SDLC: Idealer Prozess

Tsomo Taf A.

22. Januar 2024

Inhaltsverzeichnis

1 Einleitung

1.1 Die Bedeutung eines Idealen SDLC-Prozesses

Die fortschreitende Evolution der Softwareentwicklung erfordert nicht nur innovative Lösungen, sondern auch einen effizienten und zuverlässigen Prozess, um diese Lösungen von der Konzeption bis zur Bereitstellung zu führen. In diesem Kontext steht diese Projektarbeit unter dem Schwerpunkt "DevOps". Der zweite Teil meines Projekts ist gewidmet der Entwicklung eines idealen Softwareentwicklungslebenszyklus (SDLC)-Prozesses für die OmniView-Software. Dieser Prozess erstreckt sich von der Anforderungsanalyse (RE) bis zur Auslieferung (Delivery) in einem idealen Rahmen. Anders gesagt ist das Ziel von DevOps die *Time to Market* zu verringern.

1.2 Die Vision von OmniView

Omniview verfolgt die Vision, nicht nur eine Visualisierungssoftware zu sein, sondern eine Plattform, die die Messdatenerfassung in Forschungsprojekten, insbesondere AW4null, revolutioniert. Die Software soll nicht nur eine benutzerfreundliche Oberfläche bieten, sondern auch den Einsatz von künstlicher Intelligenz (KI) in der Fahrzeugdiagnose ermöglichen. Die Daten, die von verschiedenen Messgeräten erfasst werden, sollen nicht nur in der GUI angezeigt, sondern auch nahtlos an verschiedene APIs gesendet werden können. Im weiteren Verlauf meiner Projektarbeit werde ich detailliert auf den entworfenen SDLC-Prozess eingehen, um die Umsetzung der Vision von Omniview zu unterstützen. Durch die Integration von bewährten Praktiken, agilen Methoden und DevOps-Prinzipien wird dieser Prozess nicht nur die Softwarequalität verbessern, sondern auch eine effiziente Lieferung von Innovationen ermöglichen.

2 Managemententscheidung für Scrum im Omniview DevOps-Workflow

Die Wahl von Scrum als zentrales Framework für den DevOps-Workflow der Omniview-Software basiert auf der strategischen Ausrichtung, die Effizienz, Flexibilität und Zusammenarbeit im gesamten Entwicklungszyklus maximieren soll. Scrum bietet nicht nur einen agilen Ansatz, sondern auch klare Rollen, Artefakte und Veranstaltungen, die die Grundlage für einen kohärenten Entwicklungsprozess bilden. Diese Entscheidung kommt aus der Überzeugung, dass eine agile Methodik, die auf die Dynamik von Scrum setzt, die Vision von Omniview effektiver vorantreiben wird. Aber natürlich konnte ein anderes Framework für agile Methoden wie Kaban auch den idealen Prozess der Omniview-Software unterstützen.

2.1 Anforderungsanalyse von OmniView

2.1.1 Stakeholder-Analyse im Scrum-Framework für Omniview

a.Product Owner (PO)

Der Product Owner (Herr Bökelmann) ist der Vertreter des Kunden(AW4.0). Er hat eine Vision für Omniview und somit die fachlichen Anforderungen an das Produkt beziehungsweise die zu erstellende der Omniview-Software vorgibt und die ganzen Anforderungen werden im sogenannten product backlog gesammelt und priorisiert festgehalten. Das heißt das Product Backlog ist ein Dokument, in dem alles aufgelistet ist, was im Omniview enthalten sein kann. Und das soll jetzt alles innerhalb des srum Prozesses umgesetzt werden. Die Umsetzung sieht jetzt folgendermaßen aus.

b. Scrum Master:

Rolle: Der Scrum Master unterstützt das Scrum-Team, indem er sicherstellt, dass Scrum-Richtlinien eingehalten werden und Hindernisse beseitigt werden. in einem idealen Prozess mit einer, also zwei Personen mit sehr unterschiedlichen Rollen. Beziehung zu Omniview: Der Scrum Master ist auch Herr Bökelmann spielt eine Schlüsselrolle bei der Sicherstellung eines reibungslosen DevOps-Prozesses für Omniview, indem er Hindernisse für das Entwicklungsteam beseitigt.

