THM
TECHNISCHE HOCHSCHULE MITTELHESSEN

STUDIUM PLUS
DUALES STUDIUM

Praxisphasenbericht SS 20

Thema:

Erweiterung und Portierung eines CI-/DevOps-Tools zur automatischen Erstellung von Branches und Build-Konfigurationen

Vorgelegt von:	Nico Borchardt Opferweg 11 35644 Hohenahr-Erda
Matrikelnummer:	5291652
Hochschulbetreuer:	Prof. Dr. Peter Kneisel
Fachbetreuer:	Herr Dipl.-Inf. Dominic Finke
Unternehmen:	Leica Microsystems

Eingereicht am: 12.10.2020

Leica
MICROSYSTEMS

Inhalt

Liste der Abkürzungen

API	Application Programming Interface
CI	Continuous Integration
Dev-Line	Development Line
HTTP	Hypertext Transfer Protocol
ID	Identifikator
JSON	JavaScript Object Notation
LAS X	Leica Aplication Suite X
LMS	Leica Microsystems
PBG	PerforceBranchGenerator
RegEx	Regular Expressions
REST	Representational State Transfer
SDK	Software Development Kit
XML	Extensible Markup Language

Abbildungsverzeichnis

1. Einleitung

1.1 Unternehmen

Leica Microsystems ist ein Hersteller für Mikroskope, sowie verwandter Produkte, dabei fokussiert sich LMS auf drei unterschiedliche Anwendungsbereiche. Life Science beschäftigt sich mit bildgebenden Methoden zur Analyse von Mikrostrukturen. Der Industry Bereich umfasst unteranderem Qualitätstests und forensische Untersuchungen. Das Medical-Segment stellt Medizinern Equipment bereit, um operative Eingriffe durchzuführen.

LMS hat sieben Niederlassungen weltweit mit dem Hauptsitz in Wetzlar, Deutschland. Das Unternehmen gehört seit 2005 zu dem US-amerikanischen Danaher Mischkonzern.

1.2 Leica Application Suite X

Die Leica Application Suite X ist ein Softwareprodukt und stellt eine Erweiterung zur vorherigen Leica Application Suite dar. Dieses Endprodukt kommt in den Bereichen von Life Science und Industry zum Einsatz, wo es mit jedem Leica Mikroskop und einer kompatiblen Kamera verwendet werden kann.

Die LAS X ist eine modulbasierte Software. Die meisten Features können optional, für einen bestimmtem Anwendungsbereich, hinzugefügt werden.

1.3 Continuous Integration

Continuous Integration ist ein Begriff, der einen permanent fortlaufenden Prozess in der agilen Softwareentwicklung beschreibt.[1] Das direkte Einfügen von neuen Codeabschnitten in ein bestehendes Projekt, um Transparenz und Konsistenz zu schaffen, steht im Mittelpunkt der CI. Das Detektieren von Fehlern wird durch die ständige Kompilierung der Software gewährleistet. Zur Realisierung dieses Prozesses werden verschiedene Softwarekomponenten benötigt, die ineinandergreifen. Die

[1] (Duvall, 2007)

Steigerung der Softwarequalität ist das primäre Ziel bei diesem Konzept. Die Aktualität und Relevanz spiegeln sich darin wieder, dass dieser Prozess ein zentraler Baustein bei der LAS X-Produktion ist. Hierbei wird ebenfalls das Continuous Delivery Konzept, aufbauend auf der CI, verwendet.

1.3.1 Perforce

Die Software Perforce Helix Core dient der Versionsverwaltung von Dateien und Verzeichnissen. Die Hauptaufgaben liegen dabei bei der Protokollierung, Wiederherstellung, Archivierung, Koordinierung und der gleichzeitigen Bearbeitung mehrerer Entwicklungszweige (Branches) eines Projekts. Die Main-Line ist eine durchgehend stabile Version des Projekts. Dev-Lines haben den Vorteil, dass dort Raum ist, um neue Features, Verbesserungen,... etc. vorzunehmen und zu testen. Schlussendlich können Dev-Lines mit der aktuellen Main-Line zu einer neuen stabilen Version verschmelzen.

