windows-Eingabeaufforderung angesprochen werden konnte. Es fand zudem eine Integration in eine separat aufgesetzte TeamCity Umgebung statt. Die Testläufe wurden übergangsmäßig auf einer physikalischen Maschine durchgeführt. Skripts wurden zur automatisierten Installation und Deinstallation der Applikation erstellt, eine Notwendigkeit für den Testlauf.

Dies stellte sich im Laufe der Zeit als zu fragil heraus. Eine nachhaltigere Implementierung war gewünscht.

Neudesign des Testing-Tools und Integration in einen Continuous-Integration-Server

Dieser Vorgang führte dazu, dass das Testing-Tool von Grund auf neu geschrieben wurde und zwar mit einer moderneren und nachhaltigeren Architektur der Klassen. Der Fokus wurde auf den Anwendungsfall gerichtet, sodass das Tool nicht mehr von einem Benutzer bedient werden sollte, sondern per zur Verfügungstellung von geeigneten Argumenten von einem Continuous-Integration-Server per Eingabeaufforderung angesprochen wird. Ein eigenes Version-Control-System-Repository wurde eingeführt, Der Code wurde nun öffentlich für alle Mitglieder des Teams. Zudem wurde es nach mehreren Testläufen auf einem separaten Test Continuous-Integration-Server in einem offiziellen Continuous-Integration-Server des Unternehmens integriert- nun fiel die Wartung des Servers vom Team weg. Die ersten Build-Chains wurden integriert, welche ausgehend von einem fertigen Build der Applikation, über jeweils einzelne Build-Konfigurationen die Installation der Applikation durchführten, den Testlauf sowie eine Deinstallation. Anhand er Testresultate konnte nun jedes Mitglied der Teams auf die Builds der Applikation mit dem jeweiligen Commit schließen. Nachfolgend wurden die physikalischen Maschinen mit virtuellen Maschinen ersetzt. Der Testlauf war nun voll automatisiert, die Resultate transparent und man konnte Rückschlüsse auf die entsprechenden Builds der Applikation machen im Falle von fehlgeschlagenen Tests.

Evaluierung und Integration eines Test-Frameworks zur Prüfung der Benutzeroberfläche

Die Einführung einer neuen grafischen Benutzeroberfläche mittels managed-Codes bereitete uns Sorgen bezüglich Sicherstellung der Qualität. Dies führte zur Evaluierung und Einführung eines Frameworks zur Erstellung von automatisierten GUI Tests. Die Erstellung von Tests zur Abdeckung der gängigsten Use-Cases wurde vorangetrieben. Auch diese Tests wurden in den Continuous-Integration-Server integriert, um die Ergebnisse sichtbar zu machen.

Einführung einer neuen Stage und Wechsel zu einem neuen Continuous-Integration-Server

Um plattform-spezifische Anforderungen zu gewährleisten, wurden automatisierte Deployments bzw. Benutzeroberflächentests eingeführt, welche auf Produktionsumgebungen ausgeführt werden. Schrittweise erfolgte auch der Wechsel von TeamCity zu dem Team Foundation Server (TFS) von Microsoft.

- **Ableitungen und Empfehlungen**

Das Testen von Legacy-Software bedarf spezieller Anforderungen:

1. Eine Integration des Testing-Tools in einen Continuous-Integration-Server, sowie ein automatisiertes Scheduling der Tests ist unbedingt notwendig, um regelmäßig die Qualität zu überprüfen, Rückschlüsse auf Commits ziehen zu können und die Resultate für alle Mitglieder der Teams sichtbar zu machen. Zudem war mittels Continuous-Integration-Servers auch einfach möglich, die separaten Konfigurationen (Builden, Testen, Deployen, etc.) miteinander zu verknüpfen.
2. Anfangs hatten wir immer ein bestimmtes Set an Flaky-Tests in den Resultaten unserer Integrationstests, dadurch sank das Vertrauen aller Teammitglieder bezüglich der automatisierten Testroutinen. Wir investierten viel Zeit in Adaptierungen, um Flaky-Tests zu vermeiden. Das Testresultat muss aussagekräftig sein und man muss darauf vertrauen können um schnell weitere Schritte bezüglich der Fehlerbehebung zu unternehmen.
3. Die Konfiguration unserer Testmaschinen ist sehr aufwändig. Hierfür wurden alle notwendigen Schritte dokumentiert. Eine noch bessere Variante wäre ein automatisiertes Aufsetzen der Testmaschinen. Hier würde für uns z.B.: eine Container-Virtualisierung in Frage kommen. Ein Teammitglied soll nicht damit beschäftigt sein, stundenlang eine Maschine mühsam manuell aufzusetzen.
4. Unsere ersten automatisierten Benutzeroberflächentests bestanden aus Abbildung von komplexen Use Cases. Schnell lernten wir, dass die Wartbarkeit aufgrund ständiger Änderungen in der UI sehr aufwändig war, Tests schlugen oft fehl.

   Jedes Team muss für sich einen Konsens finden, wie viele Ressourcen es in die Wartung der Tests stecken möchte. Wir erstellen nun vermehrt automatisierte Tests für die Benutzeroberfläche, welche aus wenigen Aktionen bestehen, weil sie besser wartbar sind und in Summe mehr Funktionalitäten überprüfen. So kann man mehr komplexe Use-Cases in den Mittelpunkt des manuellen Testens stecken. Was auch bestimmt motivierender für die entsprechende Person ist, die die Tests händisch durchführt.

Deployment-Automation

- **Ausgangssituation**

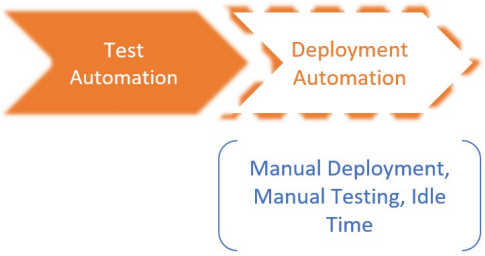


Abbildung 2: Ausgangspunkt für die Implementierung einer Deployment-Automation

Abgesehen von der Prüfung der Applikation mittels Integrationstests, gibt es vor einem Rollout eine finale (externe) Abnahme, welche aus rein manueller Verifikation auf kundenähnlichen Systemen besteht. Zu diesem Zweck findet ein Deployment der Applikation mittels spezifischen Konfigurationen auf den Systemen statt und produktübergreifende manuelle Tests werden durchgeführt. Die Abarbeitung unterliegt nicht den Mitgliedern unserer Teams, sondern Kollegen einer produkt- und teamunabhängigen Test-Abteilung. Die Testabdeckung von Funktionalitäten bzw. Kompatibilitäten ist in dieser Phase allerdings überschaubar, bedingt durch den hohen Workload der Abteilung. Die Durchführung der Tests startet, wenn ein stabiler Zustand der Applikation, sowie der abhängigen Produkte vorliegt. Als wir uns dieser Phase verstärkt gewidmet haben, um die Probleme anzusprechen (siehe Abbildung 2: Ausgangspunkt für die Implementierung einer Deployment-Automation), war die Phase bezüglich der Test-Automation schon sehr stabil und automatisiert.

- **Problemstellung**

1. Agile Softwareentwicklung trifft auf Wasserfall: Der agile Prozess stoppt abrupt bei Zulieferung des zu testenden Builds an die Test-Abteilung.
2. Es gibt weder kontinuierlich automatisierte Deployments noch einen automatisierten Test auf kundennahen Systemen.

3. Das Deployment sowie die Tests werden erst relativ knapp vor einem Rollout auf den entsprechenden System manuell durchgeführt. Selbst kleine Änderung bei den Installationsroutinen machen es oftmals notwendig, dass ein Mitglied vom Entwicklungsteam den Kollegen der Test-Abteilung Unterstützung leistet.
4. Das Deployment und die Tests starten meistens erst, wenn die Defects der Applikation behoben sind und die Autotests erfolgreich waren. Eventuelle Probleme mit abhängigen Produkten werden dadurch erst spät erkannt.
5. Wegen der vergleichsweisen kleinen Anzahl an Mitgliedern, aber der großen Diversität an zu testenden Produkten, kann es in der Test-Abteilung zu längeren Wartzeiten kommen, bis die Überprüfung startet.
6. Ergebnisse aus Test Metriken des Entwicklungsteams werden ignoriert

- **Umsetzung**

Der erste Schritt war die Implementierung von Automatismen, welche die aktuellen Builds der Applikation ohne manuelles Zutun auf den spezifischen Systemen der Test-Abteilung mittels gewünschter Konfiguration installierten. Zu diesem Zeitpunkt waren wir ein wenig erfahrener mit dem Umgang mit TFS: wir konnten schnell starten und kamen gut voran, denn das automatisierte Deployment führten wir bereit auf unseren eigenen Systemen durch um die Integrationstests dagegen laufen zu lassen.

Neu war für uns, dass wir auch andere, von unserer Applikation abhängige Produkte ins Deployment miteinbeziehen mussten, welche entweder als Addon zusätzlich installiert wurden oder als Toolbox einfach nur von der Applikation geöffnet wurden. Die notwendigen aktuellen Versionen wurden von den Kollegen der Test-Abteilung auf Netzwerklaufwerken zur Verfügung gestellt.

Zusätzlich zu dem Deployment waren aber auch Tests notwendig. Wir entschieden uns für die Erstellung von Smoke-Tests, also sehr rudimentär gehaltenen Tests, diese dafür jedoch zahlreicher. Sie wurden mit Hilfe von Kollegen der Test-Abteilung mittels des Ranorex [3]-Frameworks durchgeführt.

Denn einerseits brauchten wir das Applikations Kow-How der Kollegen bezüglich der produktübergreifenden Use-Cases, andererseits sollten auch sie die Tests durchführen, warten und die Resultate interpretieren können.

Der erste Meilenstein in dieser Phase waren nun entsprechende Automatismen in TFS, welche mittels zweier abhängiger Produktinstallationen mehrere einfach gehaltene automatisierte Benutzeroberflächentests starteten, um konkrete Use Cases abzubilden. Dies stellte sich als guter Ausgangspunkt für Erweiterungen heraus.

- **Ableitungen und Empfehlungen**

1. Automatisiertes, produktübergreifendes Deployment und Testing zeigen frühzeitig Fehler auf und verringern das Risiko, das Rollout zu dem definierten Zeitpunkt zu gefährden.
2. Die Zusammenarbeit mit Kollegen der Test-Abteilung wurde stark gefördert. Ein Know-How-Transfer bezüglich Ranorex, TFS, Applikationswissen und dem Release-Prozesses findet wechselseitig statt. Existierende kundennahe Umgebungen werden als Staging-Umgebungen genutzt bzw. ausgebaut.
3. Die Nutzung von kundenähnlichen Systemen (Staging) macht in weiterer Folge den Einsatz von Monitoring-Systemen sehr interessant, vor allem bezüglich Performance-Tests, welche auf unseren eigenen Maschinen eventuell nicht so aussagekräftig sind.

Übersicht über die Continuous-Delivery-Pipeline

Entwicklung der Pipeline über die folgenden Jahre

Die zwei folgenden Grafiken (siehe Abbildung 3 und Abbildung 4) sollen die Entwicklung der Pipeline seit Bestehen der Scrum-Teams und der Einführung der agilen Softwareentwicklung bezüglich Infrastruktur, Tooling und Testabdeckung darstellen: Man erkennt, dass wir schnell den Einsatz von virtuellen Maschinen, sowie den Version-Control-Systems vorantrieben. Als dies etabliert war, konnten wir die Eingliederung eines weiteren Testing-Frameworks wagen, bis hin zum Einsatz von Container-Virtualisierung beim derzeit aktuellen Stand der Pipeline.