c. Entwicklungsteam:

Rolle: Mitglieder von Skunforce, die für die Implementierung und Lieferung von Omniview verantwortlich sind. Sie sind direkte Umsetzer der Anforderungen aus dem Product Backlog, die die Software entwickeln und verbessern.

d. Betriebspersonal:

Rolle: Personen, die für den Betrieb und die Wartung der Omniview-Software verantwortlich sind. Betriebspersonal spielt eine entscheidende Rolle bei der Gewährleistung der Systemstabilität, Skalierbarkeit und Verfügbarkeit von Omniview in Produktionsumgebungen.

e. Das DevOps-Team:

- **Teamleiter:**
  Die Rolle des Teamleiters besteht darin, das Team zu koordinieren und sicherzustellen, dass es effizient und ohne Unterbrechungen arbeiten kann. Ähnlich wie beim Scrum Master aus dem Vorgehensmodell Scrum handelt es sich beim Teamleiter um eine Rolle, die keine Autorität über das Team ausübt. Das Team isttrotz Teamleiter selbstregulierend.
- **Service-Owner:**
  Der Service-Owner spielt eine vergleichbare Rolle wie der Product Owner in Scrum. Diese Position bildet die Schnittstelle zwischen dem Team und der Äußenwelt". Der Service-Owner kommuniziert mit Stakeholdern und Kunden, versteht die architektonische Vision des Gesamtsystems und priorisiert Aufgaben für zukünftige Iterationen.
- **DevOps Engineer:**
  Der DevOps Engineer ist verantwortlich für die Auswahl und Konfiguration von Konfigurationsmanagement-Tools sowie Deployment-Tools. Er weiß, wie diese konfiguriert und bedient werden

f. Stakeholder aus AW4.0:

Mitglieder von AW4.0, die die Forschungsziele und -anforderungen vertreten. Stakeholder aus AW4.0 beeinflussen die Produktvision und -richtung von Omniview durch ihre Anforderungen und Erwartungen. Forscher und Entwickler innerhalb von Skunforce sind direkte Beteiligte am Entwicklungsprozess und beeinflussen die technischen Aspekte des Projekts.

g. Endbenutzer:

Diejenigen, die letztendlich Omniview nutzen werden, z. B. Mechaniker in Autowerkstätten. Endbenutzer geben wichtige Einblicke und Feedback, das die Produktentwicklung und -verbesserung beeinflusst. Bei einem Forschungsprojekt wie diesem sind die Nutzer auch interne Mitglieder des Projekts.

2.1.2 Planung und Organisation den SDLC-Prozess von Omniview

Es müssen Meeting zur Arbeitsorganisation mit Teammanagern organisiert werden, um die Arbeitsstrategie zu definieren, Ressourcen zu bewerten usw. Während der Besprechung sammelt der DevOps-Ingenieur die Informationen, die es ihm ermöglichen, die Konfigurationsdateien je nach Programmiersprache der Software vorzubereiten (zum Beispiel C++ für Omniview) oder die Abhängigkeiten, die diese virtuelle Maschine benötigen würde.

a. Product Backlog

Erster Schritt ist die Erstellung eines umfassenden Product Backlogs, das alle Anforderungen, Funktionen und Verbesserungsvorschläge für Omniview enthält. Mit anderen Worten, nach dem Brainstorming von Herrn Bökelmann mit den Forschern von AW4.0 wurde eine Liste von Userstories (Z.B: Diagnostics Menu, Generate Training Data Menu and Analyze current waveform, Auto-Shop UUID VCDS oder Auto-Scan) erstellt. Die Backlog-Elemente sollten priorisiert werden, wobei die wichtigsten Funktionen und Anforderungen zuerst behandelt werden.

b. Sprint Planning

Basierend auf dem Product Backlog erfolgt die Sprint-Planung, in der das Entwicklungsteam gemeinsam mit dem Product Owner die Backlog-Elemente für den nächsten Sprint auswählt. Die Auswahl erfolgt unter Berücksichtigung der Sprint-Ziele, der Kapazität des Entwicklungsteams und der Prioritäten.

c. Sprint-Zyklus

Jeder Sprint sollte eine festgelegte Dauer (z. B. zwei Wochen) haben. Das Entwicklungsteam arbeitet während des Sprints an den ausgewählten Backlog-Elementen, um am Ende des Sprints potenziell auslieferbare Inkremente des Produkts zu haben. Die Planung erfolgt nicht nur am Anfang, sondern kontinuierlich während des gesamten SDLC-Prozesses.

