Agile Software-Entwicklung

und klare DevOps-Philosophie

bei der

mip GmbH

Jörg Kremer, Head of Consulting

mip Management Informationspartner GmbH

Inhaltsverzeichnis

1 LEITFADEN 3

2 SOFTWARE-ENTWICKLUNG IM WANDEL 3

2.1 Eine Reise durch die Zeit der Software-Entwicklung 3

2.2 Exkurs: Entwicklung der Virtualisierung 5

3 WANDEL IM SOFTWARE-ENGINEERING 7

3.1 V-Modell 7

3.2 Agile Software-Entwicklung 9

4 DEVOPS 10

4.1 Automation 10

4.1.1 Testautomation 10

4.1.2 Automation der Software-Bereitstellung 11

4.2 Teamarbeit und Kontinuität 11

4.3 Infrastruktur als Code 12

4.4 Zusammenfassung 12

1 Leitfaden

DevOps (Kunstwort aus den Begriffen Development und IT Operations) zeigt sich als einer der wichtigsten Trends in der aktuellen Softwareentwicklung und im Betrieb komplexer Systeme. Um sich dem Thema zu nähern, ist es sinnvoll zu verstehen, warum und mit welchem Ziel die DevOps-Bewegung entstanden ist.

Die mip GmbH hat in ihrer über dreißigjährigen Geschäftstätigkeit einen großen Teil dessen, was nachfolgend thematisiert wird, miterlebt und mitgestaltet. Als mittelständiges IT-Beratungsunternehmen mussten wir immer sicherstellen, dass die für uns relevanten „Eilzüge“ der IT-Innovation nicht an uns vorbeifuhren. Wir wollten und werden weiterhin möglichst als Treiber und nicht als Getriebene agieren. Dies gilt selbstverständlich auch für moderne agile Softwareentwicklung und DevOps.

Dieses Whitepaper gibt Ihnen einen historischen Abriss darüber, wie und warum der IT-Markt heute mehrheitlich im Bereich der Softwareentwicklung und des Betreibens komplexer Systeme mit agilen Ansätzen und einer DevOps-Philosophie unterwegs ist.

Schnallen Sie sich an und wir starten gemeinsam einen Sprint durch die IT-Zeitgeschichte mit den markanten Eckpunkten, welche die heutige moderne IT manifestieren.

2 Software-Entwicklung im Wandel

Die Informationstechnologie lebt vom häufigen Wandel. Klassischerweise ist die Branche hoch dynamisch in ihren Methoden und ihrer Ausrichtung und erfindet sich in gewisser Weise alle paar Jahre neu. Das ist zum einen in der rasanten Entwicklung der Computersysteme hinsichtlich ihrer Leistungsfähigkeit begründet, zum anderen aber auch in den starken Veränderungen der Anforderungen der Nutzer beziehungsweise der Kunden von IT-Systemen.

Ein IT-System besteht grundsätzlich aus Hardware und Software. Innerhalb der Software unterscheidet man zwischen Standardsoftware und Individualsoftware. Zur Erstellung von individuellen Software-Lösungen kann man zwischen komplett freien Entwicklungsumgebungen und spezialisierten zweckgebundenen Lösungen trennen. Komplett freie Umgebungen ermöglichen es, mithilfe einer Programmiersprache (z. B. Java, C++, Python etc.) Programme für nahezu beliebige Zwecke zu erstellen. Für spezielle Vorhaben können stark spezialisierte Entwicklungsumgebungen eingesetzt werden, wie etwa eine ETL-Umgebung, um Datenbewirtschaftungsprozesse zu entwickeln. Die Freiheitsgrade bei der Implementierung sind typischerweise sehr gering.

2.1 Eine Reise durch die Zeit der Software-Entwicklung

In den Anfängen der elektronischen Datenverarbeitung gab es mehr oder weniger ausschließlich individuelle programmierte Software. Bestenfalls wurde diese für eine breitere Kundenbasis bereitgestellt, aber typischerweise mithilfe geeigneter Programmiersprachen individuell entwickelt. Dabei konkurrierten individuelle Lösungen um die Gunst der Kunden. Als Beispiel seien hier verschiedene Textverarbeitungsprogramme genannt, alle individuell entwickelt und unterschiedlich in ihrer Benutzeroberfläche, ihren Funktionen und im Preis. In den ersten Jahrzehnten der EDV war praktisch jede Software kommerziell.

In dieser Phase war auch die von Unternehmen genutzte Software oft eine Individual-Lösung oder eine Zusammensetzung verschiedener Produkte. Die IT-Branche erkannte den Bedarf an gut konfigurierbaren Komplett-Lösungen und es entstanden zunehmend Standard-Systeme. Heute ist die Nutzung beispielsweise von ERP Systemen die übliche Vorgehensweise und die individuellen Lösungen der Vergangenheit sind die absolute Ausnahme. Individual-Lösungen finden sich aktuell am ehesten in sehr exotischen Branchen oder in sehr kleinen Unternehmen, die faktisch kein ERP-System nutzen, sondern zum Beispiel ihre Buchhaltung mit einem Tabellenkalkulationsprogramm erledigen.

