Conference Journal 2018

Interesting Insights into Professional Practice

Papers of Lecturers at the Software Quality Days

2018 software quality days

EXPERIENCE THE VALUE OF QUALITY

Impressum

Publisher / Herausgeber

Software Quality Lab GmbH
Gewerbepark Urfahr 6
4040 Linz
Austria

www.software-quality-lab.com
info@software-quality-lab.com
+43 5 0657-0

Published by Software Quality Lab GmbH, January 2018

Disclaimer

Haftungsausschluss

Content

2019 software Quality days

EXPERIENCE THE VALUE OF QUALITY

15th – 18th January, 2019 Vienna

SUBMIT YOUR PAPERS FOR THE SOFTWARE QUALITY DAYS!

You are kindly requested to submit your application for a presentation, tutorial or workshop to be included as part of the conference.

↗ **Submit your application starting from April 2018!**

www.software-quality-days.com

ONLY UNTIL JULY 15

WE LOOK FORWARD TO WELCOMING YOU IN 2019!

Key topic

The complexity and challenges of Software Engineering and Software Quality in the Cloud

Further topics

Requirements, testing, automation, agile, project management, quality management, processes, techniques and methods

- Practical tracks
- Scientific track
- Solution Provider Forum
- Top keynote speakers
- Lectures
- Workshops and tutorials

Highlights

TOOL CHALLENGE

BEST QUALITY TOOL AWARD

TOP KEYNOTE SPEAKERS

↗ **Submit your application starting from April 2018!**

www.software-quality-days.com

Continuous Quality

Fulfilling industry regulations and quality expectations in a world of continuous delivery

With the explosion of digital capabilities state-of-the-art development approaches from IT companies, such as continuous delivery and DevOps, are becoming increasingly relevant also for industrial companies. These approaches are however in many aspects not compatible with traditional development processes. This article provides a new perspective on understanding and reporting of development quality, which enables industrial companies to take advantage of the benefits of continuous delivery, while maintaining their high standard of quality.

Motivation for continuous delivery

Daily and more frequent enhancements of live systems are already common practice across many businesses outside industrial environments. Starting almost a decade ago, several related approaches have been developed that multiplied the deployment frequency within those companies by several orders of magnitude, leading to a paradigm shift that affects all parts of the organization. The most prominent representatives of these approaches are continuous delivery [1] and DevOps [4].

Today, those approaches have become a standard in many IT companies and are applied increasingly also in other domains as e.g. the finance industry. An indicator for the broad popularity of continuous delivery and DevOps is the fact, that they have entered the realm of "Dummies books" [5] and novel type descriptions [2].

With the boost of digital capabilities, much more frequent deliveries will also become critical for the success of industrial companies [3]. Demands from customers for speedy fixing of bugs and security issues, as well as continuous enhancements of their products during their lifetime will lead to evolving products that are continuously enhanced after delivery – "fluid" products. These fluid products can be cloud solutions operated by the company itself, as well as continuously evolving devices at customer sites operated by the customers, or a mix thereof.

Those demands require a shift from discrete product lifecycle management (PLM) and operations to continuous end-to-end development, deployment & operations, i.e. continuous delivery and DevOps. There are however significant challenges associated with this shift and PLM has to be further innovated to enable application of those approaches in industrial environments. One of the main challenges thereby is the adaptation of current approval and release processes to allow frequent deployments, which is the focus of this article.

Before investigating the challenges and potential solutions, it is worthwhile to look at the benefits that are reported by companies already using continuous delivery and DevOps approaches:

- Fast delivery of value and integration of customer feedback and change requests through delivery of customer value in much shorter cycles than today.
- Continuous value increase throughout complete lifetime of used product due to continuous product improvement based on post-deployment operational data and A/B field experiments.
- Reduced customers operational risks through highly automated delivery of small changes: the vision is that deployment is a "non-event".
- Improved quality and availability due to fast deployment of bug fixes.
- Up-to-date cyber security.

In addition to an extremely fast delivery of newly created value, bug fixes and cyber security updates to the customers, the ability to learn fast from usage data and A/B experiments, as well as the optimization of the development, delivery and roll-out procedures through controlled deployment of very small changes, will lead to a continuous enhancement of experience for the customers and maximize the benefits.

Please note, continuous delivery in an industrial environment does not necessarily mean multiple releases into a production system per day as is practice in several IT companies. It stands for the move from yearly release cycles to much more frequent deployments. Depending on the boundary conditions this could be e.g. a deployment into a production system with every sprint or a fix three months release cycle but with the ability to deploy bug fixes and cyber security updates with a defined process within days whenever desired. Even if the business decision is to not yet release the latest version to the customer, there is a benefit of keeping the system in a state that is continuously "ready" to release.

Challenges in industrial environments

When looking at industrial companies, three types of application scenarios can be distinguished that will

benefit from a significant increase of deployment frequency:

- Continuous enhancement of cloud based solutions, i.e. the organization itself operates one product instance.
- Continuous enhancement of on-premise software devices, i.e. many product instances at customer sites, operated by the customers.
- Continuous enhancement of on-premise embedded systems.

Industrial companies are however often restricted by multiple boundary conditions that are induced by the nature of their products and their use cases:

- Quality expectations
- Standards & norms
- Regulated industries
- Systems composed of hardware, software and services
- On-premise devices
- Embedded systems

This results in multiple and significant challenges that have to be addressed in order to leverage the potential of an increased deployment frequency in industrial environments:

- Adaptation of business model to leverage potential of continuous delivery.
- Addressing of customer expectations and also concerns regarding continuous delivery.
- Alignment with internal and external partners to enable a continuous delivery chain.
- Adaptation of processes to allow continuous delivery.
- Creation of an organization that fosters continuous delivery.
- Change towards a continuous delivery culture, including a strong focus on built-in quality.
- Adaptation of product architecture and technology to support continuous delivery.
- Establishing of an environment technology that enables automation of the delivery chain (often called the “delivery pipeline”).

As can be seen from the list above, moving towards continuous delivery is a paradigm shift affecting all aspects of the organization significantly and requires a rethinking of current PLM approaches. The objective is to have an approach that reaches the same high quality, but does so through a more automated and continuous approach.

The Q-gate problem

As noted in the first section, the adaptation of current approval and release processes to allow frequent deployments – which is the focus of this article – is one of the main challenges when moving into this direction. To visualize this aspect, one can consider three interrelated levels (see also figure 1):

- Level 1: The external standards and norms that have to be fulfilled within the specific industry branch.
- Level 2: The internal regulations (processes, roles and responsibilities) setup by the organization to ensure fulfillment of external standards and norms, i.e. the PLM practices.
- Level 3: The actual development working mode and working environment that shall comply with the internal regulations.

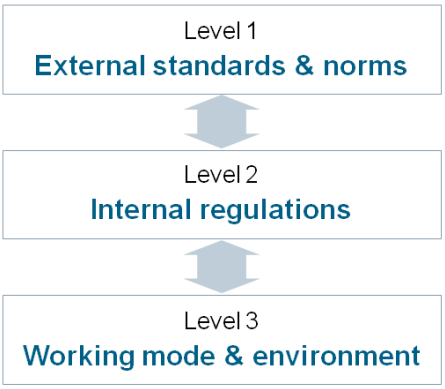


Figure 1 – External standards & norms versus internal regulations

When the goal is to implement a continuous delivery working mode with an automated delivery chain to accelerate change and ingrain built-in quality into the development process, this is a paradigm shift to previous approaches. The key problem thereby is, that today's PLM practices were setup more than a decade ago and do not necessarily support future state-of-the-art working mode and environment any more, they can actually hinder it. The connect between external regulations and desired development working mode has gone out of sync, visible in the opposition of comprehensive, “all-inclusive” Q-gate processes that take days or even weeks to prepare and the desire for frequent and fast deployments.

Although the expectation is that external standards and norms will probably adapt in the future to support modern working practices, this does not solve the challenge today. So the key question is: how can the internal regulation / PLM practices be adapted to optimally support future state-of-the-art working mode and environment, yet satisfying the requirements of external standards and norms?

Quality in the delivery pipeline

A core aspect and prerequisite for continuous delivery is building a continuous delivery pipeline with a high degree of automation, including automated tests on multiple levels (see also figure 2).

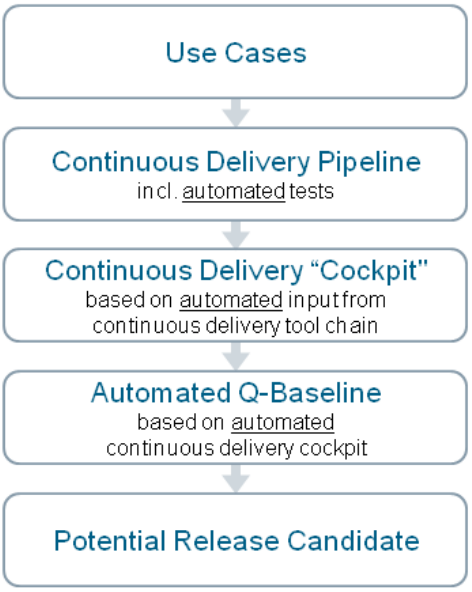


Figure 2 – Delivery pipeline

Once established, such a continuous delivery pipeline provides huge benefits regarding built-in-quality aspects and real-time quality status information. Quality information typically provided by a continuous delivery pipeline include:

- Continuous, automated real-time traceability at any point in time throughout development and test
- Linkage of all artifacts: user story → design → code → build → unit test → integration test → bug fixes.
- Explicit visibility of status of e.g. risk related user stories.
- Automated security checks (e.g. static code analysis).
- Automated decision whether a build is a "potential release candidate" based on automated checks.
- Auditable evidence (prerequisite: long term storage & retrieval of all relevant information).

Although such a continuous delivery pipeline already provides significant quality status information, it lacks some aspects required for a formal release for deployment in an industrial environment:

- Addressing of system level risks.
- Sign off of results by a function independent from development team (availability of designated quality and risk manager to approve multiple release candidates per day not realistic).
- Formal approval and release documentation (e.g. hazard summary report, risk management report, safety certification).
- Up-to-date non-development artifacts required for a release (e.g. instructions for use, translations, operations trainings).

I.e., in order to formally release and deploy a "potential release candidate" into a production environment, there are still several "manual" activities that have to be completed. The objective is to define an intelligent mechanism that takes maximum advantage of the information already generated by the automated delivery pipeline (automated Q-baseline) and keeps the manual intervention to a minimum and integrates it seamlessly into the continuous delivery cycle.

Continuous quality concept

If one starts with the checklists of today's Q-gates, one will come across a list of many criteria. To reach the objective described in the previous section, it is useful to separate those criteria into five different categories:

- Criteria that can be evaluated fully automated through a continuous delivery pipeline.

Those criteria are e.g. build status and results of static code analysis, virus check and automated tests. In a continuous delivery pipeline, those criteria are evaluated automatically with every commit.

- Criteria, for which the status can be extracted fully automated from other tools.

Those criteria are e.g. status of manual tests, defects, open issues or user documentation. If these items are managed within tools and the status of the criteria is maintained continuously in the tools, it can be automatically transferred into an overall Q-dashboard on a regular basis.

- Criteria that have to be evaluated manually.

Those criteria are e.g. the final judgment of the overall quality status by an independent quality manager, based on the aggregated information in an overall Q-dashboard.

- Criteria that are only relevant for an initial market launch of a product or specific features.

Those criteria are e.g. product safety analysis, third party software license clearing, export controls and patents. The evaluation of those criteria must be part of an initial release baseline, for small delta releases they however need to be addressed only on demand, based on evaluation and tagging of the features during the upfront planning of the next development cycles. It is important to note, that in a continuous delivery mode one should distinguish between the initial market launch and the subsequent continuous flow of value to the customers. Whereas for the initial market launch all criteria are relevant, for a small delta release later on during the lifetime of the product many criteria are not relevant, since the change typically affects only specific aspects.

- Criteria that are not related to a release decisions, but should be addressed on a regular or on-demand basis on organizational level.

Those criteria are e.g. employee qualification, internal audits, general improvements list. These aspects should be driven on a regular and on-demand basis by the organization, in parallel to the continuous value flow of product development and operations.

Figure 3 shows an approach that utilizes the structuring of release criteria into the five categories described above.

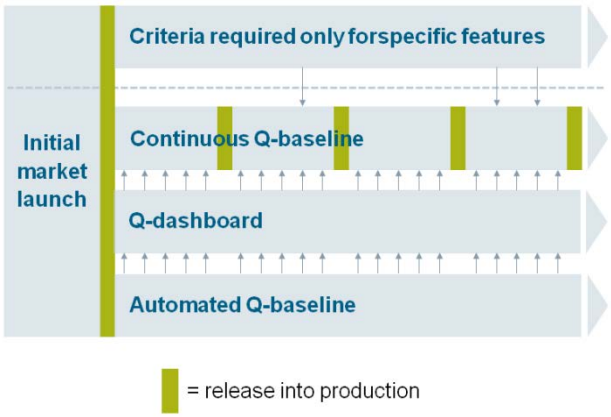


Figure 3 – Continuous quality concept

Core of the approach is the regular release into a production environment driven by a successful completion of the continuous Q-baseline. This Q-baseline is based on the status information provided by the automated delivery pipeline and the automated transfer from other tools, supplemented by the manual evaluation of the criteria that cannot be automated. The continuous evaluation of all features for applicability of criteria that are only relevant for specific features trigger tasks as required, which are then fed into the continuous Q-baseline.

Prerequisites and implementation approach

Moving into such a direction requires a high degree of automation within build, test and configuration management as well as a significant cultural change, especially regarding the following topics:

- High quality focus in every step.
- Trust in automation.
- Trust into team by management.
- Renunciation from retrospective view.
- Change in quality responsibilities.

In addition, it requires a rethinking of business models and customer relationships. The ability to reduce dependencies between features significantly supports faster release cycles.

The following steps can be used as guidance to implement the approach described in the previous section:

- Establish continuous delivery pipeline.
- Classify current Q-gates criteria, one by one:
  - Can the criteria be addressed on organizational level?
  - Is the criteria typically required for continuous enhancements, i.e. for a small release with only minor changes to the running software?
  - Can evaluation of the criteria be automated?
  - Can a delta-evaluation of the criteria be achieved (i.e. build on results of previous evaluation)?
- Define automated Q-baseline.
- Define Q-dashboard.
- Define continuous Q-baseline.
- Clarify process for evaluation and tagging of features.
- Setup redesigned release process.

Summary

Continuous delivery requires a rethinking of the current release and approval process and its decision criteria. An automated delivery pipeline already provides significant quality status information which can be utilized to speed up the release process. Systematic structuring of today's Q-gate criteria and setup of redesign quality evaluation and reporting approach provides also for industrial companies the possibility to deploy much more frequently into production systems than today.

Bibliography

[1] Humble, Farley, Continuous Delivery, Addison-Wesley, 2010

[2] Kim, Behr, Spafford, The Phoenix Project, It Revolution Press, 2014

[3] Bosch, Speed, Data, and Ecosystems, CRC Press, 2017

[4] Allspaw, Hammond, 10+ Deploys per Day: Dev and Ops Cooperation at Flickr, DevOps Days Ghent, 2009, https://www.slideshare.net/mobile/jallspaw/10-deploys-per-day-dev-and-ops-cooperation-at-flickr

[5] Sharma, Coyne, DevOps for Dummies, IBM Press, 2014, https://www.ibm.com/ibm/devops/us/en/resources/dummiesbooks/

Autor

Dr. Peter Fassbinder
Principal Key Expert Consultant
PLM Process Innovation
Siemens AG
peter.fassbinder@siemens.com

Maschinenethik

Wie (be)schreibt man ethische Software?

Wenn Sie lügen müssten, um jemandem zu helfen, würden Sie nur ganz kurz abwägen, um sich zu entscheiden. Auch wenn Sie ein Fremder bittet, für ihn ein Auto zu knacken, würden Sie die Situation schnell durchschauen. Doch: Würde es ein autonomer Roboter auch wissen? Wie? Woher?

Darum geht es in diesem Workshop: Wie erhebt, beschreibt und modelliert man Ethik-Anforderungen an autonome IT-Systeme? In kleinen Fallstudien wenden wir gängige Requirements Engineering Methoden auf konkrete Beispiele an. (bis 600 Zeichen)

Wozu brauchen Maschinen Ethik?

Autonome technische Systeme wie Allzweck-Roboter, Chatbots oder selbstfahrende Autos sind keine passiven Werkzeuge mehr, sondern treffen selbsttätig Entscheidungen. Damit übernehmen sie bzw. ihre Hersteller Verantwortung für die Folgen ihres Handelns. Diese Folgen sind in einem komplexen Umfeld wie dem Alltag oder dem Straßenverkehr nicht einfach vorhersehbar. Trotzdem sollen sie gesetzeskonform und ethisch ausfallen.

Was ist Ethik?

Wenn autonome technische Systeme und künstliche Intelligenzen sich selbsttätig in dieser Welt zurecht finden und richtig handeln sollen, dann muss ihnen unser intuitives, in Jahren erworbenes Wissen über „richtig“ und „falsch“, „gut“ und „böse“ vermittelt werden. Dazu müssen wir es zunächst uns selbst bewusst machen, also implizites in explizites Wissen umwandeln und maschinentauglich formulieren.

Verschiedene Ethikansätze sind unterschiedlich schwer zu implementieren: Allgemeingültige Regeln wie “Du musst immer…“ oder „Du darfst nie…“ wären der einfachste Fall. Aus solchen Regeln besteht die Pflichtethik, und sie lassen sich in eine Maschine hartcodiert einprogrammieren. Mit einem solchen Regelsatz gerät man jedoch an Grenzen, wenn sie einander im konkreten Fall widersprechen, zum Beispiel, wenn man eine Regel brechen muss, um zu helfen oder ein Unglück zu verhindern. Solche Widersprüche lassen sich teilweise dadurch auflösen, indem die Regeln priorisiert werden.

Der Kantsche Imperativ fragt sich, ob eine Handlung auch dann noch in Ordnung wäre, wenn es alle so machen. Diese Überlegung verlangt von der künstlichen Intelligenz die Fähigkeit, sich eine solche Situation vorzustellen und sie zu bewerten.

Besonders die „goldene Regel“, dass wir andere so behandeln sollen wie wir behandelt werden wollen, ist für Maschinen schwierig anzuwenden. Dazu müssten sie sich in einen Menschen hinein fühlen. Wir wollen ja nicht wirklich, dass eine Maschine uns so behandelt, wie sie von uns behandelt werden will. Was auch immer das sein möge. (Die Kette ölen, bitte?)

Der Utilitarismus wägt Nutzen und Schäden Handlungsalternativen gegeneinander ab. Hierzu muss die künstliche Intelligenz Folgen von Handlungen vorhersehen und bewerten können, was ein viel weitreichenderes Weltverständnis verlangt als die Anwendung von Regeln.

Noch schwieriger umzusetzen ist die Gesinnungsethik. Hierbei sollen die Handlungsalternativen bezüglich abstrakter ethischer Werte wie Ehrlichkeit oder Gerechtigkeit bewertet werden. Dieser Algorithmus verlangt ein noch weiter reichendes ethisches Verständnis von der Welt als die vorhergehenden. Auch Empathie wäre hilfreich, kann aber von Maschinen maximal simuliert werden, niemals wirklich empfunden.

Implementierung von Maschinenethik

Die oben beschriebenen ethischen Algorithmen sind für Menschen leicht durchführbar, für eine Maschine jedoch anspruchsvoll! Sie verlangen ein umfangreiches Wissen über die Welt und über die späteren Folgen von Handlungen. Aber auch das selbsttätige Erkennen, dass eine Entscheidung genau jetzt gerade nötig ist, ist nicht trivial. Das gelingt auch Menschen nicht immer.

Man unterscheidet zwei grundlegend unterschiedliche Ansätze, um Maschinen Ethik beizubringen:

1. Die ethischen Regeln werden fest einprogrammiert. Dazu müssen alle möglichen Situationen vorab bekannt sein, welche der künstlichen Intelligenz begegnen können. Ein „Ich verstehe das nicht“ kombiniert mit ratloser Untätigkeit kann in vielen Situationen unethisch sein. Der Vorteil dieser Umsetzung besteht darin, dass das System deterministisch handelt und seine Begründungen nachvollziehbar sind. Hier müsste der Hersteller die Haftung für das Verhalten des Systems übernehmen.
2. Lernen: Künstliche Intelligenzen (z.B. neuronale Netze) können durch Feedback lernen. Allerdings können sie dabei auch Falsches lernen. Ihre Flexibilität und Fähigkeit zur Weiterentwicklung bringt auch den Nachteil mit sich, dass ihre Entscheidungen weder transparent

sind noch später nachvollzogen werden können. Die Haftung für die Handlung solcher Systeme geht dann auf den Trainer, also den Besitzer über. Vielleicht wollen wir das gar nicht, wenn z.B. Verbrecher Roboter zu Autoknackern ausbilden.

