SDLC: Idealer Prozess

Tsomo Taf A.

22. Januar 2024

Inhaltsverzeichnis

1 Einleitung

Die vorliegende Dokumentation beschäftigt sich mit der Evolution des ab, beginnend beim Requirement Engineering bis zur Continuous Intergration/Delivery-Prozesses für die Desktop-Software ÖmniView". Dabei werden der Ist-Zustand, der Idealzustand und mögliche Sollzustände des Projekts analysiert und bewertet.

2 Ist-Zustand von OmniView

Die Ist-Analyse zeigt, dass OmniView als experimentelle GUI in C++ entwickelt wurde. Das Ziel ist es, die Diagnose von Autos zu erleichtern, indem es AW4.0-Mechanikern durch KI-Diagnose hilft, die von Auto-Intern OmniScopes erzeugten Daten zu verwalten, anzuzeigen und zu analysieren. Da es sich um ein Forschungsprojekt handelt, wurden die Ideen intern mit den Stakeholdern ausgearbeitet. Anforderungen für das Projekt werden in Form von Features und Verbesserungsvorschlägen durch Merge Requests eingebracht. Die Planung erfolgt über die Projekt- und Issue-Management-Tools von GitHub.

In Bezug auf die Design- und Architekturphase gibt das GitHub-Repository nicht ausreichend Aufschluss.

Die Implementierung erfolgt gemäß dem SDLC-Modell, wobei der Fokus auf der Verwendung von C++ liegt. Die Entwickler nutzen das Git-Repository aktiv für die Versionsverwaltung, wobei nach jedem Merge Request die CI-Pipeline "build.yamläutomatisch ausgeführt wird.

Die Testphase wird weder im Repository (in der CI/CD-Pipeline) nicht detailliert beschrieben, wodurch die Best Practices nicht berücksichtigt werden, selbst wenn diese Tests lokal ausgeführt werden.

Im aktuellen Continuous Delivery-Prozess von Omniview wird die Release.yaml-Datei genutzt, um neue Softwareversionen zu erstellen und die generierten ausführbaren Dateien als Version Assets herunterzuladen. Diese Artefakte, darunter Linux- und Windows-Ausführbare, entstehen im Zuge des Bauprozesses und werden in den entsprechenden Build-Verzeichnissen gespeichert. Nach Abschluss des Bauprozesses werden die ausführbaren Dateien in eine vorher konfigurierte URL hochgeladen, die durch die GitHub-Variablen festgelegt ist. Obwohl die derzeit zugängliche Version für den Kunden AW4null ein Proof-of-Concept ist, hat die Software bereits zwei Tags erhalten. Dies bedeutet, dass nach dem Tag "test"der Entwicklungszyklus mit einem neuen Sprint fortgesetzt wurde. Dabei wurden kontinuierlich neue Funktionen implementiert (Continuous Integration - CI), und nach Zufriedenheit wurde die Release.yaml ausgelöst. Der Kunde folgt dem in der README-Datei beschriebenen Verfahren, um die komprimierte Datei herunterzuladen und lokal auf seinem PC zu installieren

Für Feedback und Bug-Reporting hat man einerseits die Endbenutzer (AW4ull), die einen Proof-of-Concept der Software verwenden, den ich leider nicht manipulieren konnte. Dieser Schritt erfolgt jedoch wahrscheinlich manuell, das heiß die Stakeholder treffen sich, um Feedback/Bug der Software zu übermitteln oder neue Anforderungen vorzuschlagen. Andererseits haben die Entwickler, insbesondere die Mitglieder des Vereins Skunkforce e.V., die während der Entwicklung neuer Releases, insbesondere der Tags „Test“ und „Alpha“, eine Nachverfolgung von Bugs durch den Issue-Tracker von durchgeführt GitHub.

3 Idealzustand von OmniView

Im Idealzustand von OmniView wird ein umfassender agiler Softwareentwicklungsprozess implementiert. Die Anforderungen werden in enger Zusammenarbeit mit den Stakeholdern (DevTeam, DevOpsTeam, AW4null, OperatorsTeam) erfasst und in einem gut strukturierten Backlog priorisiert. Die Planung erfolgt in agilen Sprints, wobei die Iterationen kurzgehalten werden, um eine schnelle Anpassung an sich ändernde Anforderungen zu ermöglichen. Der Planungsprozess profitiert von Tools wie Microsoft Teams, Slack, Trello oder Jira als effiziente Taskmanager.

Die Design- und Architekturphase zeichnet sich durch eine klare, modulare Struktur aus. Eine saubere API-Dokumentation wird bereitgestellt, um eine einfache Integration und Nutzung für Entwickler zu gewährleisten. Es wird eine Microservices-Architektur angestrebt, die die Skalierbarkeit und Wartbarkeit der Anwendung verbessert.

In der Implementierungsphase wird auf leistungsstarke Frameworks wie Visual Studio Code (VsCode) und Qualitätsüberwachungstools wie SonarQube gesetzt. Diese gewährleisten einen sauberen und effizienten Code. Die Verwendung von jUnit als Testframework ermöglicht automatisierte Unit- und Integrationstests, während Github-Aktionen oder Jenkins den CI-Prozess effektiv steuern.