Im Zuge der Anforderungen entstand dann nach und nach ein komplett neuer Markt. Es wurden Entwicklungswerkzeuge bereitgestellt, die es ermöglichten, ohne tiefer gehende Programmierkenntnisse spezielle Aufgaben zu lösen. Ein klassisches Beispiel sind die Report-Generatoren, um Daten aufzubereiten und auszuwerten. Dazu zählen moderne Business Intelligence-Systeme, ETL-Tools, statistische Auswertungsprogramme oder auch „fertige“ Lösungen aus dem Bereich der künstlichen Intelligenz. Die Nutzung von Standardsoftware und von spezialisierten standardisierten Werkzeugen entwickelte sich bis in dieses Jahrzehnt hinein zum State of the Art. Klassische Software-Entwicklung spielte in vielen Bereichen nur noch eine sehr untergeordnete Rolle und fand in der Hauptsache bei den Herstellern der Standard-Werkzeugkästen oder Standard-Softwarelösungen statt.

Allerdings entwickelt sich seit Ende der neunziger Jahre die Open Source Bewegung als Gegenpol. Ziel ist es, Software ohne Lizenzkosten bereitzustellen, den Quellcode offenzulegen und somit zu ermöglichen, dass die Gemeinschaft der Nutzer und Entwickler die Software-Produkte stetig weiterentwickeln. Dabei lag der Fokus anfänglich auf privaten Endnutzern und Anwendern aus dem Bereich der Hochschulen. Es entstanden freie Betriebssysteme wie Linux, aber auch freie Web-Browser wie der 1998 erschienene Browser Netscape. In der kommerziellen Nutzung galten Open Source Produkte allerdings bis vor wenigen Jahren als zu unsicher. Im B2B-Sektor setzte man fast ausschließlich auf kommerzielle Software. Dies hat sich in den letzten Jahren massiv geändert. Man kann behaupten, dass Open Source Produkte kommerzielle Software immer stärker vom Markt verdrängen.

Eine weitere und damit einhergehende Entwicklung ist in dem Grad der Zentralisierung von IT-Systemen zu sehen. Im kommerziellen Umfeld waren die ersten Lösungen typischerweise zentralisierte Ansätze. Ein Großrechner war das Herz des Systems und der zentrale Ort für die Hardware und die Software. Die Arbeitsplätze waren nach dem Lochkarten-Zeitalter dann „dumme“ Terminals, die lediglich als Benutzeroberfläche dienten.

Es folgte eine Phase, in der IT-Systeme immer stärker dezentralisiert aufgebaut wurden. Für alle möglichen Anwendungsfälle wurden spezialisierte Server bereitgestellt. Die Software lief vor dem Internet-Zeitalter verteilt auf verschiedene Zonen, typischerweise auf dem Client-System, während die Datenhaltung auf einem zentralen Server war. Mit der Etablierung des Internets und insbesondere der Verbreitung der Web-Browser als zentrale Benutzerschnittstelle entstand dann eine typische Aufteilung in vier Zonen. Neben der Datenbankzone wurde eine Applikationszone erstellt, auf der – ob individuell oder via Standard-Umgebung – die Datenbewirtschaftung ablief. Außerdem kam der Webserver als neue Zone hinzu. Der Client, als Rechner des Endnutzers, war durchaus für die Ausführung von Teilen der Software zuständig. Diese eher dezentralen

Konstellationen waren bis vor einigen Jahren der Quasi-Standard in den meisten Unternehmen, bis sich sukzessive die Virtualisierung und dann das Cloud-Computing immer stärker etablierte.

An dieser Stelle soll kurz erklärt werden, was Cloud-Computing ist. Stark simplifiziert kann man sagen, dass IT-Systeme wieder zentralisiert werden. Der Endnutzer arbeitet über seinen Client mit den Services bzw. der Software auf der Cloud und den dort zentral gespeicherten Daten. Das Konzept erinnert an die erste Zeit der Großrechner. Allerdings mit dem riesigen Unterschied, dass eine Cloud nicht aus einem zentralen Server besteht, sondern aus einem großen Cluster von Servern bzw. Knoten. Diese Cloud-Cluster sind mit Ausfallsicherheit und ausgefeilten Desaster Recovery Konzepten aufgebaut. Alle Dienste laufen typischerweise auf virtuellen Maschinen. Während zum Beginn der Virtualisierung diese quasi wie ein Server fungierte, geht das mittlerweile so weit, dass einzelne Software oder Services virtualisiert bzw. containerisiert werden. Die Datenbank ist in dem Zusammenhang nur noch ein spezieller Container auf einer virtuellen Maschine, der problemlos auf eine andere Maschine verschoben werden kann.

