

# Wie werde ich Linux Administrator\*in bei Thalia?

Wie sieht eigentlich der Bewerbungsprozess bei einem Buchhändler aus? Was ist Thalia wichtig? Welcher Dresscode wird erwartet? Wird ein Assessment Center durchgeführt?

Transparenz gehört zu unseren Werten, und so haben wir uns überlegt, euch einen Einblick in unseren Bewerbungsprozess zu geben. Das wichtigste vorab: Wir bei Thalia sind auch nur Menschen und freuen uns immer über tolle neue Kollegen\*innen.

## Der erste Kontakt

Am Anfang steht immer die erste Kontaktaufnahme. Unsere aktuellen Stellenausschreibungen findest du neben unserer Karriereseite z.B. auch auf Stepstone, Xing, LinkedIn oder sonstigen Portalen. Wenn etwas Spannendes dabei ist, dann schicke uns deine Unterlagen entweder direkt oder über die Tools der jeweiligen Seite. Alle Wege führen dich zu den freundlichen Leuten von der Thalia Personalabteilung. Sie werden dich im weiteren Verlauf begleiten, Termine vereinbaren, deine Fragen beantworten, Feedback entgegennehmen/geben usw.

## Das erste Vorstellungsgespräch

Ein sehr spannender Moment für uns ist es, dich persönlich im ersten Gespräch kennen zu lernen. Das Gespräch findet im Thalia Bürogebäude in Münster statt

und dauert ca. 1h - 1,5h. Von Thalia sind meistens eine Person aus der Personalabteilung, ein/e Fachspezialist\*in und ein/e Teamleiter\*in dabei. Keiner der Thalia Mitarbeiter\*innen wird Geschäftskleidung tragen. Wir sind in der Regel leger gekleidet. Ziehe einfach etwas an, in dem du dich wohl fühlst.

Nach der Begrüßung werden wir dir als erstes das „Du“ anbieten. Im Bereich IT-Operations - und in vielen anderen Abteilungen von Thalia - duzen wir uns. Warum sollten wir es also in den Gesprächen anders machen?

Zu Beginn werden wir uns vorstellen und etwas über Thalia, den Bereich IT-Operations an den verschiedenen Standorten von Thalia und natürlich auch zu uns selber erzählen. Dann sind wir aber sehr gespannt auf dich! Erzähle uns einfach wer du bist, was du bislang gemacht hast und warum du glaubst, richtig gut auf die ausgeschriebene Stelle zu passen.

Der Übergang in das Fachgespräch ist dann fließend. Wir wollen gerne verstehen, inwieweit du unsere eingesetzten Technologien schon beherrscht bzw. schon mal gesehen hast. Und eines ist sicher: Du wirst nicht alles können und beherrschen. Und das ist auch OK. Es gibt nichts, was man nicht lernen könnte. Also hab den Mut auch mal zu sagen, dass du etwas noch nicht kennst oder in einer Technologie noch Lücken hast.

Es geht uns aber nicht nur um Technologie. Besonders wichtig ist uns auch, wie sehr du mit dem Team harmonierst. Was nützt uns der/die beste Techniker\*in der Welt, wenn es im Team nicht funktioniert? Nicht zuletzt ist es auch für dich sicherlich wichtig, dass du dich im neuen Team wohl fühlst. Spaß an der Arbeit ist für uns keine Floskel.

Ein weiterer wichtiger Punkt ist bei uns die Methodik. Im eCommerce von Thalia setzen wir auf agile Methoden. Auch wenn es hilfreich ist, schon mal was über Agilität gehört zu haben oder eine grobe Idee zur DevOps Kultur zu haben, so musst du kein SCRUM Master sein, um dich zurechtzufinden. Wenn du bei Thalia startest, dann wird es u.a. einen Workshop zum Thema „Agilität“ geben.

Im Gespräch stellen wir viele Fragen zu all diesen Themen. Natürlich kannst du auch schon während des Gesprächs deine stellen. Sollten dennoch Fragen bei dir offen sein, so ist nachher noch einmal explizit Zeit, um all deine offenen Punkte zu beantworten.

Nach etwa einer Stunde ist das erste Gespräch auch schon vorbei. Und wir haben einen ersten gegenseitigen Eindruck gewonnen. Genau wie wir solltest auch du den ersten Eindruck verarbeiten und dir mit den neu gewonnen Informationen zu Thalia überlegen, ob du richtig Lust auf den Job hast.

Zeitnah nach dem Gespräch wirst du wieder kontaktiert, und du erhältst Feedback zu der ersten Runde.

## **Die zweite Vorstellungsrunde**

Wenn es weiter geht, dann werden dich in der zweiten Runde weitere Personen kennenlernen wollen. So können z.B. die IT-Leitung oder weitere Team-Mitglieder beim Gespräch dabei sein. Im Gespräch stellen wir weitere Fragen. Auch kann es vorkommen, dass dich in der zweiten Runde eine kleine Aufgabe erwartet, die du natürlich zuhause in Ruhe vorbereiten kannst und im Gespräch vorstellst. Uns ist dabei nicht wichtig, ob die Aufgabe perfekt gelöst wurde. Wir wollen verstehen, wie du mit Aufgaben dieser Art umgehst.

Wenn du möchtest, führen wir dich auch gerne noch einmal durch die Räumlichkeiten, so dass du dir selber einen Eindruck von den potenziellen Arbeitskollegen\*innen und vom möglichen neuen Arbeitsplatz machen kannst. Gerne können wir auch einen Schnuppertag vereinbaren.

Mit dem zweiten Gespräch hast du dann auch schon das „Schlimmste“ überstanden. Wenn wir uns einig sind, dann stimmt die Personalabteilung mit dir noch die vertraglichen Rahmenbedingungen ab. Hier geht es dann um Themen wie z.B. Urlaub, Vorteile durch Thalia, Gehalt, Unterschriften und sonstige vergleichbare Themen.

# Es geht los!



Ab jetzt freuen wir uns auf den Tag, an dem du startest, und bereiten alles für dich vor. Am Tag 1 bei Thalia begrüßen dich die Personalabteilung und weitere Personen, die zusammen mit dir durchstarten wollen. In einer Willkommensveranstaltung informieren sie dich über die wichtigsten organisatorischen Themen. Danach kommst du in dein neues Team, wo dein vorbereiteter Arbeitsplatz dich bereits erwartet. Während der Einarbeitung in die technische Welt von Thalia begleitet dich ein erfahrener und fester Ansprechpartner aus dem Team. Jetzt kannst du bei Thalia mit deinen Ideen richtig durchstarten.

## Hast du Lust den Prozess einmal selber auszuprobieren? ☐

---



# <0110/> hackathon@thalia in Münster 2019

*Neue Technologie, die Dinge spielerisch einfach macht! Neuer Inhalt, der zählt! Neues Modell, das uns und unsere Prozesse ein bisschen besser macht. Neue Idee, die du entdecken möchtest! Welche Themen fallen dir ein? Dieser Hackathon kennt keine Grenzen.*

Für den diesjährigen Hackathon haben wir uns bewusst für ein offenes Format entschieden und damit getreu dem agilen Prinzip ‚inspect and adapt‘ das Feedback zur Themeneinschränkung beim [letzten Mal](#) berücksichtigt.

Die Beteiligung sprach für sich: 18 motivierte Menschen, 5 interessante Pitch-Vorschläge und 1 Tag Zeit.



Pitch der Ideen



Der Preis!

Das Ergebnis: 3 aus 5. Die Gruppen hatten sich schnell formiert und ihren Weg jeweils nach Bullerbü, Springfield oder Entenhausen in der Meeting-Area in Speicher 6 gefunden.

Die Türen in Winterfell waren zu jeder Zeit für den uneingeschränkten Zugriff auf die eiskalten Getränke weit geöffnet. Ebenso der Blick auf den Siegerpreis, der in den Katakomben unterhalb der Burg ... genug davon, ich schweife ab.

---

Kommen wir zu den  
Themen:



<https://www.okd.i>

## Projekt: OKD - Openshift Kubernetes Distribution

### *Worum ging es?*

Docker – auch bei uns ein Thema. Wie können wir die Administration und Orchestrierung unserer wachsenden Container-Infrastruktur besser und effizienter

managen? Dieser Fragestellung will das Team mit 8 Leuten auf den Grund gehen und schaut sich das OpenSourceprodukt OKD genauer an.

[OKD](#) ist ein Upstream-Projekt für Openshift aus der Produktpalette von Red Hat und setzt auf [Kubernetes](#) auf.

## ***Ergebnisse***

oc cluster up – mit diesem Kommando ist das Team in die Evaluierung gestartet. Das hat schon mal gut und vor allem schnell funktioniert. Ein SingleNode Cluster wird mittels Minishift und VirtualBox aufgebaut. In einem Gitlab-Projekt wurde eine Testapplikation mit Webfrontend und ein Dockerfile angelegt und deployed. Parallel haben Teammitglieder versucht, Minishift auf einem MacOS-Gerät zum Laufen zu bekommen. Nach anfänglichen Konflikten mit der installierten VirtualBox gelang auch dies.

Während der Abschlusspräsentation konnte eine schlanke Webanwendung (Login-UI) gezeigt, eine neue Version gebaut und über eine eigene Jenkins-Pipeline deployed werden. Et voilà – es funktioniert.

Während man mit Kubernetes alles bauen kann, was man so haben möchte (Load Balancer, Network Policies, Benutzerverwaltung, ...), bringt OKD alle diese Features „out of the box“ mit. OKD ist somit eine gute open source Alternative zu OpenShift.



<https://developers.google.com/awareness>

# **Projekt: Google Awareness API**

***Worum ging es?***

Unser eCommerce-Standort in Münster befindet sich in der Speicherstadt, wir arbeiten und treffen uns in unterschiedlichen Gebäuden. Es gibt zur Mittagszeit eine Kantine auf dem Campus, die ‚Hofrunde‘ ist ein allseits anerkanntes Kommunikationsinstrument und der Parkplatz ist ebenfalls auf dem Gelände. Das Team wollte analysieren, ob die [Google Awareness API](#) für die Erfassung von Zeiten genutzt werden kann, um somit anhand von Standorten zwischen Freizeit und Arbeit zu unterscheiden. Im ersten Schritt sollte dies auf Basis der geographischen Positionen von 2 Speichergebäuden erfolgen.

## ***Ergebnisse***

Die Zeit war eng bemessen. Ein Teil des Teams hat die App installiert und ein anderes die REST-Schnittstellen zum Sammeln und Anzeigen der Tracking-Daten entwickelt. Erkenntnis nach 6 Stunden: Die Anwendung wirkte instabil. Der Energieverbrauch durch fortlaufendes, aktives Ansprechen der App war hoch, d.h. durch fehlende Automatisierung wurden die genutzten Geräte stärker beansprucht. Eventuell war auch der gedachte Anwendungsfall hier einfach nicht der richtige. An dieser Stelle unkritisch - bei Thalia gibt es Vertrauensarbeitszeit:-).

Einige Java-Entwickler waren Teil des Teams und konnten so die genutzte Programmiersprache [Kotlin](#) kennenlernen und im Zuge der REST-Schnittstellen sofort damit entwickeln. Viele Vorteile wurden im direkten Vergleich mit Java gesehen - insbesondere die Null-Safety-Fähigkeit und ‚Lines of Code-Reduktion‘ konnte begeistern.

Das Team hat bei der abschließenden Präsentation alles gegeben und im Rahmen einer Live-Performance Tracking-Daten gesammelt. Sehr sportlich!



<https://www.testcontainers.org/>

# Projekt: Testcontainers

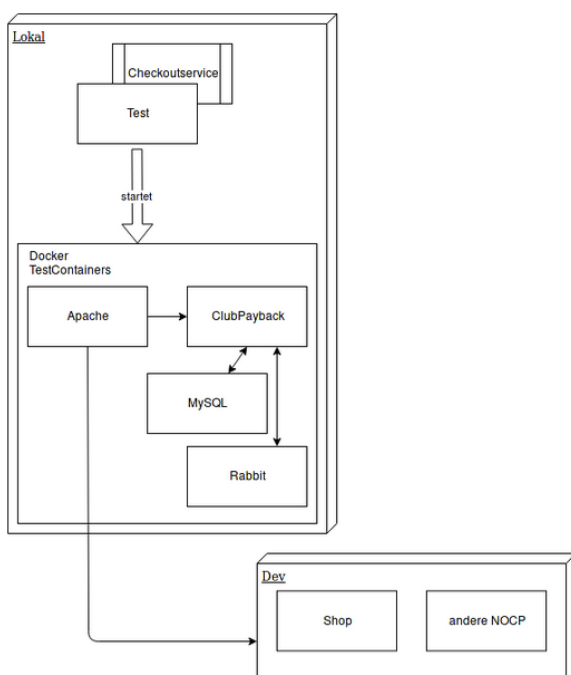
## *Worum ging es?*

Mal wieder die Dev-Umgebung kaputt oder der Test schlägt erst auf der Integrationsumgebung nach dem Deployment fehl? Dieser Zustand kostet extrem viel Zeit. In [unserer Microservice-Architektur](#) haben unsere Produktteams zunehmend die Herausforderung, Test in Abhängigkeit zu [SCS-Services](#) anderer Teams auszuführen.

In unserem [Thalia-Club-Programm](#) schließt der Kunde die Mitgliedschaft im Rahmen eines Checkout-Prozesses ab. In diesem Prozess wird eine Payback-Kontenverknüpfung durchgeführt. Dieser Prozessbestandteil liegt in der Verantwortung eines separaten Produktteams. Wie kann nun das ‚Club-Checkout-Team‘ seine End2End-Tests (Front- und/oder Backend) mit möglichst hoher Unabhängigkeit zum ‚Payback-Team‘ gestalten? Beide Services wurden auf der Docker-Infrastruktur aufgesetzt.

[Testcontainers](#) ist ein Framework, um Docker-Container innerhalb von JUnit-Tests zu verwenden (Infrastructure-as-Code). Das Team möchte prüfen, ob hierdurch die Tests auf der Dev- oder Integrations-Umgebung reduziert werden und die Ausführung von Tests ohne Deployment auf die Teststages möglich ist.

## *Ergebnisse*



## Darstellung der beteiligten Services

Der [PoC](#) zeigt, wie von einander abhängige Services aus unserer Produktentwicklung im Test auf der lokalen Umgebung zu starten und auszuführen sind, dass wir vor dem Deployment auf den jeweiligen Umgebungen Fehler feststellen können. Und eben nicht hinterher;-).

Wichtig ist hierbei der Blick auf die transitiven Abhängigkeiten. Je mehr Kaskaden an inkludierten Testcontainern es gibt, desto komplizierter kann es werden. In einem Folgeschritt sollten noch die Laufzeiten angeschaut werden, die sich aufgrund der Start/Stop-Prozesse der Container erhöht haben. Das Team sieht in der Nutzung von Testcontainers eine Menge Potential. Klasse, dass sich hier Leute aus unterschiedlichen Produktentwicklungen zusammengefunden haben. So konnte schon der erste ‚Reality-Check‘ erfolgen.



Andreas, Christian, Benjamin und Julian haben die Zuschauer-Jury überzeugt.

Die Ergebnisse und die Präsentation haben dann auch am Schluss des Hackathons die Zuschauer-Jury überzeugt. Nach einer Stichwahl stand das Team als Gewinner fest. Herzlichen Glückwunsch!