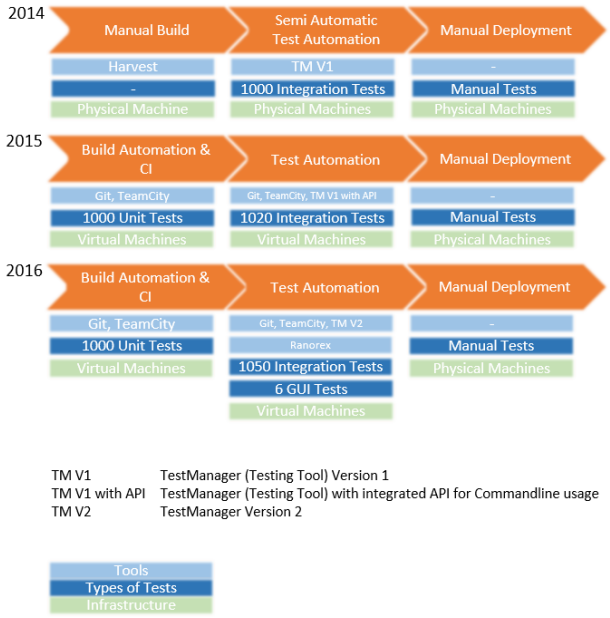


Abbildung 3: Die Ausgangsposition sowie die ersten zwei Jahre der Entwicklung

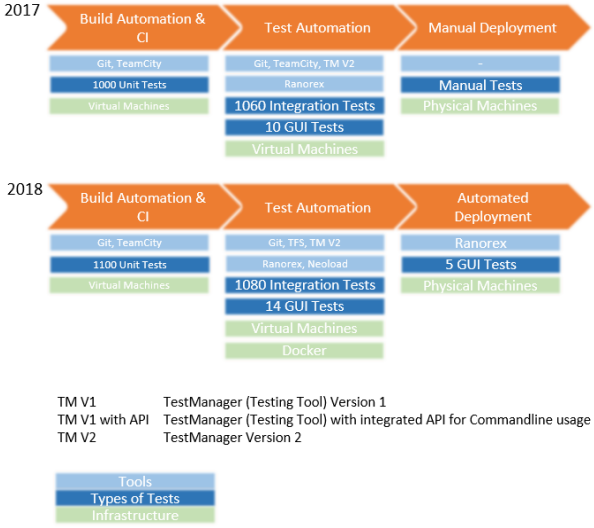


Abbildung 4: Weitere Fortschritte mit dem aktuellen Stand der Pipeline

- **Die Continuous-Delivery-Pipeline im Kontext von DevOps**

Neben Aspekten hinsichtlich Kultur und Methoden ist eine DevOps-Strategie nur dann erfolgreich, wenn auch entsprechende Werkzeuge zur Verfügung stehen. Die hier dargestellte Continuous-Delivery-Pipeline nimmt Bezug auf den „DevOps-Kreislauf" und verdeutlicht die aktuelle Tool-Abdeckung in den einzelnen Phasen.

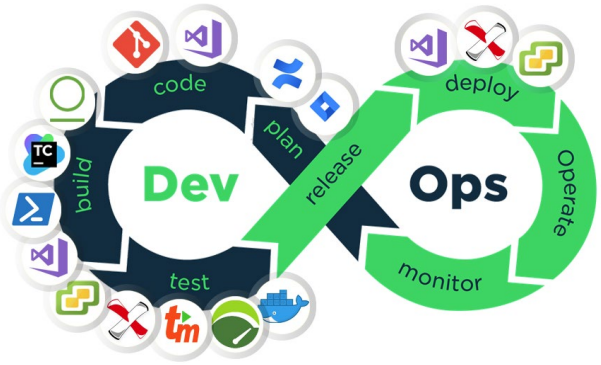


Abbildung 6: DevOps – Toolchain [5-13]

Zusammenfassung

Über die Jahre hinweg sammelten wir Erfahrungen in unterschiedlichen Teams und Rollen, in mehreren Unternehmen unter Verwendung verschiedenster Programmiersprachen, Tools und Prozessen. Die Zusammenarbeit mit Kollegen in einem internationalen Umfeld, welche unterschiedlichste Charaktereigenschaften aufweisen, prägen die soziale Kompetenz jedes einzelnen. Uns ist klar, dass es keinen universell ableitbaren Masterplan für Vorhaben in dieser Art. Jedes neue „DevOps-Unternehmen" wird seine eigene Geschichte schreiben. Rückblickend ergeben sich für uns folgende Schlussfolgerungen:

Mache das Team und die Organisation bereit für technische Optimierungen und neue Prozesse – Übe dich dabei in Geduld, trefft Entscheidungen gemeinsam.

Eine scheinbar revolutionäre Verbesserung in der Infrastruktur ist wertlos, wenn nicht das gesamte Team von Anbeginn eingebunden ist und nicht jedes Mitglied die Idee und den Mehrwert dahinter erkennt. DevOps startet vorrangig im Kopf, über Jahre manifestiertes Gedankengut wird plötzlich einer Revolution unterworfen.

Um ein schnelleres Erkennen von fehlerhaften Commits und damit eine raschere Behebung der Fehler zu gewährleisten, automatisierten wir unsere Integrationstests sowie das dafür notwendige Deployment der Applikation. Die Einbindung in einen Continuous-Integration-Server sollte die Automatismen transparent machen. Vor allem wurde angestrebt, die Testresultate für jedes Mitglied frei zugängig zu machen, um Schlüsse daraus ziehen zu können, wenn nach einer Veröffentlichung von Commits eine höhere Anzahl an nicht erfolgreichen Tests zu vermelden ist.

Nicht für jedes Teammitglied war diese offensichtliche Verbesserung anfangs eine große Errungenschaft, vor allem für jene vom Typ „einsamer Wolf, ich treffe ausschließlich meine eigenen Entscheidungen". Standhaftes Ignorieren trotz mehrfachen Hinweisens über die steigende Anzahl von fehlgeschlagenen Tests und dem Verweis auf die entsprechende Build-Konfiguration machten dies deutlich.

Ein Ansatz unsererseits war ein ebenso fortlaufendes Ansprechen der aktuellen Testresultate im (Daily-Standup). Ebenso die Hervorhebung der Vorteile einer schnellen Prüfung und die einfache Einsicht in Testergebnisse. Schließlich waren die Rückschlüsse auf die jeweiligen Commits klar und die Abarbeitung eines nachtäglichen, daraus resultierenden Defects dann doch unerwünschter Mehraufwand.

Wir erweiterten schließlich die Prüfung der Applikation durch die Integrationstests auf Sub-Branch-Ebene. So konnte jeder Entwickler per Knopfdruck die Qualität seines Branches prüfen. Die Integration in den Master konnte mit bestem Gewissen durchgeführt werden.

Wähle den Zeitpunkt der Optimierungen bzw. Erweiterungen mit Bedacht – Mühsam ernährt sich das Eichhörnchen.

Bleiben wir bei dem Beispiel bezüglich der Einführung des automatisierten Deployments und des automatisierten Testen der Integrationstests: Anfangs musste ein Kompromiss gefunden werden. Wir konnten nicht mit der Abrissbirne radikal alles niederreißen und sämtliche bis dahin etablierten, manuellen Prozesse zerschmettern. Auch war nicht ausreichendes Know-How bezüglich TeamCity, der Applikation und Funktionsweise des Testing-Tools gegeben. Zusätzlich mussten die Release-Zulieferungen gewährleistet werden, welche damals noch mehr manuellen Aufwand in Anspruch genommen haben.

Die Zwischenlösung war das Einziehen einer API in das Testing-Tool, damit eine Einbindung und eine Automatisierung mittels TeamCity möglich war. Die ersten Erfahrungen konnten gemacht und das Know-How schrittweise aufgebaut werden. Der darauf folgende Schritt war eine neue Implementierung des Testing-Tools, denn die bestehende Architektur war nicht dafür gerüstet, um neue Funktionalitäten einzuführen, die für weitere Verbesserungen bei den Testabläufen sorgten.

Erkenne und nutze Synergien – Baue Brücken.

Mittlerweile erstellen und warten nicht nur die verantwortlichen Scrum-Teams Tests für die Applikation. Jene Mitglieder des abteilungsfremden Teams, zuständig für die finalen Tests und die Freigabe zu den Kunden, zeigten größtes Interesse an Automatisierungstechniken und den entsprechenden Testing-Frameworks. Sie hatten das Applikations-Know-How in Verbindung mit weiteren Applikationen, essentiell für die letztendliche Akzeptanz. Ebenso verwalteten sie Systeme, welche sich als Produktionsumgebungen eignen. Wir lieferten tiefergehendes spezifisches Wissen über die Applikation selbst, sowie Erfahrung über Continuous-Integration-Server in Verbindung mit Deployment- und Testautomatisierung. Das Ergebnis ist nun ein Staging-Environment unter gemeinsamer Verwaltung. Ein Ausbau der Zusammenarbeit ist bereits im Entstehen. Das Ziel muss sein, das Vertrauen zwischen den einzelnen Parteien zu stärken und die Zusammenarbeit zu optimieren.

Investiere in neue Methoden und Technologien – versuche dennoch nicht, das Rad komplett neu zu erfinden

Die Implementierung einer neuen grafischen Benutzeroberfläche führte zu der Evaluierung und anschließenden Integration eines neuen Test-Frameworks. Hier wollten wir bewusst auf bestehende, sich am Markt bereits etablierte Software zurückgreifen. Eine selbstentwickelte Insellösung kam nicht in Frage. Die Skepsis war anfangs groß bezüglich des gewählten Frameworks („Dies kann nicht funktionieren, wir haben es nicht selbst entwickelt.“), ebenso hinsichtlich Erstellung und Wartung der automatisierten Tests. Nur langsam etablierte sich die implementierte Lösung innerhalb des Scrum-Teams. Wir haben Best-Practice Ansätze für das Design der Tests gefunden, konnten leicht und schnell Anpassungen vornehmen. Testergebnisse werden nun transparent dargestellt, die erstellten Tests sind leicht lesbar.

Ein weiteres Beispiel ist die Nutzung von Container-Virtualisierung: Zu Beginn war uns der Vorteil nicht klar, nicht objektive Einzelmeinungen untergruben weitere Untersuchungen. Wir mussten erst den geeigneten Einsatz und daraus abgeleiteten Nutzen dafür finden. Wir starteten mit der Auslagerung von unseren Lizenz-Server-Diensten in Container. Dies war der unkomplizierteste Wechsel.

Bibliography

[1] **Andrew Phillips,** The Continuous Delivery Pipeline — What it is and Why it’s so Important in Developing Software
https://devops.com/continuous-delivery-pipeline/

[2] **Janet Gregory, Lisa Crispin,**
Agile Testing: A Practical Guide for Testers and Agile Teams

[3] **Jez Humble, David Farely**, Continuous Delivery: reliable software releases through build, test and deployment automation, 2011, Seite 110, Kapitel 5

[4] **Ranorex GmbH**, Strassganger Strasse 289, 8053 Graz, Österreich
https://www.ranorex.com

[5] **TeamCity**, JetBains,
https://www.jetbrains.com/teamcity/

[6] **Git**, https://git-scm.com/

[7] **NeoLoad**, Neotys,
https://www.neotys.de/neoload/overview

[8] **Docker**, https://www.docker.com/

[9] **Artifactory**, JFrog, https://jfrog.com/artifactory/

[10] **Visual Studio, TFS, PowerShell**,
https://visualstudio.microsoft.com/tfs/

[11] **Jira, Confluence**, Atlassian,
https://www.atlassian.com

[12] **Vsphere**, VMware, https://www.vmware.com

[13] **Irma Kornilova**, DevOps is a culture, not a role, https://medium.com/@neonrocket/devops-is-a-culture-not-a-role-be1bed149b0

Autoren

Dipl.-Ing. Patrick Koch

Test Automation Engineer bei AVL List GmbH

Patrick Koch sammelt seit sechs Jahren Erfahrungen in der Testautomatisierung und arbeitet derzeit als Test Automation Engineer. Zudem ist er Leiter der „DevOps Community Of Practice“ der AVL. Er war verantwortlich bei dem Design und der Implementierung einer modernen Build- und Testinfrastruktur und interessiert sich für neue Technologien, die den DevOps-Gedanken fördern

patrick.koch@avl.com

Dipl.-Ing. Michael Mitter

Product Owner bei AVL List GmbH

Mit einer fundierten technischen Ausbildung an der TU Graz und mehr als 15 Jahren an praktischer Berufserfahrung im Softwareentwicklungsbereich war Michael Mitter u.a. als zertifizierter SDQA-Manager maßgeblich am Aufbau einer QA-Abteilung beteiligt. Weiters war er als technischer Projektleiter und Requirements Enginieer in einem internationalen Umfeld tätig. Als zertifizierter Scrum Master und Product Owner ist er mit der Anwendung agiler Methoden bereits seit mehr als 10 Jahren vertraut.

