Prüfungsfragen in Swen2

Erklären Sie den Begriff "SWEBOK"

SWEBOK ist eine Abkürzung für "Software Engineering Body of Knowledge". Dabei handelt es sich um ein Handbuch welches das gesammelte Wissen im Softwareentwicklungsbereich beinhaltet. Dieses Dokument wird von der IEEE Organisation weiterentwickelt. Das Ziel ist den aktuellen Stand der Technik darzustellen und zu vereinheitlichen. Es soll als Grundlage für die Qualifizierung und Zertifizierung von Informatikern dienen.

Das Projekt ist akamadisch fundiert und hat sich als Quasi-Standard durchgesetzt

Nennen Sie mindestens 4 Vor- und Nachteile von plangetriebenen Projektmethoden

Das bekannteste Beispiel unter den plangetriebenen Projektmethodiken ist das berühmte Wasserfall-Modell.

Vorteile:

- Einfach zu verstehen und anzuwenden
- Klare Abgrenzung der einzelnen Phasen mit entsprechenden abzuliefernden Ergebnissen/Artifakten (Dokumente, Software, etc.)
- Die einzelnen Phasen werden hintereinander durchgeführt ohne Überlappungen
- Funktioniert gut bei kleineren Projekten, wenn die Anforderungen klar definiert und verstanden sind und die Wahrscheinlichkeit, dass sich Anforderungen ändern klein ist.

Nachteile:

- Funktionierende Software wird erst sehr spät im Projekt Life Cycle erstellt. Dadurch sind Anpassungen aufgrund von Nutzerfeedback nahezu unmöglich.
- Sobald man in der Testphase angelangt ist, ist es äusserst schwierig in die Entwicklung zurückzugehen um Dinge zu ändern die initial nicht gut durchdacht wurden.
- Meist sehr hohes Risiko und Unsicherheit bei Softwareprojekten (weil meist sehr komplex)
- Schlechte Methodik für lange und weiterführende Projekte. Es ist schwierig ein Projekt zu planen welches auf unbestimmte Zeit in der Zukunft weiterentwickelt werden soll.
- Schlechtes Modell wenn eine mittlere bis hohe Wahrscheinlichkeit besteht, dass sich Anforderungen während der Entwicklung ändern

Geben Sie den Inhalt des Artikels "No Silver Bullet: Essence and Accidents of Software Engineering by Frederick P. Brooks" in groben Zügen wieder

Zusammengefasst kann man sagen, dass Software Engineering wie vieles im Leben, irgendwann an seine Grenzen stösst. Bis jetzt, ist keine Software zu 100% fehlerfrei oder zuverlässig. In der Softwareentwicklung gibt es einen starken Bedarf nach Softwareentwicklungsmethoden um einfach und zuverlässig Software zu entwickeln. Im Gegensatz zur Entwicklung von Hardware verbessert sich Software nicht gemäss Moore's Law sondern wird zunehmend komplexer.

Laut Brooks, hängt das Problem damit zusammen, dass Software derzeit nicht entwickelt, um Softwareentwicklungsprobleme wie "Konformität, Veränderlichkeit, Flexibilität, Unsichtbarkeit und Komplexität" anzugehen.

Aktuell gibt es zwei Arten von Komplexitäten und zwar "Accidental Complexity" und "Essential Complexity". Bei "Accidental Complexity" handelt es sich um eine Form von Komplexität die ein Entwickler meist selbst einführt und auch wieder lösen kann. Beispiele hierfür sind Optimierung von Code, damit ein Batch Prozess schneller abgearbeitet werden kann. Bei "Essential Complexity" hingegen, handelt es sich um Komplexität, die durch eine Lösung für ein bestimmes Problem eingeführt wird. Im Gegensatz zu "Accidental Complexity" kann diese Komplexität nicht entfernt werden und gilt als gegegeben. Wenn ein Entwickler 30 unterschiedliche Dinge entwickelt sind das wiederum 30 "Essential Complexities".

Durch die Einführung von Hochsprachen (High-Level Languages) wie Ada,Java,C#,etc. konnten die "Accidental Complexities" auf ein Minimum reduziert werden.

Die Kernaussage in seinem Artikel ist, dass es für die Reduzierung von "Essential Complexity" keine einzelne "Silver Bullet" gibt die das Problem grundlegend löst sondern, dass es eine Vielzahl an Innovation benötigt um signifikante Verbesserungen in diesem Bereich zu erzielen.

See also:

- https://www.grin.com/document/321505
- https://en.wikipedia.org/wiki/No_Silver_Bullet

Erklären Sie die Werte und Bedeutung des "Agile Manifesto"

Das agile Manifest ist eine Zusammenfassung gemeinsamer Werte von unterschiedlichen Softwareentwicklungsmethoden

Das agile Manifest kennt folgende Werte:

- Individuen und Interaktionen mehr als Prozesse und Werkzeuge
- Funktionierende Software mehr als umfassende Dokumentation
- Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung
- Reagieren auf Veränderung mehr als das Befolgen eines Plans

Die Werte auf der rechten Seite sind dabei weder falsch noch schlecht aber sollten für die Werte auf der linken Seite vernachlässigt werden.

Erklären Sie die 12 Prinzipien des "Agile Manifesto"

Prinzipien ohne Erklärung: http://agilemanifesto.org/iso/de/principles.html

1. **Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufrieden zu stellen.**

   Im klassischen Projektmanagement bekommt der Kunde das fertige Produkt erst am Ende des Projekts präsentiert. Agile Methoden folgen einem anderen Ansatz: Hier werden kontinuierlich, d. h. in regelmäßigen Abständen, fertige Teilentwicklungen präsentiert. Bereits in der Entwicklungsphase werden

so Rückmeldungen des Auftraggebers oder der Zielgruppe gewonnen, die wiederum in die Entwicklung einfließen. So sollen Missverständnisse minimiert, neue Erkenntnisse berücksichtigt und veränderte Rahmenbedingungen bereits im Entwicklungsprozess antizipiert werden. Der Entwicklungsprozess beginnt also ergebnisoffen. Stellen wir uns einen Stadtplanungsprozess vor, bei dem das jeweilige Zwischenergebnis in regelmäßigen Abständen in einem öffentlichen Rahmen vorgestellt wird. Die Projektentwickler holen sich in diesem Rahmen immer wieder die Rückmeldung des Gemeinderats und interessierter Bürger, die ihnen zurückspiegeln, ob sie – als Betroffene – Verbesserungsbedarf sehen. Auf diese Weise lassen sich frühzeitig konfliktträchtige Themenfelder identifizieren.

2. **Heisse Anforderungsänderungen selbst spät in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.**

   Folglich sind Änderungen der Anforderungen jederzeit willkommen, sie werden nicht als lästig empfunden. Ganz im Gegenteil: Das Ziel ist es, unter den gegebenen Rahmenbedingungen möglichst das Beste zu schaffen, das zum aktuellen Zeitpunkt für den Kunden/Auftraggeber den maximalen Nutzen stiftet. Ein agiler Grundsatz lautet: Scheitere früh, scheitere schnell. Je früher wir erkennen, dass etwas nicht zielführend ist, desto früher können wir den Kurs neu bestimmen.