2.2 Entwickler und Entwicklungsprozess

DevOps macht den Entwicklungsprozess durch Automatisierung flüssig. Das heiß: Continuous Integration (CI).
Continuous Integration (CI) ist ein Prozess, der automatisch durchgeführt wird, nachdem neu entwickelter Code in ein Versionskontrollsystem eingecheckt wird (Check-In). Dadurch soll verhindert werden, dass Entwickler Integrationsprobleme riskieren, indem sie neu entwickelten Code zu selten einchecken oder Branchs zu selten zusammenführen. Der Vorgang sollte so oft wie möglich durchgeführt werden, wenn nach dem Speichern Änderungen an der Codebasis vorgenommen werden. Für die Umsetzung ist ein CI-Server zuständig. Es verwendet sie, um Unit- und Integrationstests durchzuführen. Je nach Konfiguration können CI-Server auch Softwaremetriken verwenden, um die Codequalität und die Einhaltung von Codekonventionen zu testen. Sollten die durchgeführten Tests Fehler enthalten oder die Qualität des Codes nicht den Richtlinien entsprechen, wird der Check-In abgelehnt und der Entwickler, der sich vorgenommen hat, wird über das Problem informiert. Auch wenn der Check-In erfolgreich ist, stellt der CI-Server relevante Informationen bereit.

2.3 Operators und der Deployment-Prozess

Deployment-Prozess umfasst den gesamten CI-Prozess, erweitert um einen weiteren Schritt. Nach den CI-Schritten ist automatisch sichergestellt, dass die neu erstellte Version für den Produktivbetrieb eingesetzt werden könnte. Der CI-Prozess wird um die Phase Akzeptanztest / Staging / Performancetests

erweitert. Dabei wird die neue Version unter anderem in einer Testumgebung (Staging-Umgebung) möglichst nah an der Produktionsumgebung auf die Probe gestellt. Es sollte hier erwähnt werden, dass die Begriffe Continuous Delivery und Continuous Deployment im Fachbereich IT unterschiedlich belegt werden. Als weiterer Schritt im Continuous Deployment wird die neue Version, die sich im Continuous Delivery-Prozess als produktionstauglich erwiesen hat, automatisch für die Produktion bereitgestellt. Einer der Vorteile der kontinuierlichen Bereitstellung besteht darin, dass sich auf die Entwicklung fokussiert werden kann, ohne sich um den Bereitstellungsprozess kümmern zu müssen.

2.4 CI/CD-Pipeline

Die CI/CD-Pipeline ist ein übergeordnetes Konzept, das den gesamten Prozess von der Codeintegration bis zur Auslieferung umfasst und die dabei eingesetzte Infrastruktur. Die im Folgenden beschriebene Deployment-Pipeline ist nur ein Beispiel dafür, welche Schritte während des Prozesses im Idealfall durchlaufen und wie diese ausgestaltet werden können.
Die einzelnen Schritte sind von Projekt zu Projekt individuell gestaltet. Der Deployment Prozess ist abhängig von den Anforderungen an die zu bauende Software und auch von den Projekt- und Firmenstrukturen, in deren Kontext die Software entwickelt wird. Es ist daher wichtig, daran zu erinnern, dass ein idealer SDLC-Prozess in der Regel eine Utopie ist. Dieses Beispiel berücksichtigt also das Omniview-Projekt, aber in einem idealen Rahmen oder mit einem idealen Workflow.
Workflows: Die Pipeline hat mindestens fünf Hauptschritte Vor-Commit-Phase, Build und Integrationstests, Akzeptanztests / Staging / Performance-Tests, [partieller] Produktivbetrieb und Freigabe in den normalen Produktivbetrieb/ Monitoring und Logging