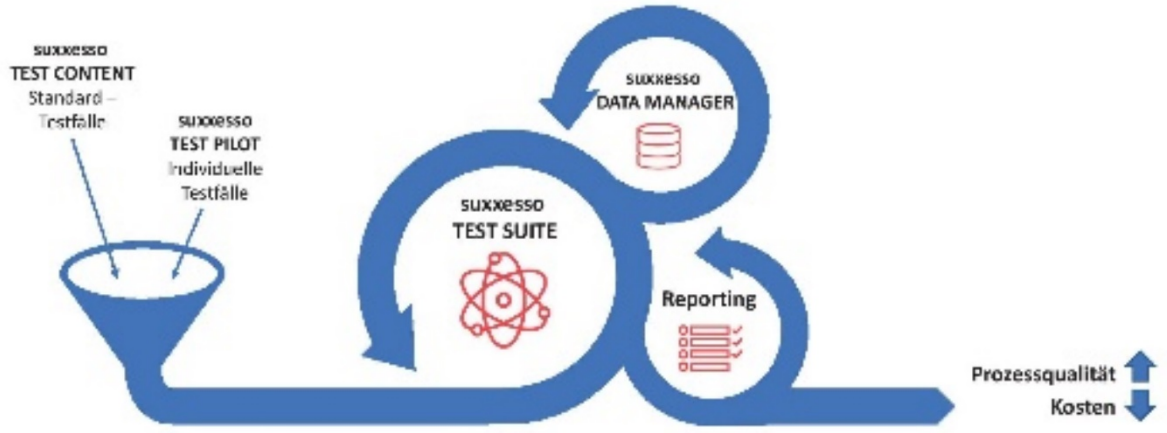
Bei allen Tätigkeiten ging der Fokus auf Software-Qualität und das ständigen Bestreben nach Verbesserungen, sowohl aus technischer als auch Prozesssicht, nie verloren.

michael.mitter@avl.com

... und Testautomation funktioniert doch!

Testen Sie Ihre **unternehmerischen Prozesse** umfassend und ressourcenschonend?
Testen Sie mit repräsentativen Testdaten?
Erhalten Sie **aussagekräftige Reports?**

Wir haben die Lösung mit unseren innovativen Produkten:



suxxesso TEST SUITE for SAP® solutions

Sofort einsetzbare, fertig automatisierte, wiederverwendbare Prozessketten
Einfache Ausführung der Testfälle inkl. Reporting
Produktiv in 4 Tagen

suxxesso DATA MANAGER

Vollautomatisierte Selektion von kundenspezifischen Testdaten
Keinerlei Aufwand für die Aufbereitung von Testdaten

suxxesso TEST PILOT

Echtzeit-Automation von individuellen Prozessketten
Einfache und effiziente Erstellung von Testfällen
Automatisierte Generierung der Prozessdokumentation

Investieren Sie nur wenige Minuten und erfahren Sie mehr: www. suxxesso.com
Wir sind für Sie da! Kontaktieren Sie uns unter: office-vienna@suxxesso.com

suxxesso

Sag's mit einem Lächeln

Emotionsbasierte Qualitätssicherung

„Was haben wir denn jetzt verbockt?“ Vor dieser Frage stehen App-Entwickler immer wieder. Nutzer löschen Apps sofort, wenn sie nicht so funktionieren wie erwartet. Typisch für Apps sind eine kurze Time-to-Market, ständige Updates und geringe Budgets. Was nun? Die gute Nachricht ist: Nutzer geben Feedback! Daraus kann man wertvolle Schlüsse ziehen. Heute lassen sich große Mengen von Feedback nicht vollautomatisch auswerten. Veränderungen der Stimmung können aber erkannt werden, ohne alle Details zu verstehen. Um das zu erreichen, analysieren wir die darin enthaltenen Emojis, die die Stimmung des Nutzers widerspiegeln. Dafür haben wir Emojis auf Emotionen untersucht, mit dem Ergebnis, dass Menschen sie ähnlich wahrnehmen. Dieser Artikel zeigt unser Vorgehen, unsere Ergebnisse und was man aus Emojis im Feedback lernen kann, um Apps zu verbessern.

Die Welt, in der wir leben, ist zunehmend vernetzt. Ein großer Teil der Vernetzung wird durch Apps bestimmt. Diese sind zunehmend intelligenter aber auch komplexer. Im Hinblick auf die Entwicklung solcher Systeme stehen wir vor großen Herausforderungen. Essentiell für die Nutzerakzeptanz ist ein fehlerfreier Ablauf, gute Qualität und innovative Ideen. Typisch für den mobilen Bereich sind auch eine kurze Time-to-Market, Updates im Wochenrhythmus sowie geringe Budgets. Darüber hinaus neigen Nutzer schnell dazu Apps wieder zu löschen, wenn sie nicht das machen, was von ihnen erwartet wird. Die Frage: „Was haben wir denn jetzt verbockt?“, ist für Entwickler mit höchster Eiligkeit zu beantworten. Der Druck im Markt bringt aber auch eine zweite Frage mit sich, die schnell einer Antwort bedarf: „Was müssen wir an unserem Produkt verändern?“. Jederzeit muss auf neue Trends und in den Markt eintretende Konkurrenz reagiert werden. Damit ist eine permanente Überprüfung der Anforderungen. Dies alles muss während der regulären Produkt-Entwicklung passieren.

Es ist notwendig diese Prüfungen sowohl effizient, als auch effektiv durchführen zu können und dabei weder hohe Kosten zu verursachen noch lange zu brauchen. Die Herausforderungen sind damit enorm.

Nutzer geben Feedback

Wie können Entwickler auf diese Situation reagieren? Die gute Nachricht ist: Nutzer geben Feedback! Dieses Feedback ist über eine Vielzahl von Kanälen wie App Stores, soziale Medien oder Kundenforen verteilt. Das Feedback der Nutzer spiegelt die Wahrnehmung von Änderungen am Produkt oder der Einsatzumgebung wider. Die breite Streuung von Feedback sorgt dafür, dass eine Vielzahl an Reviews pro Tag im Playstore, im App Store, auf Twitter, im User Voice oder als in-App-Feedback verteilt entstehen, und das auch noch in vielen Sprachen.

In diesem Feedback finden sich zahlreiche Anhaltspunkte für die Verbesserung von Produkten. Wir können Nutzerfeedback in drei Arten aufteilen: Fehlerberichte, Feature-Wünsche und Lob [1]. Während man mit letztem hauptsächlich eine bestehende Qualität wahren oder die entsprechenden Inhalte für Marketingzwecke nutzen kann, sind für die Weiterentwicklung und Verbesserung vor allem Fehlerberichte und Wünsche der Nutzer interessant.

Sammelt man all dieses Feedback kontinuierlich aus den verschiedensten Quellen ein, entsteht eine große Datenmenge. Diese gilt es zu analysieren, was aber manuell zeitlich unmöglich ist. Vollautomatisch lassen sich größere Mengen von Nutzerfeedback aktuell noch nicht auswerten. Die Entwicklung der „Natural Language Processing“ Techniken ist noch nicht fortgeschritten genug.

Dennoch steckt Nutzerfeedback voller Potentiale und kann auch mit den heute verfügbaren Mitteln für die Produktverbesserung genutzt werden. Was wir sehr wohl erkennen können, sind Themen und Trends. Genauer gesagt können Veränderungen der Stimmung im Feedback ohne genaues Verstehen von Details hergeleitet werden. Um den zahlreichen Quellen von Feedback Herr zu werden sowie um an verwendbare Ergebnisse zu kommen, benötigt man ein Maß, das intuitiv verstanden wird und überall anzutreffen ist. Beschränkt man sich nur auf Feedback, dass in App Stores abgegeben wird, hat man die Fünf-Sterne-Bewertungsskala. Diese kann man dort nutzen um eine Einstufung des Beitrages und die damit verbundenen Themen abzuleiten. Leider ist diese Skala nicht in allen Quellen wie z. B. Soziale Medien, Feedbackforen etc. vorhanden. Darüber hinaus hat ein Eintrag im App Store üblicherweise genau eine Bewertung. Dies macht es schwer Feedback zu klassifizieren das sowohl positive als auch negative Aspekte enthält.

Wünschenswert wäre also ein Maß, das man ähnlich leicht auswerten kann wie Sternebewertungen, und das idealerweise mehrfach innerhalb des Feedbacks vorhanden ist. So können die im Beitrag enthaltenen Themen jeweils unabhängig voneinander bewertet werden.

Ist ein solches Maß gefunden, kann man Entwickler unterstützen und hierbei vor allem Produktmanagern die Möglichkeit geben, fundierte Entscheidungen zur Qualitätssteigerung der eigenen Produkte zu fällen.

Emojis in Online-Texten

Wie wir gesehen haben funktioniert zur Auswertung von Feedback die Berücksichtigung von Bewertungssternen nur eingeschränkt. Betrachtet man textuelles Feedback näher, stellt man fest, dass es voller Emotionen ist. Ein Weg, wie Nutzer ihre Emotionen deutlich machen, ist die Verwendung von Emojis.

Die Historie von Emojis und ihrer Vorgänger zeigt, dass diese schon immer das Ziel hatten, Emotionen innerhalb von Texten auszudrücken. Schon im 19. Jahrhundert wurden in gedruckten Texten mit Hilfe von kleinen Druckzeichen Emotionen deutlich gemacht. Zu dieser Zeit nannte man diese Urahnen der Emojis noch Setzerscherze. Mit der Entwicklung von Schreibmaschinen und später Computern wurden diese Bemühungen auf die dort verfügbaren Zeichen eingeschränkt. So wurde mit Aufkommen des Internets den Emoticons zur Berühmtheit verholfen. Emoticons sind Kombinationen aus ASCII Zeichen, die Gefühle und Emotionen deutlich machen. In Asien entwickelten sich danach deutlich komplexere Zeichenkombinationen Namens Kaomojis, die auf asiatische Zeichensätze oder gar Unicode setzen. Durch die Verfügbarkeit von Internetanschlüssen in Privathaushalten und den immer mehr genutzten Instant Messaging Diensten wurden grafische Emoticons oder Smileys eingeführt, die im Wesentlichen aus einer Umwandlung der Emoticons in Bildchen bestanden, aber auch Neuschöpfungen enthielten.

In Japan designte schließlich Shigetaka Kurita 1999 für das aufkommende mobile Internet des Telekommonikationsanbieters NTT DoCoMo die ersten Emojis. Hier wuden zusätzliche Zeichen eingeführt, die zunächst als 12x12 Pixel große Grafiken vorlagen. Auch weitere Firmen setzen diese und ähnliche Zeichen ein. 2010 begann das Unicodekonsortium offiziell Emojis zu standardisieren veröffentlicht seitdem immer wieder neue Versionen heraus. Ihren weltweiten Durchbruch hatten die Emojis mit der Einführung der Emoji-Tastatur 2011 mit iOS 5 im. Andere Systeme zogen nach und so sind Emojis aus der modernen Kommunikation nicht mehr wegzudecken. Hunderte von Milliarden Emojis werden täglich verschickt. Man findet sie an Flughäfen zur Bewertung von Sicherheitskontrollen und Toiletten wieder sowie in einem eigens nach den Bildzeichen benannten Kinofilm. Der Wunsch Emotionen in Texten prägnant ausdrücken zu können ist in der breiten Gesellschaft angekommen.

Emojis helfen uns daher in Feedback heraufzufinden wie ein Nutzer zu bestimmen Aspekten eines Produktes steht. Da es inzwischen Tausende dieser Zeichen gibt, stellt sich die Frage, wie einheitlich Emojis verstanden werden. Dazu haben wir Emojis auf Emotionen untersucht.

Wie nehmen Menschen Emojis wahr?

Um Emojis aus Nutzerfeedback in einem Qualitätssicherungsansatz zu verwenden stellt sich natürlich zunächst die Frage: Wie nehmen Menschen Emojis wahr? Dazu ist es notwendig, ein Modell zu erarbeiten, um Emojis kategorisieren zu können sowie eine Liste von Emojis zu sammeln, welche berücksichtigt werden sollen. Zum Zweck der Einordnung von Emojis in unser Modell haben wir eine Umfrage durchgeführt.

Es existieren bereits unterschiedliche Klassifikationen von Emotionen, die sich sowohl in der Art wie sie Emotionen kategorisieren, beispielsweise über biologische Vorgänge oder Gesichtszüge, als auch in der Granularität unterscheiden. Für die Nutzung von Emotionen im Rahmen der Qualitätssicherung haben wir uns für ein Modell entschieden, dass ähnlich zu den Basis Emotionen nach Paul Ekman ist [1] und wie bei Feldmann [2] Charakteristiken der Sentimentanalyse enthält. Unser Emotionsmodell sieht wie folgt aus (siehe Abbildung 1): Es gibt zwei Ebenen. Zuerst findet wie bei Feldmann eine Einteilung in Sentiments statt. Es wird zwischen „positiv“, „neutral“ und „negativ“ unterschieden. Die zweite Ebene ist die Emotionsebene und fungiert als Unterkategorie des Sentiments. Sie enthält im Wesentlichen Basisemotionen, welche einfach zu verstehen und zu unterscheiden sind.