3. **Liefere funktionierende Software regelmäßig innerhalb weniger Wochen oder Monate und bevorzuge dabei die kürzere Zeitspanne.**

   Die Punkte 1. und 2. können nur funktionieren, wenn die Ergebnisse in regelmäßigen und möglichst kurzen Abständen präsentiert werden. Scrum sieht einen solchen Zyklus mit einer Gesamtlänge von maximal vier Wochen vor. Je kürzer desto besser. In jedem dieser Abschnitte wird ein voll „funktionsfähiger“ Teilaspekt präsentiert, getestet und begutachtet, der einen echten Nutzwert stiftet. Daraus lässt sich für die folgenden Schritte ableiten, ob und inwieweit Anpassungsbedarf besteht, um am Ende eine bessere Leistung zu liefern. Das Risiko, das „Falsche“ zu entwickeln, wird minimiert. Sie erinnern sich vielleicht an das Beispiel aus Punkt 1?

4. Fachleute aus verschiedenen Bereichen müssen während des Projekts täglich zusammenarbeiten.

   Agile Teams sind interdisziplinäre Teams. Alle erforderlichen Funktionen, die zur Erstellung eines Produkts oder einer Dienstleistung erforderlich sind, sind Teil des Projektteams. Damit die Abstimmung im Team funktioniert, treffen sich die Teammitglieder täglich zu einer kurzen Abstimmungsrunde. Auf diese Weise wird sichergestellt, dass jeder vom jedem weiß, was dieser gerade macht und wo es eventuell Überschneidungen, Probleme oder Unterstützungsbedarf gibt. Das Team „synchronisiert“ sich selbst. In aller Regel reicht dafür eine Timebox von ca. 15 Minuten. Gerade bei den Problemstellungen, denen sich die öffentliche Verwaltung zu stellen hat, ist Interdisziplinarität extrem wichtig geworden. Statt Fachbereiche wie bisher in Silos abgekapselt Teilbereiche abarbeiten zu lassen, geht es darum, gemeinsam das große Ganze im Auge zu behalten. Mögliche Probleme werden in einer frühen Phase erkannt und können bearbeitet werden.

5. **Errichte Projekte rund um motivierte Individuen. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen und vertraue darauf, dass sie die Aufgabe erledigen.**

   Agile Methoden gehen davon aus, dass jeder von sich aus motiviert und deshalb in der Lage ist, nach bestem Wissen seine Fähigkeiten in das Team einzubringen. Alles, was es hierfür braucht, ist ein Umfeld, das den Rahmen bietet. Ein solches Team muss nicht an die Hand genommen werden. Es braucht nur eine klare „Produktvision“, einen klar umrissenen Auftrag. Das Team spricht von sich aus Probleme an und

wird entsprechend an die Verwaltungsführung (Management) herantreten, das die Rahmenbedingungen schafft, die das Team zur Erfüllung seines Auftrags braucht. Im Gegenzug kann die Verwaltungsführung darauf vertrauen, dass das Team die Aufgaben meistert. Das Management wird entlastet und kann sich auf das konzentrieren, was seine eigentliche Aufgabe ist. An dieser Stelle noch der Verweis, dass die Verwaltung eben jene vertrauensvolle Zusammenarbeit im Bereich der Bürgerbeteiligung anstrebt. Werden entsprechende Werte nach innen gelebt, werden sie auch in der Außenwirkung selbstverständlich.

6. **Die effizienteste und effektivste Methode, Informationen an ein und innerhalb eines Teams zu übermitteln, ist das Gespräch von Angesicht zu Angesicht.**

   Auch wenn es banal klingen mag: Die Kommunikation von Angesicht zu Angesicht ist nach wie vor der effektivste und effizienteste Weg. Agile Methoden legen daher einen besonders hohen Wert auf Kommunikation. Statt über Aktenvermerke im Umlauf und E-Mails langwierig zu kommunizieren, treffen sich agile Teams täglich. 15 Minuten reichen aus. Was ist seit gestern passiert, was ist für heute geplant und wo gibt es Herausforderungen sowie Bedarf für vertiefenden Informationsaustausch? So ist jeder im Team im Bilde. Gemeinsam wird am Ende der Woche geplant und eine in einer gemeinsamen Rückschau überlegt, was künftig verbessert werden kann. Informationen fließen schneller, das gemeinsame Verständnis wird verbessert.Durch die enge Kommunikation mit den Betroffenen – also auch mit dem Auftraggeber, Kunden, Beteiligten aus anderen Abteilungen – und durch regelmäßige Rückblicke auf einen bestimmten Zeitraum erschließen sich Informationsquellen, die Rückschlüsse über Verbesserungspotenziale, Anpassungsbedarfe und mögliche Herausforderungen liefern. Aber auch das gemeinsame Verständnis aller Beteiligten verbessert sich, weil alle Beteiligten die Hintergründe besser verstehen. Das Verhältnis Bürger zu Verwaltung oder Verwaltung zu Gemeinderat verbessert sich eklatant. Vergessen wir nicht, dass Verwaltung oft mit komplexer Rechtsmaterie befasst ist, die für Laien schwer verständlich ist.

7. **Funktionierende Dienstleistungen/Produkte sind das wichtigste Fortschrittsmaß.**

   Hier geht es darum, möglichst einen Nutzen für den Auftraggeber bzw. für den Kunden zu schaffen. Dieser ist das Maß aller Dinge. Nicht die überkorrekt ausgefüllte Dokumentation, das exakte Reporting, der sauber dokumentierte Vorgang ist das Maß guter Projektarbeit, sondern einzig und allein das Ergebnis. Was haben wir von dem erreicht, was wir in diesem Planungszeitraum erreichen wollten? Wie oft wird gemessen, wie lange wer wofür braucht und wie oft etwas von a nach b geschafft wird – ohne jedoch drauf zu achten, was am Ende dabei herauskommt? Das Dokumentieren von Arbeitsschritten erscheint oft wichtiger als das erzielte Ergebnis. Diesem Ungleichgewicht stellt sich das 7. Prinzip entgegen. Dabei darf natürlich das Prinzip der Rechtmäßigkeit des Verwaltungshandelns nicht leiden. Aber die Erfüllung von Verwaltungsvorschriften als alleiniges Fortschrittsmaß allein wird dem öffentlichen Auftrag, den Verwaltung hat, nicht gerecht. Sie ist Dienstleisterin des Bürgers, autorisiert und finanziert vom Bürger.

8. **Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, das Team und die Nutzer der Dienstleistung sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.**

   Agilität vermeidet falsches Heldentum, im Sinne von „Verausgaben“ (Überstunden, 12-Stunden-Arbeitstage oder gar nachts durcharbeiten sind tabu). Sämtliche agile Methoden und Ansätze gehen davon aus, dass ein kontinuierlicher, gleichbleibender Rhythmus gesünder und produktiver ist und bessere Ergebnisse zeitigt als „Hauruckverfahren“, die dazu führen, dass die Beteiligten ausbrennen. Der regelmäßige Rhythmus gibt darüber hinaus die Möglichkeit, ein gewisses Maß an Verlässlichkeit zu schaffen und unnötigen Arbeitsdruck zu vermeiden. In diesem Sinne werden Arbeitsspitzen als

Warnsignale verstanden, die darauf hindeuten, dass etwas nicht funktioniert. Einer der Vorteile gleichbleibender Arbeitsrhythmen ist die Erhöhung des Durchflusses: die Bearbeitungszeiten verkürzen sich. Ein echter Mehrwert für den Bürger, aber auch für den einzelnen Mitarbeiter, der Erfolg in Form abgeschlossener Vorgänge erleben darf, statt wachsender Aktenberge.