2.2 Exkurs: Entwicklung der Virtualisierung

Mit dem Wandel von Großrechnern (z. B. IBM S/390) oder auch Midrange-Systemen (z. B. IBM AS/400) zu Client-Server-Architekturen trat beim Betrieb von IT-Infrastrukturen das zentrale Problem der Skalierbarkeit immer stärker auf. Bei diesen zentralisierten Architekturen ist es sehr komfortabel möglich, die Rechenleistung zu verteilen. Die Zuordnung der Ressourcen erfolgt dynamisch und eine Aufrüstung der Hardware kommt somit sofort allen auf dem System laufenden Programmen zugute. Genau in diesem Umfeld offenbaren sich bei Client-Server-Architekturen eklatante Probleme.

Hier kann die Hardware in einem gewissen Rahmen aufgerüstet werden, jedoch ist die Verteilung der Ressourcen bei Weitem nicht so bedarfsgerecht lösbar. Außerdem lassen sich verschiedene Programme nur bedingt voneinander kapseln, was sich sehr negativ auf die Betriebssicherheit auswirkt. Daher wurde die Skalierung einer komplexen Gesamt-IT-Infrastruktur zunächst über die Verteilung auf mehrere Server, den Server-Farmen, gelöst. Dort trennte man zudem die Applikationen von den Datenbanken und seit der Nutzung von Browser-Applikationen wurden Web-Server zumeist in isolierte Bereiche installiert. In aller Regel ist die benötigte Performance für alle diese Teile nicht immer linear, sondern es müssen Leistungsspitzen abgefangen werden können. Dadurch waren die vorhandenen Ressourcen in der IT-Infrastruktur oft ungenutzt, was die IT-Infrastruktur unnötig verteuerte.

Diese Situation war der Hauptgrund dafür, virtuelle Maschinen einzuführen, die auf einem sehr leistungsstarken Server mehrfach laufen können. Die Ressourcen der eigentlichen Hardware werden von allen vorhandenen virtuellen Maschinen geteilt. Benötigt eine virtuelle Maschine temporär mehr Leistung, wird sie zulasten der anderen temporär hoch skaliert. Im Prinzip baut Virtualisierung die Möglichkeiten nach, die Großrechner- oder Midrange-Systeme immer mitbringen.

Als nächster logischer Schritt folgte die Bereitstellung der Anwendung in Container. Das bedeutet, dass diese (z. B. eine modulare Suchfunktion) inklusive einer kompletten Laufzeitumgebung und mit allen notwendigen Abhängigkeiten bereitgestellt wird. Beispielsweise wäre das ein Textsuche-Service-Container, der genau das und nichts anderes leistet. Durch die Spezialisierung ist kein komplettes Gast-Betriebssystem notwendig. Es gibt deutlich weniger Overhead und Container kommen mit

weniger Ressourcen aus als komplette virtuelle Maschinen. Darüber werden die Anwendungen unabhängig von Hardware und Betriebssystem des Wirtssystems und somit wesentlich leichter portierbar. Schlussendlich lassen sich virtuelle Maschinen und Container sinnvoll miteinander kombinieren. Diese sich immer stärker durchsetzende Art der Virtualisierung von Anwendungen bzw. Services ist getrieben von den mittlerweile zumeist agilen Software-Projekten. Sie ist auch Treiber für passende Konzepte der Softwareentwicklung und den Betrieb von korrespondierenden Systemen.

Anstelle von monolithischen Lösungen setzen moderne containerisierte Applikationen auf Microservices in unabhängigen Containern. Aus Sicht der Virtualisierung lässt sich der Unterschied der klassischen virtuellen Maschine zu containerisierten Systemen schematisch folgendermaßen darstellen:

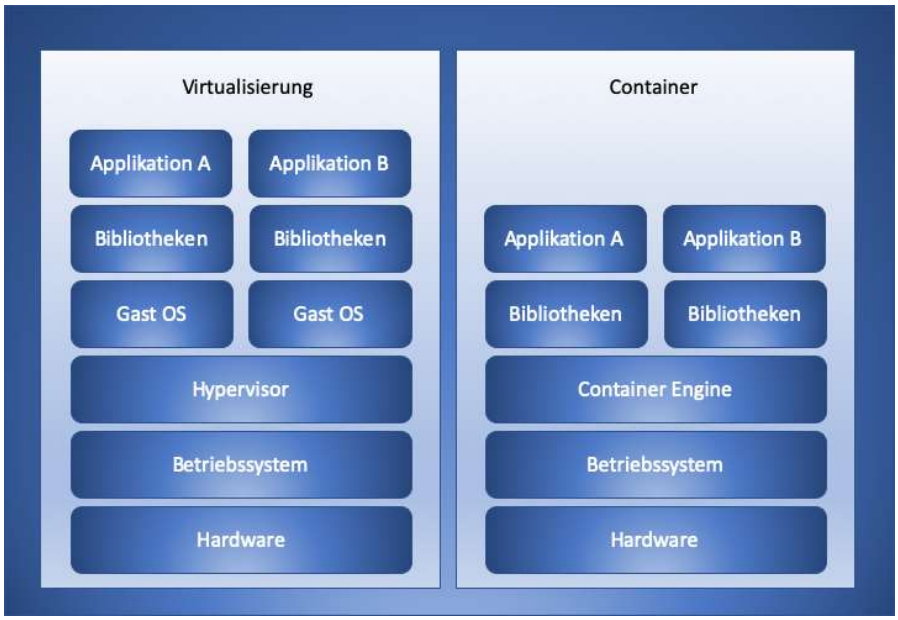

Abbildung 1: Virtualisierung vs. Containerisierung (eigne Darstellung)