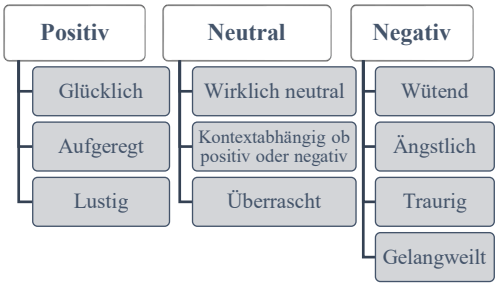


Abbildung 5 Emotionsmodell

Die Stärke des verwendeten Modells ist die Verwendung von Basisemotionen sowie die Verbindung von Sentiments - also ob etwas positiv, negativ oder neutral ist - und Emotionen. Anschließend haben wir Emojis und Emoticons gesammelt. Da es davon Tausende gibt, galt es, die Menge sinnvoll zu reduzieren. Wir haben alle Emojis, die keine Emotion übermitteln, wie Gegenstände, Buchstaben, Wortersetzungen oder Flaggen gestrichen. Die Ausgangsmenge reduzierten wir dadurch auf 612. Da es für viele Emojis Synonyme sowie alternative Darstellungen gibt haben wir anschließend die verbleibenden Emojis in Gruppen eingeordnet. Wir haben Emojis mit gleicher Bedeutung, wie den „Konfettiball“ 🎊 und den „Party Popper“ 🎉 zusammengefasst. Außerdem haben wir Emojis, die sich nur in ihrer Haut- ‚Haarfarbe oder dem Geschlecht unterschieden gruppiert. Jede Gruppe erhielt einen Repräsentanten, dessen Kategorisierung auf die restlichen Emojis der Gruppe übertragen werden kann. Die Einteilung wurde unabhängig von drei Mitarbeitern geprüft. Durch die Gruppierung der Emojis kamen wir auf 99 Gruppen mit jeweils einem Repräsentanten.

Um eine Zuordnung zwischen dem Emotionsmodell und den gesammelten Emojis zu erhalten haben wir eine Umfrage durchgeführt. Die Umfrage erlaubt uns ein allgemeines Bild der Wahrnehmung von Emojis zu erlangen. Hierfür sollten Teilnehmer Emojis nach Sentiment und nach

Emotion gemäß dem Emotionsmodell einteilen. Als Teilnehmer der Studie haben wir versucht einen Gesellschaftlichen Querschnitt abzubilden, das heißt sowohl eine Balance in den Geschlechtern herzustellen als auch unterschiedliche Altersgruppen vertreten zu haben.

Insgesamt haben wir 107 Antworten ausgewertet und darauf unsere Einteilung von Emojis in Sentiments und Emotionen gemacht. Von diesen 107 Teilnehmer waren, 53 weiblich und 54 männlich. Der Altersbereich lag zwischen 17 und 75 Jahren mit einem Mittelwert von 31,41. Betrachtet man die Teilnehmer in Bezug auf die so genannten digitalen Generationen gehörten mit 69 Teilnehmer die meisten Teilnehmer zur Generation der Generation Y, gefolgt von der Generation Z mit 16 Teilnehmern. Aus Generation X hatten wir 11 Teilnehmer und 10 aus der Generation der Babyboomer. Zusätzlich hatten wir eine Person der Kriegsgeneration. Damit haben wir Personen in der Studie gehabt die sowohl als Digital Natives aufgewachsen sind, bis hin zu Personen, die beim Aufkommen der Mobilgeräte schon älter waren.

Das Kernergebnis der Umfrage ist: Menschen nehmen Emojis sehr ähnlich wahr. Das lässt sich für Sentiment und Emotion aussagen. Betrachtet man nur die Sentiments, so konnten wir für 591 Emojis feststellen, dass mindestens 50% der Teilnehmer sie übereinstimmend eingruppierten. Dies bedeutet, dass wir nur 21 Emojis nicht einordnen konnten. Im Falle der Emotion hatten wir für 512 Emojis eine Übereinstimmung von mindestens 50%. Damit ließen sich für uns nur 79 Emojis nicht auf eine Emotion festlegen. Dieses Ergebnis bildete für uns die Grundlage, Werkzeuge zu erstellen, mit der wir die Stimmung der Nutzer gegenüber einem Produkt, einem Feature und Änderungen herausfinden. Das Ergebnis in Form der Einteilungen in Sentiments und Emotionen der Umfrage für alle Emojis ist in Tabelle 1 dargestellt.

Betrachtet man die Teilnehmer nach der von ihnen angegebene Vertrautheit mit Emojis so stellt man fest, dass zwischen „vertrauten“ und „mittel vertrauten“ kein signifikanter Unterschied in der Einordnung der Emojis besteht. Personen, die sich als „nicht vertraut“ mit Emojis bezeichneten, wichen allerdings stärker ab, besonders bei den Emotionen. Damit konnten innerhalb dieser Gruppe viel weniger Emojis klassifiziert werden. Insgesamt zeigt sich, dass je vertrauter die Teilnehmer mit Emojis waren, desto ähnlicher nahmen sie diese wahr.

Ähnliches zeigt sich auch bei den Generationen. Je jünger die Teilnehmer waren, desto gleichmäßiger wurden Emojis wahrgenommen. Gleichzeitig ist die Menge der insgesamt klassifizierbaren Emoji-Repräsentanten nach Emotion bei Generation Z, Y und X mit 81 bzw. 82 sehr ähnlich. Die Generation der Babyboomer fällt hier deutlich ab, allerdings fanden sich dort auch verstärkt Teilnehmer, die angaben, mit Emojis nicht vertraut zu sein.

Um sicherzustellen, dass Emojis wie die Sternebewertung ein gutes Mittel sind Feedback zu klassifizieren, haben wir für 178.848 Feedbackeinträge zu diversen Apps Sternebewertung und Sentiment der Emojis im Feedbackeintrag korreliert. Hieraus ergab sich eine sehr starke Korrelation beider Werte. Damit sind Emojis als Bewertungsindikator vergleichbar zur Sternebewertung und können somit ergänzend oder auch alternativ genutzt werden.

Sentiment	Emotion	Repräsentanten	#Emojis	Summe
Positiv	Glücklich	O:-) :-* :-) @};- ^0^ :-) }:-) :D <3	166	243
	Aufgeregt		12	
	Lustig	:-P	34	
	Unentschieden	;-)	31	
Neutral	Wirklich neutral	OK	3	131
	Kontextabhängig		72	
	Überrascht	:-O	24	
	Unentschieden	|-O @.@	32	
Negativ	Wütend		33	217
	Ängstlich	D:	13	
	Traurig	</3 :'-(:-/ :-(	145	
	Gelangweilt	-.-	10	
	Unentschieden	X_X :-X >.< >_<	16	
Unentschieden		*_* :$	21	21

Tabelle 1: Anzahl der klassifizierten Emojis nach Sentiment und Emotion

Qualitätssicherung mit Emotion

Basierend auf unseren Erkenntnissen zu Emojis haben wir einen Analyseprozess für Nutzerfeedback definiert (Siehe Abbildung 2). Zentrales Ziel ist es, die Entwicklungsorganisation möglichst wenig durch zusätzliche Aufgaben zu belasten, jedoch die Qualität der Software zu sichern. Kernidee ist, dass Nutzerfeedback parallel zur Entwicklung neuer Versionen der Produkte gesammelt und ausgewertet wird. Hierbei verwenden wir eine Mischung aus vollautomatisierten, teilautomatisierten und manuellen Schritten.

Kollektoren sammeln Feedback aus zahlreichen Datenquellen wie Stores, sozialen Medien, Foren etc. ein. In der quantitativen Auswertung wird bestimmt, welches Feedback aus welcher Quelle zu welchen Zeitpunkten vorkommt und ob eventuell im Feedback selbst schon Bewertungsindikatoren, wie Bewertungssterne oder Zustimmungen anderer Nutzer, vorhanden sind.

Im nächsten Schritt identifiziert der Prozess automatisch Themen innerhalb des Feedbacks. Themen können beispielsweise einzelne Features, Fehler, Anwendungsfälle oder Screens der Anwendung sein. Ziel dieses Schrittes ist es ein besseres Verständnis zu erlangen auf welche Aspekte der App sich die Nutzer fokussieren.

Der nächste Schritt ist die Erkennung von Emotionen mittels Emojis innerhalb der Feedback-Einträge und innerhalb der zuvor erstellten Themen. Dazu nutzen wir die Ergebnisse aus den in Abschnitt 3 beschriebenen Vorarbeiten. Bis zu diesem Zeitpunkt läuft der Prozess vollständig automatisiert ab.

Danach wird das Feedback mit Blick auf frühere Produktversionen ausgewertet. Je nach Thema werden geeignete Vergleichszeiträume gewählt. Damit werden Stimmungstrends,–Wendungen sowie Veränderungen der Menge innerhalb des Feedbacks erkannt.

Basierend auf diesen Analyseschritten leitet unser Ansatz Fokusbereiche für den Produktmanager ab. Ein Fokusbereich ist eine Teilmenge des Feedbacks, welches Aufmerksamkeit bedarf. Fokusbereiche an sich sind wertneutral zu sehen, da sowohl positive als auch negative Wendungen im Feedback aufgezeigt werden.

Die anschließende Ableitung von Handlungsempfehlungen ist auf Probleme im Produkt ausgerichtet. Ziel ist es, Empfehlungen zur Produktverbesserung abzuleiten und diese zu priorisieren. Für den Fall, dass eine Handlungsempfehlung die Beseitigung eines Fehlers im Produkt ist, wird der Fehler klassifiziert und festgehalten. Damit entsteht auf lange Sicht eine Sammlung von Fehlern und Fehlermustern zum Produkt. Ist die Handlungsempfehlung eine Abänderung des Produktes, werden neue Anforderungen für die Entwickler abgeleitet.

Basierend auf den Handlungsempfehlungen erfolgt ein Hinzufügen zum Release-Plan. Daraufhin findet eine Umsetzung der Empfehlungen statt. Bei der folgenden Qualitätssicherung des Produktes wird zum einen traditionelle Qualitätssicherung betrieben, aber auch ganz bewusst auf die abgeleiteten Handlungsempfehlungen geachtet. Zusätzlich werden die gesammelten Fehlermuster geprüft, um so das Wiederauftreten von alten Fehlern zu vermeiden. Wenn die Version den gewünschten Qualitätsstandards entspricht, wird sie veröffentlicht.

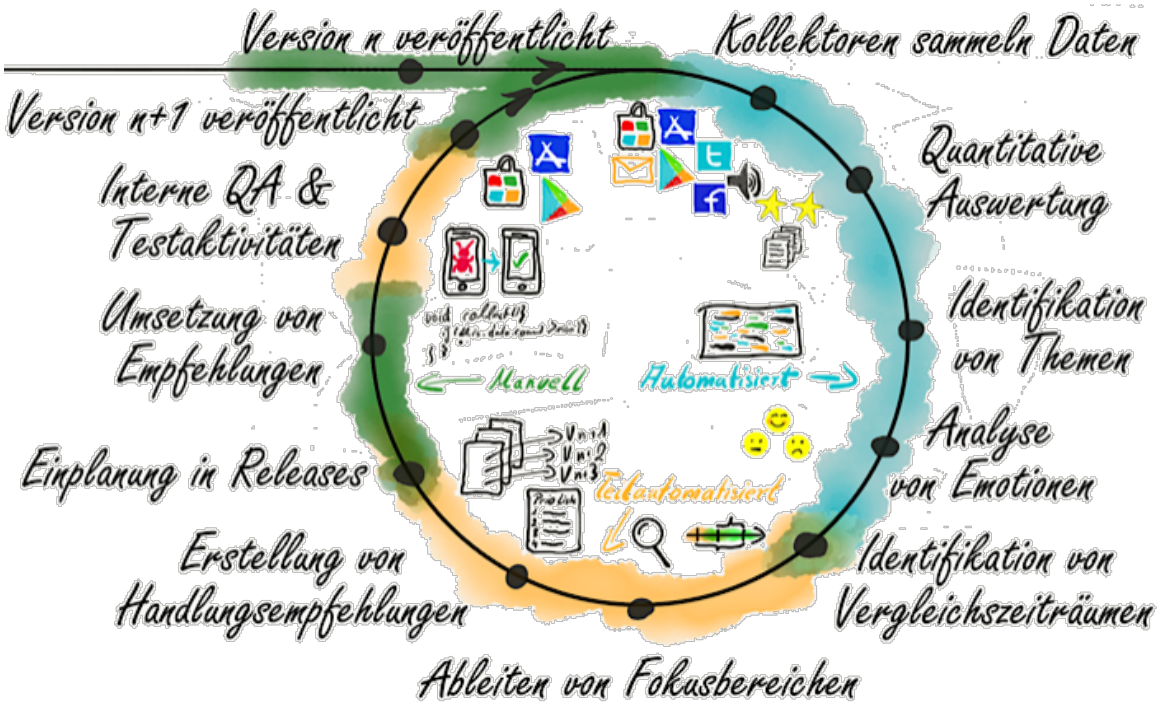


Abbildung 6: Prozess zur Qualitätsverbesserung durch Trenderkennung in Nutzerfeedback

Beispiele für Erkenntnisse aus Nutzerfeedback

Erste Erfahrungen mit unserem Prozess haben wir bei der Untersuchung von Apps aus dem App Store gesammelt. Es gibt zahlreiche Apps, bei denen plötzliche Bewertungseinbrüche existieren bzw. es spontane Anstiege in der Menge an Feedback gibt. Dies gilt auch Themen, die unser Ansatz erkennt.