9. **Ständiges Augenmerk auf fachliche Exzellenz und gute Gestaltung der Arbeitsabläufe fördert Agilität.**

   Damit das alles funktionieren kann, muss jedes Teammitglied über einen hohen Grad an sozialem und organisatorischem Können verfügen. Nur so erkennt das Team Handlungsmöglichkeiten sowie Herausforderungen und kann adäquate Lösungen erarbeiten. Die kontinuierliche Weiterentwicklung der Teamfähigkeiten, der internen Arbeitsabläufe und die permanente Weiterentwicklung der „Werkzeugkiste“ ist Voraussetzung für Agilität, da nur so auf Veränderungen reagiert werden kann. Fachliche Exzellenz ist für das Selbstverständnis einer öffentlichen Verwaltung Teil des Anspruchs. In der Realität zeigt sich, dass diese jedoch allzu oft durch die Fokussierung auf die Einhaltung von Formvorschriften zu wenig Beachtung erfährt. Wie oft bekommen junge, enthusiastische Mitarbeiter von den älteren Kollegen zu hören: „Haben wir schon immer so gemacht und es hat funktioniert. Es gibt also keinen Grund etwas zu ändern.“ Und dies, obwohl alle spüren, dass Veränderungen längst überfällig sind und sie Ballast, der irgendwann seine Daseinsberechtigung hatte, am effizienten und effektiven Arbeiten hindert.

10. **Einfachheit – die Kunst, die Menge nicht getaner Arbeit zu maximieren – ist essenziell.**

    Es mag im ersten Moment kurios klingen, dass die Kunst darin besteht, die Menge nicht getaner Arbeit zu maximieren. Aber beim genaueren Überlegen wird klar, was damit gemeint ist: Es geht darum, Nutzen zu stiften und unnötige Arbeiten zu vermeiden, die keinen Nutzwert schaffen. Statt vieler unnötiger Zwischenschritte sollen Dinge möglichst einfach gemacht und „verschlankt“ werden, wenn es möglich ist. Sprich: Alles „über Bord“ werfen, was nicht zum gewünschten Ergebnis beiträgt. D. h. auch zu hinterfragen, ob das, was Verwaltung tut, tatsächlich zum Ergebnis beiträgt oder nur aus Tradition gemacht wird. In eine ähnliche Richtung geht auch das 12. Prinzip, auf das wir noch eingehen werden.

11. **Die besten Arbeitsrahmen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.**

    Agile Teams sind selbstorganisiert. Da sich das Team täglich mit dem gemeinsamen Thema auseinandersetzt, hat es auch die nötige Expertise, um Anforderungen, Entwürfe und Arbeitsrahmen so zu gestalten, dass am Ende ein für alle befriedigendes Ergebnis herauskommt. Das Team fixiert sich nicht auf Stellenbeschreibungen, sondern definiert sich über Rollen. Nehmen wir Scrum, eine der bekanntesten Methoden der agilen Produktentwicklung. Dort wird zwischen drei Rollen unterschieden. Mehr nicht. Es gibt einen sogenannten Scrum Master, der das Team begleitet, unterstützt und berät. Es gibt den Produkteigentümer, der innerhalb des Teams die Interessen des Auftraggebers vertritt und den Kontakt zum Auftraggeber sicherstellt, und es gibt das Entwicklerteam.Das Entwicklerteam setzt sich aus allen Funktionen zusammen, die zu Erfüllung der Aufgaben erforderlich sind. Alle verfügen im Idealfall über eine breite Grundkenntnis und jeweils Spezialkenntnisse, sodass jeder im Team andere Teammitglieder unterstützen kann. Gemeinsam mit dem ScrumMaster und dem Produkteigentümer bildet das Entwicklerteam ein Scrum-Team. Ein Beispiel: Bei einem Projekt zur Einführung der eAkte übernimmt der Sachgebietsleiter des Fachbereichs zentrale Dienste die Rolle des Produkteigentümers. Der Produkteigentümer ist nicht disziplinarischer Vorgesetzter des Teams. Dem Team gehören Vertreter der verschiedenen Fachbereiche an, die als Kommunikatoren in ihre Fachbereiche wirken sowie ein Vertreter

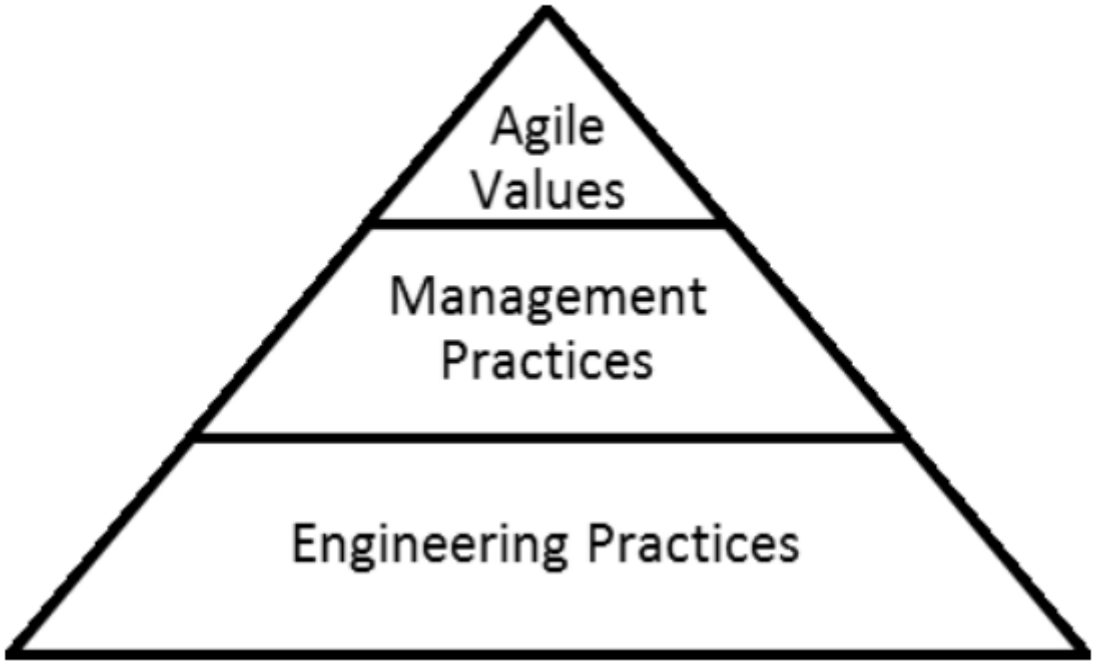
des Personalrates. Die Rolle des Scrum Masters wird von einem erfahrenen Kollegen der EDV-Abteilung ausgefüllt, der bereits in scrumgeführten Projekten gearbeitet wird. Da die Stadt vier Fachebereiche hat, gehören damit sieben Personen dem Team an. Der Bürgermeister und die Fachbereichsleiter stehen in der Rolle des Managements. Die jeweiligen Kollegen, die später mit dem System der eAkte arbeiten, sind als Anwender die wichtigsten Feedbackgeber für das Team.