Um Container zu verstehen, ist es außerdem wichtig das Prinzip der Microservices im Gegensatz zu klassischen monolithischen Applikationen zu kennen. Die nachfolgende Abbildung zeigt die Kombination unabhängiger modularer Einheiten (Microservices) zu den Applikationen. Dabei können Microservices in eigenen Containern abgebildet sein, die nur bei Bedarf „hochgefahren“ werden. Neben der sparsamen Nutzung von Ressourcen, wird auch die Bereitstellung (Deployment) und das Testen durch diesen Ansatz erleichtert. „Big Bang Deployments“ sind so nicht mehr notwendig.

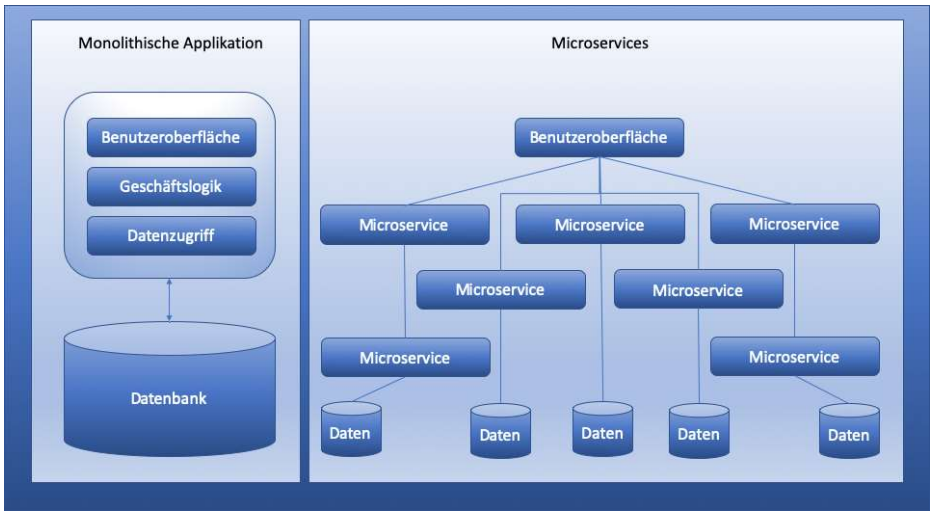

Abbildung 2: Monolithische Applikation vs. Microservices (eigene Darstellung)

In den nachfolgenden Kapiteln stehen die Veränderungen im Software-Engineering hin zur agilen Softwareentwicklung im Fokus.

3 Wandel im Software-Engineering

Parallel zu dieser technischen Entwicklung haben sich die Methoden des Software-Engineering sehr stark verändert, beziehungsweise an die aktuellen Gegebenheiten und Projektsituationen in den Unternehmen angepasst. Zwei Vorgehensweisen, welche die Weiterentwicklung des Software-Engineering aufzeigen, möchte ich Ihnen vorstellen. Zum einen das klassische V-Modell, als Repräsentant für Software-Entwicklungsprojekte in den ersten Jahrzehnten der Datenverarbeitung. Zum anderen als Gegenpol die agile Software-Entwicklung, wie sie in modernen Projekten zwischenzeitlich Gang und Gebe ist.

3.1 V-Modell

Das V-Modell teilt ein Software-Entwicklungs-Projekt in mehrere Phasen auf, die sukzessive durchlaufen werden. Jede endet dabei mit einem Meilenstein, der von der Partei, die mit den erarbeiteten Ergebnissen weitermachen soll, abgenommen werden muss. Etwaige Mängel müssen bereinigt werden, damit ein solches Quality Gate durchschritten werden kann und erst dann beginnt der nächste Schritt. Das Ganze läuft vereinfacht dargestellt folgendermaßen ab:

- Erstellung einer Fachspezifikation und fachlicher Testfälle sowie Bereitstellung von Testdaten durch den zukünftigen **Nutzer.**
- Quality Gate 1: Abnahme der Spezifikation durch die **Softwareentwicklung.**
- Erstellung einer IT-Spezifikation beziehungsweise technischen Spezifikation durch die **Softwareentwicklung.**
- Quality Gate 2: Abnahme der technischen Spezifikation durch die **Nutzer** der Software (gegebenenfalls mit Unterstützung von hinzugezogenen IT-Experten).