Als dritte Möglichkeit gibt es noch die hybride Kombination aus beiden: Das System lernt, aber innerhalb eines fest abgesteckten ethischen Rahmens. So können Regeln grundsätzliche Tabus vorgeben, beispielsweise Gesetze, die nicht gebrochen werden dürfen. Innerhalb dieses Rahmens darf das System dann lernen. Somit könnte man ungesetzliche Handlungen von vorne herein verbieten. Unbenommen bleibt die Möglichkeit, dass die Autoknacker den Code hacken und die entsprechende Regel löschen.

Requirements Engineering Techniken für Maschinenethik

Wie können die Entscheidungen einer Maschine modelliert werden? Aber auch: Welche Entscheidungen darf eine Maschine überhaupt treffen? Es ist auch unethisch, einer künstlichen Intelligenz zu viel Verantwortung zuzumuten. Solange eine Maschine nur ein passives Werkzeug ist, trägt ihr menschlicher Führer die ethische und juristische Verantwortung für ihr Handeln und dessen Folgen.

Es gibt auch Argumente dafür, dass Maschinen in manchen Situationen ethischer handeln können als Menschen, weil sie sich besser an Regeln halten und nicht emotional reagieren, auch nicht bei Gefahr (vgl. den Artikel von Arkin).

Vielleicht ist gerade ein Team aus Maschine und Mensch die perfekte Einheit, um ethisch korrekte Entscheidungen zu treffen? Dann könnten das Wissen und die Fähigkeiten der Maschine kombiniert werden mit dem Mitgefühl, der Ethik, dem Weltverständnis und der Intuition des menschlichen Bedieners.

Für die Erhebung, Konkretisierung und Darstellung von Ethik-Anforderungen eignen sich gängige Methoden wie Nutzwerttabellen, Entscheidungstabellen, Entscheidungsbäume sowie Zustandsdiagramme. Die Welt kann als Datenmodell dargestellt werden. Aber auch Methoden aus dem Sicherheitsbereich eignen sich gut, gerade auch die ethischen Risiken zu definieren, die nicht eintreten dürfen. Auch hieraus ergeben sich dann Anforderungen an die künstliche Intelligenz, ihre Entscheidungen und deren Grenzen.

Die Quellen für Ethikanforderungen sind Gesetze und andere Regeln. Hinzu kommen aber auch ungeschriebene Gesetze, Erfahrungswerte, im Fachbereich übliche Praktiken und Bräuche. Dass auch gute Menschen regelmäßig Regeln brechen, ist ein Zeichen dafür, dass die verschriftlichten Regeln nicht alle auftretenden Fälle genügend genau abdecken. Solche ungeschriebenen Gesetze erhebt man beispielsweise durch anonymisierte Interviews oder Erfahrungsberichte. Vielleicht kann man auch in vielen Situationen verschiedener Meinung sein und möchte eine repräsentative Fragebogenbefragung durchführen im Stil von: „Wie würden Sie entscheiden?“

Erst nachdem die Anforderungen klar definiert sind, welches ethische Handeln im Anwendungsbereich benötigt wird, kann im zweiten Schritt überlegt werden, welche dieser Anforderungen eine Maschine oder ein Mensch besser erfüllen.

Der Workshop

In einem halbtägigen Workshop konkretisieren wir in Gruppenarbeit für verschiedene Beispiele die Ethik-Anforderungen und deren Testfälle.

Damit geht dieser Workshop über die üblichen allgemeinphilosophischen Diskussionen hinaus und zeigt so die Herausforderungen, aber auch Lösungen auf.

Literatur

Arkin, Ronald C.: *Governing Lethal Behavior: Embedding Ethics in a Hybrid Deliberative/Reactive Robot Architecture.* Technical Report GIT-GVU-07-11 https://smartech.gatech.edu/bitstream/handle/1853/22715/formalizationv35.pdf

Erlinger, Rainer: *Moral - Wie man richtig gut lebt*, S. Fischer, 2011

Ulrich, Peter: *Integrative Wirtschaftsethik*, Haupt, Bern/Stuttgart/Wien, 2. Auflage, 1998

Autorin

Dr. habil. Andrea Herrmann

Freiberufliche Trainerin und Beraterin für Software Engineering mit über 20 Jahren Berufserfahrung, mit Schwerpunkt auf Requirements Engineering, Autorin von mehr als 100 Fachpublikationen.

herrmann@herrmann-ehrlich.de

Self Designing Teams

Wie man motivierte, neue Teams etabliert

Wie teilt man ein großes Team auf zwei kleinere Teams auf? Diese Frage habe ich mir vor einem Jahr zum ersten Mal gestellt und habe dabei schon begonnen, zwei Listen zu schreiben. Aber wie kann man sicherstellen, dass die verschiedenen Fähigkeiten der Mitglieder so verteilt sind, dass beide Teams wieder bestmögliche Resultate erzielen können? Die Geschichte eines geglückten Experiments!

Teamgröße

Wer kennt sie nicht, die berühmte 7 ± 2 Regel? Werden Teams zu groß, kommt es zu langen Scrum Events, Scrum Master und PO sind überlastet und es bilden sich Subgruppen. Eine Teamteilung ist aus mehreren Gründen ein sensibles Thema, denn sie hat Auswirkungen auf soziale und fachliche Aspekte.

Agile Teams in einem Train nach dem ScaledAgile-Framework sollten möglichst alle Fähigkeiten (z.B. Frontend, Backend, Test…) mitbringen, damit sie Features im Team eigenständig definieren, bearbeiten und testen können. In der Auseinandersetzung mit dem Thema und auf dem Weg die beste Lösung zu finden, habe ich von meinem Mentor eine Empfehlung bekommen: „Überlege dir doch mal, ob du nicht ‚self designing teams' ausprobieren möchtest"…

Das Experiment

Warum eigentlich nicht? Beflügelt vom Kulturwandel im letzten Jahr kam ich zur Überzeugung, dass sich die Mitarbeiter selbst viel besser aufteilen können, als ich es kann, denn:

- Sie kennen sich und ihre Fähigkeiten besser
- Sie werden viel motivierter sein, in den Teams erfolgreich zu arbeiten, die sie sich selbst ausgesucht haben

Natürlich müssen dafür die Rahmenbedingungen stimmen: Für zwei Teams braucht es zwei Product Owner, zwei Scrum Master, Tester und Entwickler in richtigem Verhältnis – damit am Ende zwei leistungsfähige Teams mit 7 ± 2 Personen entstehen können.

Inspiriert von einem Artikel aus dem Jahr 2013, der von der Scrum Alliance veröffentlicht wurde, habe ich alle Mitglieder der zukünftigen Teams in einen Raum mit Flipchart und interaktiver Moderationswand eingeladen. Das persönliche Zusammentreffen ist entscheidend für diesen gruppendynamischen Prozess.

Die Agenda für ca. drei Stunden sieht wie folgt aus:

- Erklärungen zum Ablauf und Hintergründe
- Vorstellung der Mitarbeiter
- Brainstorming: Wie sieht ein optimales Team aus?
- Erste Aufteilung in Teams
- Gemeinsames Review
- Zweite Aufteilung in Teams
- PAUSE
- Gemeinsames Review mit Features
- Dritte Aufteilung in Teams
- Gemeinsames Review mit SWOT-Analyse
- Festlegen von Teamnamen
- Retrospektive
- Abschluss & Dankeschön

Die Durchführung

Die Hintergründe wurden am Anfang des Artikels bereits beleuchtet, zusätzliche Infos wie z. B. strategische Ausrichtung der Firma, Zielsetzung des Wachstums helfen den Mitarbeitern, sich mit ihrer Zukunft zu identifizieren und diese proaktiv mitzugestalten. Wenn sich die Mitarbeiter noch nicht so gut kennen, können folgende Fragen das Eis brechen:

1. Was begeistert dich an deiner Arbeit?
2. Welche Sprachen sprichst du?
3. Welchen Hintergrund hast du, was hast du früher gemacht?
4. Was sind deine Stärken – bei welchen Dingen kannst du anderen weiterhelfen?
5. Was wünscht du dir für deine Arbeit in deinem Team?

Entscheidend für den Erfolg ist, dass alle Beteiligten verstehen, wohin sie wollen – nämlich gemeinsam schlagkräftige Teams bilden. Für eine gemeinsame Ausrichtung ist ein Brainstorming optimal.

Dann kann die erste Aufteilung in zwei Teams beginnen – warum „erste"? In drei Runden können verschiedene Teamkonstellationen ausprobiert werden (Aufteilung in Teams) und diese dann mit verschiedenen Aufgaben validiert werden (Gemeinsames Review). Nach einer ersten Runde des „persönlichen Beschnupperns" beginnt eine Diskussion über die Fähigkeiten des Teams.

Manchmal zeigt sich schon hier, dass in der nächsten Runde eine andere Aufteilung sinnvoll ist.

Unterstützt werden die neuen Teams dann nach der zweiten Aufteilung mit Features aus dem Program Backlog, anhand denen die Teams überprüfen können, ob es in ihrem Team alle Fähigkeiten gibt, um die Aufgabe erfolgreich abschließen zu können. Die Pause ist hier Teil des Programms, denn das gibt den Mitarbeitern die Möglichkeit, Abstand zu gewinnen und sich ungezwungen auszutauschen.

Oft gibt es dann in der dritten Runde keine Änderungen mehr. Dann hat das Team die Möglichkeit, eine Standortbestimmung durchzuführen. Dazu eignet sich eine SWOT-Analyse (Stengths, Weaknesses, Opportunities, Threats), in der die Teams sich darüber austauschen, wo sie besonders gut sind, welche Schwächen, Chancen oder Bedrohungen es gibt. Daraus können sich dann Action Items ableiten, z. B. Themen, zu denen die Teams einen Know-How Austausch starten wollen, um in ihrer neuen Konstellation effizient arbeiten zu können.

Zu guter Letzt werden noch neue Teamnamen gewählt! Da zeigte sich, dass es vorher wirklich EIN Team war: gemeinsam wurden zusammenpassende Namen gesucht, um auch länger noch in Erinnerung zu behalten, dass wir doch alle zusammengehören. Nach kurzem Brainstorming präsentierten sich: Asterix und Obelix!

Das Ergebnis

Die kurze Retrospektive zeigte, dass die Mitarbeiter es sehr positiv aufnahmen, dass sie selbst die Entscheidung treffen durften. Was sie verändern würden, war nur, dass sie neue Mitarbeiter schon gerne vorher kennengelernt hätten. Nach der ersten erfolgreichen Teamteilung in zwei hoch performante Teams durfte ich einige Monate später eine zweite Teamteilung begleiten. Hat sich auch hier die Zeit ausgezahlt?

Auf jeden Fall, denn das erste Team-Building ist immer inklusive!

Bibliography

How to Form Teams in Large-Scale Scrum? A Story of Self-Designing Teams, Scrum Alliance, https://www.scrumalliance.org/community/articles/2013/2013-april/how-to-form-teams-in-large-scale-scrum-a-story-of

Scaled Agile Framework (SAFe), Scaled Agile, http://www.scaledagileframework.com/

Author

DI Tobias Kazmierczak

Tobias Kazmierczak ist Release Train Engineer (RTE) bei Eurofunk Kappacher. Er ist begeistert von Lean Agile Leadership, Selbstorganisation und Eigenverantwortung.

tkazmierczak@eurofunk.com

Software Quality for Heterogeneous Software Development

Software Development at ams AG

Introduction of software quality processes and software development processes in a hardware driven company is a tricky and lengthy undertaking. Software development is not established by decision in such companies. It simply happens. Moreover, if things are going well, software development grows and grows. However, eventually there comes the point in time where software development and software deliverables become mission critical for projects. Then, people start asking for quality control for an extremely heterogeneous set of teams and development workflows.

This article describes how ams AG has decided to resolve the explained quality management challenge. Instead of defining one process, we have decided to introduce Software Quality Requirements that tell the software teams what to do, without mandating how to do.

ams AG is a semiconductor company. Even though software development becomes more and more important for our products and our company, ams AG could not be more of a hardware focused company. Just to give an impression, by end of 2016, only approximately 2% of employees are involved in software development.

The low number of software related employees did not yet require installing a software engineering organization. Instead, software development is organized across business lines at ams. In many cases software development is considered as legwork to hardware development projects. However, this integration into the business lines allows very close collaboration with hardware teams, and very specialized software deployment.

ams AG develops software for many different purposes such as product demonstrators, firmware, hardware drivers, algorithms, production tools, verification tools and much more. Consequently, software development follows very different strategies. Just to give illustrative examples, testing of embedded firmware (mainly stored in ROM) needs to be done differently than testing of graphical user interfaces (GUIs), which are used as product demonstrators.

Yet another differentiating factor of ams AG is that the company is growing very rapidly in an inorganic way. Acquisitions have added almost a dozen of different software development workflows to the anyway heterogeneous development landscape in the last couple of months.

Overall, this makes a one-fits-all software development process unlikely to work out. It either would be detailed but would cause an immense learning curve and very limited acceptance. On the other hand, it would be a process with very high abstraction level to give the needed flexibility but would be hardly helpful.

ams AG has decided to accept the fact of heterogeneous workflows. Instead of defining a process that all of the software teams have to follow, a set of rules have been defined which need to be followed. They are called Software Quality Requirements. How software teams follow the requirements, is their choice. The following sections describe the framework and summarizes results of audits that have been performed to check how well projects follow the Software Quality Requirements.

Software Quality Requirements and Software Quality Levels

ams AG has defined 122 Software Quality Requirements that are arranged in the groups shown below. The names of the groups should be self-explanatory. Some groups are partly overlapping. For example, doing a release documentation could be in Releasing (SQ_REL) or in Documentation (SQ_DOC). However, each Software Quality Requirement exists only in one group.

- Project Management (SQ_PM)
- Requirements Management (SQ_RM)
- Software Coding (SQ_CODE)
- Configuration Management (SQ_CM)
- Software Testing (SQ_TEST)
- Documentation (SQ_DOC)
- Bug Tracking (SQ_BUG)
- Releasing (SQ_REL)

Each Software Quality Requirement has a unique identifier and is written in the form of a requirement. As can be seen in some of the examples below, the Software Quality Requirements also define the responsible role.

SQ_PM.ACT: The software project shall have one responsible Software Project Manager.

SQ_CODE.REV: The Software Project Manager shall take care that the team performs peer reviews on the source code before a release is done.

SQ_CM.TOOL: The Software Configuration Manager shall select a Configuration Management tool to implement version control for all project related data.

SQ_TEST.TR: The Software Test Engineer shall create a Software Test Report that summarizes the testing results of a Software Release.

To limit the length of this article to a meaningful number of pages, we do not give the complete list of all Software Quality Requirements. However, from the examples the reader should be able to understand the concept and scope of the requirements.

Obviously, not all of the 122 Software Quality Requirements will be applicable for every project. To be scalable, ams AG has defined four Software Quality Levels (SQ1 to SQ4). Each of them comes with a description and the list of applicable Software Quality Requirements.

SQ1 is meant for experimental software projects, SQ2 for demonstrators, SQ3 for products and SQ4 for high volume and/or high-risk products. This allows a software project to select the appropriate Software Quality Requirements without over-stretching the quality effort. SQ1 includes 19 Software Quality Requirements, SQ2 includes 38, SQ3 includes 111 and SQ4 includes all 122 Software Quality Requirements.

The first challenge a project has to take is to identify the appropriate Software Quality Level. The definition of the levels sound trivial, but in practice the choice is a bit trickier. Many projects try to start with SQ4, which was meant to be used only for the top-most critical projects in ams AG. On the other hand, some software deliverables are set somewhere between the Software Quality Levels. For example, reference code that is sent to the customer is not a product and thus not SQ3. However, many customers take the reference code as it is and use it in their products. Thus for many projects it has to be considered as product.

Audit Results

ams AG performs Software Quality Audits on software projects. In such an audit, a Software Quality Manager reviews a team's development workflow. The goal is to identify potential room for improvement or optimization. This result is a list of areas to be considered for improvements, a number of fulfilled Software Quality Requirements (the Pass Rate) and a number of incompliances (the Fail Rate).

Figure 1 shows the results of SQ2 projects audited in 2016. The groups of Software Quality Requirements are sorted by their Pass Rate. The Pass Rate, in green, indicates the SW Quality Requirements that have been fulfilled. The red bars, show the requirements, that have not been fulfilled – the Fail Rate. Grey bars indicate not applicable requirements or inherited fails. For example, if a project plan would not exist, the content of the plan could not be audited.

As can be seen, topics that are often neglected by typical software engineers (such as testing and documentation) have a rather low rating. Also, topics that require a minimum of formalisms such as code reviews (as part of SQ_CODE) or elaboration of requirements (SQ_REQ) can be improved.

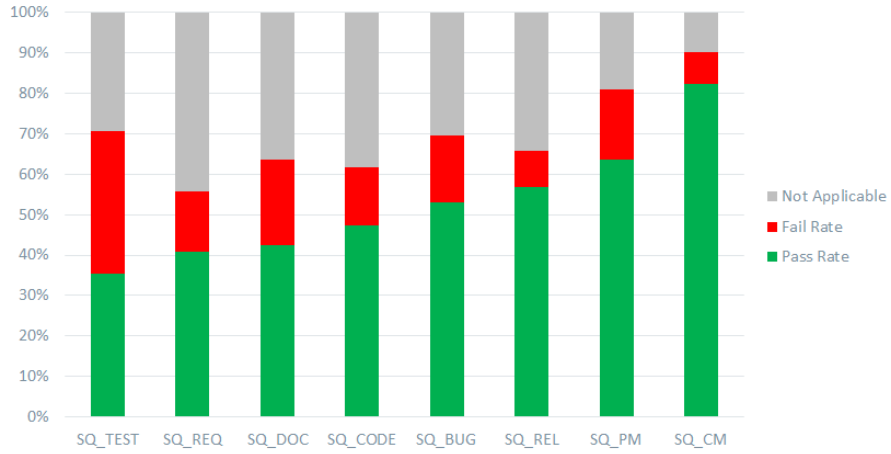


Figure 1, Audit Results 2016

Findings are of course always discussed first with the software development teams. The Software Quality Manager shows examples how other projects have implemented the Software Quality Requirement and proposes solutions.

However, it is the responsibility of the software team to improve their workflow.

Topics with low Pass Rates across multiple projects are identified as Focus Areas to be improved on corporate level. Improvement activities on Focus Areas are driven across business lines. For 2015, this was bug tracking with an overall Pass Rate of <10%. As can be seen in Figure 1 this was improved in 2016 to >50%. Focus areas for 2016 are software testing, requirements management and code reviews. Trainings to create awareness, to write test specifications and how to setup automated tests have been held. A requirements management flow was defined. Code review tools have been rolled out. The target for 2016 was to improve code reviews and testing to a pass rate of >60%. The results are not yet available but the intermediate rates look promising to reach the target.

Typical Findings of Audits

In this section, we want to discuss some typical challenges of our software teams. This also includes approaches that have been selected by projects to improve.

- **Lack of systematic testing (SQ_TEST)**

For many engineers software testing is considered as "necessary evil". It is done, but effort is typically reduced to a bare minimum. Questions about the test coverage of a software release can very often not be answered easily.

The solution to that problem is twofold. First, engineers have to be found that enjoy the "destructive nature of testing". To do that, ams AG invests in trainings and employing dedicated software test engineers.

The second solution is to increase the fun part of testing by providing the right tools and templates. These tools cover writing of test specifications, mapping tests to requirements, automatic test execution and report creation. Tools that have been successfully rolled out for many of the SW projects are, for example, Jenkins, NUnit and Gcov [1].

- **Too detailed requirements management (SQ_REQ)**

Requirements Management is a rather new topic in ams AG. Many teams have identified that writing down requirements will ease their daily work by reducing the number of Change Requests and delivering more accurate effort estimations. Most of the projects that start with requirements management start on a too high level of details. The optimistic intention to link all the requirements to test cases to have an optimal conclusion on test results fails due to underestimated effort and complexity of the approach.

On short term, we are trying to reduce requirements management to the bare minimum that still helps the project to define and meet expectations. Simple list of deliverables, supported platforms, feature lists or use cases server very well for that purpose.

On mid-term ams AG is investing in requirements management trainings and tool support. Many tools have been evaluated and one tool has been identified that will support and improve ams way of working significantly over the next docents of months.