---

## Fazit

Ist es Zufall, dass sich 2 von 3 Themen rund um die Docker-Infrastruktur konzentrieren? Wahrscheinlich nicht – wir beschäftigen uns im Moment sehr mit dem Thema und wollen eine steile Lernkurve über Einsatz, Handling und Nutzen dieser Infrastruktur erreichen. Die Erfahrungen aus dem Hackathon werden uns hierbei unterstützen. Well done!

Im letzten Jahr haben wir die [App-Entwicklung](#) von Thalia von Berlin nach Münster umgezogen. Jetzt sitzen die Kollegen/innen im Büro nebenan. Macht



richtig Spaß, diese teamübergreifende Zusammenarbeit zwischen App-Experten und Java-Spezialisten. Kotlin wird aktuell bei uns in der App-Entwicklung genutzt. Mal sehen, was jetzt in den ‚Java-Teams‘ passiert:-).

Mit folgenden Stimmungsbildern verabschieden wir uns und schließen mit den Worten:

Das machen wir wieder – see you in 05/2020.















---

**XPdays: DevOps ist keine Rolle,  
oder?**



Am 8.11.2018 haben wir auf den XPdays unsere Erfahrungen mit der Einführung einer Produkt-Organisation aus Sicht von IT-Operations geteilt.

Was passiert eigentlich mit einem klassischen IT-Betrieb, wenn ein Unternehmen sich für eine Produktorganisation entscheidet? Eines ist klar: Alles verändert sich!

In diesem Beitrag wollen wir euch auf unsere spannende Reise vom IT-Betrieb hin zu Platform Engineering und DevOps mitnehmen. Woher kommen wir? Wie haben wir uns technologisch, kulturell, prozessual und als Team verändert um optimal mit und in agilen Produktteams zusammenzuarbeiten? Wie unterstützen wir unsere Entwicklungsteams als zentrales Plattform Team aber auch als Mitglied eines crossfunktionalen Produktteam? Was passiert mit unserem Mindset „Never change a running system“ in einer Welt mit 1.000+ Deployments pro Monat? Wie haben wir uns organisiert und wie versuchen wir täglich aufs Neue das Beste aus einer agilen Produktentwicklung herauszuholen? Wie verstehen wir DevOps? Ist das, was früher mal der IT-Betrieb war, auch zu einem crossfunktionalen Produktteam geworden?

<https://www.xpdays.de/2018/sessions/150-devops-ist-keine-rolle-oder.html>

Natürlich ist es nicht möglich alle Erfahrungen in einen kurzen Betrag zu packen. Uns hat es aber viel Spaß gemacht unsere Erfahrungen zu teilen und haben uns sehr über die vielen Rückfragen und Gespräche zum Thema gefreut. Sprecht uns auch gerne weiterhin an, wenn ihr euch für das Thema interessiert.



Christoph Drosten

c.drosten@thalia.de

[https://www.xing.com/profile/Christoph\\_Drosten](https://www.xing.com/profile/Christoph_Drosten)



Matthias Wellmeyer

m.wellmeyer@thalia.de

[https://www.xing.com/profile/Matthias\\_Wellmeyer](https://www.xing.com/profile/Matthias_Wellmeyer)

Hier die Folien zum Beitrag:

Unseren gesamten Reisebericht findet ihr hier:

[\*Vom IT-Betrieb zu Platform Engineering. Ein Reisebericht \(1/3\)\*](#)

# **<#0101/> <hackathon@thalia/> in Münster**

**<#0101/>  
<hackathon@thalia.de/>  
„Acceleration Technologies“**



... fand am 8.6. in Münster statt. Was beim letzten Mal gut funktioniert hat, wollten wir beibehalten, weshalb wir uns wieder fernab des Tagesgeschäfts in einem anderen Gebäude eingerichtet haben. Kulinarisch/mit Erfrischungen hochgerüstet und nach diversem Dosen-Gepatche einsatzbereit haben wir dieses Mal unter dem Motto „Acceleration Technologies“ zu spannenden Projektvorschlägen aufgerufen. Drei Projekte haben sich daraufhin formiert und konnten passend 16 Uhr Ergebnisse vorzeigen - wenn auch anders als antizipiert...



# Projekt „OpenStack Explorer“



Ich weiß noch nicht ganz, ob Christoph mit dem Logo zum Projekt schon absichtlich den Teaser zum nächsten Mal rausgeworfen hat, aber was das Selbstmarketing des Projektes angeht, ist das schon weit vorne.

Nach dem Pitch haben sich 6 Personen (stark betriebslastig) für dieses Thema zusammengeschlossen.

## Worum ging es?

„Ich möchte [OpenStack](#) und seine Möglichkeiten kennen lernen. Ich verspreche mir davon eine Menge an coolen Features die unser Leben erleichtern können.“

- Was kann OpenStack?
- Können wir mit OpenStack schneller werden?
- Passt OpenStack zu unseren Prozessen und Infrastruktur?

## Ergebnis

OpenStack verspricht die „Cloud für zuhause“, quasi „infrastructure as a service“. Das Team ist sowohl auf virtuellen Maschinen wie auch auf Hardware in das Thema gestartet. Es wurde eine Anleitung für eine all-in-one-cloud auf einer Maschine befolgt, die zur



Konsequenz hatte, dass Puppet 20 Minuten lang Zeug nachinstalliert hat und etwa 1300 Variablen zu setzen waren. Da war dann alles dabei, Firewall, Netzwerk, alles.

Das Team konnte Maschinen hochfahren mit block storage, image service ... 6-7 Kernelemente, die GUI dazu. Als Fazit kamen die OpenStack Explorers zu dem sehr klaren Ergebnis, dass diese Technologie eher was für große Internet Provider oder die NASA ist, aber nicht passend für uns. Für Thalia Anwendungsbereiche ist OpenStack unnötig komplex.

# Projekt „GitLab Explorers“

## Worum ging es?



In diesem Projekt sollte [GitLab CI](#) evaluiert werden. 2 Personen aus dem Entwicklungsbereich überprüften Merging, Builds, das Zusammenspiel mit Jenkins – und vor allem: Was unterscheidet diese Lösung von unserer bisher im Einsatz befindlichen?



## Ergebnis

Die Ergebnisse wurden anhand des Beispiels „checkout service“ präsentiert. Das Fazit der beiden ist klar pro GitLab CI!

Das Ganze funktioniert komplett „on-premises“, jedes Team kann sich aussuchen, ob es das mit oder ohne Jenkins einsetzen will. „Geht alles!“

# Projekt „Our Vault“

## Worum ging es?

Drei Teilnehmer hatten sich dazu entschieden, den [Hashicorp Vault](#) auf seine Eignung für Thalia-eigene Anwendungsfälle im Vergleich zur heutigen Lösung zu testen. Kurz gesagt geht es bei diesem Werkzeug um die „sichere“ Ablage von



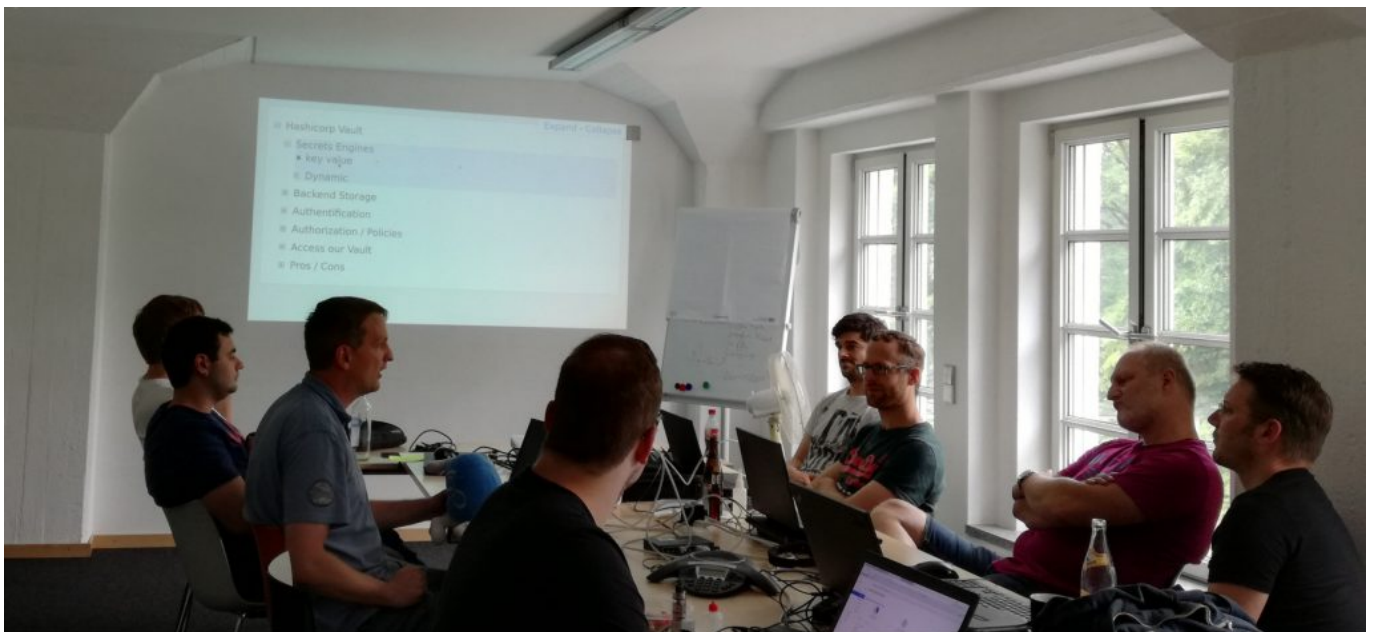
Passwörtern für Anwendungen.

## Ergebnisse

Das Team hatte sich dazu entschlossen, zweigleisig zu fahren. Auf der einen Seite wurden gängige Anwendungsfälle aus unserem Dunstkreis gegen das Produkt gehalten und umgekehrt der Featurekatalog des Vault analysiert. Auf der anderen Seite wurde eine Einbindung in unsere Infrastruktur durch zwei Entwickler umgesetzt.

Hierbei zeigte sich, dass Vault weit mehr als nur einige key/value Paare hält. Die Anzahl der Backend Storage Möglichkeiten ist hoch, diverse Authentifizierungsmöglichkeiten sind gegeben. Der Zugriff kann über Kommandozeile oder http/REST erfolgen, eine UI steht zur Verfügung. Die Verwendung von Spring Cloud Vault ermöglichte den Entwicklern das Ansteuern des Vaults mit Spring-Mitteln.

Als Fazit darf gesagt werden: da geht viel, es bringt mehr Sicherheit, dadurch dass man mehr Keys benutzt, die Integration ist sehr einfach (Kubernetes, AWS). Aber: der erzeugte „single point of failure“ ist ein Problem, das Team sah den Nutzen nicht hoch genug, um hier noch weitere Schritte Richtung Vault zu unternehmen.



## Fazit

Auch dieses Mal können wir wieder Erkenntnisse aus den Projektgruppen rausziehen – diesmal auch sowas wie „ungeeignet für uns“.

Ich möchte an dieser Stelle exemplarisch für die teilweise fantastische Kühlkette mit folgendem Stimmungsbild schließen. Bis zum nächsten Mal!



---

## Vom IT-Betrieb zu Plattform Engineering. Ein Reisebericht (3/3)



Vor einer Woche habe ich berichtet, wie wir Basistechnologien, SelfServices und Automaten etabliert haben. Das hat uns dabei geholfen, wiederkehrende Aufgaben zu automatisieren, die Umsetzungsqualität zu erhöhen, die Entwicklungsteams beim Aufbau und Betrieb von neuen Services zu beschleunigen und die Aufwände im IT-Betrieb zu reduzieren. Aber das alles war nur Werkzeug. Auf unserem Weg zum Platform Engineering Team war es auch wichtig, unser Mindset anzupassen. Wir mussten verstehen, wie unsere wichtigsten Kunden, die Entwicklungsteams, denken und was sie brauchen. Auch unsere Prozesse mussten weiter optimiert werden. Es gab also noch mehr zu tun, als nur ein paar Tools zu etablieren. Auch wenn dieser Reisebericht so geschrieben ist, als ob der Mindset-Change als Letztes stattgefunden hätte, so ist das natürlich nicht richtig. Die technologischen und die kulturellen Änderungen im Team fanden mehr oder weniger zeitgleich statt.

## ***Kapitel 3: Mindset, Methoden, Prozesse und mehr...***

**PENG! Technik ist nicht alles!**



Wenn wir schneller werden wollen, dann müssen wir einiges mit SelfService Schnittstellen machen. Cool, verstanden, fertig? Nope, never, auf keinen Fall! Was ist eigentlich mit Mindset, Kultur, Methoden,...? Auf den Konferenzen reden sie immer von DevOps, Scrum, Agile, Kanban .... Ah, Kanban, da machen wir doch schon was. Und unsere Devs und Ops mögen sich doch auch schon und gehen zusammen Bier trinken. Dennoch fehlt da noch was.

Wir müssen uns also überlegen, wie die künftigen Produkt-Teams und der IT-Betrieb zusammenarbeiten sollten. OK, noch mehr Ziele für den Umbau. Wir müssen nicht-technische Themen wie z.B. **Kultur, Zusammenarbeit, Mindset, Prozesse, Zuständigkeiten und die Schnittstellen zwischen den Teams definieren**. Da wartet einiges an Arbeit auf uns. Um die Veränderung unseres Teams nach innen und nach außen zu verdeutlichen, wollten wir uns auch einen neuen Namen geben. Nach einigen Diskussionen war uns klar: aus „IT-Betrieb“ sollte „Platform Engineering“ oder kurz „PENG“ werden.

Im Rahmen des Aufbaus der Produkt-Organisation wurden wir als Platform Engineering Team sehr früh mit einbezogen. Gute Idee! Das hat uns die Chance gegeben, neben den ganzen Team-, Technik-, Kultur- und Prozessumbauten auch die notwendigen Operations Umbaumaßnahmen mit einzubringen. Warum ist das so wichtig dieses früh und offiziell zu machen? Ich habe drei technisch Beispiele beschrieben, wo dringend Änderungen notwendig sind und SelfServices etabliert werden müssen, um schneller zu werden. Das macht man nicht mal eben so nebenbei. Notwendig dafür sind größere Investitionen und Anschaffungen in die Infrastruktur. Es muss bewertet werden, ob man SelfServices selber erstellen will oder irgendwo einkauft (Kosten/Nutzen). Zum Selberbauen brauchen wir mehr Personal, welches temporär extern beschafft und bezahlt werden muss. Auch die Themen Kultur, Zusammenarbeit, Mindset, Prozesse, Zuständigkeiten und Schnittstellendefinition zwischen Teams brauchen einiges an Zeit und sollten möglichst zusammen mit den Umbauten hin zu Produkt-Teams erfolgen.

OK, Bestandsaufnahme: Wir wissen wir sind gut, aber nicht schnell genug -> Handlungsbedarf. Wir haben uns technische und nicht technische Ziele gesetzt. Die Transformation von IT-Betrieb zu Platform Engineering kann beginnen!



Um die Transformation zu unterstützen, haben wir im Büro eine „Transformation Wall“ erstellt mit den Zielen, Regeln, Infos und was sonst mit der Transformation zu tun hat. Diese Wand lebt, hat immer die aktuellsten Themen wie z.B. die Maßnahmen einer Retro. Jeder, der ein Thema hat, darf dieses auch auf die Wand pinnen, so dass wir bei der nächsten Gelegenheit darüber reden können. Kurz gesagt: Die Wand **begleitet unsere Transformation und erinnert uns auch jeden Tag daran, was wir tun wollen und warum.**

## Setting the Stage: Agile Knowledge (the Basics)





Alle redeten über DevOps, Agile, SCRUM, Kanban, „You build it, you run it“ und so weiter .... Doch was ist das eigentlich alles? Es gab reichlich Unwissenheit oder - schlimmer - gefährliches Halbwissen. Lasst uns also eine gemeinsame Wissensbasis schaffen, um all die umherschwirrenden Buzzwords zuordnen und verstehen zu können. Auf dieser Basis können wir dann alle anderen Themen aufbauen.