12. **In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann und es passt sein Verhalten entsprechend an.**

    Die kontinuierliche Verbesserung des Teams und seiner Zusammenarbeit ist von zentraler Bedeutung. Ein neues Team funktioniert nicht von Anfang an perfekt. Es muss sich finden. Es muss sich mit den sich verändernden Rahmenbedingungen entwickeln, wenn es sich einmal gefunden hat. Im Lean Management wurde der Begriff Kaizen, der ständigen Verbesserung, geprägt. Die Idee des Kaizen spiegelt sich im agilen Manifest wieder: Es gibt keine perfekte, sondern immer nur die im Augenblick beste Lösung. Zusammengefasst: Verbesserung heißt hier nicht, einfach nur Prozesse und Strukturen fortzuentwickeln, sondern im ganzheitlichen Sinne voranzuschreiten. Die Zusammenarbeit zwischen allen Beteiligten, die soziale Ebene miteinzubeziehen und als eine der wichtigsten Stellschrauben zu sehen.

Source: https://www.borisgloger.com/blog/2018/08/23/agiles-manifest-12-prinzipien-fuer-eine-agile-verwaltung/

Zeichnen und erklären Sie die agile Kompetenzpyramide



Die Agile Kompetenzpyramide unterteilt die notwendigen Kompetenzen in drei Kategorien:

- Engineering Practices
- Management Practices
- Agile Values

Das Meistern der "Engineering Practices" bildet die Grundlage für die Entwicklung von guter Software. Unter "Engineering Practices" fallen beispielsweise "Best Practices" wie "unit testing, "clean coding", "test-driven develoopment", "continous integration" usw.

Die zweite Kategorie bilden die agilen Managament Praktiken (agile management practices). Diese definieren, wie ein agiles Projekt organisiert ist und ausgeführt wird. Darunter fallen Praktiken wie iterative Planungen, kurze Releasecyclen, kleine Release, starke Einbeziehung des Kunden, etc. Hierzu sind vor allem die richtigen Sozialkompetenzen wichtig.

An der Spitze der Pyramide befinden sich die agilen Werte (agile values) wie Offenheit, Respekt, Einfachheit, Kommunikation, etc. Die Anzahl der notwendigen Fähigkeiten nimmt zur Spitze der Pyramide ab. Trotzdem darf man nicht vergessen, dass das erlernen der Fähigkeiten mit jeder Schicht zunimmt. Das erlernen von agilen Werten benötigt meist ein starkes Umdenken der Teammitglieder und ist deshalb am schwierigsten zu erlernen.

Die Pyramide visualisiert ausserdem die Reihenfolge in der die Fertigkeiten erlent werden sollen. Eine gute technische Basis ist eine essentielle Voraussetzung um gute Software zu entwickeln. Entwicklern agile Management Praktiken zu lehren, bevor ein solides technisches Fundament existiert, ist genauso sinnvoll, als würde man versuchen eine Pyramide von oben noch unten zu bauen. Leider werden sehr häufig Entwickler von Firmen auf 2-tägige Scrum Kurse geschickt bevor diese überhaupt ein gutes technische Basis haben.

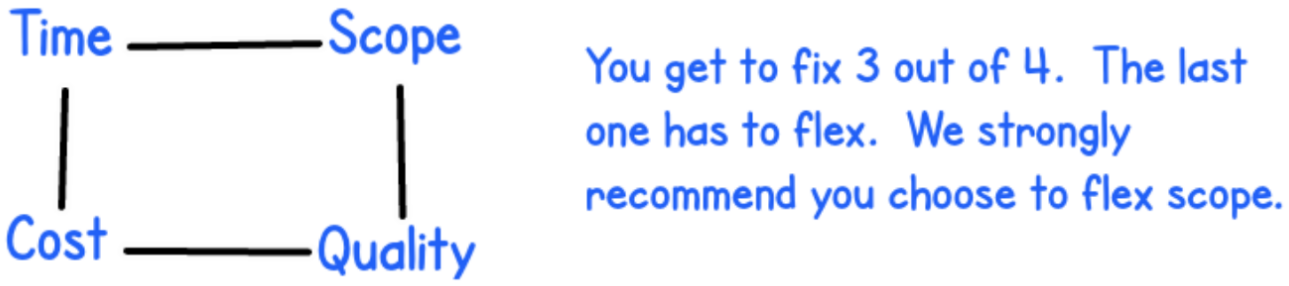
Ergänzung hierzu: "You cant be agile if your code sucks" / https://www.slideshare.net/PeterGfader/you-cantbeagileifyourcodesucksv06

Technische Fertigkeiten können leicht durch Kurse gelehrt werden. Agile Management Fertigkeiten lernt man besser mit der Durchführung von Projekten.

Source: https://pdfs.semanticscholar.org/e6d6/2ee60a8b41b154556463f8b2d30b3af834f2.pdf

---

Erklären Sie den Begriff "4 variables in Agile development"



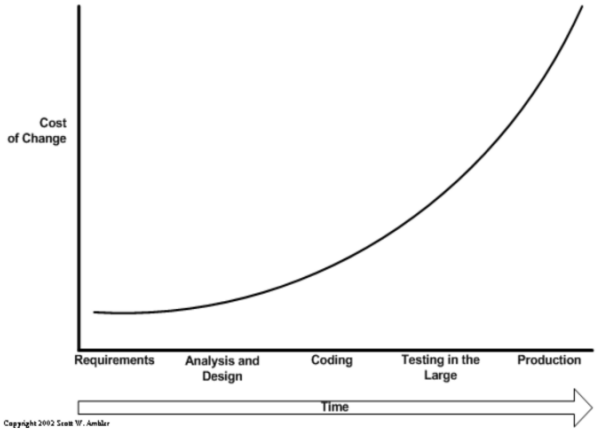
- Zeit (Time)
- Ressourcen (Resources)
- Qualität (Quality)
- Umfang (Scope)

Das Management kann beispielsweise 3 von 4 vorgeben. Das Entwicklungsteam benutzt die 4 Variable als Kontrollvariable. Wenn das Management als eine Deadline (Time), Budget (Ressources) und die Qualität festsetzt kann das Entwicklungsteam mittels dem Umfang der Features gegensteuern indem weniger Features realisiert werden.¨