1.3.2 TeamCity

Die Kernaufgabe eines TeamCity-Servers besteht darin, dass die CI gewährleistet ist. Ein Build-Prozess führt die Kompilierung und das Testen des vorhandenen Codes durch und wird sukzessiv pro Build-Agent ausgeführt. Ein wichtiger Bestandteil sind Build-Konfigurationen, Parameter, Abhängigkeiten, welche den Ablauf des Build-Prozesses bestimmen. Die Konfigurationen bestehen aus mehreren Schritten, was eine Kette von Aktionen bildet, die den Prozess definiert.

1.3.3 PerforceBranchGenerator

Der PBG ist eine bereits bestehende firmeninterne Software, die den Prozess der Softwareentwicklung unterstützen soll. Das Tool wurde von einem Leica Mitarbeiter in C# geschrieben und basiert auf dem .NET Framework. Aktuell lassen sich mit dem Programm einige Perforce-Operationen automatisieren, jedoch ist es noch nicht so weit, dass es regelmäßig von Entwicklern genutzt wird. Ein kritischer Punkt ist, dass das Tool nur auf Windows-Betriebssystemen ausgeführt werden kann, da es eine .NET Framework Applikation ist.

1.4 Motivation

Die Begriffe „Automatisierung“ und „Stabilisierung“ repräsentieren die Gründe für diese Arbeit am besten. Der aktuelle Stand ist, dass die meisten Prozesse manuell konfiguriert und gestartet werden müssen. Vor allem ist dies in der agilen Softwareentwicklung nicht wünschenswert. Ein immer wieder erneutes Anlegen von Konfigurationen für einen Perforce-Branch oder ein TeamCity-Build ist zeitraubend. Da diese Dienste im Mittelpunkt der täglichen Arbeit bzw. der Entwicklung stehen, macht es gerade dort Sinn, Optimierungen vorzunehmen und den Mitarbeitern Arbeit abzunehmen. Da das Tool beides kombiniert, ist es möglich, dass die Software regelmäßig verwendet wird. Zudem bietet dieses Thema viel Raum für eigene Ideen und Verbesserungen.

1.5 Zielsetzung

Ziel von diesem Projekt ist die Portierung, Erweiterung und Verbesserung der bestehenden .NET Applikation. Die Applikation soll zunächst auf .NET Core portiert werden.

In Fällen, wo bereits die Dev-Line existiert, welche neu angelegt werden soll, gibt es Probleme mit dem verbundenen Mapping. Probleme treten auch bei den Voraussetzungen zur Ausführung des PBG auf.

Als Neuerung soll ein erzeugter Branch mit allen zugehörigen Abhängigkeiten als Build-Konfiguration direkt an TeamCity übergeben werden. Diese Verbindung von dem Tool zu TeamCity muss noch geschaffen werden, ist aber möglich. Eine Schnittstelle für diese Überlegung stellt TeamCity schon bereit mit einer REST API zur Integrierung externer Applikationen und Interaktion mit Skripten.

Eine Abgrenzung ist notwendig, wenn es kein Eltern-Projekt gibt bzw. das Projekt, welches in Perforce gebrancht wurde, nicht in TeamCity existiert. In den meisten Fällen ist es so, dass die Projekte aus Perforce gleichermaßen auch in TeamCity vorhanden sind, somit hat jeder Branch/Build-Konfiguration ein klares Eltern-Projekt.

Ein Feature für die Erstellung eines komplett neuen TeamCity-Projekts würde somit die zeitliche Grenze überschreiten.

Daraus ergeben sich folgende Fragen:

- Ist eine Portierung der Software auf .NET Core möglich?
- Welche internen Prozesse des PBG müssen wie automatisiert werden?
- Wie können aus Perforce-Branches Build-Konfigurationen erzeugt werden?
- Wie müssen die Build-Konfigurationen überprüft werden?