Vor einigen Monaten hatte ich das Glück, einen Zwei-Tages „Agile Mindset“-Workshop zu besuchen, der Grundlagen zu SCRUM, Kanban, Agilität, agiles Manifest, agile Werte usw. vermittelt hat. Während des Workshops, der angereichert war mit kleinen Übungen, habe ich mir die ganze Zeit überlegt, wie ich das ins Team transportiert bekomme. Am Ende des ersten Tages war die Antwort klar: Gar nicht. Zusammen mit dem Agile Coach, der u.a. einen Operations Background hatte, haben wir uns dann überlegt, wie wir diesen Workshop für unser IT-Betrieb-Team abhalten können. Nach kurzer Zeit stand die Planung, und wir haben zwei echt tolle und spannende Workshop-Tage gehabt. Im Anschluss konnten wir viele „Aha Momente“ verzeichnen und hatten als IT-Betrieb ein recht gutes Basis-Verständnis zu den agilen Methoden, der Idee dahinter, den Unterschieden und auch den Vor- und Nachteilen. Klar waren wir weit ab, agile Spezialisten zu sein, aber wir hatten einen super Werkzeugkasten erhalten, der uns auf den weiteren Wegen sehr geholfen hat.

**Ausprobieren: Funktioniert SCRUM für Operations?**



So, Workshop fertig, jetzt sind wir agile! Nicht wirklich. Uns war klar, wir durften gerade einmal am Inhaltsverzeichnis schnuppern. Die wirkliche Arbeit wartete nun noch auf uns. Wir als Thalia verkaufen ja jede Menge Bücher, aber ein Buch „Agile Operations @ Thalia“ hatten selbst wir als Buch-Spezialist nicht. Vielleicht kommt das später noch ☐ Mit anderen Worten: Wir haben eine Idee bekommen, welche Werkzeuge es gibt. Welche Werkzeuge uns bei unserer Arbeit wirklich helfen (und nicht einfach nur Hip sind), mussten wir in der Praxis selber ausprobieren.

Wir hatten eine Menge über SCRUM, Kanban und Co. erfahren. Kanban hatten wir bereits einige Zeit (zumindest im Ansatz) im Tagesgeschäft praktiziert. Nun wollten wir SCRUM etwas näher kennenlernen. Zusammen mit einem SCRUM Master haben wir unser Projekt zur Einführung der Automatischen Service Bereitstellung (ASB) kurzerhand von einem klassischen Wasserfall-Projekt nach SCRUM umgestellt. Dazu haben wir die notwendigen Meetings aufgesetzt, ein Backlog angelegt und gepflegt, Rollen besetzt, Sprints geplant, durchgeführt, reviewed, verbessert usw... Gerade am Anfang haben wir uns sehr schwer getan. Ganz besonders das Review als auch das Schneiden, Schätzen und Planen von Tickets brauchten einige Übung. Wir haben viel ausprobiert, was gehen könnte. Schätzen wir z.B. Aufwände in Personentagen, Story Points, Anzahl Tickets, ...? Wir machten unsere Erfahrungen und fanden raus, was gut funktioniert, aber auch, was wir besser nicht machen sollten. Und so wurde es von Sprint zu Sprint leichter und nutzbringender. Das kleine Test-SCRUM-Team berichtete von erhöhter Transparenz, klarerer Struktur und ruhigerem Arbeiten im Sprint. Der Produkt Owner hatte einen klareren Blick auf was gemacht wurde, was wann kommen kann und hatte die Möglichkeit zu entscheiden, welches Feature er wann haben wollte. Unterm Strich war es anfangs sehr ungewohnt, jedoch sehr spannend und hilfreich. Wir haben es also geschafft, ein Operations-Projekt nach SCRUM zu führen. Die Erkenntnis stimmte uns positiv ☐

Ein Problem gab es jedoch noch: Unser SCRUM-Testballon wurde in einer reinen Projektumgebung ohne Operations-Tagesgeschäft durchgeführt. Hier gab es keine größeren Störungen, keine spontanen und dringende Anforderungen. Im Operations Tagesgeschäft wimmelt es nur so von unerwarteten Änderungen, worauf wir teilweise hochflexibel reagieren müssen. Leider bekomme ich kein Verständnis, wenn wir die Produktions-Störung des Webshops erst im nächsten Sprint in einer Woche bearbeiten. OK, kann ich verstehen, passt aber nicht so super zu SCRUM. Für das Operations-Tagesgeschäft funktioniert 100% SCRUM für uns also nicht so gut.

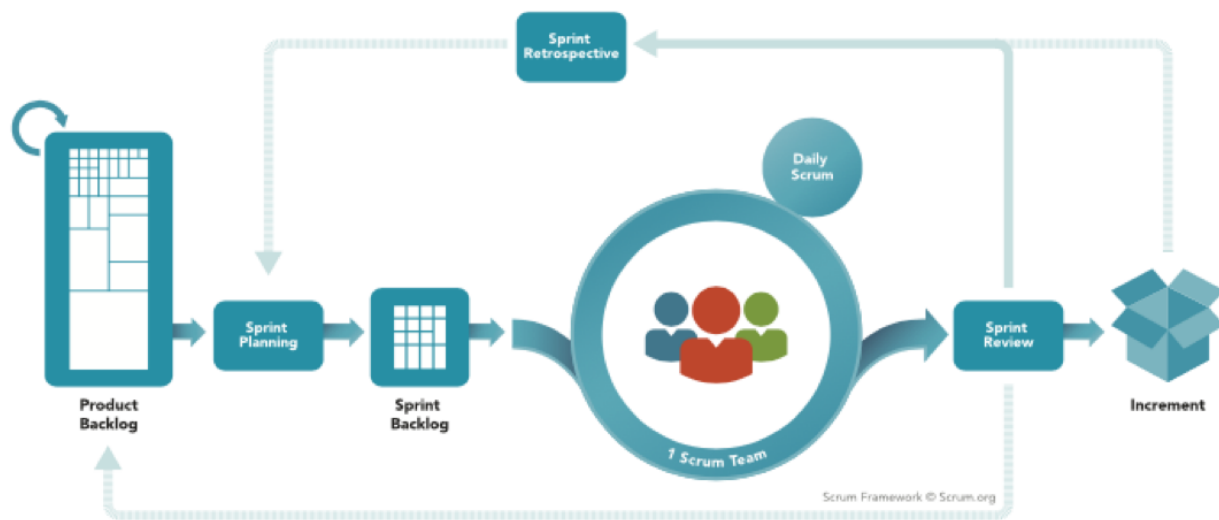
## **Nicht Hip, aber hilfreich: An SCRUM orientieren ohne SCRUM zu machen**

Warum sind wir nicht einfach bei Kanban geblieben? Das passt doch viel besser zu so unerwarteten Themen. Nun ja, wir fanden einige Elemente von SCRUM sehr spannend und hilfreich. So ist es hilfreich, dass wir uns alle zwei Wochen verbindlich zusammensetzen, um Aufgaben zu planen. Die Backlogfunktion im JIRA SCRUM-Board finden wir super, eine Retro sowie ein Review hilft uns besser zu werden, auch das Messen des Erreichten ist für uns und unsere Planung hilfreich. Das ist ein Auszug, warum wir uns aktuell an SCRUM orientieren. Klar kann es sein, dass es irgendwann Gründe gibt, wieder Kanban zu machen. Im Moment sind wir jedoch glücklich und vor allem sehr transparent. Wir haben auch gelernt, dass wir nun die gleiche Sprache sprechen wie die Produkt-Teams. Folgende Unterhaltung soll es verdeutlichen: [Dev] „Kannst du bitte folgende Ops Aufgabe diese Woche noch für mich machen? Ich erreiche sonst unser Sprintziel nicht“ - [Ops] „Das kommt etwas überraschend, um dir zu helfen müsste ich unseren Ops Sprint verändern und eine geplante Aufgabe entfernen um deine Aufgabe zu erledigen“ - [Dev] „Oh, das ist ja nicht so gut. Nein, dann plane meine Ops Aufgabe bitte in den nächsten Sprint ein. Ich kann im Zweifel warten“. Hurra, ohne das Wort „Nein“ haben sich Ops und Dev auf eine Verschiebung einer Aufgabe geeinigt und sind dabei auch noch zufrieden.

Wir wollten also SCRUM nicht nur für Projekte nutzen, sondern auch für unser Tagesgeschäft. Wie haben wir das gemacht? Erneut mit viel Ausprobieren, Überprüfen, Lernen, Bessermachen und wieder Ausprobieren. Nach einiger Zeit haben wir uns auf folgendes Setup geeinigt:



# SCRUM FRAMEWORK



This image is the copyrighted material of Scrum.org. The original can be found here: <https://www.scrum.org/resources/scrum-framework-poster>

Wir haben ein **Product Backlog**, dieses ist jedoch nur rudimentär priorisiert. Bislang kommen wir mit den Themen im Sprint gut aus :-). Alle zwei Wochen führen wir ein **Sprint Planning** durch. Wir planen dabei etwas weniger als 50% der Tickets ein, die wir schaffen können. Wir orientieren uns dabei an der Anzahl der Tickets. Wir orientieren uns nicht an genaueren Aufwänden oder Story Points (siehe auch „#noEstimation-Bewegung“ unter „Messen“). Zum einen messen wir, dass die Anzahl an Tickets im Schnitt erstaunlich konstant ist. Zum anderen würden wir zwar mit Story Points o.ä. genauer werden in der Planung, jedoch würde sich unser Planungsaufwand deutlich erhöhen. Für uns ein ungünstiges Aufwands/Nutzen-Verhältnis. Nach dem Sprint Planning starten wir den Sprint und haben ein **Sprint Backlog**. Dieses ist aufgrund der vielen Änderungen (ungeplante dringende Anforderungen oder Störungen) jedoch hoch dynamisch. Unser **Scrum Team** besteht aus Linux-, Microsoft-, Datenbank- und Security-Spezialisten. Diese arbeiten sehr eng zusammen. Das Erweitern der Wohlfühzone ist ausdrücklich erwünscht ohne den Anspruch, Spezialist in jedem Bereich zu werden. Ein richtiges **Daily Scrum** nach Lehrbuch machen wir nicht, jedoch haben wir ein Daily Standup, um den Tag zu planen und um jedem den Raum für Fragen oder für Unterstützung zu geben. Alle zwei Wochen vor dem Planning führen wir ein nicht öffentliches **Sprint Review** durch. Warum nicht öffentlich? Wir bieten allen interessierten den „Blue-Board“ Termin (Details folgen □ ) an, um

sich über unsere Arbeit zu informieren. In unserem Sprint Review bringen wir noch einmal auf den Punkt, was wir erreicht haben und was wir bewusst im letzten Planning raus gelassen haben oder was nicht während des Sprints aufgenommen wurde. Dann prüfen wir zwei Wochen nach der Entscheidung, ob unsere damalige Entscheidung heute immer noch richtig ist. Haben wir das Richtige getan? Es ist kein Blame-Game! Es geht darum zu lernen, um künftig bessere Entscheidungen treffen zu können. Direkt nach dem Sprint Review führen wir eine **Sprint Retro** durch. Was lief gut, was lief schlecht? Welche Maßnahmen leiten wir ab? Diese Maßnahmen landen dann wieder auf unserer Transformation Wall.

## **Blue Board: Die Fäden zusammenhalten**

Unser SCRUM-Board hilft uns in der täglichen Arbeit mit den vielen kleinen Aufgaben. Jedoch ergibt es auch Sinn, einen Schritt zurückzugehen, um auch die größeren Themen zu betrachten, die viele dieser kleineren Aufgaben zusammenhalten. Auf einer höheren Flugebene machen wir größere Themen sichtbar. Größere Themen können dabei unternehmensweite Projekte, Operations Projekte aber auch Penetration Test sein. Es sind also Themen, die uns längere Zeit begleiten und Kapazitäten binden. Für jedes Thema haben wir eine gelbe oder eine grüne Karte. Gelbe Karten sind für größere Themen aus dem Tagesgeschäft (Replacements, Updates, Security Scans ...) - also Themen, die wir machen müssen, um eine stabile Plattform zu gewährleisten. Grüne Karten sind für größere Themen, die unsere Produkte weiterentwickeln und Mehrwert generieren. Die Themen landen dann an einem physikalischen Board. Lustigerweise ist das ein Kanban-Board, welches wir aufgrund der Board-Farbe ganz kreativ „Blue-Board“ getauft haben. Alle zwei Wochen gibt es zusätzlich zu den Daily Standups eine größere Runde für 60 Minuten, wo wir über den Status der Themen informieren. Diese Runde ist öffentlich, so dass Vertreter aller Produkt-Teams sehen, was gerade bearbeitet wird und welches Thema wann kommt. Sie können sich informieren, ihre Fragen loswerden und Feedback geben.



„Blue-Board“ für die Transparenz und Planung von größeren Operations Themen

## Definieren unserer Produkte

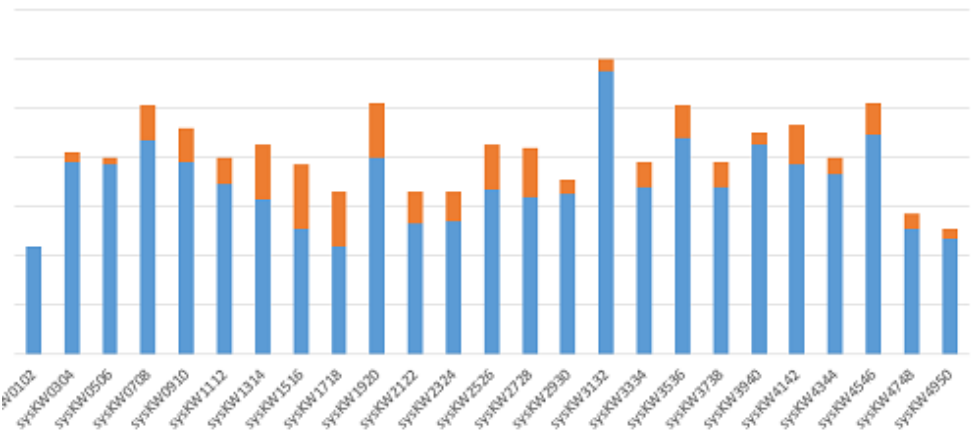
Wir wollen ein Produkt-Team sein. Aber was ist eigentlich unser Produkt? Mit dieser Frage haben wir uns lange auseinandergesetzt. Ist ein Loadbalancer schon ein Produkt? Dann haben wir ja hunderte Produkte. Wenn wir es zusammenfassen, ist dann „die Plattform“ unser Produkt? Hmmm, die Definition hilft nicht so richtig weiter bei der Planung, Strukturierung und Messung. Mit Hilfe unseres Agile Coach haben wir definiert: Ein Produkt ist all das, was einem anderen Team ein Mehrwert bietet. Etwas, wofür jemand im Zweifel auch Geld bezahlen würde.