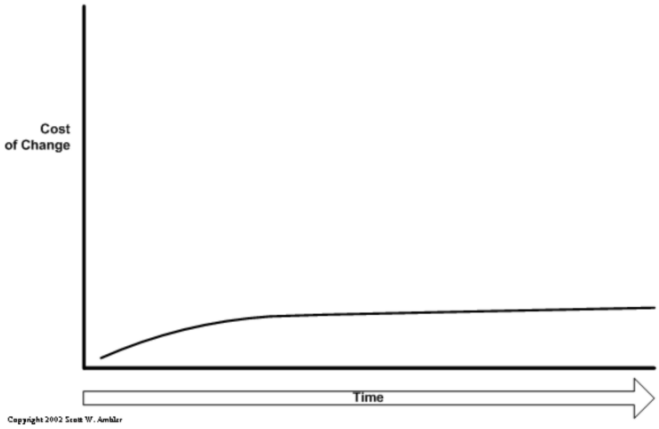
Link: https://medium.com/@jchyip/four-variables-cost-time-quality-scope-f29b4de8bfdf

---

Erklären Sie den Begriff "Cost of change"



Cost of change mit Wasserfall Methode

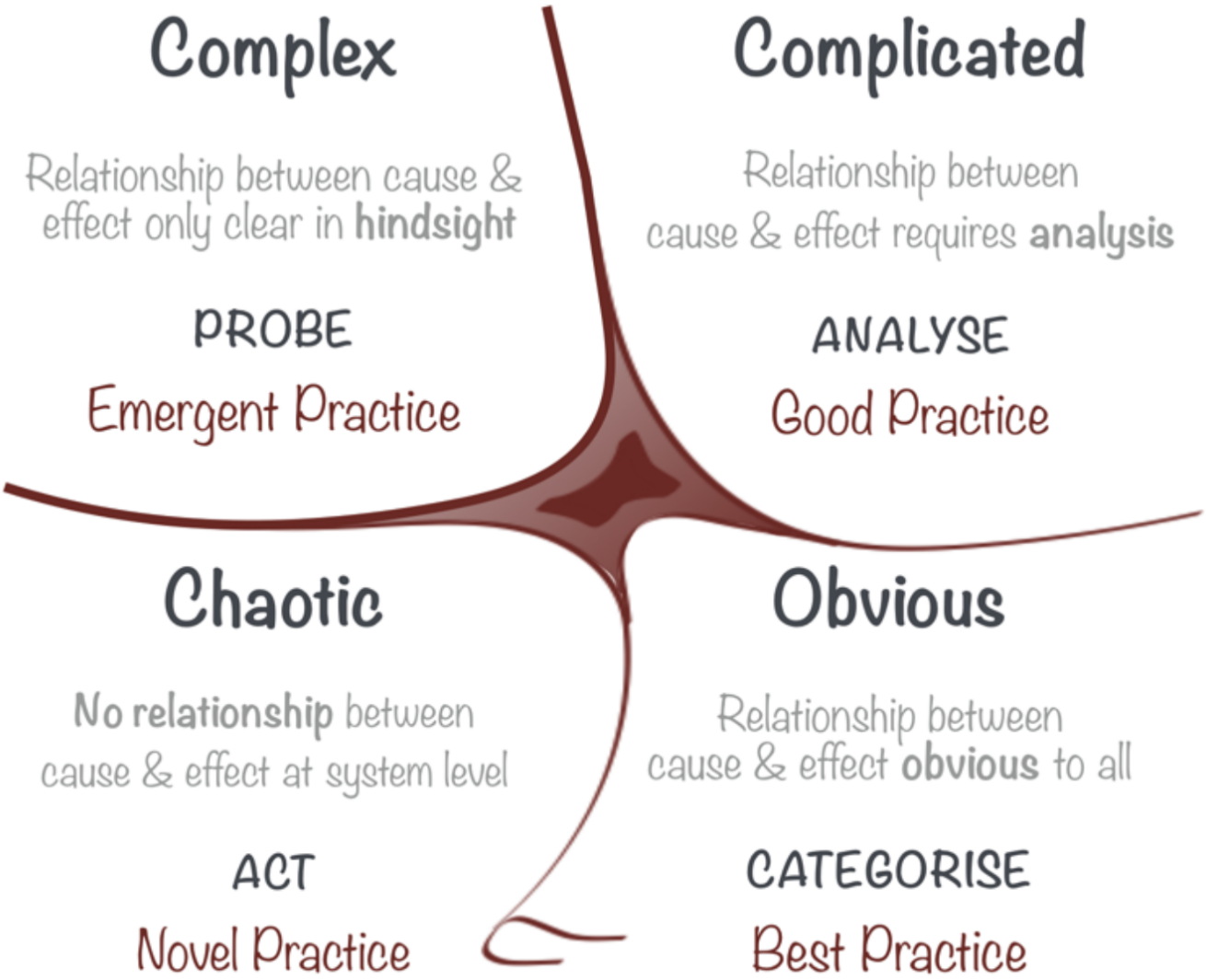


Cost of change mit agiler Methode

Die Kosten für Änderungen unterschieden sich zwischen den traditionellen Projektmethoden und agilen Projektmethiden erheblich. In der traditionellen Welt müssen bei jeder Änderung die entsprechenden Phasen durchlaufen werden, welche einen erheblichen finanziellen und meist zeitlichen Mehraufwand bedeuten.

Bei agilen Projektmethoden ist man weitaus flexibler (weniger Overhead, weniger starre Prozesse, etc.) was sich weit besser auf die Kosten auswirkt.

Beschreibung Sie das Cynefin Framework sowie den Nutzen davon im Kontext der Softwareentwicklung



Erklären Sie den Begriff "4 + 1 Domains (Simple, Complicated, Complex, Chaotic, Disorder)"

- **Simple / Obvious:**

  Die Beziehung zwischen Ursache und Wirkung ist für alle offensichtlich.

  Beispiel: Aktion A führt in 99.9% aller Fälle zu Resultat B

- **Complicated:**

  Die Beziehung zwischen Ursache und Wirkung ist nicht offensichtlich und bedarf einer Anlayse/Prüfung von Experten/Fachpersonen.

  Beispiel: Aktion A führt zu Resultat B aber es nicht offensichtlich warum Herangehensweise: Sense-Analyze-Respond Entscheidung: Good Practices (Unterschied zu Best-Practices!)

- **Complex:**

  Die Beziehung zwischen Ursache und Wirkung kann nur im Nachhinein wahrgenommen werden aber nicht im Voraus. Hier könnte man den Eindruck bekommen, dass es sich um ein nicht-deterministisches System handelt bei der die gleiche Aktion oft unterschiedliche Resultate hervorbringt. Hier handelt es sich

üblicherweise um ein System von vielen Variablen die zusammenspielen und sich gegenseitig beieinflussen.

Beispiele: Unternehmenskultur etablieren Herangehensweise: Probe-Sense-Respond Entscheidung: Stakeholder entscheiden was zu tun ist

- **Chaotic:**

  Es gibt keine wahrnehmbare Beziehung zwischen Ursache und Wirkung. Hier ist es wichtig einfach Aktionen durchzuführen (egal welche) weil es scheinbar keinen Zusammenhang gibt.

  Herangehensweise: Act - Sense - Respond Entscheidung: einfach Aktion ausführen

- Disorder:

  Disorder ist der Zustand des Nicht-Wissens, welche Art von Kausalität besteht (also chaotic,complex,simple,complicated). In diesem Zustand gehen die Leute in ihre eigene Komfortzone zurück, wenn sie eine Entscheidung fällen.

Erklären Sie den Begriff "Exaptation"

?????