- **Unknown External Components (SQ_DOC)**

While documentation of source code seems to be a natural thing for software engineers, documentation of external components is not. As external components ams AG understands everything that belongs to (or is needed for) the developed software but is not stored in the project repository. This is for example, external intellectual properties and tools such as compilers. To ensure reproducibility of releases and reported issues, developers need to remember external components long time after a release was done. A simple document listing the external components and their versions, helps remembering the details in case of need.

- **Lack of Code Reviews (SQ_CODE)**

Code reviews are a very good mechanism for finding issues that are very difficult to test and to share know how within the team. However, in order to take advantages from these benefits, the team needs to do code reviews on a frequent basis. Doing "a lot of reviews", however, is still understood as overhead.

To reduce the overhead ams AG trains projects in using review tools such as ReviewBoard and Gerrit. After an initial phase of getting used to the tool, the teams see the benefits of having immediate feedback on written code. Instead of doing lengthily reviews, the work is split in small pieces. This reduces the subjective feeling that the review is overhead.

- **Unmanaged bug tracking (SQ_BUG)**

Documenting all the reported software bugs is a very important step. Managing the found bugs is the next natural step, which needs a defined workflow. This includes analysis, prioritization, effort estimation and incorporation in release planning.

The Software Quality Requirements mentioned before ask software teams to define a workflow for issue management. On corporate level, we support the project teams to define an appropriate workflow. The main challenge is to keep the workflow as lightweight and practicable as possible. Teams tend to over engineer such a workflow to have a "perfect solution" which turns out to be too unhandy to be used on a daily basis.

- **Unreleased Software (SQ_REL)**

Projects typically have a very high quality demand on releases. However, not everything that is released to somebody is considered as software release. Consequently, unreleased software with reduced quality level is shared with other teams or colleagues.

Projects need to understand that software is not only released if it is shared with a customer. Software that is shared with a different team, a sub-team of the project (e.g. the test team, or the hardware team), or even marketing is also "released" – and thus deserves to think about the needed quality level.

Obviously not every type of a release needs to fulfill the same quality standards. A release that is sent to a customer needs certainty to be tested. A release that is shared with the test team might require passing some sanity checks but is not yet tested. However, it requires a tag in the repository and a unique identifier. By defining types and maturities of releases, the teams can define releasing effort as well as the intended purpose of a release.

- **Project documentation instead of project management (SQ_PM)**

As software development is still very small in ams AG, we do not have many dedicated software project managers. Consequently, software engineers are appointed to take over project management. Due to the lack of management experience and training, such engineers tend to confuse project documentation with project management. Instead of focusing on the future of the project the project documents what was done or achieved.

The usage of lightweight project management tools such as trackers and planning folders in CollabNet TeamForge have been implemented in many projects that are too small for a "full blown project plan". This helps the project managers to have a certain level of control while having a minimum management overhead. The concept scales very well up to medium size software projects, but is insufficient when more comprehensive resource planning is needed.

- **File and document versioning (SQ_CM)**

The idea that version control with tools such as svn, git and co is a trivial task is simply wrong. Technically, the problem to handle versions of files is resolved since a long time. However practically, a workflow that matches the teams and the project needs has to be defined carefully. In addition, the choice of the appropriate tool needs to be done careful. Users who still add dates or version numbers to file names (on network drives) need to be trained. Unfortunately, such pseudo versioning workflows have been deeply burned into the mindset of many engineers, managers and IT infrastructures.

ams AG invests a lot of time and money to resolve this challenge. CollabNet TeamForge was rolled out, supporting svn and git – replacing many local (and inappropriate) solutions. Many trainings and project workshops are held to select a useful tool and to define appropriate version control workflows.

Outlook and Conclusion

Defining Software Quality Requirements has increased acceptance on SW Quality work significantly. Requirements help to understand the purpose of certain software quality topics. Software teams understand where they have to improve while having the choice to select an implementation that works best under their circumstances. The concept of such rules is certainty not new; however, there are still many company processes and standards that define detailed workflows instead.

Defining and rolling out the Software Quality Requirements took approximately one year. Improvements have been visible almost immediately after starting the quality audits.

In parallel to the project quality work, ams AG also works continuously on an improvement of the software quality processes. Feedback from projects is taken into account in new versions of the processes. Software Quality Requirements are re-discussed and, in case of need, relaxed or removed. This allows that not only the projects but also the processes become more mature over time.

Bibliography

[1] Puri-Jobi, Stephan, Test automation for NFC ICs using Jenkins and NUnit, 2105 IEEE Eighth International Conference on Software Testing, 2015

Autor

DI Dr. Johannes Loinig
Software Quality Manager
ams AG
johannes.loinig@ams.com

Immer kürzere Testphasen?

Mit *Ticket Coverage* verhindern, dass wichtige Features ungetestet bleiben

In vielen Systemen werden die Release-Zyklen immer kürzer. Daher steht auch immer weniger Zeit für dedizierte Testphasen zur Verfügung. Viele Teams führen deshalb Tests parallel zur Entwicklung durch. Oft finden Entwicklung und Test dabei auf parallelen Branches und in verschiedenen Umgebungen statt.

Dadurch wird es immer schwieriger im Blick zu behalten, welche Tickets (z.B. User Stories, Change Requests, Bug Reports, etc.) wie gründlich getestet wurden. Und nicht zuletzt, wo auf Grund von nachfolgenden Code-Änderungen nochmal getestet werden müsste. Dadurch steigt die Gefahr, dass wichtige Funktionalität ungetestet in Produktion gelangt.

In diesem Paper stellen wir *Ticket Coverage* als Maß von Test Coverage auf der Ebene von Tickets vor. Dadurch kann pro Ticket ermittelt werden, welche Bereiche nicht (oder nicht ausreichend) getestet wurden. Wir stellen die Ergebnisse von zwei empirischen Studien vor, in denen Ticket Coverage bei manuellen Systemtests und bei Entwicklertests zum Einsatz kam. In beiden Studien konnten dadurch relevante Testlücken identifiziert werden.

Qualitätssicherung von Tests

Bei großen Systemen stehen einem als Tester oder Testmanager selten genug Ressourcen zur Verfügung, um die gesamte Funktionalität vollständig zu testen. In der Praxis müssen wir uns in jeder Testphase daher notwendigerweise auf einen Ausschnitt aller möglichen Tests beschränken.

Die Frage, wie man diese auszuführenden Tests möglichst sinnvoll auswählt, beschäftigt seit Jahrzehnten ein eigenes Forschungsgebiet. Ein zentrales Ergebnis der Arbeiten der letzten Jahre ist, dass typischerweise in denjenigen Bereichen die meisten Fehler auftreten, in denen in letzter Zeit (z.B. seit dem letzten Release) am meisten geändert wurde [2]. Daher sollte im Rahmen der Qualitätssicherung der Testaktivitäten überprüft werden, ob alle relevanten Änderungen auch getestet wurden. Diese Aufgabe wird erfahrungsgemäß um so wichtiger, je größer Systeme (und damit je unübersichtlicher Änderungen und Testaktivitäten) werden.

Um Tester und Testmanager bei dieser Überprüfung zu unterstützen, haben wir in den letzten Jahren die Test-Gap-Analyse entwickelt, die ermittelt, welche Änderungen noch ungetestet sind. Wir haben Test-Gap-Analyse u.a. auf den Software Quality Days 2017 vorgestellt [3]. Test-Gap-Analyse wird inzwischen von vielen Firmen eingesetzt.

Ein typisches Ergebnis von Test-Gap-Analyse ist in Abbildung 1 dargestellt. Jedes weiß umrandete Rechteck beschreibt eine Komponente im System under Test, jedes darin enthaltene kleinere schwarz umrandete Rechteck eine Methode. Der Flächeninhalt der Rechtecke korrespondiert mit der Größe der dazugehörigen Komponenten/Methoden in Lines of Code. Methoden, die sich seit dem letzten Release nicht verändert haben, werden grau dargestellt, veränderte farbig. Nur wenn die veränderten Methoden grün dargestellt werden, sind sie im Test durchlaufen worden. Die roten und orangenen Bereiche zeigen Methoden, die seit dem letzten Release neu entwickelt oder verändert, aber noch nicht getestet wurden.

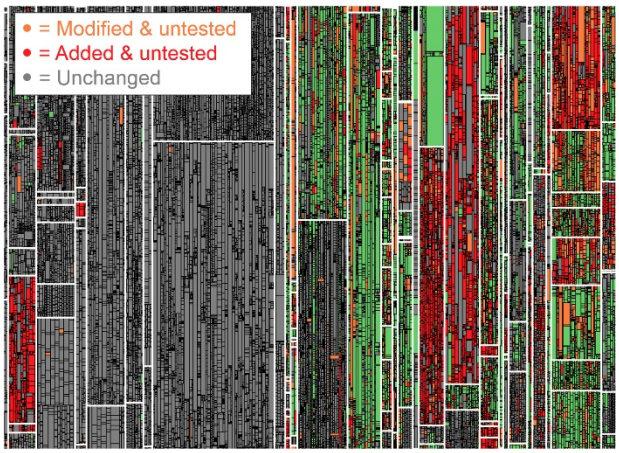


Abbildung 1 - Getestete und ungetestete Änderungen

Anforderungen durch kürzere Release-Zyklen

Test-Gap-Analyse betrachtet alle Änderungen und alle Testaktivitäten, die seit einem Referenzzeitpunkt (z.B. dem letzten Release) durchgeführt wurden. Diese Betrachtung bewährt sich vor allem in Projekten, die (wie in Abbildung 2 dargestellt) vor einem Release eine ausgedehnte Testphase durchführen, in der das gesamte Release getestet wird.

Abbildung 2 - Aufteilung von Entwicklungsphasen (Dev) und Testphasen (Test) bei wenigen großen Releases pro Jahr.

Abbildung 3 - Aufteilung von Entwicklungs- und Testphasen bei vielen kleinen Releases pro Jahr.

Agile Entwicklungsmethoden drängen seit Jahrzehnten auf kürzere Releasezyklen, um neue Funktionalität schneller zum Anwender zu bringen. In den letzten Jahren hat, in unserer Beobachtung, dieser Trend in den meisten Entwicklungsteams die Releasezyklen stark verkürzt. Mit der Verkürzung der Releasezyklen geht die Verkürzung der Testphasen einher.

Oft ist es nicht mehr möglich, vor einem Release eine ausgedehnte Testphase durchzuführen. Tests müssen stattdessen iterationsbegleitend durchgeführt werden, wie in Abbildung 3 dargestellt[1].

Tickets im Fokus der Testplanung

Wir beobachten, dass in vielen Teams, um Tests iterationsbegleitend steuern zu können, immer stärker einzelne *Tickets* im Fokus der Testplanung stehen. Testern werden dabei dedizierte Tickets zugewiesen, die sie zeitnah nach Abschluss der zugehörigen Entwicklungsarbeiten testen. Dadurch gelingt es, Tests verschränkt zur Entwicklung auszuführen, so dass dedizierte Testphasen entfallen oder kürzer ausfallen können.

Unter *Ticket* verstehen wir hierbei die Beschreibung einer geplanten Änderung am System, die in einem Change Management System[2] verwaltet wird. Je nach Art und Quelle der Änderung wird in Teams oft zwischen unterschiedlichen Ticket-Klassen unterschieden, wie bspw. User Stories, Change Requests, Bug Reports oder allgemein Issues. In diesem Paper beziehen wir all diese Ticket-Klassen im Begriff *Ticket* mit ein.

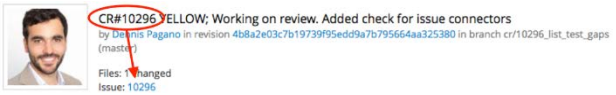


Abbildung 4 - Commit, bei dem die ID des Tickets angegeben wurde (10296), auf das sich die durchgeführten Änderungen beziehen.

Problemstellung

Wir haben in den letzten Jahren mehrere Teams begleitet, die derartig vorgehen. Dabei haben wir die Beobachtung gemacht, dass es für Tester und Test Manager sehr schwierig ist zu beurteilen, wie gründlich Tickets im Rahmen der ausgeführten Tests getestet wurden:

- Als Test-Input werden oft die Beschreibungen der Tickets verwendet. Die sind allerdings oft, gerade im Vergleich zu strukturierten Testfällen, deutlich informationsärmer, insbesondere in Hinsicht auf Sonderfälle.
- Da immer mehr Test-Arten (Unit-Tests, skriptierte UI-Tests, manuelle Tests, Systemtests, explorative Tests) und Test-Stufen (Smoke-Tests, Akzeptanztests, ...) zum Einsatz kommen, wird es immer schwieriger den Überblick zu behalten, was eigentlich wo getestet, und vor allem wie gründlich getestet wird.
- Die Entwicklung findet immer häufiger auf parallelen Branches statt. Oft fließen durch den Merge eines Feature Branches sehr plötzlich sehr große Mengen an Änderungen in den Release Branch ein. Da häufig unterschiedliche Branches in unterschiedlichen Teststufen zur Ausführung kommen, wird es immer schwieriger einen Überblick darüber zu behalten, welche Änderungen in welchen Branches schon getestet wurden.

Durch diese Faktoren steigt die Gefahr, dass auch zentrale, wichtige Tickets unzureichend getestet werden und zu Feldfehlern in kritischer Funktionalität führen. Testmanager und Tester brauchen daher ein Analysewerkzeug, um während der iterationsbegleitenden Tests einfach herausfinden zu können, welche Tickets nicht oder nicht ausreichend getestet wurden.

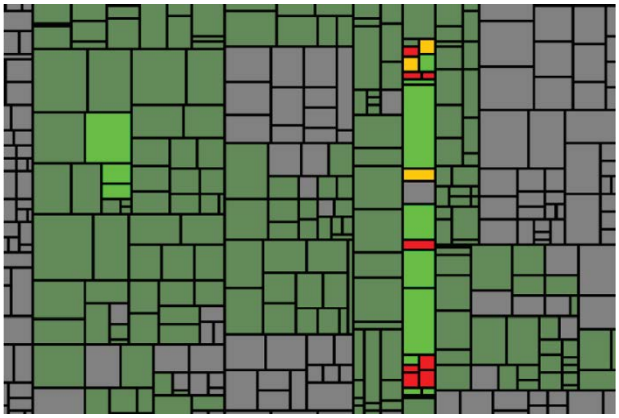
Lösungsansatz

Als Lösungsansatz schlagen wir in diesem Paper *Ticket Coverage* vor. Ticket Coverage drückt aus, welcher Anteil des Codes, der im Zuge der Implementierung eines Tickets angefasst (neu hinzugefügt oder geändert) wurde, im Test zur Ausführung kam.

Ticket-Coverage wird auf Basis von Test-Gap-Analyse berechnet. Im Unterschied zur Test-Gap-Analyse werden Informationen jedoch nicht für das gesamte System, sondern für einzelne Tickets ausgewertet und dargestellt.

[1] Um Tests iterationsbegleitend durchzuführen, können prinzipiell die unterschiedlichsten Test-Ansätze eingesetzt werden. Von automatisierten (z.B. Keyword-Driven), klassischen manuellen Tests die iterationsbegleitend ausgeführt werden bis hin zu explorativen Tests auf Basis der Beschreibung der User Story. Die Vor- und Nachteile der jeweiligen Test-Ansätze für iterationsbegleitendes Testen sind außerhalb des Fokus dieses Papers.

[2] z.B. Jira, Bugzilla, Redmine, TFS, ...



Ticket Coverage wird in folgenden Schritten berechnet:

6. Im ersten Schritt wird ermittelt, welche Methoden im Code im Rahmen eines Tickets angefasst wurden. Wie in Abbildung 4 dargestellt, geben Entwickler bei Änderungen im Code an, auf welches Ticket sich eine Änderung bezieht. Durch Analyse der Daten im Ticket-Management-System und der Versionshistorie kann dadurch für jedes Ticket die Menge der Methoden erhoben werden, die im Rahmen der Umsetzung des Tickets angefasst wurden.
7. Im zweiten Schritt werden Test-Coverage-Daten aus verschiedenen Testläufen erhoben, um zu ermitteln, welche Methoden im Test zur Ausführung kamen.
8. Im dritten Schritt wird für jedes Ticket ermittelt, welche der zum Ticket zugehörigen Methoden im Test zur Ausführung kamen.

Abbildung 5 - Ticket Coverage für ein Ticket: 12 geänderte Methoden wurden getestet (hellgrün dargestellt), 11 geänderte Methoden nicht (rot und orange dargestellt). Außerdem wurden im Test eine Reihe von Methoden durchlaufen, die nicht zum Ticket gehören (dunkelgrün dargestellt). Graue Methoden repräsentieren Code, der weder verändert, noch ausgeführt wurde.

Ticket Coverage kann in einer Treemap visualisiert werden, wie in Abbildung 5 dargestellt. Ticket Coverage lässt sich auch als Anteil der getesteten Methoden an allen im Kontext des Ticket bearbeiteten Methoden ausdrücken. Im Beispiel aus Abbildung 5 ergibt sich daher eine Ticket Coverage von 52% (12 getestete Methoden von 23 angefassten Methoden).

Grenzen von Ticket Coverage

Wie jede statische oder dynamische Analysetechnik hat auch Ticket Coverage Grenzen. Die Kenntnis dieser Grenzen ist entscheidend, um Analyseergebnisse korrekt zu interpretieren. Dabei sollten die folgenden beiden Punkte Beachtung finden:

Coverage≠Getestet: Test Coverage misst, welcher Code im Test ausgeführt wurde, nicht wie gründlich er tatsächlich getestet wurde. Da Test Coverage auch für die Berechnung der Ticket Coverage herangezogen wird, gilt das auch hier. Konkret bedeutet 50% Ticket Coverage, dass jede zweite Methode des Tickets im Test zur Ausführung kam. Die Ticket Coverage erlaubt keine Aussage darüber, wie gründlich diese Methoden getestet wurden.

Seiteneffekte: Während sich einige Methoden eindeutig zu Tickets zuordnen lassen, werden andere Methoden im Kontext der Implementierung mehrerer Tickets angefasst. Wenn eine solche Methode beim Test eines Tickets durchlaufen wird, gilt sie ggf. auch für andere Tickets als getestet. Es ist aber möglich, dass der Test anderer Tickets Fehler in dieser Methode aufdecken würde. Selbst eine Ticket Coverage von 100% gibt daher keine Garantie, dass im betroffenen Code keine Fehler mehr enthalten sind.

Für beide Fälle gilt allerdings: Wird eine Methode als ungetestet erkannt, haben für sie überhaupt keine Tests stattgefunden. Keiner der möglicherweise enthaltenen Fehler kann daher gefunden worden sein. Wir sehen Ticket Coverage daher als Werkzeug, um Lücken im Test zu erkennen, nicht als Werkzeug um ohne weitere Analyse zu entscheiden, wann genug getestet wurde.

Was bringt Ticket Coverage in der Praxis?

Um besser einschätzen zu können, wie gut Ticket Coverage in der Praxis funktioniert und wo ihre Grenzen liegen, haben wir eine Reihe von wissenschaftlichen Studien [1][5] durchgeführt.

Als Studienobjekt haben wir das von uns entwickelte System *Teamscale*[3] gewählt, weil wir hierfür die Ergebnisse der Studien besser beurteilen können, als für fremden Code. Teamscale ist eine inkrementelle Software-Qualitätsanalysesuite, die von vielen namhaften Firmen eingesetzt wird. Der Quelltext umfasste zum Zeitpunkt der Studien ca. 650.000 Zeilen Java- bzw. JavaScript-Code.

Studie 1: Finden wir relevante Testlücken im strukturierten Test?

Nicht jede Testlücke ist automatisch relevant und muss geschlossen werden. Ist etwa der zugehörige Code (noch) nicht erreichbar oder sind die betroffenen Codestellen trivial, kann es je nach Kontext durchaus sinnvoll sein, die Testlücken nicht zu schließen und stattdessen die Ressourcen anderweitig zu verwenden. Die zentrale Frage unserer ersten Studie war daher, ob wir mit Hilfe von Ticket Coverage relevante Testlücken in einem typischen Setting mit strukturierten manuellen Tests finden.