1.6 Methodik

Mit Blick auf die Ziele und die damit einhergehenden Probleme, lassen sich drei unterschiedliche Bereiche aufzeigen:

1. **Portierung:** Die Software ist eine .NET Applikation und somit nur auf einem Windows Betriebssystem verwendbar. Durch eine Portierung auf die .NET Core Plattform, müsste sich der PBG auf einer Linux-Distribution und MacOS ausführen lassen. Die Portierung könnte von Microsoft bereits unterstützt werden. Als Überprüfung der Vorhersage, kann man eine Testumgebung auf einer Linux-Distribution aufsetzen und das portierte Tool ausführen. Die korrekte Funktionalität stellt das Kriterium für die Verifikation dar.

2. **Verbesserung:** Dieser Punkt beinhaltet die Verbesserung der Funktionsweise des Tools in Bezug auf die Erstellung neuer Dev-Lines. Ein verbessertes Konzept des PBGs sollte erkennen, dass es Dev-Lines bereits gibt und die Imports des Branch-Mappings aktualisiert werden müssen. Zudem muss der Anwender vor jeder ersten Ausführung der Software die Umgebungsvariablen für Perforce setzen. Implementierungen zur Überprüfung der aktuellen Umstände sollten beide Probleme erkennen und lösen.

3. **Erweiterung:** Eine Implementierung im PBG, die zur Kommunikation mit der TeamCity REST API dient ist notwendig. Durch HTTP-Anfragen der Software an TeamCity können Informationen kopiert und ausgetauscht

werden. Aufschluss darüber, ob es funktioniert, gibt das Ergebnis nachdem der PBG dementsprechende Befehle an die API übergeben hat und die Build-Konfiguration korrekt erstellt wurde. Zur Validierung werden Test-Cases lokal, sowie netzwerkübergreifend durchgeführt, um die korrekte Funktion des PBGs zu überprüfen.

Gültig für jeden Punkt ist die Methode des Experiments. Zu beobachten, wie sich die Software in einer aufgesetzten Umgebung verhält gibt Aufschluss darüber, wo Schwächen oder Fehler liegen. Vor allem bei Änderungen der Testumgebung oder der einzelnen Parameter. Ein Experiment in einer echten Umgebung ist zunächst zu vernachlässigen, da die Software unkontrolliert ist und Schäden hinterlassen kann. Deswegen ist der Versuch, die echte Umgebung nachzustellen, umso wichtiger, aber möglich.

Die Expertenbefragung ist die Basis, um neue Informationen zu gewinnen oder eigene Probleme zu lösen. Besprechungen mit kompetenten Mitarbeitern haben einen großen Anteil an dem Fortschritt in den einzelnen Themengebieten.

2. Theoretische Grundlagen

Im Folgenden werden Grundlagen erklärt, damit die spätere Umsetzung verstanden wird. Die Portierung des PBG baut auf der .NET-Plattformen auf, deshalb werden die grundlegenden Informationen beschrieben. Das Verständnis des Prinzips und des Aufbaus von TeamCity ist wichtig, um die Anforderungen an die neue Software zu verstehen. Zudem werden essenzielle Begriffe erklärt und in Zusammenhang gebracht. Mit der REST API wird ein weiteres Konzept beschrieben und deren Funktionsweise aufgezeigt. Das Ansprechen dieser Schnittstelle und die verfügbaren Operationen werden erklärt.

2.1. NET-Plattform[2]

Die von Microsoft entwickelte Softwareentwicklungsplattform wurde bereits im Jahr 2000 offiziell bekannt gegeben. Die Sprache ähnelt in Syntax und Aufbau der von Java. Ziel dieser Entwickler-Plattform ist es, eine flexible, moderne und plattformneutrale Umgebung zu schaffen für die Entwicklung und Ausführung von Applikationen. Es heben sich drei Frameworks aus dem Konstrukt heraus: .NET Framework, .NET Core und die Mono-Plattform. Aktuell ist eine Zusammenführung der genannten Produkte unter dem Namen .NET 5.0 von Microsoft geplant.

2.1.1 .NET Framework

Das .NET Framework ist ein Teil der .NET-Plattform. Eine wichtige Komponente ist die „Common Language Runtime“. Dort werden alle Programme ausgeführt, die mit diesem Framework entwickelt wurden. Die Bereitstellung von diversen Klassenbibliotheken und die Wahl zwischen verschiedenen Windows-basierten Anwendungstypen sind zwei Kernkomponenten, die das Framework offeriert. Einschränkungen gibt es, weil nur Applikationen erstellt werden können, die ein Windows-Betriebssystem unterstützen. Das Framework des PBGs ist aktuell noch das .NET Framework.