Was ist "extreme Programming"?

Extreme Programming ist ein agiles Software Development Framework ähnlich wie Scrum. Agile Prozesse, kurze Entwicklungszyklen und schnelle Reaktionszeiten auf neue oder sich ändernde Anforderungen stehen dabei im Vordergrund.

Dabei setzt das Framework auf drei Hauptbestandteile:

- Werte (Values)
- Prinzipien (Principles)
- Techniken (Practices)

Zählen Sie die Core Principles, Values und Practices von extreme Programming auf und erklären Sie die Bedeutung davon

Values:

- Kommunikation

  Das Team kommuniziert stetig, um Informationen auszutauschen. Es gibt einen stetigen Austausch aller Beteiligten, also auch zwischen dem Entwicklungsteam und dem Kunden.

- Einfachheit

  Entwickler bemühen sich einfachen verständlichen Code zu schreiben um Zeit (Analyse von Code) und Aufwand (Wartung, Anpassung) zu reduzieren

- Feedback
- Respekt
- Mut Die Entwickler sollen mutig sein und die Kommunikation offen gestalten. Falls eine Anforderung nicht in einer Iteration umgesetzt werden kann, wird in offener und ehrlicher Art und Weise direkt darauf hingewiesen

Principles:

- Wirtschaftlichkeit
- Qualität
- Reflexion
- Verbesserungen
- Menschlichkeit
- Kleine Schritte
- Gelegenheiten wahrnehmen
- usw.

Practices:

- The Planning Game

  Neue Versionen der Software werden in einem Planning-Game, auch als Planning-Poker bekannt, spezifiziert und der Aufwand zu deren Umsetzung abgeschätzt.

- Small releases

  Releases sollen regelmässig und mit kleinem Umfang ausgeliefert werden um damit verbundene Risiken so klein wie möglich zu halten

- Simple design

  Es soll die einfachste Lösung angestrebt werden, also diejenige, die genau das Gewünschte erreicht (und nicht mehr)

- Unit testing

  Testing ist immer gut

- Refactoring

  Laufendes Refactoring, ständige Architektur-, Design- und Code-Verbesserungen, auch um Anti-Patterns umgehend erkennen und beseitigen zu können.

- Pair programming

  Beim Pair programming teilen sich zwei Programmierer einen Computer – einer codiert (der Driver) und der andere denkt mit und hat das Gesamtbild im Kopf (der Partner). Die Rollen werden regelmäßig getauscht. Anfänger sollen schneller von der Arbeit eines Spezialisten lernen. Das Wissen wird verteilt. Durch ständigen Codereview der Entwicklung und Kommunikation wird das Design verbessert und Fehler schneller gefunden.

- Collective code ownership

  Aktivitäten werden zunächst nicht an einzelne Personen verteilt, sondern an das ganze Team. Es existiert laut Methodik das Bewusstsein und die Verpflichtung nur als Team erfolgreich sein zu können. Einzelne Teammitglieder besitzen kein Wissensmonopol. Pair-Programming und wechselhafte Einsatzgebiete sollen der Strömung entgegenwirken, dass einzelne Personen Teile als ihren Besitz betrachten.

- Continuous integration

  Continous Integration der einzelnen Komponenten zu einem lauffähigen Gesamtsystem in kurzen Zeitabständen. Je häufiger integriert wird, desto höher wird laut XP die eintretende Routine. Fehler werden damit früh aufgedeckt. Vermeidet auch die berühmte "Integration Hell"

- 40 hours week

  40-Stunden-Woche, d. h. Überstunden sind zu vermeiden, weil darunter die Freude an der Arbeit, mittelfristig die Konzentrationsfähigkeit der Programmierer und somit auch die Qualität des Produktes leidet. Nachweislich sinkt die Produktivität eines Entwicklers durch Überstunden. Arbeit außerhalb der regulären Arbeitszeit wird im Einzelfall zwar geduldet, aber auf keinen Fall besonders entlohnt oder erwartet. Überstunden zeugen gewöhnlich einfach nur von falscher Planung.

- Test-Driven Programming (TDD)

  Beim Test-Driven Development werden erst die Modultests (Unit-Test) geschrieben, bevor die eigentliche Funktionalität programmiert wird. Der Entwickler befasst sich dadurch früh mit dem Design des Codes und überdenkt seine Programmierarbeit genau.

- On-site customer
- Metaphor
- Coding standards
- Slack
- Spike
- Incremental Design
- Self-organized team

Zählen Sie die Zeremonien von Scrum auf und erklären Sie jeweiligen Zweck bzw. Nutzen

- **Daily Standup (Daily Scrum)**

  Das Daily Scrum ist ein tägliches Meeting 15 Minuten für das ganze Entwicklungsteam. Das Entwicklungsteam plant dabei die Arbeit für die nächsten 24 Stunden und überprüft die Arbeitsergebnisse seit dem letzten Daily Scrum. Um die Komplexität zu reduzieren, wird das Daily Scrum an jedem Tag zur gleichen Uhrzeit am gleichen Ort abgehalten.

Fragen:

Was habe ich gestern getan?

Was werde ich heute erledigen?

Sehe ich irgendein Hindernis (Impediments)?

Dauer: maximal 15 Minuten

- **Sprint Retrospektive**

  Die Sprint Retrospektive bietet dem Scrum-Team die Gelegenheit, sich selbst zu überprüfen und einen Verbesserungsplan für den kommenden Sprint zu erstellen. Sie findet zwischen dem Sprint Review und dem nächsten Sprint Planning statt.

  Dauer: maximal 3 Stunden

- **Sprint Review**

  Am Ende eines Sprints wird ein Sprint Review abgehalten, um das Produktinkrement zu überprüfen und das Product Backlog bei Bedarf anzupassen. Während des Sprint Reviews beschäftigen sich das Scrum-Team und die Stakeholder gemeinsam mit den Ergebnissen des Sprints.

  Der Product Owner stellt den aktuellen Stand des Product Backlogs dar. Er gibt bei Bedarf eine aktualisierte Vorhersage von wahrscheinlichen Ziel- und Lieferterminen auf der Basis des Entwicklungsfortschritts. Das Entwicklungsteam führt die fertige Arbeit vor, und beantwortet Fragen zu dem Inkrement.Alle Teilnehmer erarbeiten gemeinsam, was als nächstes zu tun ist, so dass das Sprint Review wertvollen Input für die kommenden Sprint Plannings liefert

  Dauer: maximal 4 Stunden

- **Sprint Planning**

  Im Sprint Planning wird die Arbeit für den kommenden Sprint geplant. Dieser Plan entsteht durch die gemeinschaftliche Arbeit des gesamten Scrum-Teams.

  Das Sprint Planning beantwortet die folgenden Fragen:

  - Was ist in dem Produktinkrement des kommenden Sprints enthalten?
  - Wie wird die für die Lieferung des Produktinkrements erforderliche Arbeit erledigt?

  Dauer: maximal 8 Stunden

Welche Rollen gibt es in Scrum?