2.4.1 Vor-Commit-Phase

a. Aufgabe des scrum masters

- Erstellung von zweiwöchigen Sprints
- Besprechungen zur Sprintplanung und Review.
- Einführung eines Ticketmanagementsystems (mit jira oder slcack)
- Erstellung des Git-Repository
- Definition der Projektziele /Definition der Projektanforderungen /Definition der Projektarchitektur/Definition des Entwicklungsplans

b. Entwickler:

In dieser Phase wird noch in der Entwicklungsumgebung operiert. Entwickler arbeiten an einer neuen Version einer Anwendung. Mit jedem Commit - dem Beitragen von Code in die Codebasis - werden idealerweise automatisch alle Unit-Tests durchgeführt oder um Zeit zu sparen nur eine Untermenge Zur besseren Absicherung können die Entwickler vor dem Übergeben von Veränderung manuell alle Unit-Tests ausführen. Während der Entwicklung werden die Unit-Test um Tests erweitert, die die neu entwickelte Funktionalität abdecken. Die neuen Unit-Tests werden mit ausgeführt.
Der Teamleiter von Entwickler oder DevOps-Engineers konfigurieren und warten CI-Server wie Github action (in unserem Fall) Jenkins oder GitLab CI, die den automatischen Build und die Durchführung von Tests auslösen.

2.4.2 build und Integrationstest

a-Git-workflow

- Beschreibung: Hinzufugen der Funktion ”FeatureX” zu OmniView.
- Erstellung des Branches: git checkout -b feature-FeatureX
- Codeentwicklung: Nehmen Sie die erforderlichen Anderungen im Quellcode vor, um ”FeatureX” hinzuzufugen. Bei Bedarf mit anderen Beitragenden ¨ zusammenarbeiten.
- Lokale Validierung: Lokales Testen, um sicherzustellen, dass die Funktion ordnungsgemäß funktioniert.
- Commit der Anderungen: git commit -m “ Hinzufüugen der Funktion FeatureX”
- Push in das Remote-Repository: git push origin feature-FeatureX
- Erstellung eines Pull Requests (PR) / Review des Pull Requests
- Auslösen der CI/CD-Pipeline und Build des Quellcodes

b-bulid-Prozessg

Zuständig für diesen Schritt ist ein Integrationsserver (CI-Server). Von ihm werden die Ressourcen zusammengestellt, von der die Anwendung abhängig ist: Die internen Ressourcen werden organisiert, wie zum Beispiel benötigte APIs und externen Ressourcen werden konfiguriert beispielsweise eine Anbindung an eine Datenbank. Die Codebasis wird zusammen mit den Ressourcen in eine ausführbare Anwendung gebaut und diese mit der entsprechenden Konfiguration versehen. Bei diesem Vorgang entsteht das Artefakt Build. Der Build sollte ab diesem Schritt nicht mehr verändert werden, außer durch den Einfuss von Konfigurationsparametern. Der CI-Server führt an dem Build Integrationstests durch, die das fehlerlose Zusammenspiel aller Komponenten der Anwendung testen. Es ist sinnvoll, schon hier in einer Umgebung zu testen, die der Produktivumgebung so stark wie möglich gleicht.