- Entwicklung der Software gemäß der Fach- und IT-Spezifikation durch die **Softwareentwicklung** inklusive aller notwendigen Tests gegen die bereitgestellten Testfälle mit den Testdaten (Unit Test, Integrationstest). Anschließende Bereitstellung der Software an die **Nutzer** zum fachlichen Test.
- Fachlicher Test gegen die Testfälle seitens der **Nutzer.**
- Behebung aller Fehler seitens der Softwareentwicklung. Beim Auftreten von Problemen, die nicht spezifiziert waren oder für die es keine Testfälle und/oder Testdaten gibt, werden diese als Change Request definiert.
- Quality Gate 3: Abnahme der Software (alle bereitgestellten Testfälle mit den dazugehörigen Testdaten wurden erfolgreich getestet)

Die nachfolgende Abbildung illustriert das V-Modell. Auf der x-Achse wird die Zeit abgebildet und auf der y-Achse der Detaillierungsgrad, wobei dieser von grob (oben) nach detailliert (unten) verläuft.

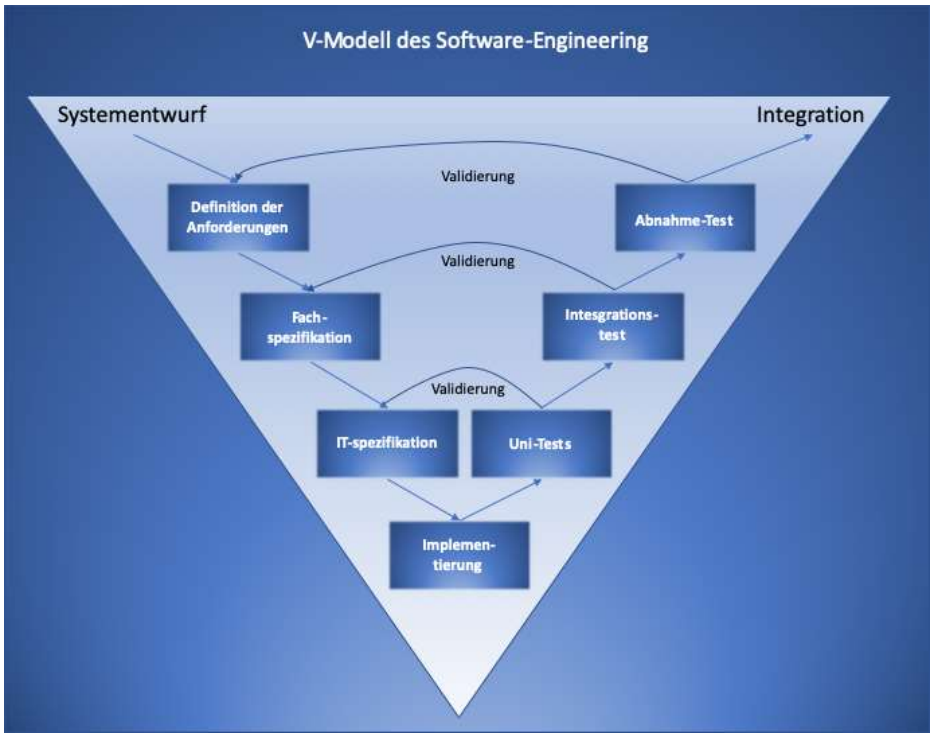

Abbildung 3: Das V-Modell des Software-Engineering (eigene Darstellung)

Die Summe der umzusetzenden Change Requests ergeben ein neues Paket, welches analog der obigen Beschreibung wieder den gesamten Prozess durchläuft.

Das V-Modell bietet den Vorteil einer streng formalen Vorgehensweise und das umzusetzende Ziel ist von Anfang an klar. Allerdings liegt eben darin der Nachteil des Verfahrens, da es oft schwierig ist, die Anforderungen an neue Software umfassend und genau zu beschreiben und das V-Modell eigentlich keine Iterationen vorsieht. Es dauert sehr lange, bis die zukünftigen Nutzer erste Ergebnisse der Software-Entwicklung nicht nur auf dem Papier sehen. Projekte, die mit dem V-Modell abgewickelt werden, sind oft träge, langatmig und teuer.

3.2 Agile Software-Entwicklung

Die agile Software-Entwicklung kann als Gegenentwurf zum klassischen V-Modell verstanden werden. Es gelten zwölf Grundprinzipien:

- Software wird an den Kunden bzw. den Nutzer kontinuierlich und früh in einer guten Qualität ausgeliefert.
- Veränderungen werden auch in späten Phasen der Entwicklung zum Vorteil des Kunden genutzt.
- Software wird funktionsfähig in regelmäßigen kurzen Zeitspannen ausgeliefert (Sprints).
- Fachexperten und Softwareentwickler arbeiten kontinuierlich eng zusammen.
- Schaffung motivierender Rahmenbedingungen und gegenseitige Unterstützung.
- Informationen werden regelmäßig im persönlichen Gespräch übermittelt.
- Die Funktionsfähigkeit der Software ist das entscheidende Fortschrittsmaß.
- Alle Beteiligten arbeiten in einem aufeinander abgestimmten gleichmäßigen Arbeitstempo.
- Technisch saubere Implementierung und gutes Software-Design sind immer im Fokus.
- KISS-Prinzip: keep it short and simple als Leitlinie.
- Selbstorganisierte Teams als Garant für gute Architektur, sinnvolle Anforderungen etc.
- Selbstreflexion als Teil der kontinuierlichen Kommunikation, um sich stetig zu verbessern.