Um diese Frage zu beantworten, haben wir zunächst zufällig 54 Tickets ausgewählt, die in den letzten 20 Monaten bearbeitet wurden. Auf Basis der Ticketbeschreibung haben wir anschließend je eine strukturierte, manuelle Testfallbeschreibung erstellt und diese vom jeweiligen Entwickler des Tickets validieren lassen. Dann haben wir diesen Testfall unter Zuhilfenahme eines Profilers ausgeführt, der die Code Coverage aufgezeichnet hat. Abschließend haben wir die Ticket Covera-

[3] http://www.teamscale.com

Bewertung	Erklärung	Häufigkeit	
kritisch: 20 (18,2%)	sollte ausgeführt werden	2	1,8%
	Testfall nicht ausreichend	18	16,4%
weniger kritisch: 56 (50,9%)	Refactoring oder nicht Ticket-relevant	54	49,1%
	Exception	1	0,9%
	überschriebene Methode	1	0,9%
nicht testbar: 3 (2,7%)	IDE plugin Code	1	0,9%
	Methode für Unit-Tests	2	1,8%
nicht relevant: 28 (25,5%)	einfacher Getter	12	10,9%
	triviale Methode	12	10,9%
	toString-Methode	4	3,6%
	keine Antwort	3	2,7%
	Σ	110	100%

Tabelle 1 - Bewertung der Testlücken durch die Entwickler.

ge wie oben beschrieben berechnet. Eine im Zuge des Tickets geänderte Methode haben wir dann als Testlücke gewertet, wenn diese bei der Ausführung des Testfalls überhaupt nicht durchlaufen wurde.

Nach der Erhebung der Ticket Coverage haben wir die Entwickler mit den gefundenen Testlücken konfrontiert und sie gebeten, diese hinsichtlich ihrer Relevanz zu beurteilen. Die Ergebnisse sind in Tabelle 1 dargestellt. Von den insgesamt 110 Testlücken wurden 20 (18,2%) als kritisch eingestuft. In unserer Studie haben wir mit Ticket Coverage also tatsächlich relevante Testlücken gefunden.

Ein Großteil der als nicht relevant eingestuften Testlücken sind Refactorings, also Semantik-erhaltende Code-Änderungen. Viele dieser Refactorings können prinzipiell automatisch erkannt und von der Berechnung der Ticket Coverage ausgeschlossen werden, um die Analyse präziser zu machen. Weitere Details zu Forschungsfragen, Study Design und Ergebnissen hierzu finden sich in [5].

Studie 2: Finden wir relevante Testlücken im explorativen Test?

Mit der ersten Studie haben wir eine Blackbox-Sicht auf das zu testende System eingenommen, wie sie ein Tester hat, der strukturierte manuelle Testfälle durchführt. In der zweiten Studie haben wir explorative Tests betrachtet, für die es typischerweise keine detaillierte Testfallbeschreibung gibt. Stattdessen wird bei explorativen Tests versucht, die entwickelte Funktionalität z.B. mit Hilfe der Beschreibung im zu testenden Ticket zu prüfen. Ein häufiges Problem hierbei ist jedoch, dass oft Sonderfälle durch den Entwickler umgesetzt wurden, von denen der Tester nichts weiß, da sie in der Beschreibung des Tickets nicht aufgeführt sind. In der zweiten Studie [1] haben wir daher untersucht, ob Ticket Coverage auch im explorativen Test relevante Testlücken aufzeigt.

Bei der Entwicklung von Teamscale kommt ein Peer-Review-Verfahren zum Einsatz. Der Reviewer führt dabei oft zunächst einen explorativen Test der umgesetzten User Story durch, um das beobachtete Verhalten zu bewerten. Uns hat in unserer Studie interessiert, ob wir mit Hilfe der Ticket Coverage Testlücken in solchen explorativen Tests aufdecken können, die im Rahmen der Reviews durchgeführt werden. Hierfür haben wir 4 Tickets zur näheren Analyse ausgewählt. Wir haben dann die jeweiligen Reviewer gebeten, explorative Tests durchzuführen und dabei die Ticket Coverage berechnet.

Im Vergleich zur ersten Studie haben wir hierbei Refactorings und triviale[4] Methoden automatisch ausgeschlossen. Außerdem haben wir Coverage zeilengenau berechnet (und nicht wie in der ersten Studie auf der Ebene von Methoden). Nach der Erhebung der Ticket Coverage haben wir dem jeweiligen Reviewer das Resultat des explorativen Tests, wie in Abbildung 6 gezeigt, präsentiert. Dargestellt ist pro Ticket eine Liste aller Methoden, die während der Entwicklung am Ticket verändert wurden oder neu hinzugekommen sind. Für jede Methode wird dargestellt, wie vollständig sie durch den explorativen Test abgedeckt wurde. Methoden, die als trivial erkannt wurden, sind ausgegraut. Diese Übersicht zeigt also aktuelle Testlücken für ein einzelnes Ticket feingranular auf.

Durch einen Klick auf eine einzelne Methode gelangt man zu einer weiteren Ansicht, auf der die Änderungen an der Methode zusammen mit der zeilengenauen Coverage zu sehen sind (vgl. Abbildung 7). Diese Darstellung erlaubt es, die Ursache der einzelnen Lücken zu identifizieren und somit zu beurteilen, ob es sich dabei um relevante Testlücken handelt.

Insgesamt waren 95 Methoden in den untersuchten Tickets geändert worden. Davon waren 30 nicht vollständig im Test durchlaufen worden, wiesen also einen Coverage-Wert von weniger als 100% auf. Bei der anschließenden Bewertung dieser Lücken durch die Reviewer wurden 23 Methoden (76,6%) als testenswert identifiziert. Die Analyse der Ticket Coverage führte also auch im Fall der explorativen Tests zur Aufdeckung von relevanten Testlücken.

[4] Triviale Methoden sind bspw. reine Getter und Setter. Die Kriterien zur Klassifikation von Methoden als zu trivial für den Test wurde im Rahmen der Studie durch die beteiligten Entwickler validiert. Details finden sich in [5].

Affected methods (22) Ticket Coverage: 86%

Class	Method	Line Coverage	Uncovered Lines	Change type
ProjectCreationService	retrieveProjectConfig	100%	0	changed
ProjectCreationService	fieldChangeRequiresReAnalysis	100%	0	added
ProjectService	elementUpdateQuery	100%	0	added
ProjectReanalysisService	process	100%	0	changed
ProjectCreator	refreshProject	100%	0	added
ProjectCreationService	nullOrToString	66%	1	added
ProjectCreationService	findConnectorByName	75%	1	added
ProjectCreationService	connectorRequiresReAnalysis	82%	3	added
ProjectCreationService	processPutRequest	84%	3	changed
ProjectCreationService	validateProjectConfiguration	80%	3	changed
ProjectCreationService	projectReAnalysisRequired	61%	5	added
ProjectCreationService	connectorsRequireReAnalysis	50%	7	added
ConfigOptionDescriptorBase	ConfigOptionDescriptorBase	100%	0	changed
NamingConventionConfiguration	NamingRegexOption	100%	0	changed
ProjectService	ProjectUpdateResult	100%	0	added

Abbildung 6 - Detailansicht der Ticket Coverage, in der alle in der Umsetzung des Tickets bearbeiteten Methoden mit ihrer Coverage aufgelistet sind.

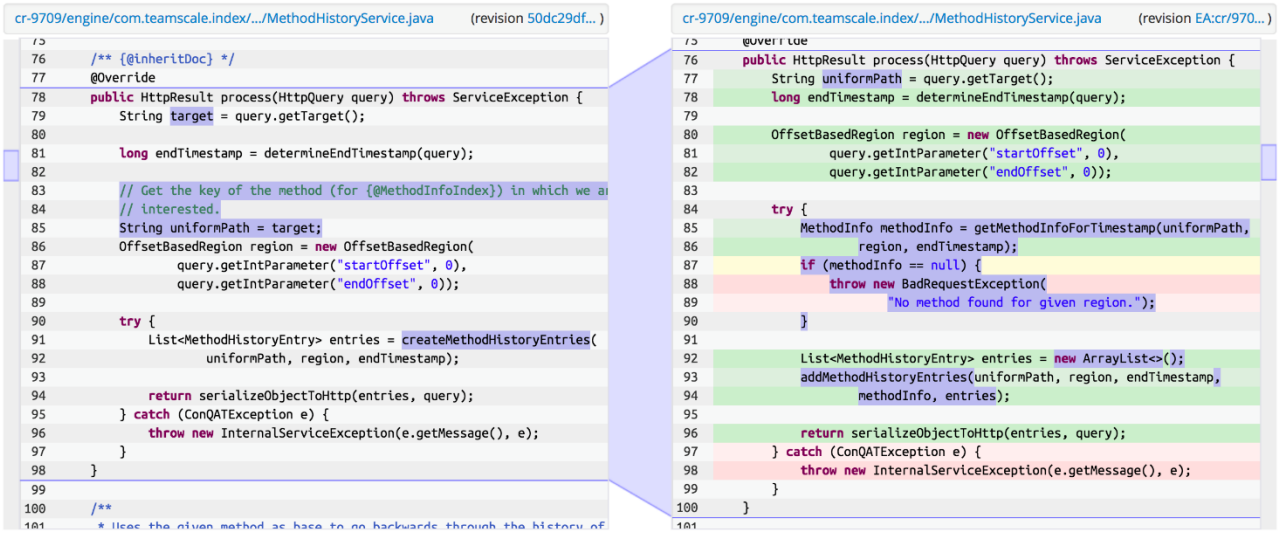


Abbildung 7 - Detailansicht für eine einzelne Methode. Änderungen im Zuge des Tickets sind aufeinander abgebildet, Coverage ist farbig dargestellt.

Interpretation

Wir haben Ticket Coverage für zwei unterschiedliche Testarten im Rahmen der Entwicklungs- und Testprozesse des kommerziellen Qualitätsanalysewerkzeugs Teamscale eingesetzt. Zum einen im strukturierten, manuellen Test, wo wir die Sicht eines Testers eingenommen haben, zum anderen im explorativen Test, aus der Sicht eines Code Peer Reviewers.

In beiden Fällen sind wir zu ähnlichen Ergebnissen gekommen: Ticket Coverage hilft zu erkennen, welche wichtigen Tickets nicht ausreichend getestet werden. Der Fokus auf ein Ticket erlaubt es dabei, sich auf die Lücken zu konzentrieren, die zu kritischer Funktionalität gehören. Dies ermöglicht es, auch bei iterationsbegleitender Testdurchführung im Blick zu behalten, ob Änderungen an kritischer Fachlichkeit gründlich genug getestet wurden und rechtzeitig nachzusteuern.

Aufgrund der Ergebnisse unserer Studien halten wir es für wahrscheinlich, dass der Einsatz von Ticket Coverage auch bei anderen Testarten und Teststufen relevante Testlücken zu Tage fördert. Dies passt auch zu den Erfahrungen, die wir über unsere wissenschaftlichen Aktivitäten hinaus tagtäglich als Berater in Kundenprojekten gewinnen.

Wie kann ich Ticket Coverage einsetzen?

Die Messung von Ticket Coverage kann durch unsere inkrementelle Software-Qualitätsanalysesuite Teamscale durchgeführt werden. Teamscale unterstützt eine Vielzahl von Programmiersprachen (u.a. Java, C#, VB.NET, ABAP, Python, C/C++ uvm.), Versionskontrollsystemen (Git, SVN, TFS, SAP, uvm.) und Ticket-Systemen (Jira, TFS, Redmine, uvm.).

Wir haben diese Analysen in den letzten Jahren bei vielen Firmen aus verschiedensten Domänen eingeführt. Fast immer haben wir etwas andere Konstellationen aus Programmiersprachen, Technologien, Infrastruktur, Testwerkzeugen und Testansätzen vorgefunden. Inzwischen haben wir daher viel darüber gelernt, wie man diese Analysen an neue Umgebungen anpasst. Wir freuen uns auf einen persönlichen Austausch, um Ihren konkreten Fall zu betrachten.

Wir haben unter www.testgap.io weiterführende Materialien zu Test-Gap-Analyse allgemein und Ticket Coverage im Speziellen zusammengestellt, u.a. Forschungsarbeiten, Blog-Einträge und Werkzeugunterstützung. Darüber hinaus freuen wir uns auch per Email über Fragen und Feedback (auch kritisches) zum Artikel oder zu Ticket Coverage allgemein.

Danksagung

Dieser Artikel baut auf Vorarbeiten und Ideen von Andreas Göb, Florian Dreier, Jakob Rott und Rainer Niedermayr auf, bei denen wir uns herzlich bedanken möchten.

Literatur

[1] **Florian Dreier, Elmar Juergens, and Andreas Goeb.** *Test Accompanying Calculation of Test Gaps for Java Applications.* Whitepaper, CQSE GmbH, 2017.

[2] **S. Eder, B. Hauptmann, M. Junker, E. Juergens, R. Vaas, and K. Prommer**. *Did we test our changes? assessing alignment between tests and development in practice.* In Proceedings of the Eighth International Workshop on Automation of Software Test (AST'13), 2013.

[3] **Elmar Juergens and Dennis Pagano**. *Haben wir das Richtige getestet? Erfahrungen mit Test-Gap-Analyse in der Praxis*. In Software Quality Days Conference Journal, 2017.

[4] **Nachiappan Nagappan and Thomas Ball**. *Use of relative code churn measures to predict system defect density*. In Proceedings of International Conference on Software Engineering (ICSE), 2005.

[5] **Jakob Rott, Rainer Niedermayr, Elmar Juergens, and Dennis Pagano**. *Ticket coverage: Putting test coverage into context.* In Proceedings of the 8th Workshop on Emerging Trends in Software Metrics (WETSoM'17), 2017.

Autor

Dr. Elmar Jürgens

Elmar Jürgens hat für seine Doktorarbeit über Qualitätsanalysen den Software-Engineering-Preis der Ernst-Denert-Stiftung erhalten. Er ist Mitgründer der CQSE GmbH und begleitet Teams bei der Verbesserung ihrer Qualitätssicherungs- und Testprozesse. Er wurde 2015 zum Junior-Fellow der GI ernannt.

juergens@cqse.eu

Autor

Dr. Dennis Pagano

Dennis Pagano hat in Software Engineering promoviert und begleitet als Berater für Software-Qualität bei der CQSE GmbH viele Firmen beim Verbessern ihrer Testprozesse. Er ist aktiv an der Entwicklung der Test-Gap-Analyse beteiligt. Seine Forschung wurde u.a. mit einem Distinguished Paper Award auf der MSR ausgezeichnet.

pagano@cqse.eu

Agilität im Projekt Management

Bedingungen für die erfolgreiche Anwendung agiler Methoden

Agile Methoden brauchen ein Umfeld, dass die zugrundeliegenden Werte und Prinzipien fördert. In vielen Firmen finden wir Strukturen vor, die Agilität eher behindern.

In diesem Artikel beschreiben wir einerseits welche Bedingungen wir vorfanden und andererseits welche grundlegenden Bedingungen notwendig sind, beziehungsweise wie man ein optimales Umfeld für die Implementierung agiler Methoden schafft.

Agilität im Projekt Management

Eine stets wachsende Anzahl neuer Technologien im Projekt und immer kürzere Projektlaufzeiten zwingen Unternehmen zu Höchstleistungen. Wenn man diesen Herausforderungen mit konventionellen Denkweisen begegnet, führt das zu hochspezialisierten Mitarbeitern, die immer kleiner werdenden Zeitscheiben in immer mehr Projekten ableisten. Dadurch geht ihnen das Verständnis für die Gesamtabläufe verloren, genauso wie die Identifikation mit dem Liefergegenstand. Außerdem muss die resultierende Menge an Arbeitspakete durch zentralen Stellen, wie das Projektmanagement, koordiniert und verteilt werden.

Der Verwaltungsaufwand im Projektmanagement steigt mit der Anzahl der Arbeitspakete, der verwendeten Tools, der eingesetzten Technologien und der Vielzahl an hochspezialisierten Mitarbeitern enorm an. Hinzu kommen können außerdem notwendige Änderungen des Plans, zum Beispiel auf Grund von Kundenwünschen. Somit wird die rechtzeitige Planung um ein Vielfaches umfangreicher und schwieriger.

Oft wird versucht dieser steigenden Komplexitäten mit zusätzlichen Prozessen, Methoden und Tools entgegenzuwirken und Unternehmensorganisationen dahingehend zu optimieren, die hochspezialisierten Mitarbeiter besser verwaltet zu können.

Agile Methoden verfolgen einen konträren Ansatz zur Leistungssteigerung von Unternehmen.

Teams von ungefähr sieben Mitarbeitern, setzen das Projekt nicht nur um, sondern tragen selbst zur detaillierte Organisation bei, indem sie schrittweise schätzen, planen, umsetzen und die Teillieferungen in Gesamtsystemen zeitnah testen.

Wir haben die Erfahrung gemacht, dass die Transition von den zuerst beschriebenen, konventionell bedingten Arbeitsweisen, hin zu agilen Methoden, nicht nur lokal auf Team- oder Projektebene stattfinden kann, sondern darüber hinaus auf allen Ebenen stattfinden muss.

In diesem Artikel beschreiben wir, welche Änderungen, auf welchen Ebenen von Nöten sind, um eine Erfolgreiche Effektivitätssteigerung durch agile Methoden zu erzielen und wie Sie beste Voraussetzungen für die Einführung dieser Methoden schaffen. Wir beginnen dazu auf der Teamebene und zeigen auf welches Umfeld ein effektives Team braucht, um

- Managementaufgaben zu integrieren
- früh eine hohe Gesamtqualität zu erzielen
- sichtbare Zwischenergebnisse in kurzen Iterationen zu generieren

In einem zweiten Schritt betrachten wir, wie man die Projektmanagementaufgaben effizient bewältigt und wie diese durch agile Methoden vereinfacht werden können. Dabei betrachten wir

- die Abgleichung von Grob- und Teamplanung
- die Organisation von Arbeitspaketen nach Funktionen
- die automatische Statuserfassung und das Reporting
- funktionsübergreifende Teams

Im letzten Abschnitt zeigen wir auf, welche Maßnahmen auf organisatorischer Ebene ein Umfeld schafft, das diese Art des Projektmanagements unterstützt. Dazu gehören

- Teams als kleinste Einheit in Unternehmen
- ein passender Führungsstil
- ein neues Karriere- und Verantwortungsverständnis

Diese Aspekte ermöglichen die effektive Anwendung von agilen Methoden auf einem breiten Feld in Ihrem Unternehmen. Die Einführung von Agilität wird ihre volle Entfaltung nur erreichen, wenn alle drei Ebenen einbezogen werden.

Das Team befähigen

Selbstverantwortlich arbeitende Teams sind das Fundament für ein effektives Projektmanagement in einem komplexen und dynamischen Umfeld.

Wie Abbildung 1 zeigt ergänzen sich die Teammitglieder im Wissensportfolio, das durch die unterschiedlichen Farben gekennzeichnet wird. Dadurch können sie sich gegenseitig bei der Entwicklung neuer Ideen unterstützen und sind in der Lage, weitgehend unabhängig von anderen Teams, End2End-Funktionen umzusetzen.

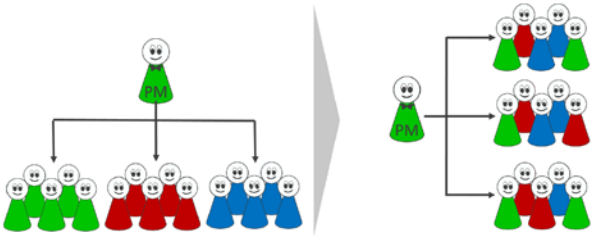


Abbildung 1 – Umstrukturierung der Teams, damit diese End2End-Funktionen umsetzen können

Sitzen die Teammitglieder in Rufweite können sie fokussiert arbeiten und längerfristig in einem stabilen Team eingebunden werden. Das steigert die Kommunikation, den Teamspirit und effektives Arbeiten. Ausfälle und Überlast in Spezialgebieten können aufgrund des verteilten Wissens besser kompensiert werden.

Kontinuierliche Optimierung der Zusammenarbeit sowohl teamintern, als auch extern, wird durch regelmäßige Koordinierung im Rahmen von Standup-Meetings und zeitnahem Feedback im Zuge von Reviews und Retrospektiven geschaffen.

Die große Herausforderung bei der Etablierung selbstverantwortlicher Teams ist die Bereitschaft zur Verantwortungsübernahme jedes einzelnen Mitgliedes. Ein Coach, der die Teammitglieder und ihre Interaktion aktiv fordert und trainiert, spielt hier eine zentrale Rolle.

Ebenfalls elementar für das Funktionieren der selbstverantwortlichen Teams, ist der Austausch von Expertenwissen. Die Arbeitsweise von Experten ändert sich also dahingehend, dass sie bei der Umsetzung von Aufgaben ihr Wissen weitergeben und möglichst viele Teammitglieder in die Lage versetzen diese Aufgaben zukünftig selbstständig abarbeiten zu können. Wir müssen uns bewusst werden, dass die Effizienzoptimierung des Gesamtprojektes auf der Effizienzsteigerung des gesamten Teams, nicht einzelner Mitglieder, beruht. Es muss darauf hingewirkt werden, dass begonnene Aufgaben auch konsequent und zeitnah umgesetzt werden, anstatt sie bei Problemen in der Pipeline versanden zu lassen.