- Product Owner

  Der Product Owner ist dafür verantwortlich, den Wert des Produktes zu maximieren, das aus der Arbeit des Entwicklungsteams entsteht. Wie dies geschieht, kann je nach Organisation, ScrumTeam und

Einzelpersonen stark variieren. Der Product Owner ist die einzige Person, die für das Management des Product Backlogs verantwortlich ist.

Aufgaben:

- Den Wert der Arbeit optimieren, den das Entwicklungsteam erledigt
- Product Backlog priorisieren und Items darin klar zu formulieren
- Sicherstellen, dass Entwicklungsteam die Einträge im Product Backlog versteht

- Entwicklungsteam

  Das Entwicklungsteam besteht aus Profis, die am Ende eines jeden Sprints ein fertiges Inkrement übergeben, welches potenziell auslieferbar ist. Im Sprint Review muss ein fertiges Inkrement vorhanden sein. Nur Mitglieder der Entwicklungsteams erstellen das Produktinkrement. Entwicklungsteams sind von der Organisation so strukturiert und befähigt, dass sie ihre eigene Arbeit selbst organisieren und managen. Scrum kennt für Mitglieder des Entwicklungsteams keine Titel. Dies ist unabhängig von der Arbeit, die diese Personen erledigen.

- Scrum Master

  Der Scrum Master ist dafür verantwortlich, Scrum entsprechend des Scrum Guides zu fördern und zu unterstützen. Scrum Master tun dies, indem sie allen Beteiligten helfen, die ScrumTheorie, Praktiken, Regeln und Werte zu verstehen. Der Scrum Master ist ein „Servant Leader“ für das Scrum-Team. Der Scrum Master hilft denjenigen, die kein Teil des Scrum-Teams sind, zu verstehen, welche ihrer Interaktionen mit dem Team sich hilfreich auswirken und welche nicht. Der Scrum Master hilft dabei, die Zusammenarbeit so zu optimieren, dass der durch das Scrum-Team generierte Wert maximiert wird.

Zählen Sie alle Scrum Artifakte auf und beschreiben Sie diese

---

Was ist der Zweck bzw. Nutzen eines Burn-Down Charts in Scrum?

Was versteht man unter einem Sprint Goal? Welchen Sinn hat es?

Was versteht man unter einem Product Backlog in Scrum?

Wer ist für die Priorisierung der Inhalte eines Product Backlogs (User Stories) in Scrum verantwortlich?

Was ist eine User Story?

Was versteht man unter User Roles in Scrum?

Was ist ein Epic in Scrum?

Was sind Themes in Scrum?

Erklären Sie die Bedeutung von Story Points in Scrum?

v

Was versteht man unter Velocity? Wie hängt die Velocity mit Story Points zusammen?

Was ist Planning Poker? Wie läuft Planning Poker üblicherweise ab?

Was versteht man unter "Conditions of Satisfaction"?

Erklären Sie was man unter "Levels of Planning" versteht

Erläutern Sie den Begriff "Techniques of Estimation"

Erläutern Sie die Abkürzung "INVEST". Wofür stehen die einzelnen Buchstaben?

Erklären Sie worum was bei "Planning for Value" geht

Erklären Sie was mit dem Begriff "Chicken and Pigs" gemeint ist

---

Was ist ein Task Board in Scrum?

Was ist eine Definition of Done? Geben Sie ein Beispiel hierfür an

---

Was ist ein Daily Scrum? Wie lange sollte laut Agile Manifesto ein Daily Scrum maximal dauern?

Was versteht man unter einem Increment in Scrum?

Was versteht man unter Continous Integration?

Bei der modernen Anwendungsentwicklung arbeiten mehrere Entwickler an unterschiedlichen Features der gleichen App. Die gleichzeitige Zusammenführung aller Quellcode-Branches an einem Tag (auch bekannt als „Merge Day“) kann einen hohen Arbeits- und Zeitaufwand bedeuten. Der Grund dafür ist, dass Anwendungsänderungen von getrennt arbeitenden Entwicklern miteinander in Konflikt treten können, wenn sie zeitgleich durchgeführt werden.

Mithilfe der Continuous Integration (CI) können Entwickler ihre Codeänderungen in einem gemeinsamen „Branch“ oder „Trunk“ der Anwendung viel häufiger zusammenführen, manchmal sogar täglich. Sobald die Änderungen eines Entwicklers zusammengeführt werden, werden sie in automatischen App-Builds und unterschiedlichen Stufen von Automatisierungsprüfungen (normalerweise Einheits- und Integrationstests) validiert. So wird sichergestellt, dass die Funktionsfähigkeit nicht beeinträchtigt wurde. Dabei müssen alle Klassen und Funktionen bis hin zu den verschiedenen Modulen der App getestet werden. Wenn die automatische Prüfung Konflikte zwischen aktuellem und neuem Code erkennt, lassen sich diese mithilfe von CI schneller und häufiger beheben.

Was versteht man unter Continous Delivery?

Nach der Automatisierung von Builds, Unit und Integrationstests mittels Continous Integration wird bei der Continuous Delivery auch die Freigabe des validierten Codes an ein Repository automatisch durchgeführt. Um also einen effizienten Continuous Delivery-Prozess zu gewährleisten, muss die CI bereits in Ihre Entwicklungs-Pipeline integriert sein. Ziel der Continuous Delivery ist eine Codebasis, die jederzeit für die Implementierung in einer Produktionsumgebung bereit ist.

Bei der Continuous Delivery umfasst jede Phase – von der Zusammenführung der Codeänderungen bis zur Bereitstellung produktionsreifer Builds – automatisierte Tests und Code-Freigaben. Am Ende dieses Prozesses kann das Ops-Team eine App schnell und einfach in der Produktionsphase implementieren.

Was versteht man unter Continous Deployment?

Die abschließende Phase der CI/CD-Pipeline ist das Continuous Deployment. Als Erweiterung der Continuous Delivery, bei der produktionsbereite Builds automatisch an ein Code-Repository freigegeben werden, wird beim Continuous Deployment auch die Freigabe einer App in die Produktionsphase automatisiert. Da der Produktionphase in der Pipeline kein manuelles Gate vorgeschaltet ist, müssen beim Continuous Deployment die automatisierten Tests immer sehr gut durchdacht sein.

In der Praxis bedeutet Continuous Deployment, dass App-Änderungen eines Entwicklers binnen weniger Minuten nach ihrer Erstellung live gehen können (vorausgesetzt, sie bestehen den automatischen Test). Dies erleichtert eine kontinuierliche Integration von Benutzer-Feedback ungemein. All diese zusammenhängenden CI/CD-Praktiken machen eine Anwendungsimplementierung weniger riskant, weil Änderungen in Teilen und nicht auf einmal freigegeben werden. Die Vorabinvestitionen sind allerdings beträchtlich, da automatische Tests für die diversen Prüf- und Release-Phasen in der CI/CD-Pipeline geschrieben werden müssen.