In der agilen Software-Entwicklung wird Software iterativ entwickelt. Anstelle einer kompletten umfassenden Fachspezifikation liefert der Kunde sogenannte User-Stories. Diese werden dann in kurzen Entwicklungs-Sprints implementiert und kontinuierlich zum Review bereitgestellt.

Nachfolgende Abbildung visualisiert den Prozess der agilen Software-Entwicklung allgemein und schlägt die Brücke zum Betrieb agil entwickelter Systeme und damit zur Philosophie der DevOps:

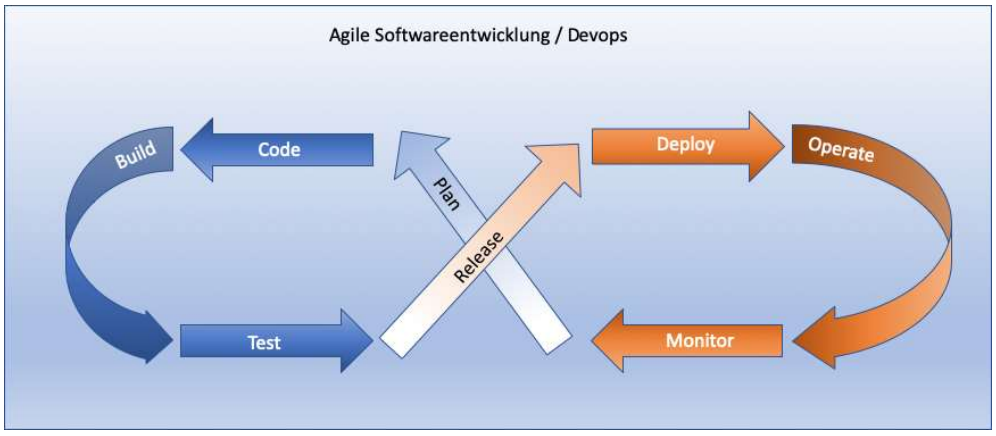

Abbildung 4: Agile Softwareentwicklung / DevOps (eigene Darstellung)

Der Nachteil an diesem Verfahren ist, dass zum Projekt-Beginn nicht klar ist, was genau im Rahmen eines bestimmten Budgets ausgeliefert wird. Der Scope passt sich im Laufe des Projektes an. Allerdings hat der Ansatz deutlich weniger Reibungsverluste und ist speziell für innovative Themen besonders zielführend. Zudem gibt es zunehmend Software-Projekte, bei denen ein iteratives Vorgehen zwingend erforderlich ist, da immer wieder Interaktionen zwischen Fachexperten und Softwareentwicklern notwendig ist. Ganz typischerweise ist das bei allen Projekten im Bereich der Künstlichen Intelligenz (KI) der Fall. KI-Projekte sind mit einem V-Modell nicht sinnvoll realisierbar.

4 DevOps

Kaum ein Begriff hat den Bereich der IT in den letzten Jahren stärker geprägt wie das Kunstwort DevOps. Herausfordernd dabei ist, dass in vielen Unternehmen eine Unklarheit herrscht, was darunter zu verstehen ist. Betrachtet man den kompletten Lebenszyklus von Software, von der Konzeption über die Erstellung bis hin zum Betrieb und technologischer Updates, ist zu erkennen, dass zwei Bereiche mit unterschiedlichen Interessen diesen Lebenszyklus begleiten. Auf der einen Seite hat die Softwareentwicklung das Ziel, die Software immer funktional als auch technisch zu verbessern und zu erweitern. Die Erfüllung der Anforderungswünsche der Anwender steht hier im Fokus. Andererseits bedeuten alle Veränderungen immer ein Risiko für den Betrieb der Software. Es können Fehler in ein bisher stabiles System geraten oder es werden andere oder stärkere Ressourcen gebraucht. Eine Optimierung des Zusammenspiels zwischen Entwicklung und Betrieb (in Englisch zwischen **Dev**elopment und **Op**eration**s**) kann und soll zu einer schnelleren und fehlerärmeren Softwareauslieferung führen. Der belgische Systemadministrator Patrick Debois prägte in diesem Zusammenhang das Kunstwort DevOps. Im Kapitel 3.2 zeigt die Abbildung 4 genau dieses Zusammenspiel auf. Aber mit welchen Maßnahmen kann dieser Prozess optimiert werden?