Im agilen Ansatz spiegelt der oben bereits erwähnte Coach, ein Scrummaster oder Agile Master, dem Team die Abarbeitungsdisziplin und hinterfragt im Rückblick liegengebliebene Arbeit kritisch. Dieses Verfahren entlastet das Projektmanagement beim administrativen Aufwand.

Das Team wird wiederrum vom Backlog entlastet, es ist die einzige Schnittstelle zwischen Projektmanagement und Team. Konkurrierende Projekte oder Projektaufgaben spiegeln sich darin wieder. Damit werden Situationen vermieden in denen verschiedenen Projektleiter oder Stakeholder den Entwicklern die Arbeit erschweren, indem sie „ihre“ Aufgaben beim Entwickler priorisieren.

Agile Methoden entlasten das Projektmanagement

Aus Sicht des Projektmanagements ist vor allem die Reduzierung des Verwaltungsaufwandes von Interesse.

Sind mehrere Teams an der Entwicklung eines Produkts beteiligt, so ist die Etablierung einer regelmäßigen Synchronisation dieser Teams unerlässlich. Die am Projekt beteiligten Teams müssen ein gemeinsames Ziel vor Augen haben, und dieses gemeinsam erreichen.

Eine klare Aufteilung der Verantwortlichkeiten sorgt für höhere Planungssicherheit und hohe Qualität. Die Verantwortung für Business & Umfang ist zentrale Aufgabe des Projektmanagements. Taktung & Qualität liegt in Händen der Entwicklungsteams. Als Schnittstelle zwischen beiden Akteuren steht, wie bereits erwähnt das Produkt Backlog. Wie Abbildung 2 zeigt, ist das Projektmanagement für das Füllen und Priorisieren zuständig, dagegen das Team für Schätzung und qualitative Umsetzung. Dieser Mechanismus sichert Selbstverantwortlichkeit der Teams und eine transparente Sicht auf geschaffenen Wert und den Projektumfang.

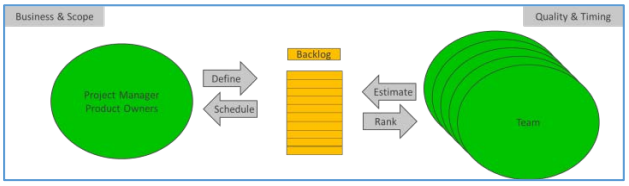


Abbildung 2 – Ein Backlog als einzige Schnittstelle

Die Planung kommt folgendermaßen zustand:. Das Projektmanagement definiert und priorisiert die benötigten Features, die Teams schätzen den Aufwand. Weicht diese Planung von den Meilensteinen des Kunden ab, ist das Projektmanagement in der Lage mit dem Team zusammen über den Umfang zu verhandeln bzw. mit dem Kunden über den Zeitraum wenn der Umfang für die Lieferung unumstößlich ist. Dies schafft Klarheit in den Erwartungen der beteiligten Parteien und schafft Transparenz über die Prioritäten der Features, welche für den Kunden umgesetzt werden müssen.

Qualität ist Teil des mit den Teams vereinbarten Liefergegenstandes. Insofern wird der Projektmanager vom Qualitätsrisiko weitestgehend entlastet, ein reduzierter Qualitätsstandard zugunsten erweiterter Features wird unterbunden. Damit diese Verantwortungsteilung gewährleistet ist unterstützt der Coach die Teams bei der Vermittlung und Kommunikation dieser Aufteilung.

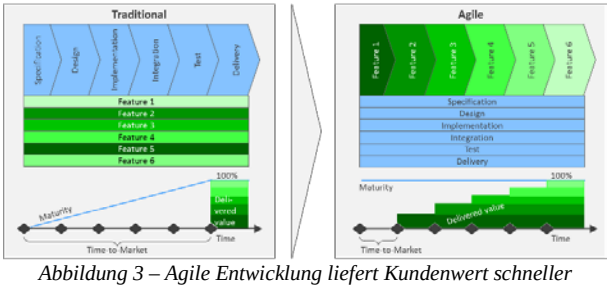


Abbildung 3 – Agile Entwicklung liefert Kundenwert schneller

Die Erstellung eines priorisierten kundenfunktionsbasierten Backlogs, wie in Abbildung 3 gezeigt, die Einführung

möglichst kurzer iterativer Entwicklungsschleifen und die Konzentration auf die Lieferung funktionsfähiger Softwarekomponenten bietet die Möglichkeit eines schnellen Kundenfeedbacks. Notwendige Änderungen können zeitnah und geplant eingesteuert werden. Trotz vieler direkter Kundenfeedbacks und kleiner Anpassungen liegt es in der Verantwortung des Projektmanagers den großen Gesamtplan im Auge zu behalten und Review Anpassungen im Kontext des Projekts einzusortieren.

Durch die Lieferung ganzer Kundenfunktionen in kleinen Abständen lässt sich der Projektstatus an der Anzahl der umgesetzten Funktionen leicht ablesen. Dies gibt dem Projektmanager die Möglichkeit Abweichungen früher zu erkennen und lässt ihm mehr Handlungsspielraum um zu reagieren. Durch diesen Schritt wird sichergestellt, dass die Teams an den richtigen Funktionen arbeiten und Sie sind in der Lage den Plan kontinuierlich zu verifizieren.

Wichtig bei der iterativen Lieferung von Features an den Kunden ist es, wiederholende Aufgaben zu automatisieren. Dies reduziert Ihre Durchlaufzeit für die Features signifikant und ermöglicht es Ihnen erst inkrementell Ihr Produkt zu erweitern. Sie sollten deshalb in den Teams das Bewusstsein schaffen, dass automatisierte Integration und das Testen für die inkrementelle Lieferung an den Kunden unumgänglich sind. Auch schaffen Sie Sicherheit für die getesteten Features, da Seiteneffekte durch neue Features rechtzeitig erkannt werden. Selbstverständlich nur dort, wo es auch automatisierte Testfälle gibt.

Diese kontinuierliche Betrachtung der Features ist der eine Teil der stetigen Verbesserung in der agilen Softwareentwicklung. Der Zweite Teil ist die Reflexion der Arbeitsweise und möglicher Behinderungen im Projekt. Dazu sollte das Projektmanagement unbedingt mit den Teams iterativ Verbesserungspotential der Prozesse, der Zusammenarbeit, der Dokumentation und der Qualität analysieren. Dieses Vorgehen ersetzt die Lessons Learned / Lessons Identified am Ende eines Projektes, weil wir kontinuierlich versuchen aus unserem Gelernten Verbesserungen am Projektsetup zu erreichen. Der Coach spielt hier als Moderator und unabhängiger Dritter eine essentielle Rolle damit alle Interessen gleiches gewichtet werden.

Im Abschnitt „Das Team befähigen“ haben wir bereits ausgeführt, dass man Teams so zusammensetzen sollte, dass sie Kundenfunktionen autark abarbeiten können. Das hilft Ihnen als Projektmanager weil sie sich auf die Definition und Priorisierung der Funktionen konzentrieren können und sich nicht mit der Einteilung von Spezialisten beschäftigen müssen. Uns ist bewusst, dass dies nicht für alle Teams möglich ist, nichtsdestotrotz sollten Verstöße gegen das Paradigma intensiv hinterfragt werden.

Organisatorische Maßnahmen unterstützen den agilen Ansatz

Organisieren Sie Ihr Unternehmen in funktionsübergreifenden, eigenverantwortlichen Teams, wie in Abbildung 4 dargestellt und machen Sie sie zu den kleinsten Einheiten in Ihrem Unternehmen. Damit entlasten Sie den Projektmanager von der Personenbesetzung des Projekts.

Funktionsübergreifende Teams sind funktional komplett ausgestattet und können somit Funktionen in einen Kundenbereich eigenständig erstellen. Dies bedeutet nicht, dass es in diesen Teams nur Generalisten geben muss. Ziel ist, dass Sie Ihre Experten in Teams so aufteilen, dass alle Teams die notwendigen Fähigkeiten haben. Es stehen also nicht Rollen (System Engineer, Developer, Tester etc.) sondern das Vorhandensein aller notwendigen Fähigkeiten im Vordergrund. Das ganze Team ist für das Ergebnis verantwortlich, einzelne Teammitglieder sollten sich nicht hinter definierten Rollen „verstecken“ können. Features werden gemeinschaftlich durch die Teams abgearbeitet, ohne einzelne Spezialisten in einer Vielzahl von Projekten zu „verbrennen“.

Der Fokus auf ein Team als kleinste organisatorische Einheit stellt auch sicher, dass ein Team gemeinschaftlich immer an einem Backlog arbeitet. Das schafft Klarheit bei der Verfügbarkeit von Teams und vermeidet, dass das Team bzw. die Organisation intern priorisieren muss, welches Projekt wichtiger ist. Die Teams fokussieren sich dadurch auf ihre Aufgaben aus dem Backlog und die Priorisierung zwischen den Projekten obliegt den verschiedenen Projektmanagern. Damit lassen sich Projektphasenübergänge und temporäre Spitzen bestmöglich abdecken.

Da jedes Team autark arbeitet, können in der Projekt Anlaufphase sukzessive Teams hinzugefügt werden bis das Projekt unter Volllast läuft. Ebenso werden im Abschluss des Projektes die Anzahl der Teams nach Bedarf reduziert und anderen Projekten zugeführt. Dies verhindert, dass im Zeitraum des Projektes Spezialwissen, weil die Teams dieses Wissen aufbauen und auch im Team verteilen.

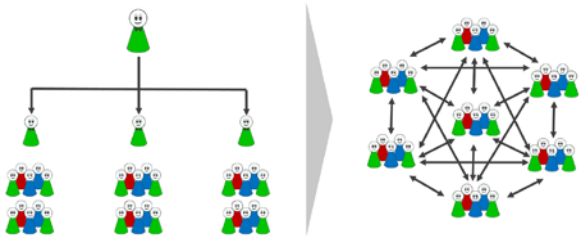


Abbildung 4 – Eine Projektorganisation als Netzwerk von autarken Teams die untereinander ihr Wissen teilen

Um die Eigenverantwortlichkeit der Teams nicht zu unterminieren ist ein Führungsstil gefragt, der die Selbstorganisation der Teams stärkt und die einzelnen Teammitglieder fördert und in ihrer selbstverantwortlichen Arbeitsweise bestärkt. Die agile Führungskraft verteilt keine Features (Push-Prinzip), vielmehr übernehmen die Teammitglieder im Rahmen ihrer Projektverantwortung und ihrer persönlichen Fähigkeiten anstehende Tätigkeiten (Pull-Prinzip). Er fordert Verbindlichkeiten und gelieferte Qualität heraus, hinterfragt Regeln und bürokratische Abläufe welche die Teams blockieren und etabliert kontinuierliche Verbesserung (Kaizen) in den Teams.

Sie müssen als Führungskraft sicherstellen, dass Teams, wie auch Einzelpersonen, Fehler machen dürfen. Ohne Experimente die fehlschlagen können, wird sich keine Verbesserung einstellen. Allerdings muss der Lerngedanke im Vordergrund stehen. Wie Albert Einstein schon sagte, „Die Definition von Wahnsinn ist, immer wieder das Gleiche zu tun und andere Ergebnisse zu erwarten“. Es ist essentiell Experimente einzuführen und Neues kontrolliert auszuprobieren. Als Führungskraft erhalten Sie am besten den Überblick, indem Sie oft und nahe mit Ihren Mitarbeitern interagieren und sich mit ihnen austauschen. Das hilft besser als jeder Report, um die Situation richtig einzuschätzen und daraus Rückschlüsse für weitere Experimente zu ziehen.

Unser letzter Aspekt in diesem Rahmen, sind Karrierepfade. Wir sind der Überzeugung, dass diese zur Bildung effektiver eigenverantwortlicher Teams eher kontraproduktiv sind. Im Fokus der Teammitglieder steht der Teamerfolg. Dadurch ist es notwendig die Karrierepfade nicht mehr nur auf die individuellen Fähigkeiten einer Person festzulegen, sondern im speziellen auf die Bedürfnisse der Teams. Bezahlung und Bewertung nach Teamerfolg sind hier erfolgversprechender.

Fazit

Uns ist bewusst, dass wir hier ein breites Spektrum an Themen anschneiden, aber leider nicht in die Tiefe eines jeden Teilaspektes gehen können.

Wir sehen, dass viele Unternehmen sich in eine Sackgasse der Effizienzoptimierung manövriert haben und darin feststecken. Als Ausweg empfehlen wir, ganz im agilen Sinne, eine Transformation durch Reflektion und Adaption zu schaffen.

In unserem Artikel haben wir Ihnen aufgezeigt, welche Handlungsspielräume Sie als Projektmanager durch die Einführung agiler Methoden gewinnen und wie Sie dadurch die Effektivität ihrer Softwareentwicklung steigern. Aber auch, dass der Erfolg dieser Methoden von den genannten Faktoren abhängig ist, die Ihre gesamte Organisation betreffen.

Sollten Sie Anregungen oder Fragen zu diesem Thema haben, können Sie uns gerne kontaktieren.

Literaturverzeichnis

Pichler, Roman, Agile Produktmanagement mit Scrum: Erfolgreich als Product Owner arbeiten, dpunkt.verlag, 2014

Sutherland, Jeff, Scrum: The Art of Doing Twice the Work in Half the Time, Random House Business, 2015

Leopold, Klaus, Kanban in der IT: Eine Kultur der kontinuierlichen Verbesserung schaffen – 2. Auflage, Carl Hanser Verlag GmbH & Co KG, 2013

Larman, Craig, Scaling Lean and Agile Development: Thinking and Organizational Tools for Large-Scale Scrum, Addison Wesley, 2008

Scaled Agile Framework, SCALED AGILE Inc., http://www.scaledagileframework.com/

Autoren

Sascha Preissler

Sascha Preissler ist Senior Expert bei Elektrobit und berät Firmen bei der Einführung und Umsetzung agiler Methoden. Als Diplom Informatiker sammelte er langjährige Erfahrung im kompletten Softwareentwicklungsprozess in kleinen und mittelständischen Unternehmen. Sein Wissen über agile Methoden erlangte er ab 2010 als Projektleiter und Berater für ein finnisches Telekommunikationsunternehmen und baute es ab 2014 bei Elektrobit, erst als Scrum-Master , später als Verantwortlicher für den Roll-Out agiler Arbeitsweisen, weiter aus.

sascha.preissler@elektrobit.com

Martin Hillbrand

Martin Hillbrand ist Experte für agile Prozesse bei Elektrobit. Als Scrum Master, Agiler Coach und Software Craftsman unterstützt er seit mehreren Jahren Unternehmen bei der Transition zu agilen Vorgehensweisen und sammelte dabei viel Erfahrung in verschiedenen mittelständischen bis großen Unternehmen, dabei lernte er die unterschiedlichsten Herausforderungen aus Glücksspiel, Logistik oder Security kennen. Diese mussten immer mit dem Fokus auf die Anpassung des gesamten Systems des Unternehmens gemeistert werden. Dabei war es immer von Vorteil die neuesten theoretischen Modelle zu kennen um daraus die besten Praktiken zur Lösung der Herausforderungen zu identifizieren. Er steht deshalb für die adaptive Einführung agiler Arbeitsweisen in einem Unternehmen, um den besten Weg zu finden.

martin.hillbrand@elektrobit.com

Der einfache Usability Test

Tipps und Tricks für Requirements und Usability Tests bei Apps und Produkten aus dem IoT Umfeld

Usability Tests im echten Testlabor sind teuer und für Apps auf Smartphones oder kleinen Bedienungsdisplays von IoT-Produkten oft nicht zielführend. Dieser Beitrag zeigt auf, wie man mit einfachen Mitteln bereits beim Erstellen der Requirements und im anschließenden Test gute Usability generiert bzw. untersucht.

In einer Untersuchung von Perfecto Mobile aus dem Jahr 2014 kam heraus, dass Nutzer Probleme bei der Usability als häufigsten Grund angaben, warum Sie eine App wieder deinstallierten. An zweiter Stelle folgt Performanz und erst dann Funktionalität, Gerätekompatibilität und einige weitere. Usability und Performanz sind aus Nutzersicht also die wichtigsten Qualitätsmerkmale, wenn es um den Erfolg einer App geht.

Bedienung von Produkten aus dem IoT Umfeld

Auf den ersten Blick geht in diesem Artikel ein ungewöhnliches Gespann an den Start. Lassen sich Erfahrungen aus dem Usability Test für Apps auf Produkte aus dem Internet of Things (IoT) Umfeld übertragen?

IoT-Produkte werden häufig per App bedient. Ist das nicht der Fall, werden häufig kleinere Touch Displays für die Bedienung benutzt. Für beide Bedienungsarten können wir den Usability Test genauso durchführen, wie für einen Usability Test von Apps auf Smartphones.

Natürlich werden einige IoT-Produkte auch anders bedient. Beispielsweise gibt es Webseiten für die Administration von ganzen IoT Systemen. Der Endanwender wird aber eben häufig Touch Displays nutzen.

Viele IoT-Produkte bedürfen auch gar keiner Bedienung, sondern sind einfach da, messen, schalten und steuern.

Nach der ISO Norm DIN EN ISO 9241-110 gibt es für die Usability der Schnittstelle zwischen Benutzer und System die folgenden Kriterien:

- **Aufgabenangemessenheit**
- **Selbstbeschreibungsfähigkeit**
- **Erwartungskonformität**
- **Lernförderlichkeit**
- **Steuerbarkeit**
- **Individualisierbarkeit**
- **Fehlertoleranz**

Daniel Knott zählt in seinem Buch Mobile App Testing die folgenden Kriterien für gute Usability auf:

- **Weniger ist mehr**
- **Selbsterklärend**
- **Fehler erlauben**
- **Wortlaut**
- **Konsistenz**
- **Muster**

Vergleicht man diese beiden Auflistungen stellt man fest, dass es einige Gemeinsamkeiten für die Usability von Apps und IoT- Produkten gibt.

> Die Benutzbarkeit von Apps und Produkten aus dem IoT Umfeld ähnelt sich stark.

Gute Usability

Was macht eigentlich gute Usability aus? Wie definiert man gute Usability?

Ich finde, diese Fragen lassen sich nur sehr schwer oder sehr umfangreich beantworten. Gute Usability fällt nicht auf, schlechte aber schon. Wenn sich eine App gut bedienen lässt, ist das so. Das bemerkt der Nutzer nicht. Man benutzt die App einfach, und gut. Ist die Usability dagegen schlecht, wird man unzufrieden und im schlimmsten Fall benutzt man eine andere App, die das gleiche tut.

Auch das wird ein Grund dafür sein, warum Usability in der Umfrage vom Anfang ganz weit oben steht.

@Martin LeBlanc twittert dazu:

Recht hat er.

Requirements und Usability

Nachfolgend werden einige Begriffe und Methoden erklärt, die bereits beim Erstellen der Requirements mit Bezug zur Usability helfen werden:

- **Zielgruppen**

Je besser wir unsere Zielgruppe für unsere Produkte kennen, desto besser können wir uns auf sie einstellen. Eine Zielgruppe kann sehr homogen sein. Beispielsweise, wenn es um eine Augmented Reality Brille für den Operateur im Operationsraum geht. Alle Menschen, die dieses Produkt benutzen, haben nahezu die gleiche Ausbildung, den gleichen Background etc.

Wenn es hingegen um eine App für die Restaurantsuche in der Umgebung des Aufenthaltsortes geht, ist die Zielgruppe sehr heterogen. In ihr gibt es junge und alte Nutzer, Nutzer, die sich mit der Bedienung eines Smartphones auskennen oder auch nicht, Leute mit viel und Leute mit wenig Geld, Menschen, die auf dem Land oder in der Stadt wohnen, etc.

Die folgenden Fragen helfen, um die Zielgruppe besser kennen zu lernen:

1. Wer ist meine Nutzerbasis?
2. Wie alt ist der durchschnittliche Nutzer?
3. Wie viele Frauen oder Männer sind in meiner Zielgruppe?
4. Welche Geräte werden von meiner Nutzerbasis am häufigsten verwendet?

Sobald wir die Zielgruppe genauer kennen, können wir daraus auch Requirements bezüglich der Usability ableiten.