Und so sind wir losgezogen und haben nach und nach Technologien zu Produkten zusammengefasst, die es uns erlauben, zu strukturieren und sowohl uns als auch allen anderen zu verdeutlichen, was wir eigentlich alles machen. Dabei herausgekommen ist folgende Übersicht:



So haben wir z.B. den Loadbalancer, Firewall, Storage, Netzwerk, Rechenzentrum ... zum Produkt **Basis Infrastruktur** zusammengefasst. Wir bieten den anderen Teams unser Produkt **Notdienst** an, welches die Teams nutzen können. Wir bieten den Teams als Produkt eine **Alarmierungs & Monitoring**-Infrastruktur an, die sie nutzen können. Und wie in jedem guten Haushalt so gibt es bei uns auch einen Bereich „Sonstiges“ mit dem klangvollen Produktnamen **3rd Party Service**. Da wir leider noch nicht alle Teams mit einem eigenen Operations-Spezialisten versehen konnten, machen wir für einige **Touchpoint Teams** noch den Operations-Teil.

## Messen: Die Basis für eine gute Planung

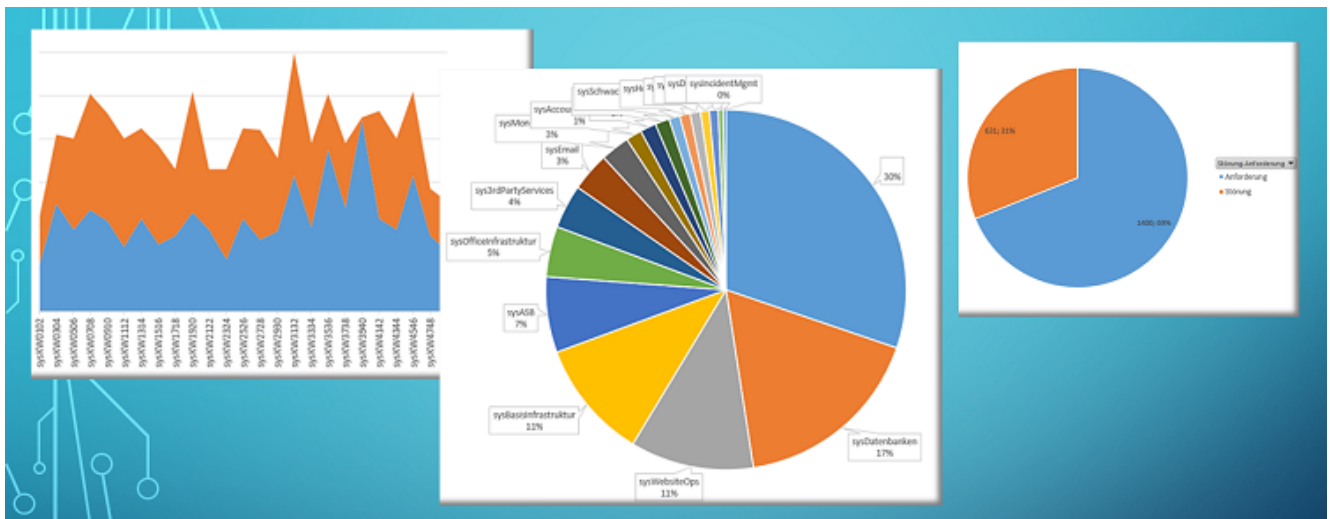


Anzahl der geschlossenen Tickets je Sprint in 2017

Anfangs haben wir überlegt, wie wir unsere Arbeit messen wollen und wie wir Aufwände planen können. Auch brauchten wir eine Möglichkeit zu prüfen, ob unsere Maßnahmen einen positiven Effekt haben. Weiter wollten wir wissen, wie viele Tickets wir in einen Sprint nehmen können. Hier haben wir einiges ausprobiert, verworfen und neu probiert. Um es möglichst einfach zu halten, um möglichst wenig Aufwand zu investieren, haben wir uns darauf geeinigt, die Anzahl der geschlossenen Tickets je Sprint zu messen. Natürlich ist uns klar, dass ein Ticket 5 Minuten oder 5 Stunden dauern kann. Jedoch haben wir festgestellt, dass bei uns die Anzahl der Tickets, die wir schließen, relativ konstant ist und somit eine Größe ist, mit der wir arbeiten können. Klar können wir die Genauigkeit weiter erhöhen, indem wir z.B. auf Storypoints gehen würden, jedoch scheuen wir den Mehraufwand für die Bewertung. Auch erscheint uns der Mehrwert, den wir damit erzeugen, zu gering. Auf der Basis der Anzahl geschlossener Tickets haben wir dann eine Anzahl Tickets abgeleitet, die wir fest in den Sprint einplanen. Leider liegt dieser Wert noch bei unter 50% der Tickets je Sprint. Das Problem ist leider immer noch, dass uns immer wieder ungeplante Aufgaben erreichen, die ihre Berechtigung haben. Bislang funktioniert die Planung basierend auf der Anzahl der Tickets recht gut und stabil. Es gibt auch noch keine Bemühungen, den Aufwand für die Planung (z.B. mit Storypoint) zu erhöhen. So sind wir mehr oder weniger geplant zu Anhängern der #noEstimation-Bewegung geworden.

Neben der Anzahl der Tickets, die uns erreichen, messen wir aber noch mehr – z.B. wie viel Tagesgeschäft und wie viel Weiterentwicklung im Sprint steckt. Wir messen, woher die Tickets kommen, für welches unserer Produkte wir die Arbeit leisten, wie viele der ursprünglich für den Sprint geplanten Tickets tatsächlich

auch fertig werden, das Verhältnis Störung zu Anforderung ...



## Eine Auswahl an Messungen

Zur Messung exportieren wir die geschlossenen Tickets aus JIRA nach Excel, wo wir sie dann via Pivot mit wenig Aufwand in ein paar Graphen visualisieren. Um die Tickets auswerten zu können, nutzen wir u.a. drei JIRA Stichwörter: (1) Ein Stichwort für die Kalenderwochen, in denen das Ticket bearbeitet wurde, (2) Ein Stichwort, um das Ticket einem unserer Produkte zuzuordnen und (3) haben wir ein Stichwort, um ein Ticket als Entwicklungsticket zu kennzeichnen (ohne dieses Stichwort ist es ein Tagesgeschäftsticket).

## XFT - oder der Microsoft Spezialist der MySQL DBs auf Linux installiert

Super, jetzt haben wir ein wenig „agiles Mindset“, einen Agile Coach und ein paar Boards. Wir können nun priorisieren, um das Richtige zur richtigen Zeit tun. Aber was, wenn der Spezialist für das Richtige gerade im Urlaub ist? Im Team waren wir so organisiert, dass wir für Linux-, Datenbank-, Security- und Windows-Themen Spezialisten haben, die ihr Handwerk wirklich gut verstehen. Dadurch hatten wir jedoch, wenn man so will, vier Teams in einem Team. Diese künstlichen Hürden wollten wir entfernen und dazu die Wohlfühlzonen, die wir hatten, erweitern. Warum sollte nicht der Windows-Spezialist eine MySQL-Datenbank auf einer Linux-Maschine installieren, wenn das gerade die wichtigste



Arbeit ist? Als kleiner Nebeneffekt hat er seine Wohlfühlzone und sein KnowHow erweitert. Zudem macht es auch noch Spaß, neue Sachen zu versuchen. Aber funktioniert das auch im Alltag? Um das herauszufinden, haben wir jeden Ausflug raus aus der Wohlfühlzone auf einer einfachen Matrix gemessen und in der Retro besprochen. Die Reaktionen im Team waren ziemlich positiv, und die Fokussierung auf die umzusetzenden Themen im Sprint hatte positive Auswirkungen auf die Bearbeitungsgeschwindigkeit. Grenzt das nicht an ein Cross-Functional-Team (kurz XFT)? ☐



Erweitere deine Wohlfühlzone. Wer hat mit wem zusammengearbeitet?

## DevOps ist keine Rolle, oder? Operations KnowHow in die Produkt-Teams

Dank der guten und nicht ganz einfachen Arbeit der Kollegen wissen wir bereits, welche Produkt-Teams es gibt und welche Software-Services sie entwickeln und

betreiben sollen. Auf dieser Basis konnten wir Team-Mitglieder suchen, die Lust auf ein neues Abenteuer haben. So haben wir Linux Spezialisten gefunden, die bereit waren, zwei oder auch drei (je nach Teamgröße) Produkt-Teams zu betreuen. Aber was bedeutet das? Um die Teams zu betreuen, sind die Linux Spezialisten sowohl fachlich als auch physisch in die Produkt-Teams gegangen und fester Bestandteil der Teams geworden. Hurra! Endlich haben wir unseren eigenen Linux Spezialisten, der alle Operations Aufgaben erledigen kann! Nope! Die Idee ist, dass er zwar der Operations Spezialist im Team ist. Jedoch ist seine Aufgabe, sein Operations-KnowHow so weit wie möglich zu verbreiten und das Team in die Lage zu versetzen, eigenständig alle für ihr Produkt notwendigen Operations Aufgaben umzusetzen. Wir erinnern uns: Wenig Reibungsverlust an Schnittstellen. Möglichst viel selber machen. Gilt auch im Kleinen. Wollen wir nun z.B. die Entwickler zu Operations Spezialisten machen? Nein! Wir wollen, dass die Teams in der Lage sind, alles, was nötig ist, selber zu machen. Im Optimalfall gibt es einen einfachen SelfService mit einer Entwickler Schnittstelle, die man mit geringem KnowHow bedienen kann. Die ganze Operations Magie passiert dann automatisch im Hintergrund. Um zu verstehen, wie die optimale Schnittstelle zwischen Platform Engineering und dem Produkt-Team aussieht, haben wir zusammen mit den Linux Spezialisten in den Teams definiert, was in den Teams in welcher Tiefe passieren soll und wo Platform Engineering eine einfache Schnittstelle bereitstellen muss. Auch wenn jede Schnittstelle im Optimalfall perfekt mit einer großartigen GUI oder API automatisiert ist, so ist das besonders zum Start nicht realistisch. Im Zweifel kann eine Schnittstelle auch ein Ticket oder eine gute Dokumentation sein. Automatisiert aber so schnell und so früh wie möglich die Schnittstellen. So, nun sind die Linux Spezialisten in den Produkt-Teams, weg, raus aus dem Platform Team. Und wie bleiben die Ops Spezialisten in den Teams nun am Operations Puls der Zeit? Das ist recht einfach. Zum Einen nehmen sie weiter an den bereits etablierten Meetings teil.





Unser Daily Standup mit Teilnehmern vor Ort, in Hagen, Berlin und im Homeoffice.

So können Sie an den Daily Standups oder an den alle zwei Wochen stattfindenden „Blue Board“ Meetings teilnehmen (da wo wir die größeren Themen und deren Priorität und Fortschritt besprechen). Das ist alles nice und hilft. Besonders wichtig ist jedoch ein neu geschaffenes Meeting. Noch ein Meeting? Och nöö! Doch, ergibt Sinn. Tut nicht wirklich weh, bringt dafür aber viel. Neu etabliert haben wir die „Operations Community of Practice“ – kurz: OpsCoP. Hier treffen sich alle zwei Wochen die Ops Spezialisten aus allen Produkt Teams sowie ein Vertreter aus dem Platform Engineering Team. Dort tauschen wir unsere Erfahrungen zwischen den Teams aus. Was kann ich aus Team A lernen und für Team B übernehmen? Welche Entwicklung passiert gerade im Platform Team oder welche Known Issues gibt es? Und so weiter ....

Irgendwie ist es dann doch noch passiert, das unsere Linux-Spezialisten in den Produkt-Teams nur noch „die DevOps“ genannt werden. Im Grunde total falsch! DevOps ist keine Rolle. Wir haben sogar einen Versuch gestartet das wieder raus zu bekommen. Erfolglos. Die falsche Bezeichnung DevOps erfreut sich großer Beliebtheit bei uns. Nun lassen wir es einfach so, jedoch immer mit der Ergänzung dass es eigentlich falsch ist ☐

## **Die Schnittstelle zwischen Platform Engineering und den Produkt-Teams**



Cool, jetzt haben wir die Produkt Teams mit Operations KnowHow ausgerüstet, verteilen dieses, machen die Teams dadurch schneller. Auch die Schnittstellen sowie der Informationsfluss zwischen Platform Engineering und Produkt Teams ist geklärt. Aber was machen eigentlich die Operations-Kollegen, wenn die Produkt Teams ihre Sachen selber betreiben? Wir brauchen ein neues Ziel. Früher war unser Ziel, alle Systeme und Anwendungen 7x24h zu betreiben. Nun machen das (zumindest in großen Teilen) die Produkt Teams. Die Antwort ist einfach, die Angst völlig unberechtigt. Wir sind inzwischen kein echter IT-Betrieb mehr. Da war doch was. Hatten wir uns nicht umbenannt? Platform Engineering.... Was ist das eigentlich? Wir engineeren (gibt es das Wort im deutschen überhaupt? Sorry) eine Plattform für alle Services. Aber gehört alles zur Plattform? In großen Teilen helfen uns da die Schnittstellen, die wir zusammen mit den Produkt Teams definiert haben. Alles was ein Produkt Team braucht, um einen Service zu bauen und zu betreiben, machen die Produkt Teams. Alles andere macht das Platform Engineering. Beispiel: Das Produkt-Team braucht einen Knopf, aus dem ein Server mit einem Service rauskommt. Den Knopf mit all seiner Logik und seinen Automatismen baut das Platform Engineering Team. Ein neuer Blickwinkel. Wir bauen und betreiben nun Services, die es erlauben, Server zu erstellen. Den erstellten Server betreibt nun jedoch jemand anderes. Oder ein anderes Beispiel aus dem Bereich Datenbanken: Wir definieren nun, welche MySQL Versionen automatisiert bereitgestellt werden. Wir definieren verpflichtende Sicherheitskonfiguration. Aus unserer Erfahrung schlagen wir Best Practice Konfigurationen vor, die von Produkt Teams genutzt werden können, nicht müssen. Wir stellen getestete Patches und zugehörige Dokumentation bereit. Die Produkt Teams müssen jedoch all diese Änderungen selber implementieren und betreiben. Dadurch ergeben sich für uns neue Möglichkeiten. Endlich haben wir eine Chance, uns auf neue Innovationsthemen zu stürzen. Wir haben die Chance, die Automation weiter auszubauen. Wir als

Platform Engineering verstehen uns als Anbieter von Technologie für die Produkt Teams, ohne diese auf unsere Technologie limitieren zu wollen. Warum sollte ein echt schnelles Produkt-Team Monate darauf warten, von uns eine Technologie zu bekommen? Warum sollte nicht auch ein Produkt-Team zusammen mit dem Linux Spezialisten in Abstimmung mit dem Platform Engineering Team eine neue Technologie evaluieren und diese Technologie dann über das Platform Engineering Team allen anderen Teams zur Verfügung stellen? Wichtig ist hier wieder die enge Abstimmung der Bereiche. Sobald mehr als ein Team eine neue Technologie nutzen möchte, so übernehmen wir aus dem Platform Engineering den aktuellen Stand und stellen es allen Teams zur Verfügung. Müssen wir deswegen jede Technologie selber entwickeln? Nein, natürlich nicht. Was spricht dagegen, eine Monitoring Lösung, ein CDN, einen DDoS-Schutz und so weiter einzukaufen und den Produkt-Teams zur Verfügung zu stellen? Warum nicht Technologie aus der Public Cloud einkaufen und allen anbieten. Das macht uns wieder schneller. Es ist immer eine Frage des Preis-/Leistungsverhältnisses. Hier darf man jedoch nicht vergessen, dass auch der billiger aufgebaute eigene Service am Ende sehr viel teurer sein kann als der teuer eingekaufte Service. Ausschlaggebend ist natürlich auch der Faktor „time to market“. Wenn der teuer eingekaufte Service es uns ermöglicht, zwei Monate schneller am Markt zu sein, und somit zwei Monate mehr Umsatz generiert, ist das ein wichtiges Entscheidungskriterium.