4.1 Automation

Ein Grundpfeiler der DevOps-Philosophie ist eine fast vollständige Automatisierung bestimmter Prozesse, die einen starken Impact auf den Betrieb von Software haben. Im Fokus stehen dabei vor allem das (technische) Testen sowie das Bereitstellen (Deployment) der Software.

4.1.1 Testautomation

Ein Softwaretest umfasst mehrere Schritte, die prinzipiell automatisiert werden können:

- Erstellung der Testfälle
    - Erstellung von Testdaten
    - Erstellung von Testskripten
- Durchführung
- Auswertung
- Dokumentation
- Administration

Zur Erstellung von **Testfällen** für automatische Tests ist es notwendig, dass zur Beschreibung ein geeignetes Format genutzt wird. Hierzu werden Entwicklungssprachen verschiedener Abstraktionsstufen verwendet. So können Testdaten tabellenartig erstellt werden, während die Funktionsaufrufe mit Skripten (z. B. Python, Perl etc.), objektorientierten Ansätzen (z. B. JUnit), imperativ (z. B. in C) oder modellbasiert (z. B. Time Partition Testing) festgelegt werden. Vom Grundsatz werden so aus eher fachlichen Testfällen maschinennahe (und damit automatisierbare) Testfälle erzeugt. Bei der Automatisierung bietet das Datenmodell der Software den systematischen Rahmen. Bei der Erstellung von Testfällen wird oft unterstützende Software eingesetzt, die beispielsweise erlaubt, diese mit einer grafischen Benutzeroberfläche zu modellieren und mit dem zu Grunde liegenden Datenmodell zu verknüpfen, um dann komfortabel die passenden Testdaten zu erstellen.

Die eigentliche **Testdurchführung** erfolgt nun idealerweise mit automatisierten Testwerkzeugen. Hierbei gibt es zahlreiche Tools für automatisierte Unit-Tests oder Performance- und/oder Lasttests. Auch Integrationstests oder weitere komplexere Tests können durchaus automatisiert werden. Allerdings ist hierbei ein guter Tradeoff zwischen Aufwand der Implementierung dieser Tests und dem tatsächlichen Nutzen kritisch zu hinterfragen. Generell wird durch die Automation ein möglichst lückenloses Testen aller elementaren Units der Software stark erleichtert bzw. beschleunigt und damit auch die Wahrscheinlichkeit von Fehlern bei weiterführenden Tests stark minimiert.

Bei der **Testauswertung** wird je Testfall das Testergebnis erwartete Ergebnis verglichen. Hierbei ist zu bemerken, dass bei sehr komplexen Systemen Probleme auftreten können und die Automation an dieser Stelle durchaus herausfordernd sein kann. Die automatisierte Testauswertung ist typischerweise Teil von Test-Automations-Tools, genau wie die darauf aufbauende automatische Erstellung einer einfach lesbaren **Testdokumentation**.

Schließlich gehört zu einem guten System zur Automation von Tests auch eine Möglichkeit der Versionierung der Tests. Dies ist ein entscheidender Teil der **Testadministration**.

4.1.2 Automation der Software-Bereitstellung

Die eigentliche Bereitstellung der Software ist der nächste logische Schritt. Auch hier ist es mit geeigneten Tools möglich, mithilfe von Skripten den Prozess komplett zu automatisieren. Dabei spielt das Versionsmanagement eine entscheidende Rolle. Alle notwendigen Metadaten werden hierzu in Repositories abgelegt. Idealerweise wird der Prozess der automatisierten Software-Bereitstellung logisch und inhaltlich verknüpft mit den automatisierten Tests des jeweiligen Releases.

Im Rahmen der agilen Softwareentwicklung wird Software kontinuierlich konzipiert, erweitert, getestet und bereitgestellt, wie es in Abbildung 4 des Kapitels 3.2 dargestellt ist. Auch muss in einer solch agilen Welt ein kontinuierliches Monitoring des Betriebs stattfinden. Das Arbeiten mit Containern ist im Zuge dessen hervorragend geeignet, die bisher beschriebenen Automationsprozesse zu unterstützen. Der Aufbau moderner Systeme mit Microservices und die Containerisierung sind daher ganz elementare Bestandteile einer DevOps-Philosophie.

4.2 Teamarbeit und Kontinuität

Komplexe Software-Projekte werden in der Regel von Entwicklerteams erstellt. Bei der agilen Softwareentwicklung ist es von entscheidender Bedeutung, dass die Quellcode-Updates aller Entwickler in möglichst kurzen Zeitabständen kontinuierlich

zu einer Hauptlinie integriert werden. Das bedeutet einen Paradigmenwechsel, da diese fast täglich miteinander kommunizieren und diese **kontinuierliche Integration** sicherstellen müssen.