- **Personas**

Nachdem wir die Zielgruppe genauer beschrieben haben, können wir Personas erstellen. Eine Persona ist eine fiktive Person genau aus dieser Zielgruppe. Aus der Zielgruppe für die Restaurantsuche-App könnte man dann ganz konkret eine Persona namens Kerstin erstellen:

Information	Profil
Name	Kerstin
Geschlecht	Weiblich
Alter	35
Monatliches Einkommen	3.500 €
Bildungshintergrund	Masterabschluss in Informatik
Wohnort	München
Benutzte Apps	Spotify, Twitter, Facebook
Geräte	iPhone7, iPad
Persönliche Merkmale	Freundlich, intelligent, geht gerne aus

Mit Hilfe mehrerer Personas können wir die Usability Requirements weiter verbessern, indem wir die Usability aus Sicht dieser konkreten Personas betrachten.

- **Szenarien**

Szenarien helfen, konkrete Usecases für die App oder das IoT-Produkt zu bestimmen.

Ein Szenario sollte den folgenden Inhalt haben:

1. Beschreibung des Szenarios – was erwartet der Nutzer, was wird benötigt
2. Genaue Beschreibung des Ablaufs
3. Beschreibung von möglichen Fehlern und der Umgang damit
4. Zeitlich parallel laufende Aufgaben
5. Wann ist das Szenario zu Ende – Zustand der App bzw. des IoT-Produkts

Beispiel für ein Szenario U-Bahn Überweisung:

> *Martin fährt gerade mit der U-Bahn von der Arbeit nach Hause. Jetzt fällt ihm ein, dass er noch eine Überweisung machen muss.*
>
> *Er erinnert sich, dass seine neue Bank eine App zur Verfügung stellt.*
>
> *Er lädt sie mit dem zur Verfügung gestellten W-Lan in der U-Bahn herunter und startet sie.*
>
> *Martin meldet sich mit den Daten vom Online-Banking an.*
>
> *Er sucht die Rechnung in seinem Emailprogramm und führt die Überweisung durch.*

Mit Hilfe dieses konkreten Szenarios können wir abermals die Requirements bezüglich der Usability genauer untersuchen und gegebenenfalls verbessern.

- **Weitere Hilfsmittel**

Story Boards können helfen komplexe Konzepte oder Umgebungen genauer zu beschreiben. Bilder sagen eben häufig mehr als tausend Worte.

Abbildung 6- Beispiel Story Board

Neben dem Story Board können auch UML-Anwendungsfälle oder User Stories die konkrete Beschreibung eines Uses Cases hinsichtlich der Usability noch eindeutiger darstellen.

- **Prototyping**

Sowohl das Low Fidelity Prototyping (LFP, Prototyping auf dem Papier) als auch das High Fidelity Prototyping

(HFP, Prototyping mit klickbaren Oberflächen) sind gerade für die Benutzbarkeit ein geeignetes Hilfsmittel in jedem Projekt. In beiden Fällen können Stakeholder, Product Owner, Kunden, Entwickler aber auch Tester einen ersten Eindruck bezüglich der Usability erhalten. Selbst im LFP kann man sich einen ersten Eindruck über die Benutzbarkeit der App machen. Eventuell sind die Buttons für unsere Zielgruppe zu klein oder zu versteckt - oder Kernfunktionalitäten sind nur schwer zu erreichen.

Natürlich kann man die Usability beim HFP besser beurteilen, selbst wenn ein Button-Klick keine wirkliche Funktionalität hat, da ein echter Prototyp beispielsweise auf dem Smartphone installiert wird.

Gerade für die Usability ist das Wissen über die Zielgruppe unseres Produkts essentiell. Mit Hilfe der Zielgruppe können wir aus der Sicht einer konkreten Persona die Usability der App oder des IoT-Produkts beurteilen. Konkrete Beschreibungen des Usecases mit Hilfe von Szenarien oder Story Boards helfen, mögliche Anforderungen hinsichtlich der Usability zu erstellen oder zu hinterfragen.

Usability Tests

Ein echter Usability Test im Labor ist folgendermaßen konzipiert. Mehrere Versuchspersonen benutzen eine Anwendung und werden dabei gefilmt. Manche Kameras beobachten aus der Ecke des Raumes, andere beobachten die Bewegung der Pupillen. Zusätzlich wird jeder Klick, jede Interaktion mit der Anwendung aufgezeichnet. Der Proband muss dann mehrere ihm vorgeschriebene Usecases durchführen. Im Anschluss werden die Videos, die Klicks und weitere Daten ausgewertet. Außerdem müssen die Probanden einen Fragebogen ausfüllen. Alles zusammen ergibt ein gutes Testergebnis bezüglich der Usability.

Dieses Vorgehen verspricht detaillierte Ergebnisse, ist aber auch sehr teuer. Außerdem ist fraglich, ob die vielen Kameras gerade bei der Bedienung eines Smartphones noch wesentliche Dinge aufzeichnen können.

Das im Folgenden beschriebene Vorgehen zeigt Ihnen, wie Sie mit einfachen Mitteln einen kostengünstigen und trotzdem zielführenden Usability Test durchführen können.

Wir können alles, was wir bereits bei der Erstellung der Requirements benutzt haben, wiederverwenden:

- **Personas**
- **Szenarien**
- **Anwendungsfälle**
- **Prototyping**

Mit Hilfe der Personas können wir einzelne Tests aus unterschiedlichen Perspektiven durchführen. Szenarien und Anwendungsfälle helfen, Testfälle zu erstellen oder ein grobes Vorgehen für explorative Tests zu haben.

Sofern es Prototypen des IoT-Produkts oder der App gibt, sollte der Tester früh in das Projekt einsteigen.

Zusätzlich benutzen wir die folgenden Methoden:

- **Usability Heuristiken**

Im Web findet man viele Usability Heuristiken. Mir haben die von der Nielsen Norman Group am besten gefallen. Auf deren Webseite sind die zehn besten Usabilty Heuristiken erklärt und aufgelistet (siehe Quellen). Mit diesen Heuristiken kann man für sein Projekt eine Checkliste erstellen und diese dann im Test einfach durchgehen.

- **Error guessing**

Ein weiteres Testvorgehen ist das Error Guessing. Beim Error Guessing oder auch dem intuitivem Vorgehen werden von erfahrenen Usability Testern mehr oder weniger explorative Tests durchgeführt. Die Tester schauen sich vor allem die Teile der Anwendung an, die in anderen Anwendungen bereits Usability Auffälligkeiten aufwiesen. Außerdem erstellt der Tester bei diesem Vorgehen neue Testfälle für das aktuelle Projekt.

- **Exploratives Testen**

Exploratives Testen ist mit Nichten ein „darauf los testen“. Auch beim explorativen Testen gibt es Richtlinien und Vorgehensweisen.

Eines davon sind Testtouren, die uns auch beim Usability Test helfen können. Die folgenden Touren eignen sich insbesondere für den Usabilty Test:

- **Supermodel Tour**
- **Lonely Businessman Tour**
- **Guidebook Tour**
- **Landmark Tour**
- **Coach Potato Tour**

Mit Hilfe dieser Touren und Szenarien, Story Boards oder Anwendungsfällen führen wir einen explorativen Test durch und identifizieren dabei Usability Auffälligkeiten.

- **Testing in the Wild**

Diese Methode eignet sich besonders für mobile Anwendungen oder Produkte – also Anwendungen, die von unterwegs aus benutzt werden.

Nehmen Sie das IoT-Produkts bzw. die App und testen Sie diese von unterwegs aus. Beispielsweise sollte eine App, die die Schneehöhen anzeigt auch tatsächlich einmal in den Alpen untersucht werden. Funktioniert die App auch, wenn die Verbindung nur sehr langsam ist? Wird das GPS-Signal auch an Steilhängen gut verarbeitet? Kann man die App auch mit kalten Fingern oder gar mit Handschuhen benutzen? All diese Dinge sind nur im wirklichen Live-Betrieb zu testen. Nach unseren Erfahrungen finden Sie hier eine Menge Auffälligkeiten oder gar Fehler – vor allem auch bezüglich der Usability.

Tests bezüglich der Usability lassen sich einfach mit bereits bekannten Testmethoden durchführen. Seien Sie kreativ und verwenden Sie alles, was bei der Erstellung der Requirements verwendet wurde, wieder. Benutzen Sie z.B. Personas, um in einem explorativen Test bei der Supermodel Testtour eine andere Perspektive einzunehmen und beurteilen Sie aus dieser Sicht die Usability.
Außerdem helfen Heuristiken und Error Guessing, ein zielführendes Testkonzept bezüglich der Usability zu erstellen.

Wrap Up

Die Erstellung von Usability Requirements und der anschließende Usability Tests können auch kostengünstig und trotzdem zielführend durchgeführt werden.

Seien Sie kreativ und verwenden Sie Ihnen bereits bekannte Testmethoden wie Exploratives Testen oder ähnliches.

Damit Sie das IoT-Produkt oder die App aus anderen Perspektiven hinsichtlich der Usability bewerten können, helfen vor allem Personas.

Bibliography

Knott, Daniel, Mobile App Testing, dPunkt Verlag 2016

https://www.nngroup.com/articles/ten-usability-heuristics/

https://www.interaction-design.org/literature/article/don-t-build-it-fake-it-first-prototyping-for-mobile-apps

http://wiki.iao.fraunhofer.de/index.php/Qualit%C3%A4tsmerkmale:_Auszug_des_Leitfadens_%C2%BBEntwicklung_und_Bewertung_von_Automaten%C2%AB#Beispiel:_Qualit.C3.A4tsmerkmal_Usability

Autor

Master of Science in Angewandter Informatik Nils Röttger
Speaker, Berater Mobile Testing und Leiter Ausbildung imbus AG
Email: Nils.Roettger@imbus.de
Twitter: @nilsroettger
Xing: xing.com/profile/Nils_Roettger

accenture

NEW ISN'T ON ITS WAY. WE'RE APPLYING IT NOW.

NEW APPLIED NOW

Erfahren Sie, wie wir innovative Lösungen und unser Branchenwissen nutzen, um die Herausforderungen von Unternehmen anzugehen – jetzt auf accenture.de

NEW APPLIED NOW

strategy | consulting | digital | technology | operations

The Modern Software Factory. The blueprint for digital transformation.

The Modern Software Factory is the blueprint for digital transformation. It's where agility, automation, insights and security come together to help your business compete in the rapidly changing application economy.

See how CA can help you build your own Modern Software Factory at ca.com.

ca technologies

Business, rewritten by software™

Schärfen der Softwarequalität

Continuous Quality in Softwareentwicklungsprozessen

Ein Softwareentwicklungsprozess ist mehr als Continuous Integration und Deployment, denn auch die Qualitätssicherung muss als kontinuierlicher Bestandteil darin integriert werden. Testmaßnahmen müssen häufiger und gezielter gesetzt werden. Das führt zu Herausforderungen, welche durch ein entsprechendes Konzept gelöst werden können. Dabei kommen neben Continuous Integration und Deployment noch weitere Werkzeuge zum Einsatz. Der Beitrag erklärt die Grundidee, das Vorgehen und das daraus resultierende Grundkonzept von Continuous Quality um die Softwarequalität zu schärfen.

Softwareentwicklungsabteilungen werden oft als Softwareschmieden bezeichnet. Nehmen wir an, dass eine solche Schmiede ein edles Schwert herstellt, so liegt es nahe, die Qualitätssicherung als das „Schärfen“ jenes Schwertes zu sehen.

Wir wissen, dass ein Schwert nur durch viele, kontinuierlich wiederholte Arbeitsgänge rasiermesserscharf gemacht werden kann. Doch wir übersehen häufig, dass dies auch für die Qualität der Software gilt, analog zu unserm Beispiel des Schwertes. Wollen wir also die Qualität schärfen wie ein edles Schwert, so müssen wir an mehr als nur einen Arbeitsschritt und somit auch mehr als nur eine Testphase denken.

Bildlich gesprochen bedeutet das, dass wir ein Vorgehen vom groben Schliff bis hin zur feinen Politur unserer Software benötigen. Dazu müssen wir alle Werkzeuge gekonnt in den Softwareentwicklungsprozess einbinden, so dass die Software durch kontinuierliches Testen und weitere qualitätssichernde Mechanismen, immer wieder über den Schleifstein gezogen und somit verbessert wird. Und dadurch, wie ein Schwert, wertvoller und besser wird.

Kontinuierliches Wiederholen - Das Schleifen

Continuous Integration, Delivery und Deployment sind fixe Bestandteile von modernen Softwareentwicklungspraktiken, wie auch in der Form von zentralen Elementen in DevOps. Sie ermöglichen erst das schnelle und effektive Arbeiten der Entwickler. Darum ist es wichtig, dass die Qualitätssicherung ebenso agiert und durch kontinuierliches Testen die Qualität immer weiter schärft. Also sprechen wir, analog zu Continuous Integration, Deployment und Delivery von einem Continuous Quality Ansatz.

Testmaßnahmen werden dabei in mehreren Ebenen des Softwareentwicklungsprozesses festgelegt und verankert, um mit den Ansprüchen moderner Softwareentwicklung mithalten zu können.

Testen muss gezielt und vor allem kontinuierlich durchgeführt werden, wobei neben Tests neuer Features, häufige Regressionstests über bereits getestete Softwareteile durchzuführen sind, um Regressionen und ungewollte Veränderungen aufzudecken.

Dies ist in schnellen und inkrementellen Umfeldern, wie DevOps oder auch bei agiler Softwareentwicklung, manuell nicht realisierbar und Testautomatisierung kommt zum Einsatz. Solche automatischen Qualitätssicherungsmechanismen können in unterschiedlichen Ausprägungen an verschiedenen Stellen im Prozess eingesetzt werden.

Das Vorgehen - vom groben Schliff bis zur Politur

Continuous Quality ist nicht nur das Erstellen eines automatisierten nächtlichen Testlaufs, sondern auch das Einrichten weiterer automatisierter Qualitätssicherungsmaßnahmen durch den gesamten Softwareentwicklungsprozess hindurch. Denn auch beim Schärfen eines Schwertes wird nicht nur ein einzelner Schleifstein verwendet.

So beginnt der grobe Schliff schon mit den Unittests und Komponentenintegrationstests. Diese werden von den Entwicklern erstellt und sollten schon am Anfang des Prozesses kontinuierlich durchgeführt werden. Am besten werden diese also bei jedem Check-In bzw. Push des Codes in das zentrale Repository als Teil des Continuous Integration Ablaufs ausgeführt.

Somit wird in der CI nicht nur die Software gebaut, sondern auch schon grob in ihren Komponenten durch die Entwicklertests getestet. Oft macht es Sinn, ein Check-In oder Push in das zentrale Repository erst nach bestandenen Unittests zuzulassen, um die Qualität im Repository zu sichern.

Nun haben wir eine paketierbare und in den Komponenten getestete Software, die in verschiedenen Testumgebungen eingespielt werden kann. Hier kommen weitere Werkzeuge zum Einsatz:

- **Provisioning**: Ist das Vorgehen zur Erstellung bzw. zum Einrichten einer Umgebung. Dies kann bereits automatisiert und in der jeweilig benötigten Konfiguration der Teststufe ausgeführt werden. Ebenso ist es möglich Testumgebungen “on demand” in Form von Containern zu erstellen, so dass diese nach dem erfolgreichen Build der Software automatisch erstellt, hochgefahren, das System im Test

eingespielt, getestet und der Container wieder heruntergefahren wird.

- **Test-Environment-Management**: Ist die übergeordnete Verwaltung der Testumgebungen in ihrem Umfang und Nutzen in der gesamten Prozesskette. Ein Beispiel einer typischen Struktur, die sich an den jeweiligen Teststufen orientiert, ist:
  - **Ebene 1 - Development**: Software wird mit den neuesten Änderungen am Code eingespielt und in den Komponenten, Schnittstellen, sowie wichtigsten fachlichen Anforderungen getestet.
  - **Ebene 2 - Test**: Hier eingespielte Software wird durch Tester und vor allem automatisierten Tests funktionell genauestens gegen die Anforderungen geprüft.
  - **Ebene 3 – Pre-Production**: Diese Testumgebung entspricht so gut wie möglich der echten Produktionsumgebung und dient den Abnahmetests durch den Betrieb bzw. Fachbereich.

Wir haben nun einen durch Continuous Integration erstellten Build inklusive durchgeführter Unittests und ein automatisches Deployment in einer ebenso automatisch erstellten Testumgebung. Damit wir nun die Testebenen schnell durchlaufen und Tests mit jedem Build immer wieder in selber Qualität durchgeführt werden können, benötigen wir in den Ebenen eine Testautomatisierung von Schnittstellen und UI Tests.

Damit dies auch schon in Ebenen wie zum Beispiel Development möglich ist, wo noch nicht alle Komponenten, Daten sowie benötigten Services zur Verfügung stehen, werden zusätzlich folgende Werkzeuge verwendet:

- **Testdatenmanagement** - wird eingesetzt um Testdaten in der jeweiligen Testebene bereitzustellen. Dies kann durch einen Abzug und Anonymisierung von Produktivdaten geschehen oder durch synthetische Erstellung. Auch Mischformen beider Varianten sind möglich.
- **Service Virtualisierung** - Durch Simulation von Services, sind Tests dieser Serviceschnittstellen schon in Testebenen möglich, in denen die betroffenen Services selbst noch nicht zur Verfügung stehen. Dies minimiert die Fehler in späteren Testebenen, wo das reale Service integriert wird und reduziert somit die Probleme und Kosten spät entdeckter Fehler in der Software.

Da nun in jeder Ebene automatisiert getestet wird, benötigen wir eine Möglichkeit den gesamten Prozess auch zu überwachen. Das funktioniert über Monitoring-Tools, welche alle Systeme sowie Build- und Deployment-Ergebnisse abgreifen und überwachen.

Darüber hinaus ist es meist notwendig eine weitere Testebene für Last und Performanz-Tests einzuplanen und entsprechende Tests der nicht-funktionalen Anforderungen an Lastbeständigkeit und Performanz durchzuführen.

Mit all diesen Werkzeugen ist es nun möglich ein Konzept umzusetzen, welches die Qualität der Software vom Build über die Testebenen bis in die Produktion schärft und sicherstellt.

Das Konzept - Schärfen durch Continuous Quality

Wie die zuvor beschriebenen Werkzeuge im Detail eingesetzt werden hängt von mehreren Faktoren ab. Darunter die Art des Softwareentwicklungsprozesses, die Architektur der Software und das Testkonzept an sich. Wichtig dabei ist, dass das Schärfen der Qualität, durch mehrere Stufen kontinuierlich durchgeführter Testmaßnahmen, fest in den Softwareentwicklungsprozess integriert und zum überwiegenden Teil automatisiert wird.

Damit ist es möglich eine voll automatisierte Softwareentwicklungsprozesskette zu erschaffen, welche in der Grundidee wie folgt verläuft:

1. Der Entwickler stellt seine Änderung am Code fertig und checkt sie ein bzw. pusht diese in das zentrale Repository
2. Der Build-Prozess wird durch das Check-In bzw. den Push automatisch gestartet, baut die Software und führt alle Unit-Tests und Entwickler-Integrations-Tests aus.
3. Nach bestandenen Tests wird die Software in der Development-Umgebung eingespielt und mit automatisierten Schnittstellen- und Smoke-Tests in ihren Komponenten getestet.
4. Sind auch diese Tests fehlerfrei, so wird das Softwarepaket automatisch auf die Test-Umgebung eingespielt, wo alle automatisierten Schnittstellen, Integrationstests und UI-Tests aller Features durchgeführt werden.
5. Waren alle Tests der Vorstufen erfolgreich, so wird die Software auf der Pre-Production-Umgebung ausgerollt und steht somit für die (potentiell ebenfalls automatisierten) Abnahmetests durch den Betrieb bzw. Fachbereich zur Verfügung.
6. Sind auch diese Abnahmetests abgeschlossen, wird die Änderung auch im Produktionssystem eingespielt und ist somit produktiv.

In allen Testebenen kommen dabei Provisionierung, Testdatenmanagement und Service Virtualisierung zum Einsatz. Zusätzlich können noch Last- und Performanz-Tests automatisiert ausgeführt werden.

Mit dieser Konzeptidee kann jeder Prozess durch Continuous Quality erweitert und verbessert werden. Im Falle von DevOps, wird erst durch Continuous Quality das Optimum der DevOps-Idee erreicht: Eine schnelle, effiziente und dabei qualitativ hochwertige Entwicklung von Software.

DevOps ist also mehr als Continuous Integration und Deployment. Es besteht zusätzlich aus Provisioning, Test-Environment-Management, Testautomatisierung, Service Virtualisierung, Testdatenmanagement, Last- und Performanz-Tests, sowie Monitoring.