[2] (Dotnet-bot, n.d.)

2.1.2 .NET Core

Das .NET Core Framework ähnelt nur in den Grundzügen den anderen Teilen der .NET-Plattform. Vor allem unterscheidet es sich von dem monolithischen .NET Framework. .NET Core ist eine universelle Entwicklungsplattform, deshalb können Applikationen systemübergreifend und unabhängig von der Rechnerarchitektur entwickelt werden. Zusätzlich lässt sich noch sagen, dass Applikationen mit .NET Core wesentlich performanter sind als mit .NET Framework. Dies sind ausschlaggebende Punkte, um eine Portierung auf .NET Core zu begründen.

2.2 Der Aufbau von TeamCity

Die Navigationsstruktur von der TeamCity-Software ist ähnlich aufgebaut wie der Windows-Explorer. Einzelne Projekte können Sub-Projekte oder direkt TeamCity Build-Konfigurationen enthalten – vergleichbar mit Ordnern/ Sub-Ordnern/ Dateien.

Build-Konfigurationen verfügen über eine Vielzahl von Einstellungsmöglichkeiten und Abhängigkeiten zu anderen Konfigurationen. Diese Abhängigkeiten sind ein wichtiger Bestandteil der Herstellung und der Entwicklung der Software. Sie verweisen auf andere Konfigurationen, welche benötigt werden, um das Ausgangsprojekt herzustellen. Es wird zwischen zwei verschiedenen Arten unterschieden:

1. Snapshots: Diese legen fest, welche anderen Projekte involviert bzw. gebaut werden müssen, bevor das aktuelle hergestellt wird. Snapshots implizieren Artefakte, jedoch muss nicht für jeden Snapshot ein Artefakt vorliegen. Hinter jedem Snapshot steht eine Build-Konfiguration.
2. Artefakte: Dies sind Dateien, die aus Snapshot-Verweisen bereits gebaut und zentral persistiert wurden, um so die Performance zu verbessern. Ein Artefakt ohne Snapshot wird ignoriert.

Ein Projekt hat immer einen Namen und eine daraus teilweise entstehenden ID. Die ID ist essenziell für die Arbeit mit der Software, da diese den Speicherpfad wie folgt repräsentiert: „Projekt_SubProjekt_SubSubProjekt_..._Konfiguration“.

2.3 REST API

Eine API, auch Programmschnittstelle genannt, ermöglicht es komplett unabhängigen bzw. unterschiedlichen Systemen, miteinander zu kommunizieren. Speziell die REST (Representational State Transfer) API orientiert sich an den Paradigmen des World-Wide-Web. Die Nutzer bekommen eine einfache Möglichkeit, für eine Kommunikation in der Netzwerkumgebung. Dabei basiert der Kern dieser Schnittstelle auf den grundlegenden HTTP-Operationen. Diese Operationen sind:

- ➔ GET, DELETE, POST und PUT

Allgemein ist das Prinzip, dass Informationen in einem verteilten System, auf eine einfache Weise, abgerufen, sowie persistiert werden können. TeamCity stellt dem Nutzer auch eine REST API zur Verfügung. Mithilfe von HTTP-Anfragen können darüber Informationen über Builds, Build-Server,... etc. abgerufen werden. Es kann u.a. gezielt nach einzelnen Unterpunkten in der Build-Konfiguration gefragt werden. Die Ausgabe wird dann durch ein XML-Dokument dargestellt.

3. Realisierung

Im folgenden Kapitel wird die Vorgehensweise dargelegt, um die gesteckten Ziele zu verwirklichen. In 3.1 wird ein Lösungsweg aufgezeigt, der die Portierung einer Software ermöglicht. Die Verbesserungen, die benötigt werden, um das Branch-Mapping zu prüfen und die Umgebungsvariablen zu setzen, werden in 3.2 erklärt. Die neue Software für TeamCity und deren Funktionsweise wird in 3.3 beschrieben.