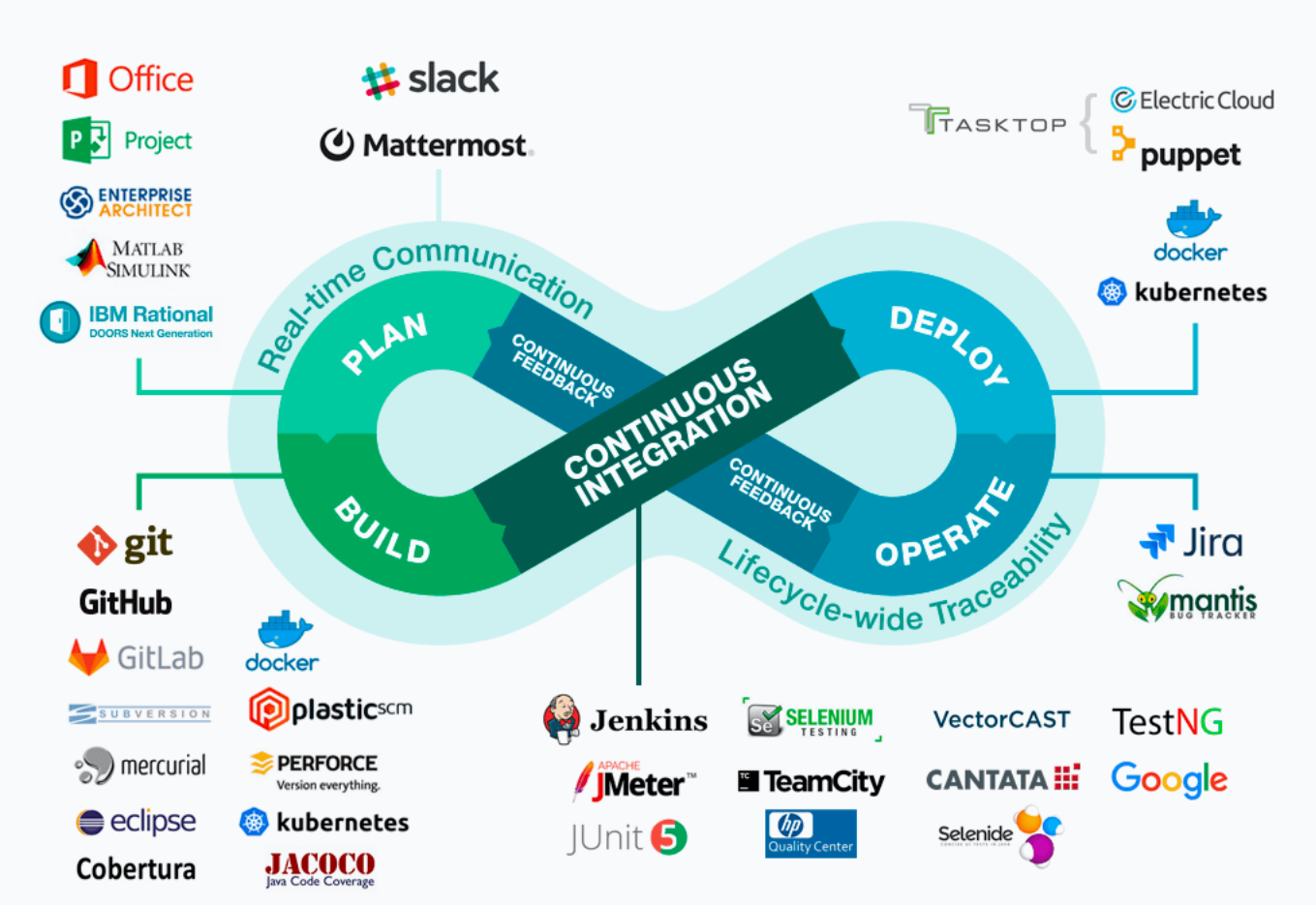
Worin unterscheidet sich Continous Deployment von Continous Delivery?

Bei Continous Delivery wird der Build Prozess so weit automatisiert, dass eine App jederzeit auf Produktion ausgerollt werden kann falls notwendig. Dabei wird die Software in einem Repository abgelegt (z.B. Docker Images in Docker Registry oder Maven Artifakte in Maven-Repository, etc.)

Bei Continous Deployment geht man noch einen Schritt weiter und deployed die neue Version der App nach Validierung der Tests sofort in der Produktionsumgebung.

Was versteht man unter DevOps?

DevOps ist ein Kunstwort aus den Begriffen Development (englisch für Entwicklung) und IT Operations (englisch für IT-Betrieb). DevOps soll durch gemeinsame Anreize, Prozesse und Software-Werkzeuge (englisch: tools) eine effektivere und effizientere Zusammenarbeit der Bereiche Dev, Ops und Qualitätssicherung (QS) ermöglichen.[1] Mit DevOps sollen die Qualität der Software, die Geschwindigkeit der Entwicklung und der Auslieferung sowie das Miteinander der beteiligten Teams verbessert werden.



Was sind die Aufgaben eines Software Architekten?

Erklären Sie was man unter einer Software Architektur versteht

---

Zählen Sie mind. 4 Architektur Stile auf und erklären Sie die Vor- und Nachteile

---

Geben Sie den Inhalt des Artikels "Evolutionary architecture and emergent design: Investigating architecture and design, by Neal Ford" in groben Zügen wieder

---

Was versteht man unter dem Begriff "ORM"?

Was ist ein "O/R mismatch"?

Erklären Sie was man unter einem Domain Model versteht

Was versteht man unter einem Pojo in der Softwareentwicklung?

Was ist "Location Transparency" in Bezug auf ORM/JPA?

Erklären Sie was man unter Entity Mapping in ORM/JPA versteht

---

Erklären Sie den Typ "Request/Response" in einem verteilten System

---

Erklären Sie was man unter "Message Passing" in einem verteilten System versteht

---

Was sind die Unterschiede zwischen den Typen "Request/Response" und "Message Passing" in einem verteilten System?

Nennen Sie drei Architekturstile und beschreiben sowie vergleichen Sie dieser untereinander

- Pipes & Filters
- Schichten
- Interpreter

Beschreiben Sie den "Web Service Technology Stack"

Beschreiben Sie die Operationen Publish, Find und Bind des Web Service Technology Stacks

---

Welche Standards werden in den Ebenen Transport, Message, Description und Discovery des Web Service Technology Stacks eingesetzt?

Wofür steht die Abkürzung REST?

Was versteht man unter einem RESTful, Webservice?

Was ist SOAP und aus welchen Teilen setzt sich eine SOAP zusammen

Was ist das SOLID Principle? Erklären Sie alle Prinzipien und geben Sie jeweils ein Beispiel an

- **S** ingle Responsibility Principle
- **O** pen Closed Principle
- **L** iskov Substitution Principle
- **I** nterface Segregation Principle
- **D** ependency Inversion Principle

Was versteht man unter Dependency Injection?

Als Dependency Injection wird in der objektorientierten Programmierung ein Design Pattern bezeichnet, welches die Abhängigkeiten eines Objekts zur Laufzeit steuert. Benötigt ein Objekt beispielsweise bei seiner Initialisierung ein anderes Objekt, ist diese Abhängigkeit an einem zentralen Ort hinterlegt (z.B. Spring Container), es wird also nicht vom initialisierten Objekt selbst erzeugt. Mittels "Inversion of Control" wird die Kontrolle/Verantwortung der Instanzierung üblicherweise an ein Framework (z.B. Spring) übergeben.