Autor

Dominik Schildorfer

Dominik Schildorfer fand nach Abschluss seines Informatik-Studiums an der FH Technikum Wien seinen Schwerpunkt in den Bereichen Testautomatisierung und Service Virtualisierung. Seine Expertise konnte er bei ANECON bereits in verschiedensten Projekten praktisch anwenden und durch wertvolle Erfahrungen weiter ausbauen.

dominik.schildorfer@anecon.com

Continuous Integration

In developing complex Embedded systems

It is always crucial to test new functionality as soon as possible and to make sure it does not have any negative impact on the existing one. Our experience proves that during development complex Embedded systems, the Continuous Integration is the best choice for functional and regression verification. It means that parallel development with integrated Automated build system (ATS) and Automated testing system (ABS) will enable you to achieve a high quality and reliable solution.

Comtrade Digital Services

Comtrade Digital Services including Embedded team helps our clients expand into new technologies and shorten the time to market. With deep knowledge and over 20 years of experience serving world leaders in automotive, measurement devices, telecommunications, multimedia, medical, and energy fields, we are excellent partners even for most demanding projects.

We are planning to present our findings from years of delivering cost-effective embedded solutions.

Challenges during development

During development process we face with many challenges such as building a firmware from one place, testing a new functionality, finding a bug and functional and regression verification. Constantly trying to overcome these challenges we aim to lower verification and development cost and improve quality and deliver the best to our clients.

Continuous Integration

As shown in the next figure, the build server checks out new code from source control, compiles/builds it and tests the code (primarily unit tests though static code analysis is also possible). The build server can also launch scripts to perform integration testing, user interface testing, advanced security testing and other tests requiring a running version of the software. Consistent with Agile requirements that emphasize a continually working version of the software, CI server automatically reverts to the last successful version of the software, keeping a working QA system available even if integration tests failed.

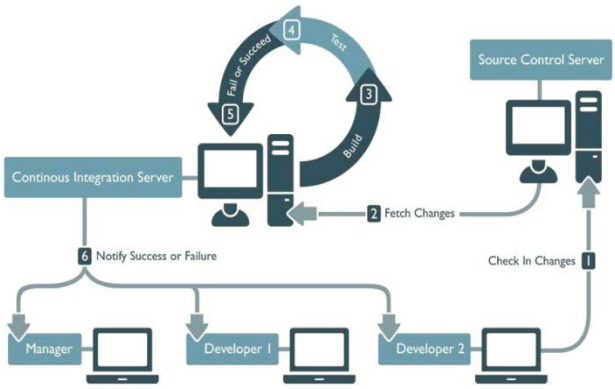


Figure 1: Continuous Integration example

It is proven that Continuous Integration with Automated build system (ABS) and Automated testing system (ATS) is the best way to develop embedded systems. Since it is always critical to test new functionality as soon as possible to achieve functional and regression verification, ABS and ATS are the best choice.

Automated Testing System

Comtrade has designed and developed a test automation solution for one of our previous projects and an ATS setup is shown on the following figure.

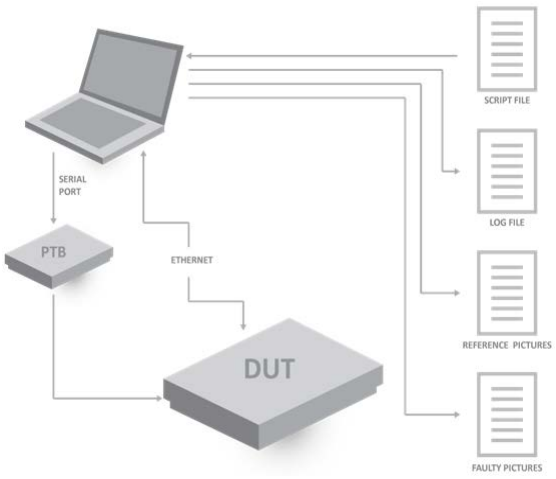


Figure 2: Automated Test System (ATS)

The communication flow between the PTB and DUT was done via the SCPI protocol. It means that DTU has to be

SCPI compliant and this enables us to properly configure DUT before the test is executed against the DUT.

Components of ATS

As it is shown on the next figure, the main components of our automated test system are:

- Script Generator
- ATS client
- PTB – Production testing board - proprietary HW developed by Comtrade
- DUT – Device under test – vehicle diagnostic system in our case
- Report generator

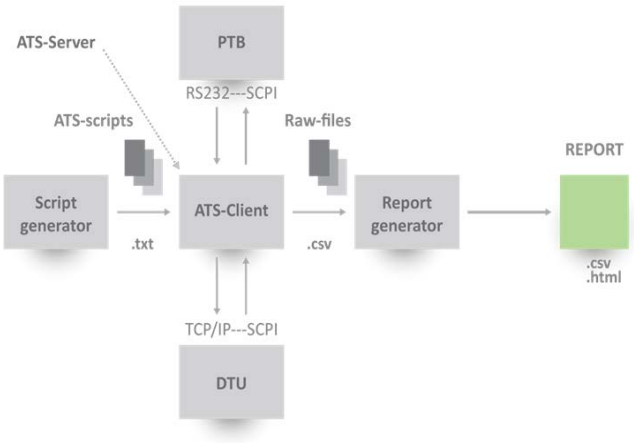


Figure 3: Interaction between ATS components

The ATS client executes test scripts and logs the results, the Production testing board (PTB) simulates HW signals – such as voltages, currents, and resistance – which are connected to a Device Under Test (DTU), the Script Generator generates the test scripts and the Report Generator evaluates test results and creates test reports.

Automated Build System

ABS is a part of a server with automated continuous integration features and automates the integration process by monitoring the source control repository directly. Every time when a developer commits a new set of modifications, the server automatically launches an integration build and automated tests in order to validate the latest changes.

Conclusion

Find out how we can help you develop even better products and services with our technically advanced and skilled engineering teams underpinned by highest quality standards.

Bibliography

Head of Embedded Systems Engineering at ComTrade Digital Services company (http://comtradedigital.com/), with over a decade of experience in Embedded Software development in different roles, starting as Developer, through Technical Lead to Project and Account manager, in many fields, such as Medical, Industrial, Automotive to IoT and Smart Energy. Responsible for the Embedded team in Comtrade company which is able to develop remarkable products and services with technical knowledge and skills underpinned by highest quality standards.

Author

Ivan Veličković
ivan.velickovic@comtrade.com

Die Vermessung des Endanwenders

Psychologische Nutzerfaktoren in der agilen Entwicklung

Eine hochwertige User Experience (UX) gilt als wesentliches Kriterium für den Markterfolg von interaktiven Produkten. Daher sollten UX Design Aktivitäten integrale Bestandteile von Softwareentwicklungsprojekten sein. Gerade für den agilen Bereich stellen sich hier jedoch besondere Herausforderungen, unter anderem aufgrund von speziellen zeitlichen Constraints. Im vorliegenden Artikel soll für den Bereich der Anwenderforschung (User Research) anhand von Beispielen beschrieben werden, welche Möglichkeiten bestehen, „psychologische" Anwenderfaktoren nutzerzentriert in einen agilen Kontext einfließen zu lassen.

Agile Softwareentwicklung und UX Design

Agile Ansätze sind aus der modernen Softwareentwicklung nicht mehr wegzudenken. Ebenso ist eine hervorragende User Experience (UX) im Großteil aktueller Softwareentwicklungsprojekte eine wesentliche Zielsetzung. Herausforderungen ergeben sich immer dann, wenn die beiden Aspekte agile Entwicklung und UX Design in der Praxis kombiniert werden sollen, da die jeweiligen Methoden und Prozesse nicht in jedem Fall miteinander kompatibel sind. Häufig tut sich beispielweise ein Spannungsfeld auf hinsichtlich der zeitlichen Erstreckung verschiedener Aktivitäten, die für einen reibungslosen Projektablauf parallel ablaufen oder effizient verzahnt werden sollen, da dies wegen sehr unterschiedlicher Zeitbedarfe nicht ohne Weiteres möglich ist. Wenn dieser Problematik nicht angemessen begegnet wird, sind Probleme für Projektverlauf und -resultat unvermeidlich. Im vorliegenden Beitrag soll für den UX Teilbereich der Anwenderforschung (User Research) dargestellt werden, welche Probleme sich in der agilen Praxis auftun und wie diesen angemessen begegnet werden kann.

User Research allgemein und im agilen Kontext

User Research zielt darauf ab, relevante Einblicke in die Lebenswelt von Anwendern zu gewinnen und die gewonnenen Erkenntnisse für die Gestaltung benutzerfreundlicher interaktiver Systeme nutzbar zu machen. Hierzu kommen beispielsweise Methoden wie der Anwenderbesuch zum Einsatz, beim dem der User Researcher sich in den realen Lebens- bzw. Arbeitskontext der Anwender begibt und intensive Beobachtungen durchführt, um auf diese Weise Informationen zu erheben, die für die Entwicklung maßgeblich sind, zum Beispiel hinsichtlich der Priorisierung von Anforderungen oder auch der Aufdeckung bisher unbekannter Anforderungen, die mittels alternativer Verfahren, wie zum Beispiel Interviews, nicht zutage traten. Ein wesentliches Anliegen von User Research ist es, über die Erstellung einfacher Requirements-Listen hinaus einen Einblick in die reichhaltige Lebenswelt von Anwendern, zum Beispiel hinsichtlich grundlegender Bedürfnisse, Motivationen und Erwartungen, zu ermöglichen und hierauf aufbauend den Entwicklungsprozess steuern zu können, um auf diese Weise letztendlich interaktive Systeme zu gestalten, die wiederum genau in der realen Lebenswelt der Anwender einen Mehrwert für diese bieten.

Vielen der „klassischen" User Research Maßnahmen, die den Anfängen des vergleichsweise recht jungen Felds UX Design entstammen, ist gemein, dass sie nicht explizit auf zeitliche Constraints der agilen Entwicklung ausgelegt sind, wo eine strikte Taktung von Prozessen, beispielsweise in Form von zweiwöchigen Sprints, die Regel ist. Dies bedeutet, dass die Einbindung von User Research Maßnahmen in agile Prozesse kein „Selbstläufer" ist, was im ungünstigsten Falle dazu führen kann, dass User Research Maßnahmen aufgrund scheinbar grundsätzlich fehlender Passung kaum oder gar nicht zur Anwendung kommen. Verschärft wird diese Problematik noch dadurch, dass ein Softwareentwicklungsprojekt auch ohne User Research Maßnahmen nicht ins Stocken gerät und so zunächst das offensichtliche Warnzeichen der Verzögerung von Arbeiten fehlt, das im Normalfall dazu führt, entsprechende Gegenmaßnahmen zu ergreifen. Vielmehr ist es so, dass ein Projekt ohne User Research Maßnahmen natürlich einen Aktivitätsbereich weniger hat, in dem sich Verzögerungen ergeben können. Die Kombination dieser genannten Faktoren kann schließlich dazu führen, dass User Research als „Luxus" oder sogar als Gefahr für die Durchführung eines Softwareentwicklungsprojekts wahrgenommen wird.

Herausforderungen für User Research im agilen Kontext

Aus den zuvor beschriebenen Umständen ergeben sich zwei wesentliche Herausforderungen für die Einbringung von User Research in den agilen Kontext.

- Grundsätzliche Widerstände und Hemmschwellen zur Einbringung von User Research in agile Projekte überwinden.
- User Research auf eine qualitativ gute und für den agilen Kontext geeigneten Art und Weise durchführen.

Diese beiden Punkte machen deutlich, dass zur Platzierung von User Research in agilen Projekten ein Vorgehen mit mehreren Facetten erforderlich ist. So kann es neben der Auswahl, Gestaltung und Durchführung geeigneter User Research Maßnahmen auch erforderlich sein, entsprechende Überzeugungsarbeit im Projektteam zu leisten, um ein Mindset zu erreichen, durch das die Anwendung von User Research auf einen fruchtbaren Boden fällt.

Summarisch gesehen besteht die Herausforderung für User Research im agilen Kontext also darin, die Beteiligten für die Perspektive des Anwenders zu sensibilisieren, eine Bereitschaft zu erzeugen, diese Perspektive ernsthaft und durchgängig für die Entwicklung zu berücksichtigen und geeignete Maßnahmen in das Methodenrepertoire aufzunehmen, mittels derer diese Perspektive im Rahmen der agilen Praxis nutzbar gemacht werden kann. Über all dem steht also die Frage: „Wie kann der Nutzer im agilen Prozess in den Mittelpunkt gerückt werden?“

Der Nutzer im Mittelpunkt

Selbst wenn Projekte für sich in Anspruch nehmen, „nutzerzentriert“ vorzugehen, so schlägt sich dies in der Praxis allzu oft in einer lediglich sehr oberflächlichen und/oder einseitigen Berücksichtigung von „Nutzerfaktoren“ nieder, beispielsweise indem nur einfach zu erhebende soziodemographische Daten berücksichtigt werden, wie zum Beispiel Alter oder Beruf der Nutzer, oder indem das Verhalten von Anwendern auf binäre Maße zum Erreichen/Nicht-Erreichen von Arbeitszielen reduziert wird. Dies wird der Reichhaltigkeit des Erlebens und Verhaltens von Menschen jedoch nicht gerecht und lässt viele für ein Softwareentwicklungsprojekt höchst relevante Aspekte außer Acht. Dass relevante „psychologische“ Anwenderfaktoren häufig unter den Tisch fallen liegt unter anderem daran, dass hierfür vergleichsweise wenig Bewusstsein bei Projektverantwortlichen und Softwareentwicklern besteht und dass, selbst wenn ein solches Bewusstsein in Ansätzen vorhanden ist, die Erfassung, beziehungsweise Messung, der entsprechenden Anwenderfaktoren als schwierig und zeitaufwändig eingeschätzt wird, so dass sie zugunsten der Erfassung „einfacher“ Faktoren, wie die oben genannten Alter und Beruf, vernachlässigt wird oder gleich ganz entfällt.

Ein weiterer Grund liegt im alltäglichen Zeitdruck in der der Softwareentwicklung, da dieser in vielen Fällen dazu führt, dass zu einem Softwareprodukt erst sehr spät im Entwicklungsprozess – wenn überhaupt – Nutzerfeedback eingeholt wird. Und selbst wenn dies dann der Fall ist, handelt es sich häufig um „klassisches“ Prototyping in Form von implementierungstechnisch relativ weit fortgeschrittenen Systemen mit dem Ziel, „greifbare“ Lösungen für konkrete Kundenanforderungen auf *funktionaler* Ebene zu kommunizieren und abzugleichen. Diese reduzierte Sichtweise wird auch durch die zuvor genannten (falschen) Erwartungen hinsichtlich des Aufwands zur Erhebung psychologischer Nutzerfaktoren gefördert. Im Rahmen eines solchen „klassischen“ Prototypings sind eine Justierung des Prozesses und insbesondere nachgelagerte Anpassungen und Änderungen, die sich aus der Untersuchung der Nutzerfaktoren ergeben, wie alle anderen Änderungen auch, erfahrungsgemäß aufwändig und teuer – sofern sie in einem (weit) fortgeschrittenen Projekt überhaupt noch möglich sind.

Durch die angemessene Berücksichtigung von „psychologischen“ Nutzerfaktoren kann ein Entwicklungsprojekt jedoch frühzeitig auf eine viel umfassendere und stabilere Informationsbasis gestellt werden, die die Gestaltung von innovativen und nutzerfreundlichen Lösungen erlaubt. Als Beispiele für psychologische Nutzerfaktoren seien hier Bedürfnisse und Erwartungen genannt. Beiden Konzepten ist gemein, dass es sich dabei *nicht* um funktionale Anforderungen an ein User Interface handelt, sondern dass sie auf einer sehr viel grundlegenderen Ebene angesiedelt sind, die wiederum funktionale Anforderungen von Anwendern wesentlich formen und beeinflussen kann.

Stark vereinfacht gesagt sind es Bedürfnisse, die den Menschen antreiben und ihn dazu bringen, Handlungen zu vollziehen, die auf die Befriedigung eben dieser Bedürfnisse ausgelegt sind. Da auch die Nutzung interaktiver Systeme einen Teilbereich des menschlichen Handelns darstellt, darf angenommen werden, dass auch hier die angemessene Berücksichtigung von Bedürfnissen angezeigt ist. Zur Klassifikation von Bedürfnissen schlagen verschiedene Autoren unterschiedliche Modelle vor, wobei als eines der bekanntesten die Bedürfnispyramide von Maslow (1943) gilt, die in ihrer ursprünglichen Form physiologische Bedürfnisse, Sicherheitsbedürfnisse, soziale Bedürfnisse, Individualbedürfnisse und Selbstverwirklichung in dieser aufeinander aufbauenden Reihenfolge enthält. Speziell für den Kontext der Gestaltung von interaktiven Systemen schlagen Diefenbach und Hassenzahl (2017) die Bedürfnisse Kompetenz, Verbundenheit, Bedeutsamkeit, Stimulation, Sicherheit, Popularität und Autonomie vor. Die Rolle der einzelnen Bedürfnisse und die Möglichkeiten zu deren Befriedigung fallen unterschiedlich aus, je nachdem, ob ein interaktives System freiwillig genutzt wird oder dessen Nutzung obligatorisch ist. Aber auch bei der obligatorischen Nutzung eines interaktiven Systems, beispielsweise einer Produktivanwendung im Rahmen der Berufsausübung, kann davon ausgegangen werden, dass auf der Grundlage von Wissen über Anwenderbedürfnisse motivationale Effekte bei der Nutzung erzielt werden können, die sich wiederum auf die Benutzerfreundlichkeit des betreffenden Systems auswirken. Liefert ein System beispielsweise zeitnah und präzise Informationen zu Prozessfortschritten, so kann dies zu einer Kompetenzerfahrung des Anwenders beitragen, was wiederum ein Baustein dazu ist, das entsprechende Bedürfnis zu befriedigen. Der Stellenwert der einzelnen Bedürfnisse variiert hierbei zwischen Anwendern beziehungsweise Anwendergruppen und zusätzlich auch über die Zeit.

Das Konzept „Erwartungen“ bezieht sich im Kontext dieses Artikels auf die Wünsche und Vermutungen, die ein Anwender bezüglich der Art und Weise hat, wie er mit einem interaktiven System interagiert. Es handelt sich

also nicht um konkrete *Lösungs*wünsche von Anwendern, sondern um Vorlieben und Mutmaßungen, wie das interaktive Wechselspiel mit einem System sich aus der persönlichen Sicht des Anwenders vollziehen soll, also um „Interaktionserwartungen“. Als Klassifikationsschema für diesen Bereich kann auf das Interaktionsvokabular zurückgegriffen werden, das von Diefenbach und Hassenzahl (2017) vorgestellt wird. Hierbei handelt es sich um ein Set von elf Begriffs-Gegensatz-Paaren, die dazu dienen, Interaktionen unabhängig von der konkreten Form ihrer Umsetzung zu beschreiben.

langsam	schnell
abgestuft	fließend
sofort	verzögert
gleichförmig	gegensätzlich
stabil	unbeständig
vermittelt	direkt
räumliche Trennung	räumliche Nähe
ungefähr	präzise
behutsam	kraftvoll
beiläufig	gezielt
offenkundig	verborgen

Abbildung 1: Interaktionsvokabular nach Diefenbach & Hassenzahl (2017)

Mit diesem Vokabular kann eine (erwartete) Interaktion beschrieben werden, zum Beispiel indem die Ausprägungen auf jedem der elf Begriffspaare angegeben werden, etwa auf jeweils einer Likert-Skala, um auf diese Weise dann ein „Interaktionsprofil“ zu gewinnen, das vom Anwender erwartet wird. So ist es beispielweise denkbar, dass ein Anwender, dem wie zuvor beschrieben an einer Kompetenzerfahrung liegt, von einer Interaktion erwartet, dass diese sich unter anderem durch die Eigenschaften „schnell“, „sofort“ und „offenkundig“ auszeichnet.

Nutzerzentrierte Messungen im agilen Kontext

Die zuvor beschriebenen Bedürfnisse und Interaktionserwartungen haben eine hohe Relevanz für die Gestaltung interaktiver Systeme, ohne dass jedoch unmittelbar konkrete Gestaltungslösungen daraus abzuleiten sind. Vielmehr können die Bedürfnisse und Interaktionserwartungen als wichtige Inspirationsquellen für Überlegungen, als Eckpfeiler für Gestaltungsarbeiten sowie zur Bewertung von Lösungsansätzen herangezogen werden. Im agilen Kontext stellt sich wie zuvor beschrieben speziell auch die Frage, wie derartige Aspekte auf geeignete Weise überzeugend und nachhaltig in den Entwicklungsprozess Eingang finden können. Wichtig ist hierbei unter anderem, dass Messmethoden angewendet werden können, die sich schnell und ressourcenschonend durchführen lassen, um kompatibel mit den zeitlichen Aspekten eines strikt getakteten agilen Entwicklungsprojektes zu sein.