3.1 Portierung

Für die Portierung existiert eine Möglichkeiten der Umsetzung seitens Microsoft.

➔ Das Erstellen einer Applikation direkt über .NET Core, jedoch muss geprüft werden wie oder ob der Code optimiert werden muss, um portierbar zu sein.

Hierfür werden Visual Studio mit .NET Core 3.0 oder höher und der Portability-Analyzer[3] benötigt. Der Analyzer zeigt auf, wie viel Prozent des vorhandenen Codes nach .NET Core portiert werden können. Anschließend muss aus dem aktuellen Projekt eine .csproj-Datei erstellt werden. Normalerweise ist die Datei bereits vorhanden. Danach muss die csproj.-Datei in den Stil des SDK migriert werden. Daraufhin wird sie in der Projektmappe geöffnet und der Wert der Eigenschaft „<TargetFramework>“ auf „netcoreapp3.0“ gesetzt werden. Abschließend müssen die entstandenen Fehler manuell beseitigt werden.

Assembly Name	Target Framework	.NET Core,Version=v3.1
ConfigurationGenerator	.NETCoreApp,Version=v3.1	100
PerforceBranchGenerator	.NETFramework,Version=v4.7.2	100

Abbildung 1- Ergebnis des Portability-Analyzers

3.2 Verbesserung des Branchvorgangs

Abstrakt betrachtet ist eine Dev-Line eine Kopie der Main-Line. Im Zuge dessen verfügt diese Kopie auch über alle Eigenschaften (Imports,… etc.) der Main-Line. Der PBG ist eine Konsolenanwendung, welche Dev-Lines erstellt. Dabei kann die Erstellung aus einer Main- oder Dev-Line erfolgen.

[3] (Dotnet-bot, n.d.)

3.2.1 Aktualisierung des Branch-Mapping

Imports eines Branches sind definierte Abhängigkeitspfade zu anderen Branches/Projekten. Wenn eine Dev-Line existiert und diese erneuert werden sollte, kam es zu Problemen. Erneuern bedeutet, dass die Dev-Line nochmal erstellt wird und dabei die Eigenschaften der Main-Line und der bestehenden Dev-Line kombiniert. Bisher wurden dabei die neuen Imports hinzugefügt, aber die alten nicht gelöscht, was zu Kollisionen führte.

Die Implementierung von Methoden, welche die Stelle innerhalb des Branch-Mapping finden mit Hilfe von RegEx, war nötig. Bei erfolgreicher Suche werden die schon vorhandenen Imports gelöscht. Dabei muss darauf geachtet werden, dass sonst nichts innerhalb der String-Variable verschoben wird. Nach dem Löschen werden die neuen Abhängigkeitspfade in dem richtigen Format eingefügt und gespeichert.

3.2.2 Umgebungsvariablen

Bei jeder ersten Ausführung des PBG müssen drei Umgebungsvariablen für den Perforce-Port, -User, Client gesetzt werden. Um diesen Prozess zu automatisieren, prüft eine Methode bei dem Programmstart, ob diese schon existieren.

Ausgehend davon, dass sie nicht existieren, erscheint eine Konsoleneingabe für den Nutzer, um die entsprechenden Werte festzulegen. Nachdem die Eingabe bestätigt wurde, werden diese Werte mit einer Methode als Konsolenbefehl verpackt und ausgeführt. Bei jedem weiteren Einsatz des PBG werden die Variablen nicht abgefragt.

3.3 Configuration-Generator

Das im folgende beschriebene Programm (Configuration-Generator) ist in C# geschrieben und liegt zusammen mit dem bereits vorhandenen PBG in einem Projektverzeichnis. Beide Projekte basieren nach der Portierung auf der .NET Core Plattform. Die Programmierung erfolgte mit Hilfe von Visual Studio. Das Ziel ist es, passend zu einer in Perforce neu erstellten Dev-Line, eine passende Build-Konfiguration zu erstellen.