Die App Snapchat beispielsweise hatte am 27.7.2017 eine hohe Menge an Feedback. Ursache war eine temporäre Störung des Dienstes. Einen ähnlichen Bewertungsverlauf konnten wir Ende Juni 2017 beobachten, als die App ein Update erhielt, das es ermöglichte, seine Freunde auf einer Kartenansicht darzustellen. Die Nutzer schienen nicht zu wissen, wie die Funktion zu benutzen ist und es gab einen Einbruch der Durchschnittsbewerbung sowie verstärkt negative Emotionen.

Des Weiteren untersuchten wir Instagram, bei dessen Bewertungsverlauf es zwei deutliche Einbrüche Anfang und Ende Juli 2017 gab, die gleichzeitig mit einer erhöhten Menge an Feedback und vielen negativen Emotionen einhergingen. Bei der ersten Situation handelte es sich um ein kurzzeitiges Problem des Dienstes, bei dem Benutzerkonten irrtümlich als gelöscht markiert wurden und so eine Nutzung des Dienstes nicht mehr möglich war. Die zweite Situation war eine etwa zwölf Stunden andauernde Downtime für einen Teil der Nutzer.

Uber zeigte einen deutlichen Anstieg von Feedback im Januar 2017. Die Menge von Feedback erhöhte sich von 1.903 Einträgen im Dezember 2016 auf 3.984 Einträge im Januar 2017 und fiel dann wieder auf 1.840 Einträge im folgenden Monat. Die Ergebnisse im Dashboard zeigen in diesem Monat eine häufige Verwendung des Themas „löschen“. Tatsächlich waren viele Nutzer sehr verärgert über die Zusammenarbeit zwischen Ubers CEO und einem Politiker, weshalb Uber viele Nutzer verlor. Im Februar trat CEO aufgrund dieser Ereignisse zurück [4]. Wir haben auch beobachtet, dass Uber oft mit seinem Mitbewerber Lyft verglichen wird. Als Nutzer sich für die Löschung von Uber entschieden, gaben viele an, sie seien zu Lyft gewechselt. Feedback über den Wechsel zu Lyft oder den Vergleich zwischen Uber und Lyft hat eine Durchschnittsbewertung von 1,28 Sternen und enthält meist negative Emotionen. Dieses Beispiel zeigt, dass Nutzer Feedback Einblicke in die Beziehung zwischen Nutzern und dem Ruf und der Ausrichtung einer Firma gibt sowie in konkurrierende Produkte.

Pinterest erlebte einen Einbruch im Bewertungsverlauf von 3,7 auf 3,0, als der „like“-Button aus der App entfernt wurde [5]. Im Juni 2017 gab es 722 Feedback Einträge, die das Wort „Button“ enthielten und viele Nutzer beschwerten sich darin über den fehlenden Button. Die Emotionen in diesem Feedback waren meist negativ, wobei „traurig“ die häufigste Emotion war. Eine mögliche Handlungsempfehlung für dieses Thema wäre das Zurückbringen des „like“-Buttons.

Ein weiteres Problem haben wir für Tinder im März 2018 entdeckt, als die durchschnittliche Sterne-Bewertung um 0,8 auf 2,5 sank. In diesem Monat war die Menge an Feedback mehr als dreimal so hoch wie im vorherigen. Unsere Analyse zeigt zu dieser Zeit eine häufige Verwendung der Themen „Profil“ und „fake“ und der Großteil des Feedbacks, das eines dieser Themen enthält stammt vom März 2018. Darüber hinaus war „fake“ in vielen Einträgen mit „Profil“ kombiniert. In der Tat gab es eine große Menge an Feedback, in dem Nutzer sich über zu viele Fake Profile auf Tinder beschweren. Die entsprechende Handlungsempfehlung wäre die Implementierung eines Features zur Reduktion von solchen Profilen oder eine Echtheitsprüfung. Alternativ könnte ein Feature entwickelt werden, das „verifizierte Profile“ zeigt. In den folgenden Monaten nahm die Menge an Feedback über das Thema „fake“ deutlich ab.

Die aufgezeigten Beispiele machen deutlich, dass Probleme nicht zwangsläufig rein in der Mobilanwendung zu verorten sind. Apps sind immer mehr Teil von größeren Systemen, die eine sofortige Reaktion auf Probleme erforderlich machen.

Fazit

Um die Produktakzeptanz von mobilen Apps zu erhöhen haben wir die Notwendigkeit erkannt, leichtgewichtige Textanalysemethoden zu entwickeln. Dazu haben wir einen Qualitätssicherungsprozess definiert. Ein Kernbestandteil des Prozesses bildet die Analyse von Emojis innerhalb des Feedbacks und die schnelle Erkennung von Trends, um auf dieser Basis Handlungsempfehlungen ableiten zu können.

Damit dies möglich ist, haben wir ein Emotionsmodell erstellt und eine Umfrage zu Emojis durchgeführt. Kernergebnis der Umfrage ist, dass Menschen eine sehr ähnliche Wahrnehmung von Emojis haben. Folglich können große Mengen an Texten auf Grund der verwendeten Emojis schnell analysiert werden. Dies ermöglicht eine umfangreichere Kategorisierung von Texten, verglichen mit der regulären 5-Sternebewertung, die beispielsweise im Apple App Store üblich ist.

Die Ergebnisse flossen in unseren Qualitätssicherungsprozess zur Trenderkennung in textuellem Feedback ein. Zentrale Anforderung an den Prozess sowie an die technische Umsetzung ist eine einfache Bedienbarkeit und die Einsatzfähigkeit für hochiterative Produktentwicklung.

Die Analyse von Emojis erlaubt uns dort Feedback zu analysieren, wo keine bewerteten Metriken, wie Bewertungsterne, mit dem Feedback mitgeliefert werden. Dies trifft insbesondere auf Feedback im Supportbereich und soziale Medien zu.

Der Vergleich von Sternebewertung und des emotionalen Sentiments zeigt eine starke Korrelation der beiden Metriken auf. Damit können wir eine stärker automatisierte Auswertung von Nutzerfeedback in heterogenen Datenquellen realisieren. Zahlreiche Beispiele zeigen auf, dass unser Prozess bei realen Produkten und deren Feedback wirken kann.

Einer der nächsten Schritte ist Feedback über verschiedene Datenquellentypen wie App Stores, soziale Medien und Kundenforen hinweg gezielt zu vergleichen. Da die Menge an Emojis stetig weiterwächst, haben wir eine Folgeumfrage in Arbeit. Im Wesentlichen besteht diese aus der Klassifizierung der im Jahr 2017 neu eingeführten Emojis, sowie detaillierten Fragen zu den schwer einzuordnenden Emojis.

Bibliografie

[1] F. Elberzhager and K. Holl, "Towards Automated Capturing and Processing of User Feedback for Optimizing Mobile Apps," in *Procedia Computer Science*, Leuven, 2017.

[2] P. Ekman and W. V. Friesen, "Constants across cultures in the face and emotion," *J. Pers. Soc. Psychol.,* vol. 17, no. 2, pp. 124-129, 1971.

[3] R. Feldman, "Techniques and applications for sentiment analysis," *Communications of the ACM,* vol. 56, no. 4, 2013.

[4] J. C. Wong, "Uber CEO steps down from Trump advisory council after users boycott," 3 2 2017. [Online]. Available: https://www.theguardian.com/technology/2017/feb/02/travis-kalanick-delete-uber-leaves-trump-council. [Accessed 02 07 2017].

[5] K. O., "Goodbye, Like button," 20 4 2017. [Online]. Available: https://newsroom.pinterest.com/en/post/goodbye-like-button. [Accessed 5 07 2017].

Autor

Simon André Scherr
Engineer User Experience and Requirements Engineering
Fraunhofer IESE
simon.scherr@iese.fraunhofer.de

Tricentis
Continuous Testing
Platform

Automated Continuous Testing

Distributed Load Testing

Agile Dev Testing

TRICENTIS Tosca

Resilient regression testing across any architecture or application stack at the speed of change.

TRICENTIS flood

Cloud-based performance labs at the fingertips of every developer and tester for on-demand load testing.

TRICENTIS qTest

Scalable, in-sprint test management for open source test automation, exploratory testing and BDD.

Software Testing **Reinvented** for Agile and DevOps

TRICENTIS

Discover Quality Requirements

… with the Mini-QAW, a short and fun workshop for Agile teams

Good quality requirements help software engineers make the right architectural design decisions. Unfortunately, collecting requirements is not always easy. The Quality Attribute Workshop (QAW) helps teams effectively gather quality requirements but it can be costly and cumbersome to organize. The mini-QAW is a short (a few hours to a full day) workshop designed for inexperienced facilitators and is a great fit for Agile teams. Due to the shortened agenda and visual tools that we use in the workshop, we find that we can discover a fair number of quality requirements in half a day or less.

There could be millions of ways to design a software system from the same set of functional requirements. Features are binary. The software either does something correctly or it doesn't. Software architects spend most of their time deciding how the software will provide that functionality. To helps software architects in this task, they consider quality requirements in addition to functionality. Quality requirements, sometimes also called quality attributes or non-functional requirements, characterize the externally visible properties of the system in the context of some bit of functionality. Learn the quality requirements for your system and you'll stand a chance of designing an architecture that truly solves your stakeholders' needs (Keeling 2017).

The Quality Attribute Workshop (QAW), introduced by the Software Engineering Institute (SEI) in 2003, describes a repeatable method for gathering quality requirements (Barbacci et al. 2003). During the two-day QAW, trained facilitators help a development team collaborate with stakeholders to gather detailed quality requirements as a precursor to software architecture design. The workshop produces a prioritized list of quality attribute scenarios, which support writing specific and measurable quality requirements.

The QAW is a great method but requires an investment few agile teams are willing to make. The QAW is also only as good as the facilitator who leads the workshop. Quality requirements are so essential to the success of a software system that forgoing structured methods for gathering them does not seem prudent. We created the mini-QAW to overcome deficiencies in the traditional QAW and make a workshop focused on quality requirements more palatable for agile teams.

Creating the Mini-QAW

The mini-QAW has two goals. First, reduce the time and effort needed to gather good quality requirements. Second, make it easier for novice facilitators to host a successful workshop. These goals are achieved by reducing the scope of the traditional workshop and by introducing a quality attribute taxonomy to guide the workshop. For a detailed description of the changes read our article in RE Magazine (de Gooijer et al. 2018).

Mini-QAW Agenda

1. Introduce the mini-QAW and quality attributes (10 minutes)
2. Introduce quality attributes and the quality attributes taxonomy (15 minutes)
3. Facilitate the stakeholder empathy map activity (15 minutes) – new in mini-QAW
4. Facilitate scenario brainstorming (30 minutes to 2 hours)
   - "Walk the Quality Attribute Web" activity, or
   - Structured brainstorming
5. Prioritize raw scenarios using dot voting (5 minutes)
6. Refine scenarios (as time permits)
   - While time remains, remainder is homework

Schedule a follow-up meeting to review the refined quality attribute scenarios with stakeholders. This meeting usually lasts about an hour.

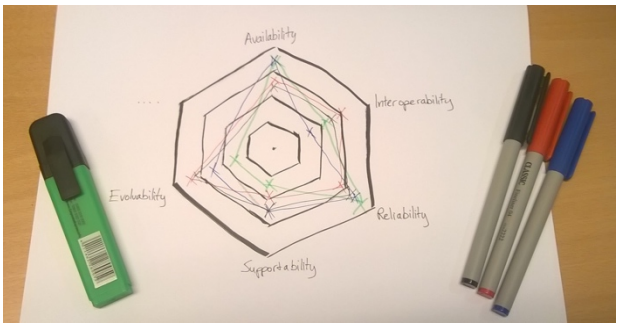


Abbildung 7 - Sample empty quality attribute web

Quality Attribute Web - Mini-QAW Visualization

Beyond changes to the agenda, we also introduce the quality attribute web as a visual tool for the workshop. The quality attribute web (Abbildung 1) is a radar chart

created from a taxonomy of quality requirements likely to be important for the type of system under development. The axes of the web are used to score the importance of a quality attribute and as a visualization for capturing requirements during the workshop. Marks closest to the center indicate low priority or interest, and marks towards the outer border indicate a higher score. This way we can visualize differences in stakeholder priorities, or how different systems require different qualities. The visualization of the web also enables us to use standard workshop techniques such as dot-voting for prioritization.

The Activities in the Mini-QAW

There are five primary steps in the Mini-QAW. Each step is driven by a collaborative, visual activity designed to engage stakeholders and encourage discussion about a system's quality requirements.