## ***Was haben wir gelernt?***

- Mache kein SCRUM, nur weil alle es machen. Mache es, weil es dir hilft.
- Sei mutig. Mache Fehler. Lerne.
- Lass dir helfen. Hilfe anderen. Vertraue deinem Team.
- Probiere aus. Laufe vor die Wand, um zu verstehen, was man besser machen kann.
- Reduziere Schnittstellen auf ein Minimum und automatisiere den Rest und noch mehr.
- Sei kein Blocker. Wenn du ein Blocker bist, verändere es.

- Messe, was du tust, und mach die Ergebnisse sichtbar.
- Tue das Richtige zur richtigen Zeit. Mache dazu deine gesamte Arbeit sichtbar und priorisiere sie nach dem Nutzen, den die Arbeit erzeugt.
- Mache sichtbar, was du nicht machst.
- Stelle die Aufgabe in den Mittelpunkt und bearbeite sie, auch wenn du länger brauchst als der Spezialist.
- Der Klassiker: Stop starting, start finishing. Es macht dich langsam, wenn du alle Themen gleichzeitig machst.
- Gib Raum für Eskalationen. Eskalationen sind hilfreich und nicht böse.
- Spreche die gleiche Sprache wie dein Dev Kollege.
- Erweitere deine Wohlfühlzone und habe Spaß dabei.
- Lass dich inspirieren. Gehe auf Konferenzen, rede mit Kollegen aus anderen Unternehmen oder ganz einfach: Rede mal mit deinen Entwicklern ☐

Sicherlich habe ich einige Punkte vergessen. Ich möchte mit diesem Reisebericht jedoch unsere wichtigsten Eckpunkte, Erfahrungen und Gedanken vom IT-Betrieb hin zum Platform Engineering teilen. **Ich würde mich total über Feedback in den Kommentaren freuen.** Was sind eure Gedanken zu unserer Reise? Was sind eure Erfahrungen? Was können wir noch besser machen? Wir sind auch offen für einen persönlichen Erfahrungsaustausch. An dieser Stelle ein großes Dankeschön an Kollegen von Otto.de oder dem LVM für tolle inspirierende Gespräche. Ich hoffe, wir konnten auch etwas zurückgeben.

## Alle drei Kapitel im Überblick



[Kapitel 1: Woher kommen wir?  
6 Jahre im Schnelldurchlauf](#)



[Kapitel 2: Basistechnologien,  
SelfServices & Automation](#)



[Kapitel 3: Mindset, Methoden,  
Prozesse und mehr...](#)

---

# Vom IT-Betrieb zu Platform Engineering. Ein Reisebericht (2/3)



Vor einer Woche habe ich berichtet, woher wir gekommen sind, welche Probleme wir hatten und wie wir mit Veränderungen wie z.B. der Einführung einer [SOA-Architektur](#) umgegangen sind.

Wir hatten schon einen wichtigen Schritt gemacht, indem wir unsere Arbeit sichtbar gemacht haben. Dadurch konnten wir priorisieren. Wir haben bewusst Themen abgemeldet bzw. erst für später eingeplant, um wiederum andere Themen mit einer höheren Priorität früher zu machen.

Wir hatten aber immer noch Probleme. Die Ticketanzahl stieg, wir waren immer noch zu langsam, und von Weiterentwicklung wollen wir gar nicht reden. Die Reise war also noch nicht zu Ende...



## ***Kapitel 2: Basistechnologien, SelfServices & Automation***

Wir wussten, dass wir nicht schnell genug waren. Die Anzahl der Tickets stieg und acht Wochen Vorlauf für einen neuen Service in Produktion klingt irgendwie nicht zeitgemäß, oder? An welchen Schrauben sollten wir drehen? Die Teams, die wir unterstützen sollen, warteten zu oft auf uns. Immer wieder kam z.B. die Aussage, dass ein Server in der Cloud in wenigen Minuten verfügbar ist. Wir als Team waren mit operativer Arbeit überlastet und hatten nur wenig Zeit für Weiterentwicklung. Irgendwie mussten wir im Tagesgeschäft schneller werden, um mehr Zeit für die Weiterentwicklungen unserer eigenen Themen zu bekommen. Wenn wir einen Mehrwert bieten wollen, dann müssen wir uns mit neuen Technologien auseinandersetzen. Nicht zuletzt machen neue Technologien und Weiterentwicklung auch mehr Spaß als das hoch standardisierte und wiederholte Einrichten von neuen Servern, Monitoring, Datensicherung, Logging, .....

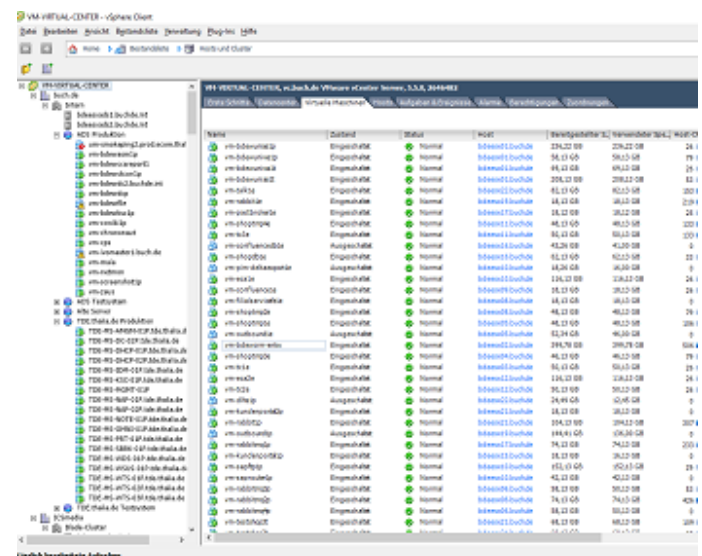


Was fehlte uns, um schneller zu werden? Wir hatten das Glück, dass wir die Chance hatten, Einblicke in die Arbeitsweise und das Mindset von Otto.de zu bekommen (vielen Dank und viele Grüße an die Kollegen von Otto.de!). Ein Satz ist mir besonders hängen geblieben, weil ich ihn anfangs erschreckend schlimm fand. Kurz zusammengefasst: Kommunikation macht langsam! Das meinten die Jungs jetzt nicht ernst, oder? Gute Kommunikation ist doch die Basis! Klar habe ich es erst missverstanden. Gute Kommunikation ist natürlich sehr wichtig. Gemeint war damit die Idee, dass **wenn zwei Teams miteinander reden müssen** bzw. Themen ein Team verlassen, im nächsten Team bearbeitet werden und dann wieder zurück müssen, dort geprüft werden, Fehler gefunden, wieder zurück an das andere Team zur Nachbesserung, zurück zum Test, ..... Kommt das

jemandem bekannt vor? Klingt das nach Topperformance? Solche Kommunikation an Schnittstellen macht einfach langsam. Versuche, Schnittstellen zu reduzieren. Versetze den Anforderer in die Lage, seine Anforderung selber umzusetzen, ohne Spezialisten-KnowHow zu haben. **Biete SelfServices an!** Ein SelfService basiert dabei auf einem Automaten mit einem einfachen Frontend. Über das Frontend ist der Anforderer in der Lage, seine wiederkehrenden Aufgaben einzugeben. Der Automat erledigt dann die hoch standardisierte Umsetzung der Aufgabe in einer fehlerfreieren Qualität als ein Mensch es könnte.

Nur wo fange ich an? Leichter gesagt als getan. Wir haben uns daran orientiert, wo wir als IT-Betrieb die größten Schmerzen hatten und wo wir den neuen Produkt-Teams den größten Nutzen bringen konnten. Hier drei technische Beispiele, die uns nach vorne gebracht haben.

## Beispiel 1: Automatisierte Service Bereitstellung (ASB)



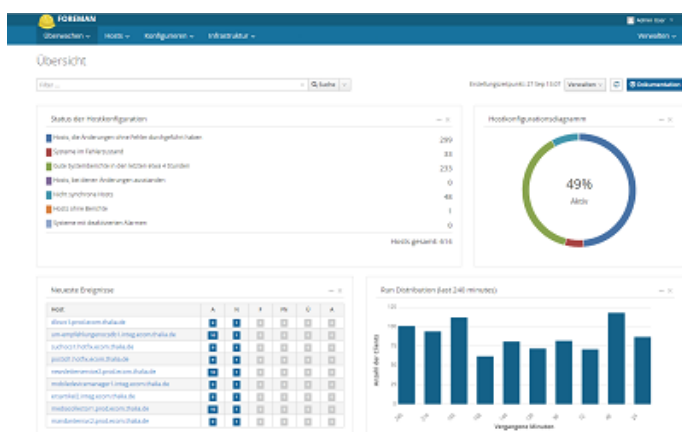
Name	Status	Power	Storage	Network	VMware Tools
vm-kcs-prod-01	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-02	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-03	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-04	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-05	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-06	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-07	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-08	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-09	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-10	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-11	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-12	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-13	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-14	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-15	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-16	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-17	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-18	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-19	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-20	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-21	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-22	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-23	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-24	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-25	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-26	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-27	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-28	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-29	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-30	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-31	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-32	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-33	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-34	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-35	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-36	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-37	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-38	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-39	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-40	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-41	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-42	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-43	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-44	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-45	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-46	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-47	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-48	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-49	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-50	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-51	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-52	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-53	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-54	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-55	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-56	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-57	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-58	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-59	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-60	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-61	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-62	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-63	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-64	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-65	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-66	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-67	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-68	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-69	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-70	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-71	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-72	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-73	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-74	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-75	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-76	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-77	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-78	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-79	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-80	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-81	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-82	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-83	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-84	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-85	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-86	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-87	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-88	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-89	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-90	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-91	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-92	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-93	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-94	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-95	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-96	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-97	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-98	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-99	Engaged	Normal	100GB	vmxnet3	Installed
vm-kcs-prod-100	Engaged	Normal	100GB	vmxnet3	Installed

Vorher: Server klonen mit VMware

Wie ihr wisst, haben wir acht Wochen Vorlauf für einen Service in Produktion benötigt. Das muss natürlich auch schneller mit weniger Vorlauf gehen. Warum haben wir acht Wochen Vorlauf gebraucht? Primär aufgrund der Abstimmung zwischen IT-Betrieb und Entwicklung. Von den acht Wochen haben wir drei Wochen für die Abstimmung eingeplant. Eine Woche ist für den Bau der gesamten Infrastruktur eingeplant worden, so dass nach vier Wochen die Infrastruktur für den neuen Service bereitstand. Aber warum dann acht Wochen? Vier Wochen vor Produktion startet das erste Testdeployment. Hier muss die Infrastruktur fertig

sein, damit die Tests starten können. Auch in der Testphase ist immer wieder aufgefallen, dass die bereitgestellte Infrastruktur in sich nicht immer konsistent war. So waren Server trotz Standard VMware Image gerne mal unterschiedlich konfiguriert, es fehlten Berechtigungen oder Benutzer waren nicht angelegt. Um die Fehler in der Massenbereitstellung zu reduzieren, hilft nur eines wirklich: **Automatisiere!** Stelle Services automatisch und ohne Expertenwissen bereit.

Baue einen Automaten mit einem Webfrontend. Gebe dem Entwicklungsteam einen Knopf, auf den man drücken kann und aus dem dann nach wenigen Minuten ein Server rausfällt. Ein Server? Wer will den einen Server haben? Erweitere den Knopf, so dass nach der automatischen Server-Installation (das kann eine public Cloud eh besser) auch gleich noch das Betriebssystem passend für den **Service** konfiguriert wird, dann der Service installiert und konfiguriert wird und in Produktion gestellt wird. Das kann eine public Cloud für unsere Services „out of the box“ nicht -> Cool, Mehrwert generiert. Klingt das gut? ☐



Nachher: Service Bereitstellung auf Foreman & Puppet Basis

Leute aus den Entwicklungsteams und dem IT-Betrieb haben also damit begonnen, basierend auf Foreman, Puppet und ein paar anderen Helferlein eine automatisierte Service Bereitstellung (Intern kurz: ASB) aufzubauen. Der Fokus lag im ersten Schritt auf dem Basis Betriebssystem und JAVA/Tomcat Services. Davon hatten wir die meisten, und es würde somit den größten Nutzen bringen. In dieser Zeit haben wir auch die Basis-Architektur für den Automatismus definiert und aufgebaut. Darauf basieren bis heute alle neu entwickelten Automatismen z.B. für DNS oder Apache Webserver. Wir haben einen starken Fokus auf den Aufbau der ASB-Infrastruktur gegeben, wodurch sicherlich auch

andere Arbeit liegen geblieben ist. Jedoch konnten wir nur so den Entwicklungsteams die Möglichkeit geben, Server inkl. Services ohne Rückfrage beim IT-Betrieb aufzubauen. Egal in welcher Stage. Konsequenz: Der SelfService wird so gut genutzt, dass wir heute über alle Stages weit über 1.000 Server betreiben.

Was haben wir damit erreicht? Wir haben die Server- & Service-Installation nun via einfaches Webfrontend an die Entwicklungsteams gegeben. Wir müssen keine Termine finden, es ist keine lange Abstimmung mehr notwendig, keine langen Rückfragen, keine Nachbesserung, keine acht Wochen Wartezeit... Der Aufwand für den IT-Betrieb und die Entwicklungsteams wurde stark reduziert, und die Durchlaufzeit für Server- und Service-Installationen wurde von Wochen auf wenige Minuten reduziert. Win/Win Situation!

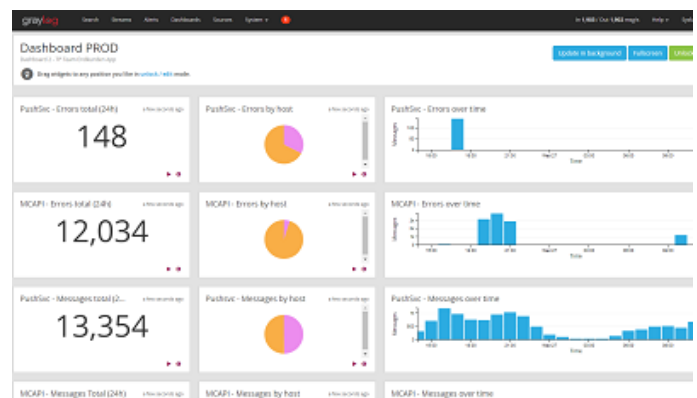
## Beispiel 2: Zentrales Logdaten Management (ZLM)

[illegible]

## Vorher: Logdaten Analyse auf der Console

OK, ein SelfService, um Services automatisch bereitzustellen, das ist schon mal super. Aber auch nur ein Baustein. Oft ist uns in der Vergangenheit das Logdaten Management System auf ELK-Stack-Basis (ELK = Elastic Search, Logstash und Kibana) unter der Last zusammengebrochen. Das lag weniger an der Technologie „ELK-Stack“ sondern eher an der eigenen Architektur, die, als wir sie vor Jahren initial für IT-Betrieb Systeme designed haben, nicht für die inzwischen vorliegenden Last ausgelegt war. Das verursacht wieder unfrohliche Stimmung in den Entwicklungsteams, weil wichtige Daten fehlen und das System nicht

verfügbar war. Gleichzeitig hat der IT-Betrieb wieder ein neues, ungeplantes, dafür dringendes Ticket, um das alte System mit viel Klebeband wieder ans Laufen zu bringen. OK, lass deine Arbeit *jetzt* liegen, kümmere dich (schon wieder) um einen Störfall im Logdaten Management. Macht weder glücklich noch bringt uns die Störung nach vorne. Von schneller werden reden wir hier auch nicht. Aber es gibt ja nicht nur Störungen. SOA und Microservice sei Dank gibt es ja fast täglich neue Services, die auch in das Log-Management möchten. Mal abgesehen von den Kapazitäten im Logdaten Management System beginnt das traditionelle Spiel im Ticketsystem: Neues Ticket: „Ich benötige Logging“, klar baue ich dir gerne, so fertig, hmm ich sehe nichts, ups jetzt aber, OK funktioniert, aber warum kann ich nicht mehr Daten ins System geben, .... aus den Büchern der menschlichen Schnittstellenkommunikation □ . Wie kann ich solche Kommunikation vermeiden? Baue einen **SelfService**! Plane und implementiere eine Logdaten Management Infrastruktur, die leistungsstark genug ist, um dein Volumen locker zu handhaben und wenig störanfällig ist. Sorge dafür, dass die Infrastruktur skaliert (nach oben und nach unten). Stelle eine gut dokumentierte Schnittstelle bereit, über die ein Entwicklungsteam selber seine Daten in das Logdaten Management schreiben kann und die Möglichkeit hat, die Daten selber auszuwerten.