3.3.1 Status Quo

Ein Perforce-Branch kann als Projekt/Build-Konfiguration in TeamCity vorliegen, jedoch ist das nicht immer der Fall. Eine Prüfung muss erfolgen, um sicherzustellen, dass der aktuelle Perforce-Branch in TeamCity existiert. Der Projektname, welcher als Ausgangspunkt betrachtet wird, wird aus Perforce übergeben. Die Prüfung auf TeamCity-Seite geschieht über eine „GET"-Abfrage der bestehenden Projekte mit Hilfe der REST API. Die Antwort der API ist ein XML-Dokument, welches alle existierenden Projekte mit deren Namen, IDs und weiteren Sub-Projekten beinhaltet. Bei erfolgreicher Suche nach dem Eltern-Projekt, wird dies im nächsten Schritt als Vorlage zur Erstellung des TeamCity-Projekts genutzt. Das gefundene Projekt wird mittels einer „POST"-Operation kopiert, jedoch mit einem vorher festgelegten Namen und der daraus resultierenden ID.

Bei einer misslungenen Suche wird ein Hinweis ausgegeben und das Programm geschlossen, da dies die Abgrenzung ist, die in 1.5 festgelegt wurde.

3.3.2 Abhängigkeiten

Der kopierte Projektbaum wird nun auf der Suche nach Build-Konfigurationen durchlaufen. Eine Liste der Konfigurationen eines Projektes wird durch eine „GET"-Abfrage zur Verfügung gestellt. Bei einem Treffer werden zunächst die Abhängigkeiten dieser Konfiguration betrachtet, da diese die größte Priorität genießen.

Zuerst werden die Snapshots geprüft, danach folgen die Artefakte. Der Vorgang für beide Abhängigkeiten unterscheidet zwischen der existenziellen Prüfung und der inhaltlichen Aktualisierung. Das XML-Dokument mit den Snapshots/Artefakten der aktuellen Build-Konfiguration steht im Mittelpunkt. In diesem Dokument werden nacheinander die einzelnen Abhängigkeiten und deren Eigenschaften betrachtet.

Snapshots

Nachdem die Snapshot-ID geprüft wurde, um zu verifizieren, dass der Snapshot existiert, wird dieser kopiert. Bei dem Kopiervorgang muss unterschieden werden, ob die hinter dem Snapshot liegende Konfiguration in dem gleichen Projektbaum wie das Ausgangsprojekt liegt. Trifft dieser Fall ein, wurde dieser Snapshot bereits kopiert und der Vorgang wird

übersprungen. Liegt der Snapshot in einem externen Projektbaum, wird dort ein neues Sub-Projekt für die Kopie des Snapshots angelegt. Alle zukünftigen Snapshots in diesem externen Projektbaum werden in diesem neuen Sub-Projekt gespeichert. Der Name und die ID werden direkt übernommen und die restlichen Eigenschaften setzen sich aus dem ursprünglichen Snpashot zusammen.

Unabhängig von der Lage, muss jeder Snapshot mit seinen neuen Eigenschaften in die Kopie des Ausgangsprojektes korrekt eingebunden werden. Die Build-Konfiguration darf nicht auf den anfänglichen Snapshot verweisen, sondern auf die neu angelegte Kopie.

Das Editieren von Abhängigkeiten einer Build-Konfiguration ist via REST API grundsätzlich möglich. Dennoch traten Fehler auf, bei dem Versuch dies zu tun. Die Hypothese bestand, dass für eine Aktualisierung des Snapshots zuerst der momentane Zustand mit der „GET"-Operation abgerufen werden muss. Das erhaltene XML-Dokument so bearbeitet wird, dass alle Eigenschaften auf den neu kopierten Snapshot ausgelegt sind. Der alte Snapshot kann dann mit einer „DELETE"-Operation aus der Konfiguration gelöscht werden und der neue mit der „POST"-Operation gespeichert werden. Diese Hypothese wurde falsifiziert (3.3.3 Konflikt mit der REST API). Nach Gesprächen mit den zuständigen Mitarbeitern seitens JetBrains, lag keine funktionierende Lösung vor.