Stakeholder Empathy Map

The Stakeholder Empathy Map is a brainstorming and visualization exercise that gives absent and underrepresented stakeholders a voice during the workshop. During the exercise participants empathize with absent stakeholders and specify quality concerns from the absent stakeholder's perspective. As a result, participants experience firsthand the conflicting priorities and requirements that may exist in the system's requirements. This awareness makes it easier to deal with tension among the participants on conflicting priorities and requirements later in the workshop.

To build a stakeholder empathy map, participants start by creating a list of stakeholders for the system. Next, each participant picks a stakeholder to empathize with and reflects on this stakeholder's requirements. Finally, the participant records a best-effort estimate of the stakeholder's quality priorities on the quality attribute web.

All of this happens before the participants get the space to share their own requirements. The intention is to avoid projection of the participants wishes or needs on the absent stakeholder represented. One key benefit of the exercise is that it is possible to run a meaningful workshop with a smaller set of stakeholders. This activity also makes it feasible to run a workshop for the same system in multiple locations with subsets of stakeholders, removing travel overhead and potentially getting access to stakeholders that otherwise could not or would not participate.

Steps to create a stakeholder empathy map

7. Brainstorm: who are the stakeholders in the software?
8. All participants pick a stakeholder who is not in the workshop
9. Estimate quality priorities for the chosen (absent) stakeholder
10. Record estimates on flip charts or worksheets
11. (optional) the facilitator guides the participants to reflect on the results and areas of conflict

Scenario Brainstorming

The scenario brainstorming exercise in the mini-QAW results in raw quality attribute scenarios, just like the similar exercise in the traditional QAW workshop. The goal is to discover what the participants need and expect from the system. During scenario brainstorming participants should not worry about creating a formal quality attribute scenario. Facilitators should invite participants to think about events (stimuli), behavior (responses), and changes to the environment in which the system operates.

Raw quality attribute scenarios informally describe stakeholder concerns. One way to formulate these is through concrete examples of a demonstration of quality. For example, we need the system to work during business hours (an availability requirement); or we expect a peak load of 150 requests per second (a performance requirement). Ideas for features and functional requirements usually come-up during the workshop. Don't suppress these, instead put them in a dedicated area outside of the quality attribute web for later review.

The base form for this exercise is to walk through each axis on the quality attribute web. Pause on each quality; consider whether it is relevant; and if it is, spend about five to seven minutes brainstorming and recording raw scenarios. In some groups, active listening questions are enough to keep the scenario ideas flowing, others may benefit from structured help. Since the web is built from a quality attribute taxonomy, a taxonomy-based questionnaire can be used to help participants adequately explore the problem space. Such questionnaires can also help novice facilitators consistently deliver good workshops. The SEI has example questionnaires available for the traditional QAW.

Steps for scenario brainstorming

12. Start with a quality on the web, ask "Is this quality attribute relevant to our system?"
13. If Yes, spend 5 minutes brainstorming scenarios / concerns on that quality.
14. Write raw scenarios on sticky notes and put on the quality attribute web.
15. After 5 minutes, move to the next quality.

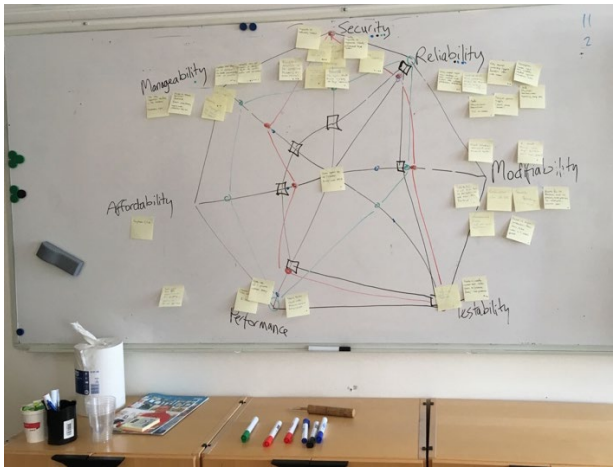


Abbildung 8 - Visual cues from the quality attribute web

Raw Scenario Prioritization

Prioritizing raw scenarios helps the software architect focus on what is important to the stakeholders so the architect can make the right trade-offs when designing the architecture. There is also no use in refining scenarios that stakeholders do not find useful or important. Finally, prioritization may expose areas of disagreement requiring further exploration during the workshop.

The mini-QAW uses visual dot voting to establish priorities. Both the quality attributes (web axes) and raw quality scenarios are prioritized. Based on the number of scenarios, participants are assigned a set number of dots. Participants may distribute these dot allocation as they wish: all dots on one scenario; one dot on each scenario that is important to them; or anything in between. Voting for quality attributes works in a similar way.

After voting, the quality attribute web contains a lot of visual information. Facilitators can use this information to identify inconsistencies and gaps in the requirements analysis. Abbildung 2 shows an example from one workshop. In this example, the result of the empathy exercise is shown in the colored lines running through the web. We can also see that manageability, security, reliability, and modifiability received most of the scenarios, which aligns with what most stakeholders cared about. In this example one stakeholder strongly valued affordability but few raw scenarios were identified. This may be an opportunity for follow-ups and further analysis. Additionally, the facilitator should look for cluttered areas that might benefit from a quick consolidation round.

Step for dot voting

16. Each stakeholder gets:
    - n / 3 + 1 dots for raw quality scenarios, where n = # scenarios
    - 2 votes to choose “top quality attribute”
17. Participants place their assigned dots, taking turns or at the same time.

Scenario Refinement

The scenario refinement step is very important to get from raw scenarios to testable requirements. The traditional QAW rightfully spends significant time in the agenda on this activity. Since the mini-QAW is a time-constrained workshop, we only continue this activity until time runs out. This usually leaves several high or medium priority raw scenarios untouched. The architect, facilitator, or requirement engineer will refine the remaining scenarios after the workshop as homework.

In a typical mini-QAW, the facilitator guides the team in refining scenarios. Start with the top voted scenarios (those with most dots) and use a visual worksheet (Abbildung 3) to create specific, measurable scenarios. Each scenario is specific to a quality attribute and consists of six parts (Bass et al. 2003). Note any assumptions made regarding response measures, scenario context, and anything else that may help understand and test the scenario later. The facilitator or architect should ask questions to clarify requirements. Usually also in this activity, like in the brainstorming activity, functional requirements disguised as quality attributes show up: be aware, add them to a list, and review later.

Steps for scenario refinement

18. Start with high priority scenario
19. Fill out the visual worksheet, identifying/specifying each of the components of a quality attribute scenario
20. Complete or put a time limit on each scenario, continue until workshop time runs out.

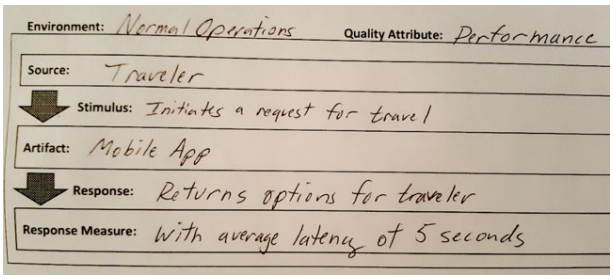


Abbildung 9 - quality attribute scenario worksheet example

Scenario Review

The workshop facilitator or architect formalizes the scenarios after the workshop. During this homework, the scenarios may change slightly and no longer reflect exactly what the stakeholders had in mind. Therefore, it is important to review the scenarios with the stakeholders.

After the final tweaks to the scenarios, present the refined scenarios to your stakeholders during a review to gather their feedback. We recommend giving stakeholders the opportunity to read the material before you meet them for the review. During the review, you may discover previously unstated requirements, record these. You could find out the scenario is misunderstood easily, try to

explain how the scenario helps the stakeholder do a job or achieve a goal. The refinements to scenarios may change their scope: one becomes multiple or vice versa. Therefore, it is useful to verify the priority of such scenarios.

Steps for scenario review

21. Present top and architecturally significant scenarios to stakeholders.
22. Gather feedback & check that scope matches expectations.
23. Trawl for new requirements (!).

Experience so Far

The mini-QAW method has been used successfully by several groups throughout the world. It is finding its place as a standard tool among many software architects. Despite the shortened format, we find that we can easily discover 30 quality requirements through 50+ rough scenarios in less than half a day. The mini-QAW format will not give results of the same depth and breadth as a traditional QAW, but is a great tool for smaller, low-risk or iterative projects.

Hints and Tips for Facilitators

We want more software projects to deliver long-term value which requires a sound architecture. Without a proper understanding of quality requirements, it is very difficult to build such a foundation. Therefore, we want you to be successful with the mini-QAW and wrap up with some hands-on hints and tips.

- ☑ Prepare, prepare, prepare. You'll feel more confident and get more out the workshop. Know what you will say, and practice the logistics of the workshop.
- ☑ Match your tools to the stakeholders. Some people dislike sticky notes, be creative and find an alternative. For remote workshops using video conferencing, you will want to use online collaborative tools. Skype (for Business) works, for example, you can collaboratively annotate PowerPoint slides.
- ☑ Choose the right taxonomy for your system: your experience is needed to pick quality attributes that a relevant for the quality attribute web. Don't use too many, about 5 or 6 properties is a good number. Remember you can brainstorm just-in-time if needed or leave some axes on the web open.
- ☑ Give each stakeholder a voice: ensure everyone is heard. This may mean you must split your workshops to ensure everyone feels comfortable to speak. For example, in a strongly hierarchical culture subordinates may not want to speak up with their superiors in the room.
- ☑ Speed-up your ATAM architecture evaluation using the mini-QAW when there is no proper record of quality requirements.

Conclusion

The mini-QAW is an additional tool in one's toolbox, but of course no silver bullet for quality requirements engineering. We have neither conducted a formal review of effectiveness, nor done a direct comparison to effectiveness of the traditional QAW. Yet, it has been useful to us and increasingly others as a quick way to document key quality requirements.

We shortened the traditional QAW workshop by tailoring some original activities and deferring part of the work as homework. The core mini-QAW activities include scenario brainstorming using a "quality attribute web", creating stakeholder empathy maps, visual voting, and refinement using a visual quality attribute scenario template. Refreshed activities and a tuned agenda create fast, effective, and fun workshops.

If you want to get started, you will find resources to run the workshop freely available on our webpage: http://bit.ly/mini-qaw. There you will also find the material to and videos of our talks. The method is still young and we very much appreciate hearing about your feedback and experiences. Quality attribute taxonomies and scenario discovery questionnaires help us all, so let's collect them together.

Bibliography

Barbacci, M., Ellison, R., Lattanze, A., Stafford, J., Weinstock, C., & Wood, W. (2003). Quality Attribute Workshops (QAWs). Architecture Tradeoff Analysis Initiative.

Bass, L.; Clements, P. and Kazman, R. Software Architecture in Practice, 2nd Edition. Reading MA: Addison Wesley, 2003

de Gooijer, T., Keeling, M., Chaparro, W. (2018). Discover Quality Requirements with the Mini-QAW. RE Magazine, IREB GmbH.

Keeling, M. (2017) Design It! From Programmer to Software Architect. Raleigh, NC: The Pragmatic Bookshelf

Authors

Thijmen de Gooijer

IT architect at Kommuninvest working as technical lead for its digitalization initiative and growing CI/CD and Agile practice in the company. Thijmen's work on the mini-QAW started in his role as researcher at ABB, where he worked on software architectures with colleagues around the globe. He graduated cum-laude in software engineering from VU University in the Netherlands and Malardalen University in Sweden.

thijmen@acm.org

Michael Keeling

Michael Keeling is a software engineer at LendingHome and author of the book Design It! From Programmer to Software Architect. Michael has a Master of Software Engineering from Carnegie Mellon University and a Bachelor of Science in Computer Science from the College of William and Mary.

Will Chaparro

Will Chaparro is a software development manager in IBM's Watson Group. He spent over 5 years designing and building complex enterprise search solutions as a managing consultant. Prior to IBM he spent 11 years as a software engineer. He has a BS in Computer Science from the University of Pittsburgh.

How Software Engineers Read your Quality Processes

Or: How to write better Software Quality ProcessesQuality Processes are necessary in today's industrial environments to ensure delivery of high quality products in an efficient way. However, writing and deploying such processes is a non-trivial activity.

In this article, we would like to present some of our learnings at ams AG, which we collected at defining and implementing our software development processes. We hope to provide a very practical guide for quality personnel that are going to create a new engineering process.

Why do we need processes?

Engineering processes define a workflow for engineers. In that sense, the opposite of having processes (having no process) could be understood as chaos. This might sound a bit overstressed and engineers might disagree and say that the opposite of having processes is certainly flexibility. However, as a quality manager I have not seen many projects taking advantage of such flexibility on long term. A good process, in turn, helps teams to meet project goals in a reliable and efficient way.