- **Erhebung von Bedürfnissen als Grundlage für Design-Überlegungen**

Basierend auf dem von Diefenbach und Hassenzahl (2017) vorgestellten Bedürfnismodell kann eine Exploration der Relevanz von Bedürfnissen in der Nutzerzielgruppe relativ unkompliziert erfolgen. Hierzu können „Bedürfniskarten“ eingesetzt werden, die jeweils eines der Bedürfnisse benennen und in kurzer Form beschreiben. Um einen Eindruck der subjektiv empfundenen Bedürfnisprioritäten bei Anwendern zu gewinnen, können Anwender beispielsweise gebeten werden, die Karten in eine Rangfolge zu bringen, um auf diese Weise darzustellen, was ihnen wichtig ist und auf dieser Grundlage Tendenzen für Anwender oder Anwendergruppen identifizieren zu können. Das Ziel besteht bei einem solchen Unterfangen nicht in einer objektiven und finalen Quantifizierung von Anwenderbedürfnissen. Hierzu sind Menschen zu unterschiedlich, Bedürfnisprioritäten zeitlich und kontextabhängig variabel und letztlich ist auch keines der Bedürfnisse irrelevant, sondern jedes einzelne ist ein „Antrieb“ für menschliches Verhalten. Die Erhebung von Bedürfnissen dient vielmehr dazu, die Bedürfnisperspektive für den agilen Prozess überhaupt einmal anschaulich und greifbar zu machen. Dies stellt dann eine wertvolle Grundlage für Brainstormings und vergleichbare Überlegungen dar, die unter anderem darauf abzielen, den Raum möglicher Lösungen zu explorieren. Kompatibel mit agilen Ansätzen ist es leicht möglich, die Erkundung von Anwenderbedürfnissen mit zeitlich minimalem Aufwand durchzuführen. Je nach (technischen) Möglichkeiten kann die Arbeit mit den Bedürfniskarten auch in elektronischer Form erfolgen, so dass Anwender räumlich und zeitlich autonom die Karten in eine Reihenfolge bringen, wobei die Ergebnisse dann automatisiert gesammelt und aggregiert werden können. Steht im agilen Prozess noch etwas mehr Zeit für User Research zur Verfügung, so kann das Arbeiten mit den Bedürfniskarten (vor Ort oder remote) auch im Rahmen einer Interviewsituation erfolgen, in der die Anwender dazu aufgefordert werden, ihre Priorisierung der Karten zu erläutern. Dies ermöglicht die Sammlung zusätzlicher qualitativer Daten in Form von Aussagen und Anekdoten, die informativ für die folgenden Projektaktivitäten sind und die mit Projektentscheidungen und (Zwischen-)Ergebnissen abgeglichen werden können, um die Anwenderperspektive durchgängig im Prozess berücksichtigen zu können.

Unabhängig von der konkreten Art der Erhebung besteht ein wesentlicher Zweck einer derartigen Beschäftigung mit Anwenderbedürfnissen darin, den Beteiligten an der agilen Entwicklung einen leicht durchführbaren Weg aufzuzeigen, wertvolle und anregende Erkenntnisse über Anwenderbedürfnisse zu gewinnen, die Diskussionen und Denkprozesse in Gang setzen, ohne dass hierfür Metho-

den erforderlich wären, die mit einem agilen Entwicklungsprozess nicht verträglich sind. Hierdurch wird auch eine Grundlage dafür geschaffen, derartige Explorationen nicht nur punktuell, sondern projektbegleitend immer wieder durchzuführen.

- **Erhebung von Interaktionserwartungen zum Abgleich mit Design-Lösungen**

Ähnlich wie Bedürfnisse können Interaktionserwartungen zu verschiedenen Zeitpunkten im Projektverlauf erhoben werden, um unterschiedliche Zielsetzungen zu verfolgen. So kann beispielsweise in einem Fall, in dem es ein bestehendes Produkt gibt, das optimiert werden soll, von Anwendern ein Interaktionsprofil erstellt werden, das beschreibt, wie der Anwender gerne mit einer zukünftigen Version des Produkts interagieren würde. Haben die Anwender mit dem bestehenden Produkt bereits Erfahrungen gesammelt, so ist dies eine gute Grundlage für die Erstellung eines solchen Interaktionsprofils, da sich die Anwender einfach in die Nutzungssituation für das zukünftige Produkt hineinversetzen können. Grundsätzlich ist ein solcher Ansatz aber auch möglich, wenn es sich um ein neuartiges Produkt handelt und es den Anwendern ermöglicht wird, sich in die zukünftige Nutzungssituation zu versetzen, beispielsweise indem sie gebeten werden, sich die Situation der Nutzung eines vergleichbaren Produkts detailliert vorzustellen. Unterstützt werden kann dies beispielsweise durch die Beschreibung entsprechender Szenarien in Wort und/oder Bild, etwa durch die Betrachtung von Illustrationen der zukünftigen Nutzungssituation.

Weiterhin kann das Interaktionsprofil zur Evaluation eines bestehenden Produkts beziehungsweise eines Prototypen genutzt werden, um zu bewerten, inwieweit die Lösungsgestaltung den Erwartungen der Anwender entspricht.

Ähnlich wie bei der Erhebung der Bedürfnisse kann die Ermittlung von Interaktionserwartungen online oder offline erfolgen, abhängig von der zur Verfügung stehenden Zeit und den technischen Rahmenbedingungen. In bestimmten Fällen kann die Online-Erhebung direkt im jeweiligen Interface erfolgen, beispielsweise indem die elf Begriffspaare des Interaktionsprofils eingeblendet werden, verbunden mit der Bitte an den Anwender, die zuvor erlebte Interaktion zu bewerten, entweder für jedes Begriffspaar in „binärer“ Form oder feiner aufgelöst in Form einer Likert-Skala.

Durch die Erhebung von Interaktionserwartungen kann die Anwenderperspektive über die Berücksichtigung von Bedürfnissen hinaus durch eine Perspektive erweitert werden, die in einen engeren Bezug zur Lösungsgestaltung gesetzt werden kann. Dies bedeutet nicht, dass diese Perspektive damit „besser“ ist als die Bedürfnisperspektive, sondern lediglich, dass sie einen zusätzlichen Erkenntnisauschnitt bezüglich des Wissens über die Anwender darstellen kann.

- **Implizite Messungen: Erfassung des Anwenderverhaltens bei der Nutzung**

Die zuvor beschriebenen Ansätze zur Erfassung der Perspektive des Anwenders können als „explizite“ Messungen aufgefasst werden, da die Anwender jeweils mit Fragen bezüglich der zu erhebenden Konstrukte konfrontiert werden und sie auf diese Rückmeldung geben. Zusätzlich kann die Wissensbasis über den Anwender auch noch mittels impliziter Messungen erweitert werden, die auf einer Erfassung des Anwenderverhaltens beim Umgang mit einem System beruhen. Mit impliziten Messungen werden bestimmte Anwenderfaktoren erfasst, ohne dass dem Anwender jedoch explizite Fragen danach gestellt werden. Dies bedeutet auch, dass die Bewertung der erhobenen Maße häufig einen größeren interpretativen Anteil erfordert als dies beim Umgang mit explizit erhobenen Maßen der Fall ist. Weiterhin ist zur Erhebung dieser Maße in der Regel ein existierendes System oder zumindest ein Prototyp erforderlich, um ein entsprechendes Modul zur Verhaltensmessung einzubauen und diese Daten zu speichern oder an eine Datenbank zu übermitteln, von wo sie ausgelesen werden können. Prinzipiell sind auch Datenerhebungen in Form von Beobachtungen der Anwender bei ihrer Interaktion mit einem System denkbar. Dies ist jedoch vor dem Hintergrund der zuvor beschriebenen Rahmenbedingungen, denen User Research in einem agilen Kontext unterliegt, oft nur sehr einschränkt möglich, so dass automatisierte Ansätze in diesem Kontext besondere Vorteile bieten können.

Ein mögliches implizites Maß ist etwa die Häufigkeit des Aufrufs der Systemhilfe pro Zeiteinheit. Als Systemhilfe gilt in diesem Kontext nicht nur eine etwaige dedizierte Hilfe-Sektion im betreffenden Interface, sondern zum Beispiel auch die Information, die per Kontext-Button (etwa „?“ oder „i“) zur Verfügung gestellt wird und mittels derer sich der Anwender bei Bedarf über das System und dessen Funktionen informieren kann. Dadurch, dass der Aufruf derartiger Hilfe-Elemente in Bezug zum zeitlichen Verlauf der Systemnutzung gesetzt wird, ist es möglich, den Verlauf der Intensität der Hilfenutzung zu messen anstatt nur ein schlichtes summarisches Häufigkeitsmaß zu erheben. Auf diese Weise eröffnet sich ein Ansatz, Episoden zu identifizieren, in denen der Anwender möglicherweise besonders verunsichert ist und diese mit den dann gerade genutzten Systemfunktionen in Bezug zu setzen. Wie erwähnt ist dies mit einem interpretativen Anteil behaftet, da es natürlich auch weitere Gründe gibt, eine Systemhilfe in Anspruch zu nehmen, zum Beispiel das prinzipielle Interesse an Funktionen, die das betreffende System bietet. Es ist daher empfehlenswert, solche impliziten Maße mit weiteren Datenquellen und Informationen zu kombinieren, um die Verlässlichkeit der Interpretation zu erhöhen. Nicht zuletzt kann auch in diesem Zusammenhang Wissen über relevante Anwenderbedürfnisse zur Interpretation der Daten bedeutsam sein, zum Beispiel, wenn bekannt ist, dass die Anwender ein vergleichsweise hohes Sicherheitsbedürfnis haben und eine intensive Hilfe-Nutzung dann anders zu interpretieren ist als bei einer Anwendergruppe, die sich durch ein

vergleichsweise geringes Sicherheitsbedürfnis auszeichnet.

Weiterhin kann bei der Hilfenutzung speziell, aber auch bezogen auf die generelle Nutzung eines Systems durch die Anwender, auch ein Modell des idealen Nutzungsverhaltens zum Abgleich mit den implizit erhobenen Maßen herangezogen werden. In seiner einfachsten Form besteht ein solches Modell schlicht aus einer oder mehreren Idealsequenzen, in denen Systembereiche und Funktionen aufgerufen werden sollten, um eine bestimmte Aufgabe so effizient wie möglich zu erledigen. Auf der Grundlage eines solchen Modells kann dann ein Maß für die Abweichung von Anwendern von diesem Idealpfad ermittelt werden, zum Beispiel indem bestimmt wird, wie viele Bereiche, die nicht auf dem Idealpfad liegen, vom Anwender aufgerufen werden oder auch wie oft Bereiche wiederholt aufgerufen werden, obwohl dies für das Beschreiten des Idealpfads nicht erforderlich ist. Für derartige Abgleiche mit einem Idealmodell ist es bedeutsam, ob die implizite Messung der Anwenderfaktoren in einer kontrollierten Situation passiert, in der die Aufgaben vorgegeben werden, oder ob die Nutzung des Systems unkontrolliert vonstattengeht, in dem Sinne, dass die Entscheidung für eine Aufgabe und der Beginn der Bearbeitung derselben völlig im Ermessen des Anwenders liegt. In letztgenannter Situation stellt sich in der Praxis die besondere Herausforderung, zu bestimmen, zu welchem Zeitpunkt die Bearbeitung einer Aufgabe für die Interpretation angenommen werden kann und ob der Anwender seinen mentalen Fokus durchgängig auf die Aufgabe setzt oder sich zwischendurch anderen Aufgaben oder Aspekten des Systems zuwendet, bevor er die Aufgabenbearbeitung fortsetzt und abschließt. Insbesondere wenn die Nutzung des Systems im Ermessen des Anwenders liegt und ansonsten unbeobachtet stattfindet, kann der Abgleich mit einem Idealmodell schwierig sein. Wird ein System oder ein Prototyp an viele Anwender ausgeliefert und besteht die Möglichkeit, auf diese Weise viele Messpunkte zu erheben, so können Data Mining Ansätze hier Lösungsmöglichkeiten bieten. Derartige Ansätze können dazu dienen, Nutzungsmuster und Nutzungsepisoden aus der Datenbasis zu extrahieren, auch ohne dass von vornherein bekannt ist, wann Nutzer eine konkrete Aufgabe beginnen, unterbrechen oder beenden. In einem Folgeschritt können die durch das Data Mining identifizierten aussagekräftigen Nutzungsepisoden dann dem zuvor beschriebenen Abgleich mit dem Idealpfad und weiteren Analysen unterzogen werden, um das Verhalten der Anwender im Umgang mit dem System zu beleuchten.

Im agilen Kontext können sich zur Messung des Anwenderverhaltens somit Ansätze als praktisch erweisen, bei denen die Erhebung relevanter Daten (weitestgehend) automatisiert ablaufen kann. Je nach Rahmenbedingungen, Fragestellungen und eingesetzten Verfahren kann hierbei der Schwerpunkt entweder auf eher qualitativen oder auf eher quantitativen Herangehensweisen liegen.

Der User Researcher im agilen Kontext

Die zuvor beschriebenen Ansätze stellen Möglichkeiten dar, wie Wissen über den Anwender nachhaltig und gewinnbringend in einen agilen Entwicklungsprozess integriert werden kann. Um dies erfolgreich umzusetzen muss nicht zuletzt auch der Einbindung der Rolle des User Researchers im agilen Kontext besondere Aufmerksamkeit gewidmet werden, da eine solche im „klassischen“ agilen Ansatz nicht vorhanden ist. Die Einführung neuer Rollen birgt immer die Gefahr der Ablehnung; daher ist es essentiell, dass das Entwicklerteam auf das Mindset der User Research eingestimmt wird, damit die neue Rolle des User Researchers nicht als Fremdkörper oder getarnte Kontrollinstanz angesehen wird, sondern als Instrument der Qualitätssicherung auf fachlicher, das heißt insbesondere dem Anwender zugewandten, Ebene. Dies kann unter anderem dadurch unterstützt werden, dass der Mehrwert der entsprechenden Aktivitäten für das Entwicklerteam demonstriert wird, beispielsweise durch die schnelle Generierung relevanter Erkenntnisse und Denkanstöße mittels der zuvor beschriebenen Ansätze. Die Akzeptanz und das Verständnis vorausgesetzt kommen keine zusätzlichen Aufgaben oder erhöhte Aufwände auf das eigentliche Entwicklerteam zu, lediglich die Sichtweise auf die Arbeitsgrundlage und die Arbeitsergebnisse ist eine andere: (a) Keine Anforderungen, sondern Bedürfnisse bestimmen das Backlog. (b) Es werden keine Features programmiert, sondern Lösungen für die Bedürfnisse und Interaktionserwartungen der Nutzer kreiert. Zusätzlicher Aufwand entsteht allerdings auf der Seite der Planung, da die User Research Maßnahmen zum Beispiel vom Product Owner als weiterer Aktivitätsbereich in die Taktung der Sprint-Planung einzupassen sind.

Ein radikalerer Ansatz wird von Gothelf und Seiden (2016) aus dem Umfeld von Lean UX vorgeschlagen, nämlich (User) Research als weitere Disziplin in einem interdisziplinären agilen Team zu verankern. Dies erfordert jedoch über die Akzeptanz und das Verständnis für User Research hinaus ein entsprechendes Skill Set bei allen Mitgliedern des Teams.

Praxiserfahrungen bei der Integration von User Research in einen agilen Kontext

Grundlegende Erkenntnisse und Erfahrungen bei der Integration von User Research in einen vorhandenen agilen Kontext werden aktuell in dem vom Bundesministerium für Bildung und Forschung (BMBF) geförderten Verbundprojekt „Optimierung für mobile Applikationen“ (Opti4Apps) gewonnen. Im Rahmen dieses Projekts wird die „klassische“ agile Entwicklung einer mobilen Anwendung in einem realen Umfeld durch die Anwendung von Lean UX Prinzipien und insbesondere durch User Research Maßnahmen angereichert.

Bisher stellte sich dabei unter anderem heraus, dass die Abkehr von einer lösungsorientierten Denkweise hin zu einer bedürfnisorientierten genau den vermuteten Einblick in die Lebenswelt des Anwenders mit sich bringt und damit einhergehend eine stärkere Identifizierung und

Bindung des Entwicklerteams mit den relevanten Personas ermöglicht. Somit können insbesondere Entwurfsentscheidungen schneller und zuverlässiger getroffen werden, da diese nicht mehr eigene Annahmen oder Interpretationen des Teams widerspiegeln, sondern auf Aussagen der betroffenen Anwender beruhen und daher begründet sind. Ebenso ist eine zielgerichtetere, weil an den Bedürfnissen der Anwender ausgerichtete, Release-Planung möglich.

Im weiteren Projektverlauf sollen implizite Messungen direkt in mobilen Anwendungen durchgeführt werden, um das Nutzerverhalten beim Umgang mit Anwendungen detaillierter zu erfassen. Durch die gemeinsame Auswertung und Interpretation der Ergebnisse aus expliziten und impliziten Messungen ist zu erwarten, dass der Erfolg von neuen oder geänderten Funktionen oder Funktionsgruppen beziehungsweise eines gesamten Releases geprüft und bewertet werden kann. Ziel ist es, eine Methodik zu entwickeln, mit der die durch die Messungen gewonnenen Erkenntnisse über die Anwender systematisch in die iterative Lösungsgestaltung einfließen können.

Fazit

Die ernsthafte und durchgängige Berücksichtigung der Anwenderperspektive bedeutet mehr als nur die Erhebung schlichter demographischer Zielgruppencharakteristika. Soll der Anwender ernsthaft und nachhaltig in das Zentrum der Entwicklung gerückt werden, so sind die Beschäftigung mit „psychologischen“ Anwenderfaktoren wie Bedürfnissen und Interaktionserwartungen sowie die Untersuchung des Anwenderverhaltens beim Umgang mit den betreffenden interaktiven Systemen essenziell. Wie gezeigt wurde ist dies auch im Rahmen von agilen Entwicklungsprojekten mit ihren besonderen Anforderungen hinsichtlich zeitlicher Taktung von Aktivitäten möglich. Dies stellt darüber hinaus eine Bereicherung solcher Projekte dar, indem eine deutlich bessere Grundlage für die Gestaltung und Evaluation nutzerfreundlicher Lösungen geschaffen wird.

Es ist wünschenswert, möglichst viele Ansätze zur Erfassung psychologischer Anwenderfaktoren im Kontext agiler Projekte zu erproben, um abhängig von konkreten Projektkontexten Stärken und Schwächen unterschiedlicher Herangehensweisen zu erkennen und auf diese Weise ein Methodenrepertoire zu entwickeln, aus dem für einen Projektkontext das jeweils passende Methodenset zusammengestellt werden kann. Dies betrifft sowohl die Möglichkeiten zur expliziten Messung von Anwenderfaktoren wie auch deren (automatisierte) implizite Messung.

Für jeden Kontext und jede Methode sollte schließlich auch reflektiert werden, welche Rolle dem User Researcher bei der praktischen Einbindung derartiger Aktivitäten in ein Projekt zukommt und wie die Rolle so gestaltet werden kann, dass sie im agilen Team möglichst gut angenommen wird.

Danksagung

Teile dieses Artikels entstanden im Rahmen des Forschungsprojekts „Optimierung für mobile Applikationen“ (Opti4Apps), gefördert durch das Bundesministerium für Bildung und Forschung (BMBF), Förderkennzeichen 02K14A180.

Weblink

Opti4Apps (Verbundprojekt): http://www.opti4apps.de

Literatur

Diefenbach, S. & Hassenzahl, M. (2017). Psychologie in der nutzerzentrieren Produktgestaltung. Berlin: Springer.

Gothelf, J. & Seiden, J. (2016). Lean UX: Designing Great Products with Agile Teams (2. Aufl.). Sebastopol, CA: O'Reilly.

Maslow, A. (1943). A Theory of Human Motivation. Psychological Review, 50(4), 370–396.

Autoren

Dr. Markus Weber
Head of Usability Analysis
Centigrade GmbH
markus.weber@centigrade.de

Daniel Ried
Projektleiter Forschungs- und Förderprojekte
Heidelberg Mobil International GmbH
daniel.ried@heidelberg-mobil.com