Nachher: Logdaten Management auf Graylog Basis

Auch den Aufbau eines neuen und leistungsfähigen Logdaten Managements konnten wir recht früh starten. Auch hier setzen wir auf das bewährte Prinzip, Entwicklerteams und IT-Betrieb zusammenzubringen. Schnell wurde aber klar, dass es nicht besonders schlau ist, alle Logdaten, die es gibt, ohne Sinn und Verstand in eine Datenbank zu pressen. Das ist zwar möglich, aber nicht schlau. So haben wir uns u.a. mit den Fragen auseinandergesetzt „Was und wie loggen



wir überhaupt?“, „Wann brauchen wir eine Logausgabe?“ und „Was machen die Logdaten?“. Nach vielen Gesprächen über Anforderungen und Optionen haben wir einen Vorschlag erarbeitet, wie die Entwicklungsteams Logging in ihren Services nutzen können. Parallel dazu haben wir eine neue Logdaten Management Infrastruktur basierend auf Graylog aufgebaut. Graylog bietet zusätzlich zum ELK-Stack noch ein paar nette Features wie z.B. ein User Management. Auch erschienen uns das Handling und der Betrieb im Vergleich zum ELK-Stack etwas einfacher. Gesized wurde die neue Infrastruktur für Spitzenlastzeiten. Bei uns ist das neben dem Schulbuchgeschäft natürlich das Weihnachtsgeschäft. So kamen wir auf eine Infrastruktur von u.a. zwei Elastic Search Master Nodes und acht Elastic Search Data Nodes in zwei Datacentern. Durch die Data Nodes ist die Infrastruktur jederzeit ohne viel Aufwand skalierbar. Im Peak kann die Infrastruktur bis zu 100.000 Messages die Sekunde verarbeiten. Um die künftigen Nutzer der Infrastruktur zu informieren und möglichst früh Feedback zu bekommen, haben wir zum Thema eine von uns so getaufte „Bier-Session“ angesetzt. Eine „Bier-Session“ ist in etwa vergleichbar mit einer Brownbag-Session, jedoch war es nicht in der Pause und es gab (alkoholfreies) Freibier. Das Feedback haben wir gerne aufgenommen und in die Infrastruktur eingebaut.

Was haben wir mit der neuen Infrastruktur erreicht? Die Produkt-Teams können nun ihre eigenen Anwendungen durch eine einfache Konfiguration in der „logback.xml“ anbinden. Auch die Dashboards zur Auswertung der Logdaten können sich die Teams ohne Unterstützung des Platform Engineering Teams einrichten. Ein weiterer schöner SelfService, der die Aufwände im IT-Betrieb reduziert und die Arbeit in den Produkt Teams beschleunigt.

## **Beispiel 3: Automatisiertes Monitoring**



Vorher: CACTI Monitoring hat seinen Dienst getan

Monitoring war auch so eine super Sache. Wir monitoren ja echt viel – mit einigen unterschiedlichen Monitoring- & Alarmierungssystemen bekommen wir einen super Überblick über den Zustand unserer Services aus technischer Sicht und aus Kundensicht. Jedoch mit was für einem Aufwand! Sämtliche Sensoren wurden von einem Menschen (immerhin haben wir für das Monitoring einen dedizierten Menschen) manuell in die Monitoring Systeme eingetragen. Also erstellt das Entwicklungsteam ein Ticket für das Monitoring, da fehlen aber noch Information im Ticket, wieso fehlen da noch Information, ich erkläre es dir, jetzt habe ich ein Ticket mit allen Information, die Sensoren wurden eingebaut, da fehlt aber noch ein Sensor, OK, warum hast du das nicht gleich gesagt, nun ist der Sensor auch eingebaut, aber wo ist der Aktionsplan für die Alarmierung, der kommt später, .... Hatten wir das mit den Schnittstellen nicht schon mal? Gibt es eine Lösungsidee? Klar: Bau eine **SelfService** Schnittstelle! Im gleichen Zuge, in dem wir die SelfService Schnittstelle gebaut haben, haben wir auch unser solides, jedoch in die Tage gekommenes CACTI durch eine moderne Lösung abgelöst. Von der neuen Lösung versprochen wir uns mehr Möglichkeiten, Dashboards, Graphen, Informationen etc. bedarfsorientiert zu visualisieren, um dadurch schneller und einfacher entstören und planen zu können.



## Nachher: Performance Monitoring mit InfluxDB & Grafana

Dank der frühen Planung und der Rahmenbedingungen, die geschaffen wurden, konnten wir auch damit beginnen, einen weiteren SelfService für das automatisierte Monitoring zu bauen. Möglich wurde das u.a. dadurch, dass wir uns externe Unterstützung ins Team geholt haben, die uns den Rücken im Tagesgeschäft frei gehalten hat, so dass die internen Mitarbeiter sich um die Entwicklung der neuen Technologie kümmern konnten. Das klingt alles super und hat uns sehr geholfen, ein Überschuss an Mitarbeitern war aber dennoch nicht zu erkennen. Wir haben einige Themen parallel bearbeitet. Doch wir haben das Beste daraus gemacht. Und so kam es z.B., dass wir u.a. einen unserer Datenbank Spezialisten überzeugen konnten, zusammen mit einem Monitoring-Spezialisten und Software-Entwicklern ein neues Performance Monitoring System basierend auf InfluxDB, Telegraph und Grafana zu bauen. Die Arbeiten wurden stark unterstützt von Produkt-Teams, die ihre Ideen und Anforderungen eingebracht haben. Nachdem der erste brauchbare Prototyp stand, haben wir vor der Pilotphase eine weitere „Bier-Session“ durchgeführt. Die Session hat allen Nutzern einen frühen Blick auf das künftige System gegeben und uns gutes Feedback gegeben, welches wir einarbeiten konnten.

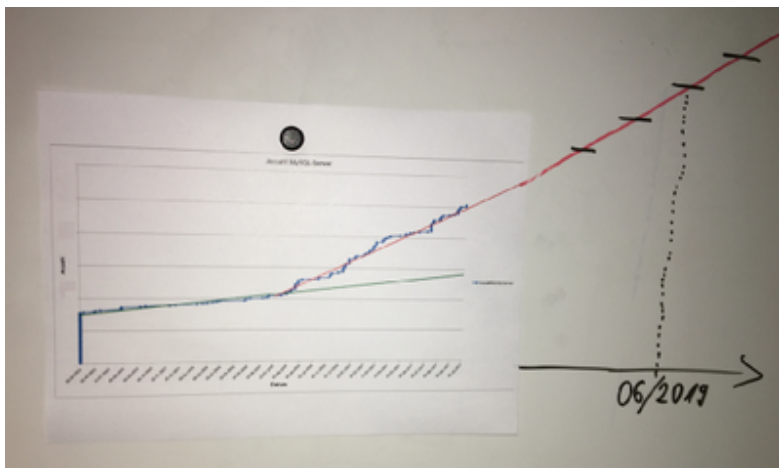
Ein neues Performance Monitoring hilft aber nur in Teilen weiter. Es ist leistungsstark, voller Features und kann ohne Hilfe des IT-Betriebs genutzt werden. Jedoch fehlt noch eine Lösung für unser Alarmierungssystem und dessen Aktionspläne. Daher haben wir zusätzlich zusammen mit Software Entwicklern, Qualitätstestern und Administratoren einen komplett neuen Service gebaut, der es erlaubt, aus Informationen aus einer YAML-Datei die Sensoren in der Alarmierung sowie dem Performance Monitoring zu erstellen und zu konfigurieren. Dadurch haben wir den Entwicklungsteams eine Schnittstelle gegeben, die sie aus ihrem Tagesgeschäft kennen, haben die Monitoring

Konfiguration zu Code gemacht und versioniert, die manuellen Aufwände reduziert und die Qualität erhöht. Klingt nach einem coolen SelfService? Ist er auch! ☐

## **Was haben wir mit den Änderungen an den Basistechnologien erreicht?**

Wir haben eine Menge Arbeit in die Bereitstellung von SelfServices und Automaten investiert. Dadurch haben wir es geschafft, die aufwändige Abstimmung zwischen Entwicklungsteams und IT-Betrieb zu reduzieren. Wir haben durch Automation und technische Schnittstellen die Qualität in der Umsetzung von Anforderungen erhöht und den Bedarf für Nacharbeiten reduziert. Dadurch haben wir einen wichtigen Betrag zur Beschleunigung der Entwicklungsteams geleistet. Die Aufwände für wiederkehrende Arbeit wurden im IT-Betrieb reduziert, wodurch mehr Zeit für Weiterentwicklung entstanden ist.

## **Gibt es auch Schattenseiten?**



Wachstum DB-Systeme der letzten 2 Jahre inkl.  
„Prognose“

Natürlich ist nicht immer alles rosarot, und auch wir haben durch Schmerzen gelernt. So wird z.B. unser SelfService zur automatischen Service Bereitstellung sehr gerne und viel genutzt, was erst einmal sehr positiv ist. Hier ist ein Beispiel, wie sich die Anzahl unserer Datenbanksysteme über die letzten 2 Jahre geändert

hat und welches Wachstum wir noch erwarten. Ratet einfach mal wann die automatische Service Bereitstellung verfügbar war □ Eines der verbundenen Probleme ist, dass die Anforderung an alle Komponenten der Plattform (Beispiel: Storage, CPU, RAM, Virtualisierung, ...) deutlich gestiegen sind. Auch die Anforderungen an die für den Betrieb notwendigen Systeme (Beispiel: Datensicherung, Monitoring, Logging, ...) sind davon nicht ausgenommen. Natürlich haben wir mit einem spürbaren Anstieg der Systeme und somit auch mit den Anforderungen gerechnet. Dennoch wurden wir ein Stück vom Erfolg überrascht. Fehlende Ressourcen in der Plattform haben zu Störungen in den Services geführt, was wir nachträglich im laufenden Betrieb mit Schmerzen in allen Teams korrigieren mussten. Auch die steigenden Kosten sowie die Kostenkontrolle für Plattform Ressourcen müssen bei der Einführung und dem Ausbau von SelfServices berücksichtigt werden. Aktuell verrechnen wir die Kosten nicht an die Teams, jedoch stellen wir über Reporting sicher, dass jedes Team weiß, was ein System kostet und welche Kosten für Plattform Ressourcen ein Team produziert.

## ***Kapitel 3: Mindset, Methoden, Prozesse und mehr...***

In einer Woche werde ich im dritten und vorerst letzten Kapitel unserer Reise darüber berichten, dass neben der Technologie auch kulturelle und methodische Änderungen notwendig waren. Z.B. war unser Mindset das eines klassischen IT-Betriebs. Das Mindset unserer wichtigsten Kunden, der Entwicklungsteams, war jedoch das einer agilen Software-Entwicklung. Und was ist eigentlich dieses DevOps? Hier gab es also noch etwas zu tun. Abgerundet wird das dritte Kapitel durch ein Fazit, was wir gelernt haben. To be continued ...



# Alle drei Kapitel im Überblick



[Kapitel 1: Woher kommen wir? 6 Jahre im Schnelldurchlauf](#)



[Kapitel 2: Basistechnologien, SelfServices & Automation](#)



[Kapitel 3: Mindset, Methoden, Prozesse und mehr...](#)

---

## Vom IT-Betrieb zu Platform Engineering. Ein Reisebericht (1/3)



Es gibt einige Artikel auf unserem Tech Blog, die über unseren Wandel zur Produkt-Organisation berichten. Auch wenn der IT-Betrieb bei solchen Aktivitäten gerne mal vergessen wird, so war es bei unserem Wandel zur Produkt-

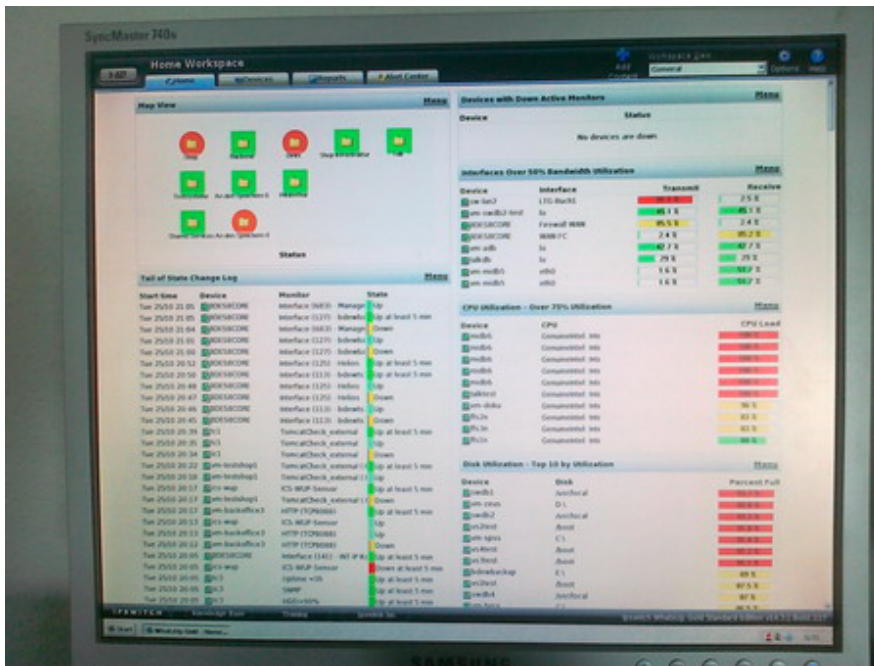
Organisation nicht der Fall. Als Learning aus anderen Umstellungen und Unternehmen wurde der IT-Betrieb sehr früh in den Prozess einbezogen. Nach gut einem Jahr kann man sagen: Das hat sich bezahlt gemacht!

Die folgenden Zeilen sind ein Reisebericht aus dem Blickwinkel von IT-Operations. Woher kommen wir? Warum wollten wir uns verändern? Wie haben wir das gemacht? Was haben wir dabei gelernt? Diese kleine Artikelserien in drei Teilen gibt einen Überblick über die Themenblöcke, deren Hintergründe und Inhalte. Ich gehe dabei jedoch nicht auf jedes einzelne Detail ein. Eventuell folgen diese ebenfalls sehr spannenden Details in späteren Artikeln.