Die Testphase wird durch automatisierte Unit- und Integrationstests mit jUnit als Test-Framework unterstützt, die in die CI/CD-Pipeline integriert sind. Eine umfassende Testabdeckung wird angestrebt, um die Stabilität und Zuverlässigkeit der Software sicherzustellen. Die Bereitstellung der Software erfolgt über die Datei "Release.yaml", die durch Github-Aktionen automatisch ausgelöst wird, wenn eine neue Version gewünscht ist. Diese automatische Bereitstellung wird durch Docker, Kubernetes, Ansible und Terraform unterstützt, um eine effiziente, skalierbare und zuverlässige Bereitstellung zu gewährleisten.

Im Bereich Feedback und Bug-reporting setzen wir auf Tools wie Prometheus und Grafana, um eine klare und umfassende Rückmeldung von Benutzern zu erhalten. Diese Tools ermöglichen eine kontinuierliche Verbesserung und schnelle Fehlerbehebungen

Zusammenfassend wird im Idealzustand von OmniView ein agiler Entwicklungsprozess mit klarem Fokus auf Qualität, Skalierbarkeit und Benutzerfeedback implementiert. Dies schafft eine dynamische und adaptive Softwareentwicklungsumgebung. Omniview ist jedoch eine native Software mit einer sehr spezifischen Vision und konnte daher nicht alle dieser Best Practices berücksichtigen. Eine weitere Option für Continuous-Delivery könnte daher eine Website oder ein Artefakt-Manager sein, wo der Client die ausführbare Datei herunterladen könnte, ohne das Github-Repository offenzulegen

4 Sollzustand von OmniView

Im Sollzustand von OmniView werden prinzipiell die Anforderungen an den Continuous Integration/Continuous Deployment (CI/CD)-Prozess auf der Grundlage des idealen Zustands optimiert. Der Fokus liegt darauf, die Ausführungszeit der Pipeline zu minimieren, um eine schnellere Feedbackschleife zu ermöglichen. Dies wird durch die Integration von bedingten Ternäroperatoren und parallelen Ausführungsschritten erreicht, um den Prozess effizienter zu gestalten.

Die Design- und Architekturphase bleibt monolithisch und behält ihre Ausrichtung auf die Vision des Projekts bei. Die Einführung von Microservices wird nicht als geeignet erachtet und könnte zu unnötiger Komplexität führen. Diese Entscheidung hat Auswirkungen auf den Continuous-Deployment-Prozess, der sich weiterhin auf einen monolithischen Ansatz stützt. Hierbei wird jedoch die Verwendung von Docker als entscheidender Schritt eingeführt, um die Software portierbar zu gestalten und sie mit ihren Abhängigkeiten in einer isolierten Umgebung zu platzieren. Docker bietet eine konsistente Umgebung für die Ausführung der Anwendung auf verschiedenen Systemen und erleichtert die Skalierung.

Der Continuous-Delivery-Prozess wird optimiert, indem die Software über einen zentralen Artefaktmanager wie den Nexus Repository Manager bereitgestellt wird. Dies ermöglicht es, verschiedene Versionen der Software mit entsprechenden Zugangsrechten direkt von Nexus herunterzuladen. Kunden können die gewünschten Versionen einfach und sicher abrufen. Diese Methode gewährleistet nicht nur den einfachen Zugriff auf die Software, sondern bietet auch eine zentrale und verwaltbare Plattform für die Bereitstellung von Artefakten. Die ausführbaren Dateien sollen auf Nexus bereitgestellt und über eine URL zugänglich gemacht werden. Dadurch könnte der Continuous-Delivery-Prozess optimiert und der manuelle Aufwand minimiert werden.
In der Realität könnten alternative Methoden wie das Brennen von neuen Versionen auf leere CDs oder das Bereitstellen auf einem USB-Stick in Erwägung gezogen werden. Diese Lösungen erfordern jedoch erhebliche physische Ressourcen und bieten nicht die Skalierbarkeit und Verwaltbarkeit, die durch die zentralisierte Nexus-Plattform erreicht werden. Der Fokus liegt darauf, den Endbenutzern einen einfachen und sicheren Zugriff auf die Software zu ermöglichen und gleichzeitig die Verwaltung und Verteilung von Softwareversionen zu rationalisieren.

5 Schlussfolgerung

Das Projekt OmniView wurde erfolgreich abgeschlossen, und die dokumentierten Verbesserungsvorschläge bieten einen klaren Pfad zur Optimierung des SDLC-Prozesses für die Software. Die Dokumentation steht als Referenz für weitere Forschungen zur Verfügung und ist im THGA GitLab Repository verfügbar. Obwohl Die Metriken wie Time to Deployment und Fehlerrate aufgrund mangelnder Zugriffsrechte nicht ausgewertet werden konnten, bietet die theoretische Analyse dennoch eine solide Grundlage für zukünftige Entwicklungen.