A poor process, however, will cause a lot of frustration and inefficient work items that are perceived as overhead. In the next subsections, we are going to discuss some characteristics of poor processes and we will give some hints on how to avoid them.

The three …ings

We can summarize the main objectives of good work on process definition and implementation into three words.

- Wording
- Tooling
- Training

We will discuss the detailed content in the subsections below. We would like to emphasize their importance, as the words sound pretty simple and obvious.

Wording: One might wonder how relevant the correct wording of a process is. One might argue that the substance of a process is much more important than its words. Some years ago, I would have agreed. Today, I claim that a process is nothing more than words. A process is meant to be taken literally and engineers will do so. Motivated engineers will get frustrated on poor wording because they do not feel supported by the process. Engineers that are not motivated to follow a process will use poor wording as excuses that the process is inefficient - maybe even impossible to follow.

Tooling: Engineers love tools. In many cases, engineers like introducing a tool for a certain activity much more than actually doing this activity. A process should certainly be supported by proper tooling. Nevertheless, the tools need to be selected carefully. The most efficient tools I have seen are checklists and templates. Tool vendors, however, are trying to sell highly sophisticated tools (and most of them are legitimate), but their introduction comes with an expensive learning curve. This initial effort often stands in the way of a successful introduction of a new process and should thus be done after the role-out of a new process.

Training: Last but not least, training should be mentioned here. No process is that simple, that it can be understood and used without trainings and thorough communication. The wording of the processes can be close to perfect but it will still cause questions that need to be answered. One should be prepared for a lot of effort to discuss the process, its interpretations and potential realizations. A process that is not questioned and challenged by affected engineers is not a perfect process but an ignored one.

Documents are for the sake of quality only and are thus pure overhead.

In our first version of our software development process we had many phrases like "A Test Specification shall be documented at milestone x." or "The Risk Sheet shall be filled out at project start." We discussed the importance of such deliverables with the engineering managers. There was no doubt that these documents are essential in a development project. However, the feedback we got

from the engineers was that the quality overhead of our process was far too high. The engineers have understood the mentioned sentences as pure documentation exercises that have to be done to fulfill the process.

It was certainly not our intention when we wrote the process, but by simply using the word "documenting", an engineering activity became suddenly a quality activity without a lot of recognized benefit for a project. After all, we have been told repeatedly that we are not selling documents. The usage of the word "documenting" seemed to supersede the real purpose of an activity.

We tried to correct these misunderstandings with communication and trainings – but finally we reworded most of such sentences to sentences like "Test Cases shall be specified." or "Risk assessment and mitigation shall be executed.". This puts more focus on the engineering activities instead of the documents.

Legitimated copy and paste of project data

Bigger companies typically have a landscape full of different, but overlapping processes. In ams AG, for example, we have a hardware development process and a software development process. As we execute many projects including hardware and software development, project teams have to follow both processes.

What is the most pragmatic way for an engineer to fulfill two processes that ask for the same action but are using different terms? For example, one process is asking for a Product Architecture, the other process is referring to a System Architecture. Or, one process requires a Test Plan, the other one a Test Specification. This will confuse engineers, wear out the confidence into the processes, and finally result into duplicated information in separate files with different names (to make sure each process is fulfilled).

There are other examples of inconsistent wording in processes that happen because people use their preferred vocabulary. Deliverables have to be "accepted", "approved", "signed-off" or "reviewed". They have to be "delivered" at a certain milestone or "completed" at a certain milestone. While the idea of these words is often the same, the exact meaning is slightly different.

Establishing and using a common set of vocabulary for deliverables, roles and actions in the process landscape is tedious and exhaustive. However, it is absolutely required to build a common understanding on the processes. When you define the terms together with the process owners, do not be surprised if you find out that even the process owners have completely different opinions on the definition of the terms.

Engineering processes are written for engineers. Or for process experts?

Processes need to be definite, unambiguous and easy to read. Luckily, there are languages available that allow to document processes in such way. RACI matrices (and all its flavors) and process models like [BPMN] or process chains are a few examples of them. They all share the goal that readers of the process get the exact same understanding of the process' content. But they also share the same disadvantage. Hardly any engineer understands that kind of language. Engineers are obligated to learn that language as self-study. This, however, opens the floodgates for misinterpretations.

Let us have a look into a simple RACI example based on the explanation found in Wikipedia. We understand that Wikipedia is not a credible reference for publications. However, where else would a "standard engineer" start searching for an explanation of a RACI matrix? The C (Consulted) on the German RACI explanation states that a person has relevant information on a subject and should or has to be asked. The English version says that it is a person whose opinion on a subject is sought. There is a subtle but important difference between "relevant information" and "opinion"! Other explanations of the C in the top 10 Google findings are "the person that can advise"[1], "responsible subject manner expert"[2] and "who will be communicated with regarding decisions and tasks"[3].

In ams AG, we currently prefer to use flow charts, block diagrams and descriptive text. The process documents are a bit longer as if a modeling language would have been used, but can be understood by our engineers. We accept the blur of the natural language because we believe that a more formal language might be more precise for experts but also more often misunderstood by non-experts.

The perfect tool supports the complete process flow, is fully automated, easy to use … and does not exist.

The market of tools is full of highly sophisticated tools that cover the complete software development cycle. There is hardly any niche that is not yet filled by a tool. We are not challenging these tools. However, we want to emphasize that rolling out a new process together with a new tool is a bad idea. Even though it sounds so obvious that the engineers will like the new process much more if some of the process steps are automated by a tool.

[1] https://projekte-leicht-gemacht.de

[2] https://www.projektmagazin.de

[3] https://project-management.com

Introducing a new process together with a new tool will cause an unnecessary steep learning curve. Shortcomings of the process will backfire to the acceptance of the tool. Moreover, shortcomings of the tool will slow down the introduction of the process. However, there is no process question that could not be answered by an engineer with the introduction of a tool. That's the nature of engineering. It is a noble intention, but we have not seen that working with test automation, requirements management, bug tracking, code reviews or any other engineering topic in ams AG.

We propose to concentrate on relatively trivial tools such as document templates and checklists. Such tools have a very high return-of-investment rate. They are easy to create, easy to use and easy to change when the process becomes more mature.

Tools that are more sophisticated should be considered if the process is already applied by the engineers and the engineers are considering optimizations to increase efficiency.

In such a case, make sure the tool is prepared (customized) for your process. Most of the tools are very flexible and allow many ways to do certain things. It is inefficient if tool users have to find their own way on how to use the tool to be compliant to a process.

Good processes are self-explaining but still require training.

Do not expect that all engineers will automatically read an announced process document. Some of them might not realize that the process is applicable for their work. Others might think that they know the content anyway. Most others might want to read it but will simply not find the time to do so. Last but not least, remember that process documents are terribly boring to read.

Try to get as many people into trainings as possible. Prepare a comprehensive training on the process. Take your time for lengthy discussions with engineers. Discussions are good! They show that the engineers are taking the topic serious. Be available for face-to-face trainings. Rolling out a process by the shotgun approach will not work.

The intranet is an insufficient communication tool.

We have experienced that many engineers do not know the content of our processes intranet page. It lists all the required information, gives examples and proposes best practices. However, we are still getting many questions where the answers would exist on the intranet page already. This shows that we did an insufficient job in communicating the information.

Do not expect that many engineers will actively search for information. This is not part of the daily business of an engineer. Instead, make sure that the information is brought to the engineers. Take care that the level of detail is sufficiently high. Just sending the link to the intranet page by email will not work. Invite for trainings, feedback rounds, workshops, presentations (e.g. how well the process works in one team or project), etc.

The Agile-Question.

Agile development is still a fancy subject and many software teams see such a workflow as a role model for their work. However, not all processes can take advantage of such a flow – especially if hardware development is involved. Be prepared to face the question on how agile development can be used to implement the process.

In ams AG, we are not yet using agile development. Our processes are based on the waterfall model. However, some of the software teams are trying to find their way to apply agile development. Both workflow variants are not necessarily contradicting. However, it is not an easy exercise to map an agile workflow to a milestone-based process.

Teams that want to experiment with agile development require special support. Especially the concept of iterations/sprints and the concept of project phases and milestones requires attention. There is no obvious mapping between sprints and milestones. Not every sprint can run through all milestones. Passing the milestones in sequential sprints, however, renders the agile approach useless. As a rule of thumb, we are using sprint 0 and 1 to setup the project including the requirements (epics and user stories) and a very high-level architecture/concept. The main sprints execute refinements of the architecture and the implementation and testing of the user stories. The last couple of sprints are reserved for release testing, bug fixing and releasing.

Summarizing the …ings

Many things can go wrong with poor wording in a process. Avoid phrases that cause reluctance at engineers (like documentation). Be precise and consistent but use a language that can be easily understood by the readers.

Support the process with appropriate tooling. Checklists and templates are sufficient at the beginning. Introduce sophisticated tools when the process is in use already and the workflow should be made more efficient.

Pay attention to communication and training on adequate level of detail. Be invasive in communication and responsive on feedback and questions. Do not underestimate the effort for these activities.

Bibliography

[BPMN], **Object Management Group**, Business Process Model and Notation Specification Version 2.0, 2011, https://www.omg.org/spec/BPMN/2.0/

Author

Johannes Loinig

Johannes Loinig is the software quality manager at ams AG. He works on definition, roll-out and audits for several processes such as software development and requirements management.

johannes.loinig@ams.com

The IoT-Testware

Functional and non-functional testing for the IoT

The Internet of Things (IoT) is omnipresent. More and more hardware devices get connected and will collect and share huge amounts of data in the near future. This progress will lead to a digital and hyper-connected world. Though, in such growing networks of interconnected things, quality assurance (QA) will become a continuous challenge. Especially aspects like conformance, interoperability and security but also performance and robustness will require an increased attention from QA perspective.

Introduction

According to market research institutes, the IoT is one of the most emerging technologies today and the new saviour of an ideal digital world. However, in addition to the huge potential, there are still open issues arising from the adoption of IoT. Special attention must be paid to the interoperability and security. Although these topics are not new, IoT will require an increased focus for a successful future. While e.g. security issues within the traditional internet [Tech18] mostly affect the data privacy, security breaches in the IoT can be potentially dangerous for human lives. The heterogeneity of the connected systems and the challenge of interoperability between those systems adds up the complexity and thus directly affects the overall quality of these systems.

Having the complexity and increased the threat of IoT in mind, security and interoperability aspects need to be approached in a systematic and standardized way. The Eclipse project IoT-Testware [EcIoT] is intended to pioneer such an approach and deliver executable tests which are derived systematically from test purposes standardized at ETSI [ET04].

IoT from a Tester's perspective

The IoT itself is not just a technology, but rather an architectural paradigm which aims to connect many different and heterogeneous technologies via communication networks. From this perspective, the IoT is technologically not new at all. However, the ubiquitous interconnectivity of heterogeneity of the devices – even devices which were never designed to communicate to the internet – is fairly new. Fortunately, QA of heterogeneous interconnected devices does not need to be invented from scratch. The telecommunications industry has developed and matured methodologies and tools over the past decades which can be easily reused and applied to the IoT. In particular, the assurance of protocol conformance and technical interoperability as defined in [Vee08] and [ISO9646] are applicable to IoT.

Methodology

As discussed in the previous sections the QA process activities are very important for IoT solutions. But at the same time, the increasing complexity and the vast amount of different combinations make a comprehensive QA extremely expensive and almost impossible to achieve by a single vendor alone. For this reason, we rely on a standardized approach and focus on black-box testing against existing standards and specifications. This facilitates the IoT-Testware to be vendor-agnostic and be still useful for vendors and customers from different areas of the IoT-landscape.

Figure 1 illustrates an overview regarding two development procedures and its relationships: (a) the definition and implementation of the target system (under test) that needs to address subjects, assets and requirements as well as threats, policies and assumptions, and (b) the test development steps including test architecture, test purposes and test suite structure. System and test engineers need to derive the test implementation to be executed and analysed, e.g. for certification purposes.

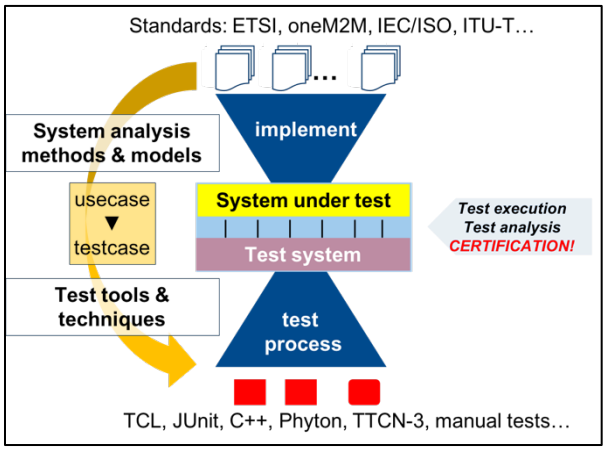


Figure 1: The role of testing for quality assessments

The test development methodology derived from [Vee08] and [ISO9646] is depicted in 2. With this phased approach

standards and specifications can be translated into digital artefacts which build on top of each other.