Auch die Testprozesse bzw. die Qualitätssicherung stehen bei agiler Vorgehensweise vor neuen Herausforderungen. Mithilfe der beschriebenen Automatisierungskonzepte und geeigneter Toolunterstützung ist **kontinuierliches Testen** in engen Zyklen bereits durch die Entwickler von entscheidender Bedeutung für den Projekterfolg. Damit wird die Qualität dessen, was durch separate QS-Prozesse im Qualitätsmanagement getestet wird, auf ein möglichst hohes Niveau gehoben. Im besten Fall finden sich während der klassischen QS-Maßnahmen nur noch fachliche Fehler, die sich nicht durch Testautomatismen abbilden lassen.

Gelingt es, kontinuierliche Integration, kontinuierliches Testen und Versionierung zu automatisieren, ist sichergestellt, dass alle Build-Artifakte immer automatisch erfolgreich einen adäquaten Testprozess durchlaufen haben müssen und somit quasi bereitstellungsfertig sind. Der Deployment-Prozess ist dann idealerweise vollständig automatisiert und eine Bereitstellung der integrierten getesteten Software kann mehr oder weniger ad hoc bzw. schnell und häufig erfolgen. Um dieses kontinuierliche Ausliefern zu erreichen, bieten die für die Testautomation genannten Tools, integriert in einer Umgebung, oft die Möglichkeit, das Deployment mit zu automatisieren.

Durch den stark iterativen und agilen Prozess ist es nicht möglich, Software in einer strengen Weise zu testen, wie es im Rahmen des V-Modells vorgesehen ist. Auf der anderen Seite wird durch das kontinuierliche Vorgehen die Qualität der Software stark verbessert. Dennoch ist es in einer agilen Welt wichtig, die Software **laufend zu überwachen**. Deshalb obliegt in einem DevOps-Ansatz die Software einem permanenten Monitoring. So werden Qualitätsprobleme schnell identifiziert und in einer kommenden Iteration behoben.

4.3 Infrastruktur als Code

Ein weiteres wichtiges Verfahren im Bereich der DevOps ist die Automatisierung der Provisionierung bzw. zur Verfügung Stellung von Infrastruktur (etwa Bedarf an Storage Volumen) über „Infrastruktur-Code“. Damit können Umgebungskonfigurationen besser überwacht werden und Änderungen an der Konfiguration nachvollzogen werden. Etwaige Rollbacks zu alten Versionen der Konfigurierung werden damit deutlich vereinfacht. Der „Infrastruktur-Code“ ist stark simplifiziert ausgedrückt alles, was ein Container konfiguriert. Neben der Provisionierung ist selbstverständlich auch der umgekehrte Weg möglich, um etwa nicht mehr benötigten Storage in einem Container wieder frei zu geben.

4.4 Zusammenfassung

Eine DevOps-Philosophie in der Softwareentwicklung bedarf eines gewissen Aufwandes zur Schaffung möglichst perfekter Voraussetzungen, um die Kontinuität in Kombination mit der Zusammenarbeit in Entwicklerteams sicherzustellen. Ansätze hierzu sind:

- Einführung von Containerisierung.
- Erstellung und Nutzung von Microservices anstelle monolithischer Ansätze.

- Toolgestützte Automation von Integration, Test und Bereitstellung von Software.
- Agile Software-Engineering-Methoden im IT-Management (wie etwa SCRUM).
- Kooperatives Arbeiten von Software-Entwicklung, Software-Betrieb, Qualitätsmanagement und Kunden.
- Kontinuierliches Streben nach Verbesserung aller Prozesse durch regelmäßige Reflexion mit allen Beteiligten (Softwareentwicklung, Betrieb, Nutzer, Management, Qualitätssicherung).

Sind diese Voraussetzungen erst einmal geschaffen, können erhebliche Verbesserungen in der Qualität der Software, deren Betreibbarkeit und auch der Kosten der Entwicklung erreicht werden.

Das sind insbesondere:

- Qualitativ hochwertigere und schnellere Bereitstellung neuer Software.
- Schnellere Lösung der Anforderungen oder Probleme und Verringerung der Komplexität.
- Deutlich flexiblere und bessere Skalierbarkeit.
- Steigerung der Verfügbarkeit der Systeme und Verbesserung der Verfügbarkeit von Betriebsumgebungen.
- Optimierte Auslastung aller Ressourcen.
- Steigerung der Transparenz.
- Förderung von Innovation.

Die mip GmbH ist bei sehr unterschiedlichen Kunden hinsichtlich der Unternehmensgröße und der Branche tätig. Viele sind noch stark im klassischen V-Modell des Software-Engineering unterwegs, während andere bereits eine DevOps-Philosophie leben. Aus unserer Sicht geht mittel- bis langfristig kein Weg an agiler Softwareentwicklung und DevOps vorbei. Da wir sowohl im konventionellen Softwareengineering als auch in modernen Methoden zu Hause sind, können wir insbesondere Kunden dabei helfen, in dieser „neuen Welt“ anzukommen. Selbstverständlich arbeiten wir genau so gerne und gut in Projekten mit klassischen Ansätzen wie dem V-Modell.

Kontakt:

Jörg Kremer
Head of Consulting

Mail: Joerg.Kremer@mip.de
Tel.: +49 89 58939440