## ***Kapitel 1: Woher kommen wir? 6 Jahre im Schnelldurchlauf***

### **Die Reise beginnt**

Vor rund sieben Jahren haben wir nach und nach den IT-Betrieb aufgebaut. Dazu haben wir über die Zeit Datenbank-, Linux- und Windows-Spezialisten an Bord geholt und zu einem Team formiert. Eines war dabei immer klar: Bleib nah an der eigenen Entwicklung, die unsere eCommerce Software baut. Da die gesamte IT damals noch eine überschaubare Größe hatte und auf einem Flur saß, war das auch recht einfach. Genau wie heute war bereits damals die Stimmung sehr positiv, der Zusammenhalt und die gegenseitige Unterstützung sehr groß.



## Monitoring & Alarmierung (2010)

Im Jahr 2010 hatten wir für [thalia.de/ch/at](http://thalia.de/ch/at) lediglich rund 100 Server, die große Monolithen und einige wenige Services beherbergten. Eine vergleichsweise überschaubare Systemlandschaft mit Linux, Tomcat, JAVA, Apache, MySQL. Schon damals hatten wir kein eigenes Datacenter, sondern haben Datacenter as a Service (DCaaS) genutzt. Wir waren nicht darauf aus, unsere Zeit primär mit einem Schraubendreher im Rechenzentrum zu verbringen. Uns war immer klar, dass wir als Team den größten Mehrwert nah an den vom Kunden genutzten Services bringen können. Server in ein Rack einschrauben und Betriebssysteme installieren können viele Dienstleister und Cloudanbieter besser als wir.

Unsere Aufgabe war es, den Betrieb der Systeme und der Services rund um die Uhr sicherzustellen. In guter IT-Betriebs-Tradition haben wir durch Prozesse, Standardisierung und Professionalisierung einen guten Beitrag zur Verbesserung der Verfügbarkeit unseres Gesamtsystems und somit zum Umsatz geleistet. Auch wenn längst nicht alles perfekt war, so machten wir doch gute Fortschritte, und wir wurden stetig besser – was man u.a. auch an der Verfügbarkeit der Shop Systeme erkennen konnte.

## Insourcing der Entwicklung & SOA-Architektur

Doch nicht nur der Betrieb entwickelte sich weiter. Auch die Software-Entwicklung machte große und tolle Fortschritte. Damals wurde Software für

unser eCommerce System zu einem großen Teil extern entwickelt. Diese Entwicklung wurde ins Haus geholt (hmm, das klingt gerade leichter als es wirklich ist – bestimmt schiebt einer der Kollegen mal einen Artikel dazu ☐ ). In diesem Zusammenhang wurde auch das Deployment, welches zuvor vom externen Dienstleister durchgeführt wurde, in den noch jungen IT-Betrieb übergeben. Ungefähr zur gleichen Zeit hatte die Software-Entwicklung eine echt gute Idee: [SOA-Architektur](#)! Tötet die Monolithen! Nach einem kurzen Termin mit unserem Software-Architekten, der die Idee vorstellte, sagten wir völlig ahnungslos dafür aber um so selbstbewusster: Klar, kein Problem! Und so kam was kommen musste. Der IT-Betrieb wurde von Anforderungen überrollt und konnte die Software-Entwicklung nicht bedarfsgerecht bedienen. Hinter Prozessen und dem Ticketsystem haben wir versucht, in Deckung zu gehen und die Anforderungen zu verstehen, zu priorisieren und sequenziell abzuarbeiten. Waren wir damals schnell? Nun ja, es gab da durchaus Verbesserungspotential.



Sysadminday (2012)

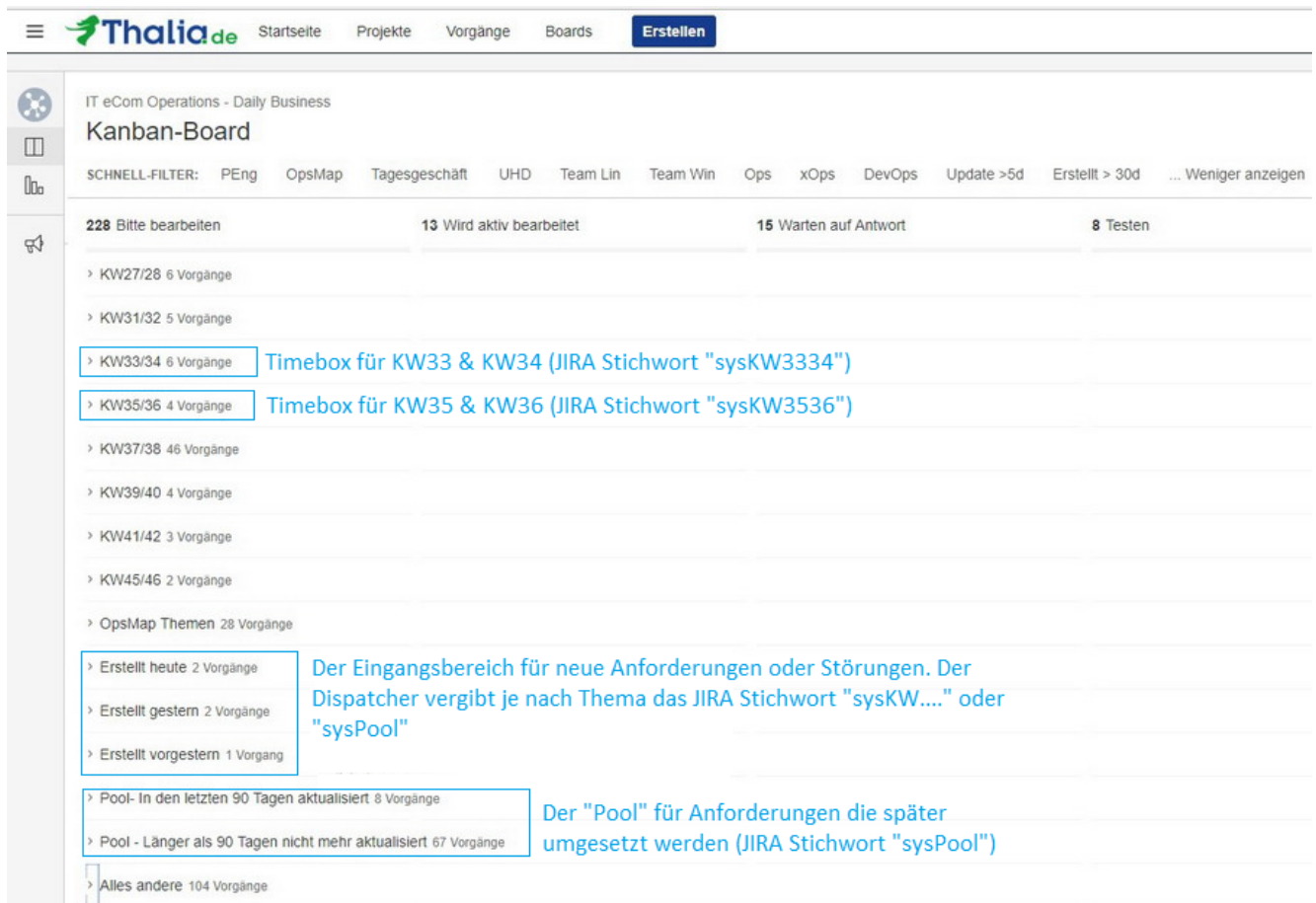
Die gesamte Schnittstelle zwischen der Entwicklung und dem IT-Betrieb hatte einen massiven Overhead und verursachte hohe Reibungsverluste. Klar waren wir (Entwicklung & IT-Betrieb) immer noch gute Kollegen, haben uns geholfen wo es ging und haben freudig (inzwischen auf zwei Etagen) zusammengearbeitet. Events unterschiedlichster Art wie z.B. den Sysadminday haben wir gerne gefeiert und machen das immer noch. Der Stresspegel auf beiden Seiten stieg jedoch permanent an. Jeder im IT-Betrieb hatte im Ticket System seine 100 Vorgänge und mehr zu tun als er schaffen konnte. Transparenz für den Anforderer und für den Bearbeiter gab es trotz Ticket System jedoch schon lange nicht mehr. Klares Priorisieren der Masse war nahezu unmöglich. Zeit für eine

Veränderung!

## **IT-Operations meets Kanban**

Überrollt von Anforderungen aus der Entwicklung sowie aus dem IT-Betrieb mussten wir mehr Struktur und Transparenz in unsere Arbeit bringen. Wir waren ein klassischer IT-Betrieb und hörten zum ersten Mal etwas von „Kanban“. Hmm, was ist das? Was kann das? Sieht erst einmal spannend aus. Da wir damals schon JIRA mit dem AGILE Plugin im Einsatz hatten, haben wir einfach mal unsere Tickets in einem Kanban Board anzeigen lassen. Toll, 500 Tickets auf einem Board. Das soll jetzt helfen? Es hat viele Anläufe gebraucht, bis wir es tatsächlich geschafft haben, ein Kanban Board zu entwickeln, welches uns als IT-Betrieb den Fokus auf die „jetzt“ wirklich wichtigen Tickets gegeben und dem Anforderer ein Feedback gegeben hat, wann ein Ticket aller Wahrscheinlichkeit nach umgesetzt wird. Von der Grundidee haben wir unsere Themen sichtbar gemacht und Timeboxes von 2 Wochen eingeführt. Jedes Ticket wurde einer Timebox zugewiesen oder gezielt in den Pool gelegt, wenn es später bearbeitet werden sollte. Auch Tickets unbearbeitet jedoch kommentiert schließen kann sinnvoll sein, führt aber gerade am Anfang zu Diskussionen.





Damit das Board funktioniert, haben wir einen Filter auf alle offenen Tickets des IT-Betriebs gelegt. Das Board selber hat drei Kategorien von Swimlanes: (1) **Timeboxen** für die Bearbeitung, (2) den **Eingangsbereich** und (3) den **Pool**.

Eine **Timebox** dauert immer zwei Wochen. Tickets in einer Timebox werden mit dem JIRA Stichwort „sysKWxxxx“ getagged wobei „xxxx“ die jeweiligen zwei Kalenderwochen angeben (Beispiel sysKW0102, sysKW0304, sysKW0506, ...). Wird ein Ticket z.B. mit dem Stichwort „sysKW3334“ versehen, so erscheint das Ticket in der Swimlane der Timebox „KW33/34“. Diese Timebox gibt dem Bearbeiter die Chance sich nur auf Tickets in der Timebox zu fokussieren (und nicht auf alle 500 offenen Tickets). Der Anforderer kann am Stichwort erkennen, wann sein Ticket gemäß Planung bearbeitet werden soll.

Neue Anforderungen oder Störungen landen automatisch im **Eingangsbereich**. Ein Dispatcher entscheidet nun anhand von Prioritäten (ggf. sind hierfür Rückfragen beim Anforderer notwendig), in welcher Timebox ein Ticket planmäßig abgearbeitet werden soll. Um ein Ticket in eine Timebox zu packen, muss er lediglich das JIRA Stichwort „sysKWxxxx“ vergeben.

Ist die Priorität einer Anforderung oder eine Störung aktuell nicht besonders

hoch, so wird das zugehörige Ticket im **Pool** abgelegt. Dazu vergibt das Dispatcher das Stichwort „sysPool“. Auch ist es der Dispatcher, der immer wieder den Pool prüft, ob Tickets aufgrund veränderter Prioritäten in eine Timebox geschoben werden sollten. Der Anforderer wird automatisch per Mail über die Vergabe von JIRA Stichwörtern und somit den Planungsstatus des Tickets informiert. Ist er mit der Timebox oder dem Pool nicht einverstanden, so kann er beim Dispatcher die Priorität besprechen und ggf. verändern.

## **Was haben wir bis hier erreicht?**

Wir hatten nun eine gute Übersicht über alle Themen und in welchem Status die Themen waren. Wir konnten Themen priorisieren, abmelden oder auf später verschieben. So konnten wir die wichtigen Themen früher machen und die vermeintlich unwichtigeren Themen später oder gar nicht. Im Team hatten wir mit einem Mal Struktur in den Themen. Die Team-Mitglieder hatten nicht mehr den Druck, alles auf einmal machen zu müssen. Sie hatten einen klaren Blick auf eine handhabbare Anzahl an Tickets. Dadurch konnten sie fokussierter an den wichtigen Themen arbeiten.

**Ergebnis:** Die Zufriedenheit bei Anforderer und Bearbeiter stieg. Die Zahl der Eskalationen reduzierte sich. Ein großer Schritt nach vorne!

## **Warum mussten wir uns dennoch weiterentwickeln?**

Waren wir jetzt schnell genug? Nein! Auf keinen Fall!

Per Prozessdefinition musste ein virtueller Server, der in Produktion eingesetzt werden sollte, rund acht Wochen vor Deployment angemeldet werden (3 Wochen Planung, 1 Woche bauen, 4 Wochen die neue Software testen). Die echte Arbeitszeit in den acht Wochen lag jedoch nur bei ~3PT. Ein Großteil der Zeit während der acht Wochen ging für die Terminfindung, Warten, Abstimmung, Rückfragen, ... bei den beteiligten Teams drauf. Nicht jeder fand das super. Auch wurden die Aufgaben (insbesondere aus dem Tagesgeschäft) immer mehr. Wir hatten kaum Zeit für Innovation und Weiterentwicklung von Themen aus dem

Bereich IT-Betrieb. Wir waren zwar entspannter mit einem besseren Blick – jedoch immer noch Getriebene.

Die Reise geht also weiter ...

## ***Kapitel 2: Basistechnologien, SelfServices & Automation***

In einer Woche werde ich im zweiten Kapitel unserer Reise darüber berichten, was wir im Bereich der Technologien gemacht haben. Wie konnten wir Werkzeuge nutzen, um uns zu beschleunigen und die Schnittstellen zu den Produkt-Teams genauer definieren? To be continued ...

## **Alle drei Kapitel im Überblick**



[Kapitel 1: Woher kommen wir? 6 Jahre im Schnelldurchlauf](#)



[Kapitel 2: Basistechnologien, SelfServices & Automation](#)



[Kapitel 3: Mindset, Methoden, Prozesse und mehr...](#)

---

# **OVSoftware Scrum Workshop**

Für die Mitarbeiter von Thalia gab es am 13.09.2017 ein besonderes Angebot:

## **„Scrum“ in einem Workshop ohne Software-Entwicklung kennenlernen**

Das Unternehmen OVSoftware bietet ein Scrum – Spiel an, zu dem die Teilnehmer einfach ein gewisses handwerkliches Geschick mitbringen müssen.