2.4.3 Akzeptanztests / Staging / Performance-Tests

Nutzung einer isolierten Testumgebung zur Durchführung unterschiedlicher Tests. Beim Staging wird versucht eine (Staging-)Umgebung zu schaffen, die der Produktivumgebung so nahe wie möglich nachempfunden ist. Die Anwendung sollte in dieser Phase dieselbe Konfiguration haben, wie sie sie auch im (vorläufigen) Produktivbetrieb haben wird.
Es werden Akzeptanztests mit einer Auswahl von Nutzern durchgeführt, die die Anwendung wie vorgesehen (z.B. über die GUI) nutzen und Feedback über eventuelle Probleme geben. Durch den Einsatz von automatisierten Akzeptanztests, wird auch, wie bei manuell ausgeführten vorgeschriebenen Akzeptanztest, ein hoher Grad an Wiederholbarkeit erreicht.
In diesem Schritt werden auch nicht-funktionale Tests durchgeführt, die Performance, Sicherheit etc. testen.
Obwohl man sich in einem idealen Prozess befindet, ist es wichtig, darauf hinzuweisen, dass zu viele Tests zu größerem zeitlichem Aufwand führen, zu wenig Tests erhöhen die Chance, dass Fehler nicht gefunden werden. Tests sollten mit Bedacht geschrieben werden und nicht mit dem Ziel, alle Fehler in der Software zu finden, da dies unmöglich ist. Es wird immer Fehler geben, die sich erst bei längerem Produktivbetrieb auftun.

2.4.4 [partieller] Produktivbetrieb und Freigabe in den normalen Produktivbetrieb

In Hinsicht auf das Design sollte der DevOps-Ingenieur die Architektur der Software verstehen, um sicherzustellen, dass Deployments reibungslos verlaufen und dass Ressourcen effizient genutzt werden. Diese ist für die Optimierung oder die Verwendung von der IaC oder Containerisierung von entscheidender Bedeutung.

a. So wird es auf herkömmliche Weise gemacht

Bereitstellung mit Docker, Kubernetes, Ansible und Terraform (Webanwendung als Beispiel)

Vorbereitung der Produktionsumgebung

Dazu muss man Docker, Ansible und/oder Terraform auf einem Remote-AWS-Server oder in einer von Vagrant und VirtualBox erstellten oder bereitgestellten VM installieren

Bereitstellung der Software mit Docker, Kubernetes, Ansible und Terraform

- Verwendung von Docker für den Bau und die anfängliche Bereitstellung (Dockerfile, Docker images, Docker container)
- Push des Images in eine Docker-Registry (z. B. Docker Hub)
- Verwendung von Terraform für die Erstellung der Kubernetes-Infrastruktur
- Verwendung von Ansible für die Bereitstellungskonfiguration

Überwachung der Systemleistung und -stabilität

- Nutzung von Kubernetes Dashboard oder Monitoring-Tools (Prometheus, Grafana)
- Alarmierung mit benutzerdefinierten Regeln konfigurieren
- Verwendung von Docker Stats zur Überwachung einzelner Docker-Container
- Docker stats name_des_container
- Einrichten von Systemprotokollen zur Überwachung der Leistung

Hier sollte man die Konfigurationsdateien, Bildnamen und Einstellungen entsprechend Ihren spezifischen Bedürfnissen anpassen.

b. Zur vollständigen Bereitstellung der OmniView-Software

Mithilfe der am häufigsten verwendeten DevOps-Tools (GitHub Actions, Terraform, Kubernetes und Docker) wurde der GitHub Actions-Workflow so aktualisiert, dass er die Schritte enthält, die zum Erstellen, Verpacken, Bereitstellen und Überwachen der Anwendung erforderlich sind. Hier ist ein Beispiel für eine ideale Version für die Bereitstellung von Omniview
Leider hatte ich keinen Zugriff (ssh-Key), um die CI/CD-Pipeline auszuführen, Aber hier ist ein Stück Code, das zeigt, wie die Bereistellung sein sollte:

```yaml
deploy:
  name: Deploy to Kubernetes
  needs: build
  runs-on: ubuntu-latest

    steps:
    name: Checkout repository
    uses: actions/checkout@v3

    name: Set up Kubeconfig
    run: echo ${{ secrets.KUBECONFIG_DATA }} | base64 --decode > kubeconfig.yaml

    name: Deploy to Kubernetes
    run: kubectl apply -f kubernetes-deployment.yaml

    monitor:
    name: Monitor Performance
    needs: deploy
    runs-on: ubuntu-latest

    steps:
    name: Checkout repository
    uses: actions/checkout@v3

    name: Install kubectl
    run: |
    sudo apt-get update && sudo apt-get install -y kubectl

    name: Monitor Kubernetes Pods
    run: kubectl get pods


```

2.4.5 Monitoring und Logging:

Durch die Integration von Monitoring und Logging in den Entwicklungsprozess strebe ich an, nicht nur Bugs effektiv zu managen, sondern auch eine solide Grundlage für Systemverbesserungen und erfolgreiche Teststrategien zu schaffen. In Bezug auf Teststrategien, sind Beta-Tests und Canary-Tests verschiedene Teststrategien, die in der Softwareentwicklung eingesetzt werden, um die Stabilität, Zuverlässigkeit und Benutzerfreundlichkeit einer Anwendung zu gewährleisten.