The process generates the following artefacts:

- Test Suite Structure (TSS): The specification of the TSS is the first step. There are no hard rules for specifying the TSS. However, the ETSI provides many best-practices and recommendations. This step is essential for identifying proper test groups and test architecture.
- Test Purpose (TP): The description of TPs emphasizes the test objective rather than on the implementation. Thus, not only the complexity is separated, but also the reusability of TPs is given. The next section outlines the test description.
- Abstract Test Suite (ATS): The ATS is the implementation of the previously defined TPs. The ATS results usually in a collection of Test Cases (TCs) which can be mapped against the corresponding TPs and reflect the TSS. The ATS can be implemented in different technologies (e.g. Python, Tcl, C++, TTCN-3, etc.).
- Executable Test Suite (ETS): This step comprises the executable artefact. Depending on the chosen technology this process step might be optional or combined with the ATS. For example, for interpreted programming languages (e.g. Tcl or Python) the ATS is also executable. Compiled languages (e.g. TTCN-3 or C++) on the other hand produce clearly distinguishable artefacts.

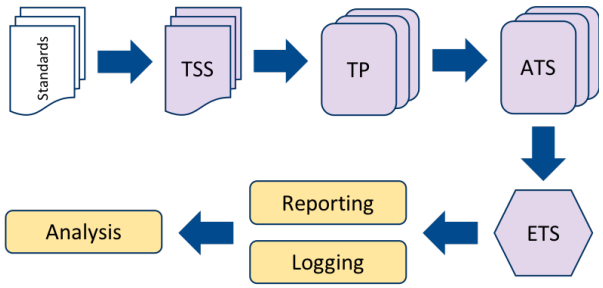


Figure 2: Conformance Testing

Test Descriptions

In industry and standardization, you may find multiple notations and design templates for the definition of test purposes. The contents differ due to the scope of aspects but also on the degree of details. Since it is our intention to provide our results and documents, also for certification or labelling purposes we decided to apply a more detailed description including structure and styles already used in standardization bodies. Therefore two techniques appear suitable, the UML-based notation (UTP) from the OMG and the TDL-based approach from ETSI

Especially in the context of non-functional testing and due to its simplicity, we followed the test development approaches from ETSI and its Test Description Language TDL. In particular, we apply TDL-TO, i.e. the part 4 [ET02] of TDL that is dedicated for the test objective definitions. This notation has been developed as TPLan in the past and is used now in several technical committees at ETSI, e.g. for the Intelligent Transport Systems [ET01].

TTCN-3

The Testing and Test Control Notation TTCN-3 is the only standardized test specification and implementation language used in the industry an research to define and execute different types of tests in multiple domains like e.g. telecommunication or automotive. The notation supports an abstract description of test scenarios (ATS). Due to multiple tool support, including powerful open source projects, it is successfully worldwide in use e.g. for testing and certification of mobile phones by 3GPP or the oneM2M products.

Eclipse and ETSI

Test projects at ETSI TC MTS, working group TST, currently cover test purposes for MQTT, CoAP, LoRaWAN and foundational security IoT-Profile of IEC 62443-4-2. The work of ETSI MTS-TST is correlated with active Open Source projects currently hosted by the Eclipse Foundation. The technical contributions from the Eclipse members are coordinated by ten dedicated Eclipse committers. The overall work includes Test purposes in TDL-TO but also TTCN-3 test code developments that is important for test campaign execution in the test labs. In particular, ETSI members from MTS-TST control the test purposes developments and are responsible for the utilization of the resulting TP definitions for the ETSI working items and technical specifications. This approach allows to get input from active developers from the Eclipse community and a fast implementation of the target test suites for the interested industry but also support a faster development of ETSI specifications.

Software Tools

Although the described methodology is openly available, the corresponding tool support is largely unknown beyond standardization and research bodies. The following non-exhaustive list will give an overview of some of the open-source tools we used to implement the methodology and the IoT-Testware.

Description of Test Purposes

Despite the fact, that the Test Purposes (TPs) can be described in plaintext, TDL-TO enriches the TPs with a certain formalism. This formalism does not only increase the quality of the TPs but is also suitable for tool support. This tool support is directly provided by the ETSI as an Eclipse IDE plugin and can be found in the TDL Open Source Project (TOP) at https://tdl.etsi.org.

Implementation of Test Cases

As described in the previous section, we used TTCN-3 for the implementation of the executable Test Cases (ETS). For this purpose, we had the choice between multiple TTCN-3 IDEs and runtime environments. But while Spirent's TTworkbench or Elvior's TestCast would have fit perfectly our technical requirements, we have chosen Eclipse Titan. The major advantage of Eclipse Titan is its availability as open-source without any drawbacks in quality or support.

Load- and Performance-Testing

Thanks to TTCN-3's and Titan's sophisticated architectures, this technology stack is also highly suitable for the implementation of load- and performance tests. For this purpose, Titan provides the RIoT application which is built on top of Titan itself and its Core Load Library (CLL) as shown in Figure 3.

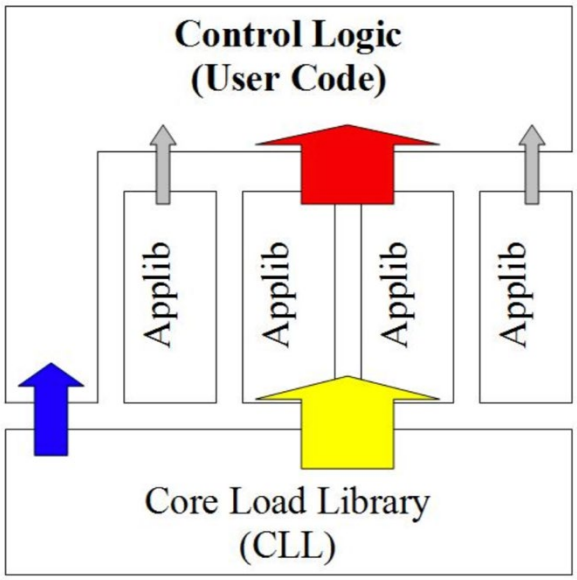


Figure 1: Eclipse Titan RIoT

Besides the application libraries (Applib) for CoAP and MQTT and some other IoT related protocols, RIoT provides also libraries for Model-based testing (MBT) and the support for using finite-state machines (FSM) for describing load- and performance tests. With this feature, RIoT enables a tester to specify the behaviour-model once and derive as much as required realistic test data for load generation.

Security- and Robustness-Testing

The robustness of an implementation is an important attribute in the context of security. For this purpose, the IoT-Testware utilizes the concept of fuzzing to ensure a robust behaviour on malicious input data. Because the input space for invalid data can grow very quickly additional tools are required to manage the vast amount of possible input data. The generic fuzzing data generator Fuzzino, which is developed and maintained by Fraunhofer FOKUS, was integrated with the IoT-Testware.

In *Figure 4* an exemplary integration based on the CoAP conformance test suite is depicted. In this example, Wireshark was used to capture the sent packets from a CoAP conformance test run. In the next step, the captured packets can be used as input data for Fuzzino to generate the fuzzed data. These data can be stored and used afterwards to be played against the same or another SUT. In this experiment, Fuzzino proved its powerfulness. Using the CoAP test suite, we were able to capture 46 CoAP packets which we used to feed Fuzzino. Through mutation of these conformant CoAP packets, Fuzzino was able to derive several thousands of fuzzed CoAP packets.

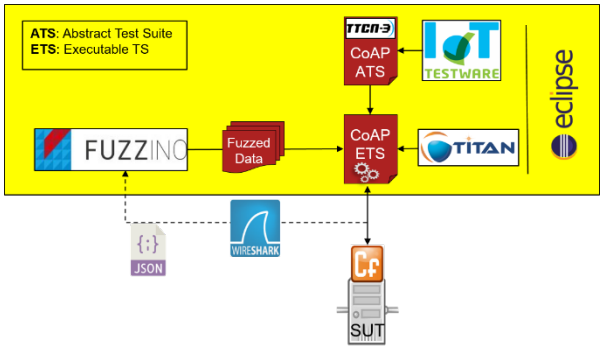


Figure 2: Fuzzing based on CoAP-Testing

Due to this approach, the fuzzing integration is extremely flexible and robustness testing can be performed even without Fuzzino. For this purpose, we will provide a selected set of approved and tested fuzzed data for each test suite. This will allow to compare the robustness of different SUTs and also enable deterministic regression tests in case of any vulnerability and weakness findings.

Results and Findings

Prior to the IoT-Testware, it was almost impossible to impartially compare different implementations of an IoT protocol. Simplifying, the quality measure "protocol conformance" was rarely assured in the IoT domain. While this fundamental requirement is mandatory in many industries (e.g. telecommunication, automotive or industrial automation) the IoT domain is still lacking maturity and awareness in this area.

In the scope of testing the IoT-Testware, we decided to run it against different publicly available MQTT brokers and compare these results (see Table 1).

Table 1: MQTT Brokers

SUT	Pass	Fail	Inconc.
HiveMQ	86,67%	8,89%	4,44%
Mosquitto	84,44%	11,11%	4,44%

VerneMQ	82,22%	11,11%	6,67%
EMQ	77,78%	17,78%	4,44%
Lannister	68,89%	26,67%	4,44%
ActiveMQ	64,44%	31,11%	4,44%
aedes	57,78%	37,78%	4,44%
Moquette	35,56%	64,44%	0,00%

Although the results might lead to the assumption that MQTT brokers are poorly implemented, this assumption does not hold in general. Most of the tested brokers interoperate "out of the box" with most of the publicly available MQTT clients (e.g. Eclipse Paho) and thus are suitable for small PoCs or "hobby projects".

However, our main goal was to investigate the behaviour of the Systems Under Test (SUTs) on edge cases and ensure the correct error handling. While on small PoC the correct negative behaviour of a message broker could be negligible, in productive use non-conformant behaviours can lead to sporadic non-interoperability or even provide additional attack surfaces for hackers. If we think of IoT installations in industrial manufacturing, smart cities or critical infrastructures such rare scenarios might have horrific impacts.

Conclusion

No question – the IoT has a lot of potentials and will be one of the key pillars of digital transformation and Industry 4.0. However, to meet the expectations and requirements the IoT still needs to evolve and mature. A systematic QA and standardized approaches – especially for security assurance – will be indispensable.

The findings revealed that the demand for standardized open-source testing solutions exists. This demand will even grow with the blanket adoption of IoT and increasing interconnections. While security awareness is widely disseminated, the assurance of security will stay a constant challenge in IoT.

Finally, we like to thank our colleagues from the IoT-T project (www.iot-t.de) for their input to this article, in particular, Mr. Sascha Hackel and Mr. Dorian Knoblauch for their implementation support.

Bibliography

[Tech18] Techworld Staff, „Techworld," IDG, 25 October 2018. [Online]. Available: https://www.techworld.com/security/uks-most-infamous-data-breaches-3604586/. [Accessed: October 2018]

[Vee08] Hans van der Veer, Anthony Wiles, „Achieving Technical Interoperability - the ETSI Approach," ETSI, Sophia Antipolis, 2008.

[ISO9646] ISO, ISO/IEC 9646-1:1994 Information technology -- Open Systems Interconnection -- Conformance testing methodology and framework -- Part 1: General concepts, ISO, 1994.

[ET01] ETSI TS 102 868-2 V1.4.1: Intelligent Transport Systems (ITS); Testing; Conformance test specifications for Cooperative Awareness Basic Service (CA); Part 2: Test Suite Structure and Test Purposes (TSS & TP), 2017.

[ET02] ETSI ES 203 119-4 V1.3.1: Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 4: Structured Test Objective Specification (Extension), 2018.

[ET03] ETSI MTS Test Specification for foundational Security IoT-Profile, https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=54751.

[ET04] ETSI MTS Working Group TST https://portal.etsi.org/TBSiteMap/MTS/MTSTSTToR.aspx

[EcIoT] Eclipse IoT-Testware https://projects.eclipse.org/projects/technology.iottestware

Authors

Axel Rennoch

Axel is computer scientist at the Fraunhofer Institute for Open Communication Systems in Berlin. As a member of the System Quality Competence Center he is involved / responsible for validation and testing projects on next generation networks and software technologies.

axel.rennoch@fokus.fraunhofer.de

Alexander Kaiser

Alexander is computer scientist at Relayr GmbH in Berlin and responsible for researching IoT related technologies and methodologies.

alexander.kaiser@relayr.