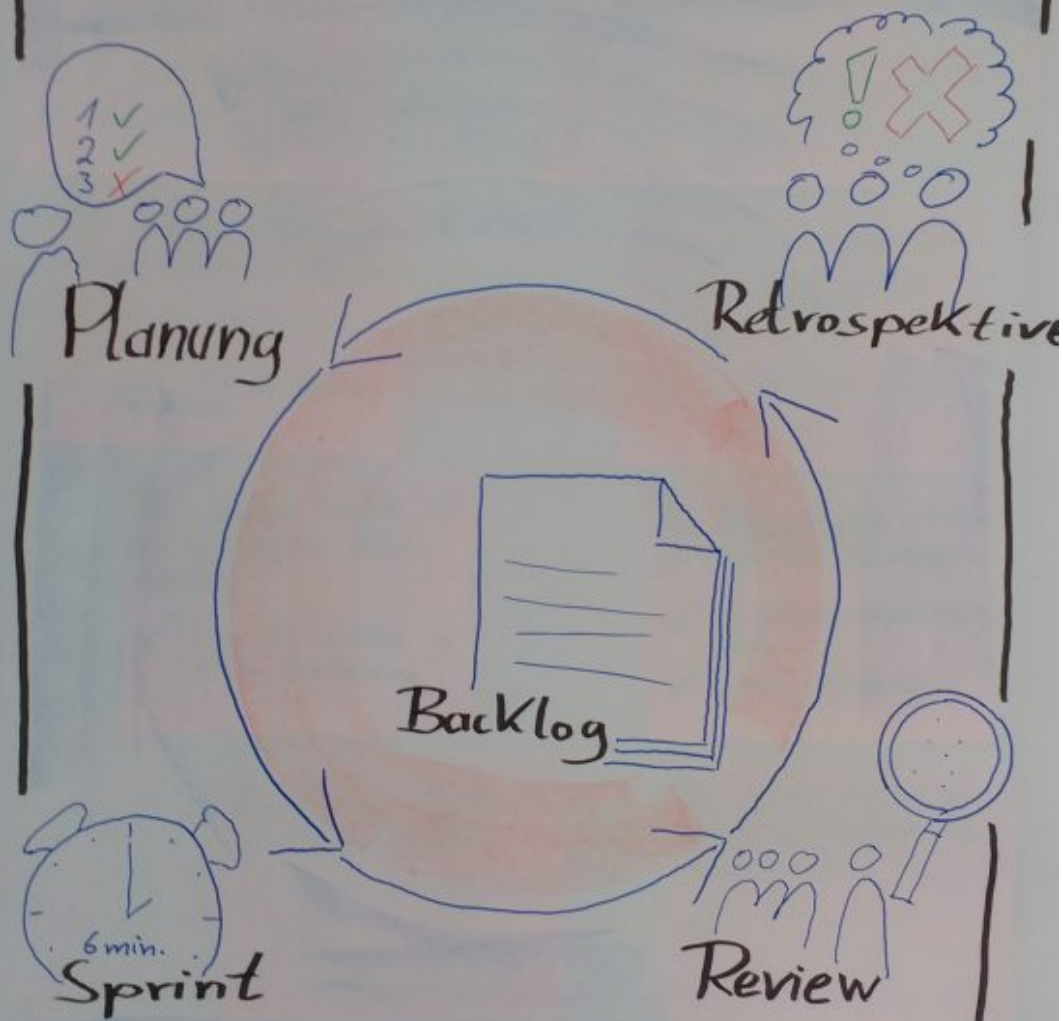








# Scrum Regeln



Die

Workshop-Leiter gaben zu Beginn eine kurze Einführung in die Entwicklungsmethode. Anschließend wurde die Vision für das zu entwickelnde Produkt vorgestellt. Für den Anreiz untereinander wurden die Teilnehmer in 5 Teams aufgeteilt.











Jedes Team

erhielt einen Arbeitsplatz. Nach Vorstellung der ersten „User Storys“ ging jedes Team mit einem Commitment in den ersten Sprint. Nach 6 Minuten wurde die Arbeit eingestellt.



Alle Teams traten zum ersten Review an und die Workshop Leiter bewerteten die Ergebnisse hinsichtlich zu erfüllender Abnahmekriterien. Nach einer kurzen Retrospektive wurden alle Teams wieder zu einem Commitment gebeten.



Getreu dem agilen Ansatz „Inspect & Adapt“ wurden weitere Sprints im

Workshop durchgeführt. Am Ende ging jedes Team mit einem vollendeten Prototyp und viel Spaß an einer neuen Arbeitsmethode aus dem Workshop.

Solltet ihr mir bis hierhin gefolgt sein und Euch fragen: „Worum geht`s hier eigentlich?“

**...Ich habe bewusst hier das Produkt nicht näher vorgestellt, um noch Appetit für weitere Workshops anzuregen.**



---

## Münster macht Agil

### Wow, was für ein schöner Start!

Ca. 50 Interessierte aus Münster und Umgebung sorgten bei regionalem Wetter für einen Speicher 6 gefüllt mit Gedanken, Austausch und interessanten Gesprächen rund um das Thema „Agile“, insbesondere „Scrum“.





Um 19 Uhr gab es zunächst ein „social warmup“ bei italienischem Heißgebäck und Kaltgetränken.

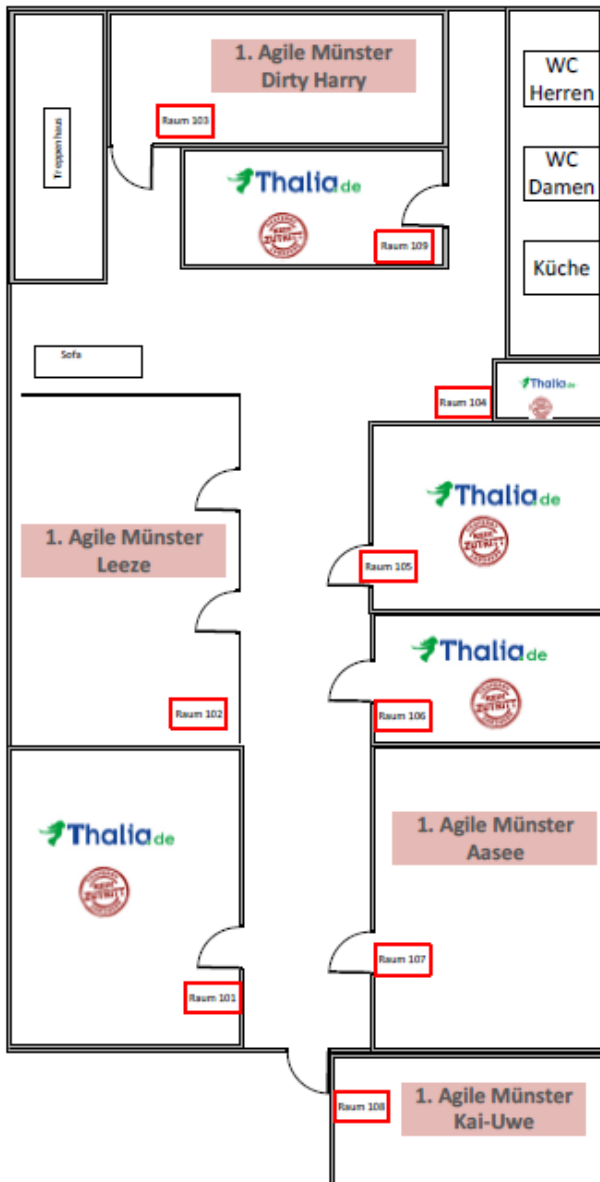
Patrick und Matthias haben nach einer kurzen Begrüßung die Teilnehmer mit Kennenlernfragen rund um das Thema Agilität bunt gewürfelt und anschließend den Marketplace für die Themen und Fragen eröffnet.



Bereits 5 Minuten später war die Matrix mit 13 Themenvorschlägen gefüllt und die ersten Sessions wurden gestartet. Zur Verfügung standen die 4 Räume Leeze, Dirty Harry, Aasee und Kai-Uwe.



## 1. Agile Münster Raumplan



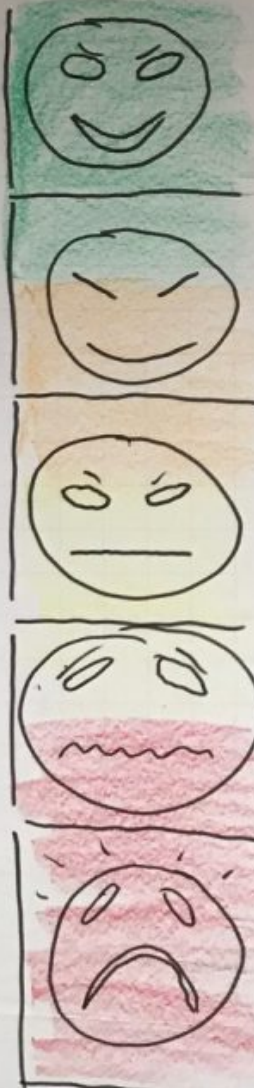
Von Fragen wie „**Scrum vs Kanban**“ beziehungsweise „**Was mache ich mit Daily Business in geschützten Sprints?**“ über **Best Practices für die Ausbildung von Product Ownern** gab es sogar eine DIY-Session mit Lego zum Thema „**Agile Transition - vorher, nachher**“.



Um 21:45 Uhr schlossen Matthias und Patrick den Marketplace und damit auch den Open Space mit einer Retrospektive zum ersten Abend.



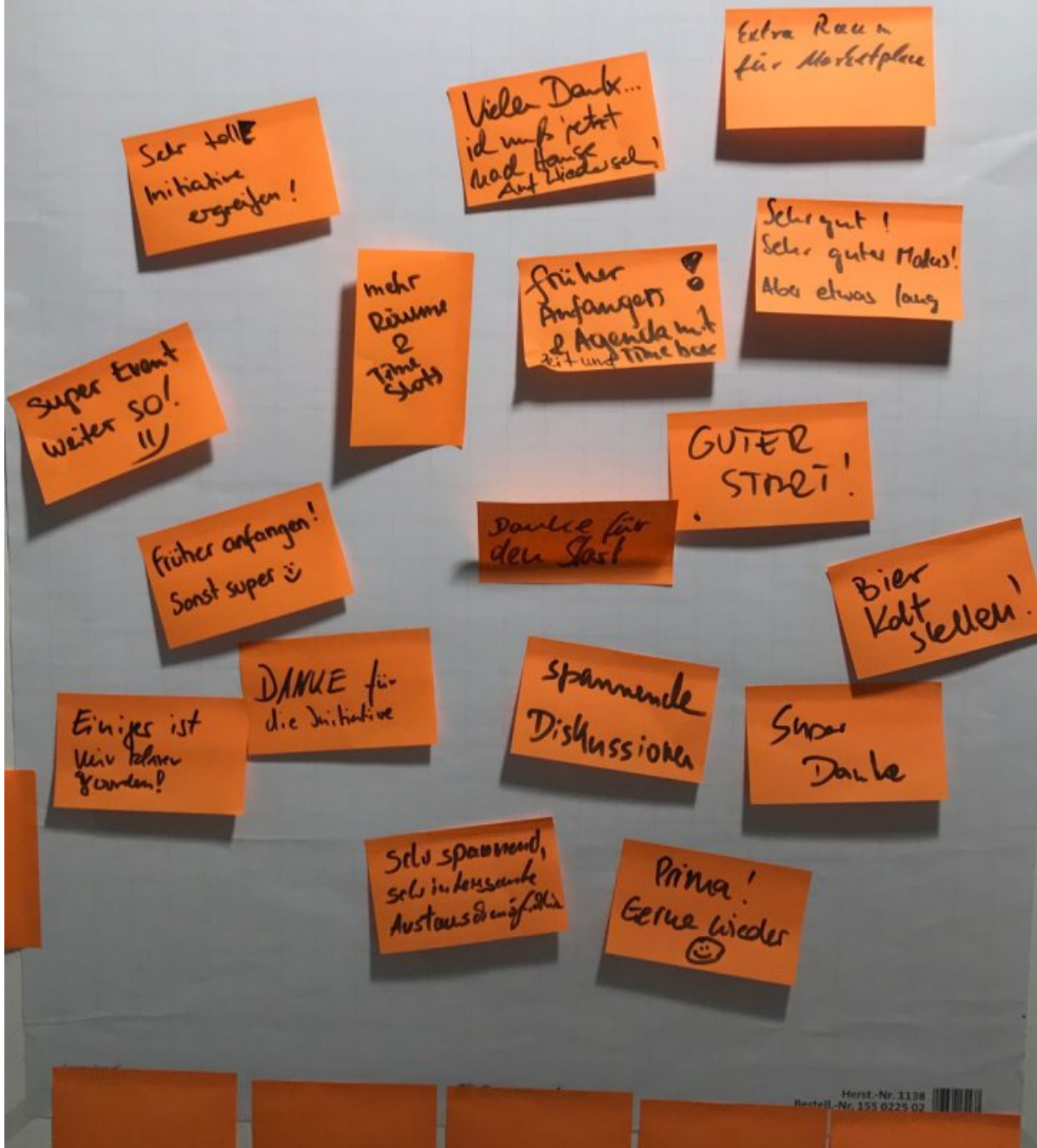
Wie fühlst Du Dich?





# Feedback

Wie war es für Euch?



Ein Dank gilt neben dem Sponsor Thalia auch den freiwilligen Helfern, die auch nach Abreise der Gäste noch halfen, die Büros wieder herzustellen.

Wir freuen uns schon auf das nächste Mal, wenn es heißt:

# Münster macht Agil

Dieser Beitrag entstand mit:



Matthias Hochschulz Agile Coach /  
Scrum Master

---

## Happy SysAdmin Day!

Es ist der letzte Freitag im Juli. Die Flure sind mit Danksagungen tapeziert. Die Leute sind noch freundlicher zu den System-Administratoren. In der Kaffee-Küche stehen Süßigkeiten.

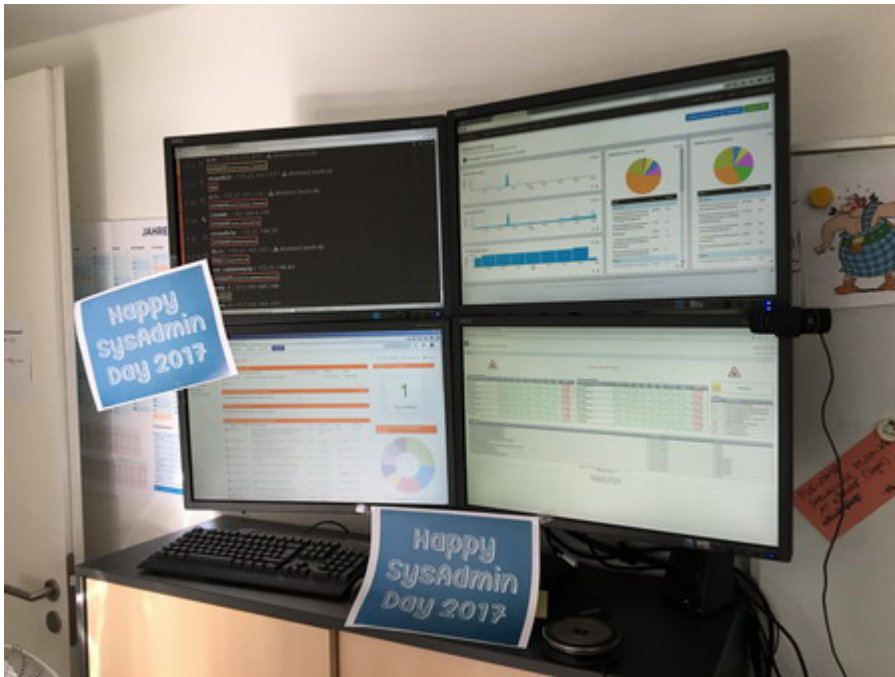
Das kann nur eines bedeuten.... **Es ist SysAdmin Day!**

**Vielen Dank allen System-Administratoren** für die unermüdliche Arbeit in und für unsere Crossfunctional Teams.









Btw: Liebe Entwickler, habt noch ein wenig Geduld. Am 256 Tag des Jahres ist der „[Day of the Programmer](#)„. Auch die QA kommt nicht zu kurz. Am 9.11. ist „[World Quality Day](#)„