3 Zusammenfassung

3.1 Einige Werkzeuge für agile Prozesse

- Github für Code-Repository
- Trello für das Ticket-Management
- Jira für das Bug-Management

- Confluence für die Dokumentation
- Github Actions

3.2 Detaillierte Beschreibung des DevOps-Prozesses

3.2.1 Fall der Builds

Wenn der Benutzer eine Pull Request genehmigt, startet Github Action das Script build.yml. Dieses Script führt folgende Aktionen durch: Ausführung von Unit-Tests Ausführung von Integrationstests Erstellung des Builds Bereitstellung des Builds Die Unit- und Integrationstests werden in einer isolierten Testumgebung ausgeführt. Der Build wird mithilfe eines automatisierten Build-Tools wie CMake (in unserem Fall), Maven oder Gradle generiert. Die Bereitstellung des Builds erfolgt in einer Produktionsumgebung.

3.2.2 Fall der Releases

wenn das Release erstellt wird, startet Github Action das Script release.yml. Dieses Script führt folgende Aktionen durch:

- Ausführung von Unit-Tests
- Ausführung von Integrationstests
- Erstellung des Builds
- Erstellung des Releases
- Bereitstellung des Endartefakts

Die Unit- und Integrationstests werden in einer isolierten Testumgebung ausgeführt. Der Build wird mithilfe eines automatisierten Build-Tools generiert. Das Release wird mithilfe eines Release-Management-Tools erstellt. Das Endartefakt wird in einer Produktionsumgebung bereitgestellt. Eventuell kann man auch ein Artefaktory Manager benutzen.

4 Schluss

In der zweiten Phase meines Projekts "DevOps"habe ich einen idealen DevOps-Workflow beschrieben, der mit dem Requirements Engineering beginnt und bis zur Auslieferung reicht. Nachdem wir die Problemstellung erläutert haben, können wir abschließend die Bedeutung von zwei Schlüsselkonzepten im DevOps-Kontext hervorheben: kontinuierliche Integration (CI) und kontinuierliche Bereitstellung (CD).
Die kontinuierliche Integration, die sich durch die automatische Ausführung von Unit- und Integrationstests bei jeder Codeänderung auszeichnet, spielt eine entscheidende Rolle. Dieser Ansatz ermöglicht eine schnelle Fehlererkennung und trägt so dazu bei, die Codequalität zu gewährleisten. Die schnelle Erkennung von Problemen reduziert das Risiko von anhaltenden Fehlern und fördert somit einen zuverlässigeren Entwicklungsprozess.
Die kontinuierliche Bereitstellung ist ein entscheidender Schritt, der die automatische Bereitstellung in der Produktion sofort nach Validierung des Builds ermöglicht. Diese Automatisierung beschleunigt den Markteinführungsprozess und reduziert die Zeitspanne zwischen Entwicklung und Verfügbarkeit für Endbenutzer erheblich. Infolgedessen trägt dieser Ansatz zu einer erhöhten Kundenzufriedenheit bei, da neue Funktionen und Korrekturen schnell und regelmäßig zur Verfügung gestellt werden.
Durch die Übernahme dieser Praktiken im Softwareentwicklungsprozess von Omniview können wir die Codequalität garantieren, die Time to Market verkürzen und die Kundenzufriedenheit verbessern.

Indem der Entwicklungs- und Bereitstellungsprozess von Omniview diesen idealen DevOps-Workflow übernimmt, kann er optimiert werden, um diese Ziele zu erreichen. Dies bietet eine leistungsfähigere und reaktionsfähigere technologische Lösung für die sich ändernden Anforderungen der Automobilbranche.