An dieser Stelle ist der Vorgang für einen Snapshot abgeschlossen und der rekursive Aufruf rückt in den Mittelpunkt. Die kopierte Abhängigkeit ist nichts anderes als eine weitere Build-Konfiguration, welche wiederum eigene Abhängigkeiten besitzt. Diese müssen genau so durchlaufen, geprüft und kopiert werden, deshalb wird der beschriebene Vorgang wieder aufgerufen. Eine Liste, die bereits bearbeitete Snapshots beinhaltet, verhindert eine Dopplung von Operationen. Einen weitere Liste gibt am Ende aus, welche Snapshots fehlen. Der Aufruf endet, wenn alle notwendigen Build-Konfigurationen abgearbeitet wurden.

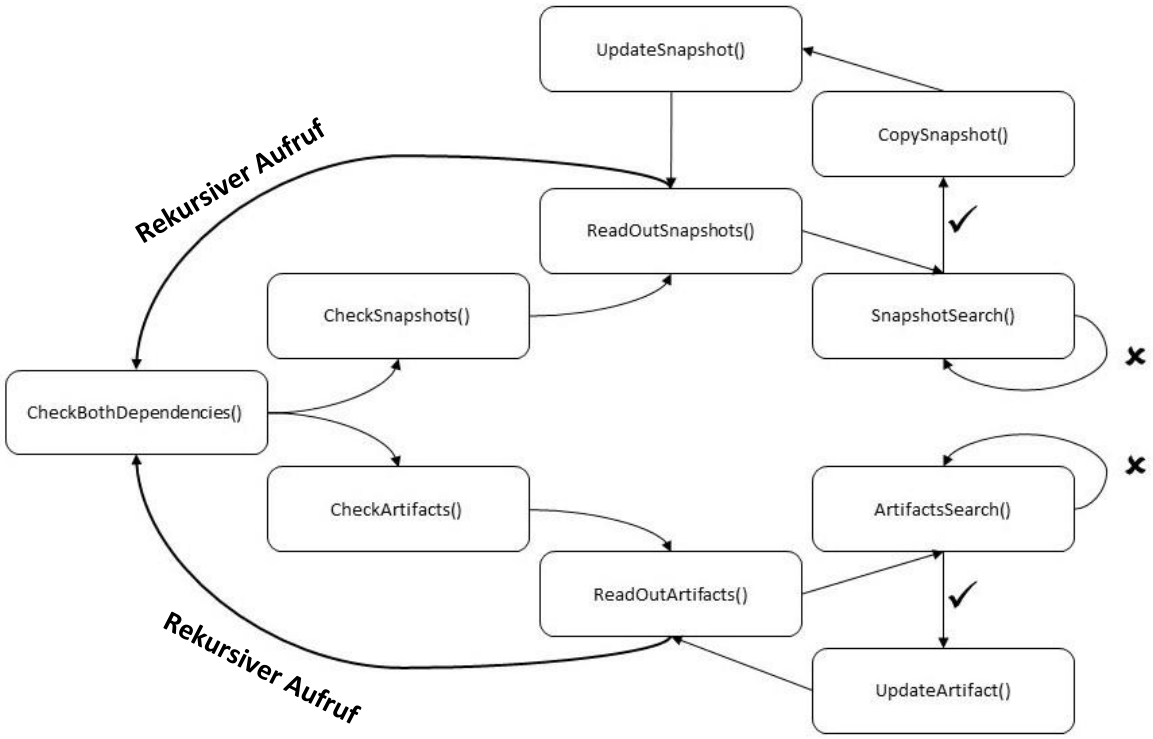

Abbildung 2 - Rekursiver Aufruf

Artefakte

Mit der „GET“-Operation liegt ein XML-Dokument mit allen Artefakt-Informationen vor, die zur aktuellen Build-Konfiguration gehören. Durch dieses Dokument werden die einzelnen Artefakte und deren Eigenschaften gefunden. Der Aufbau der Artefakte ist unterschiedlich zu Snapshots. Das Kopieren von Artefakten ist nicht nötig, lediglich das Prüfen, ob ein zugehöriger Snapshot in der selben Konfiguration vorhanden ist. Tritt der Fall ein, dass ein Artefakt ohne Snapshot vorliegt, wird dies am Ende des Programms dem Nutzer mitgeteilt. Bei einer erfolgreichen Prüfung folgt anschließend eine Aktualisierung, sodass auf den neuen Snapshot verwiesen wird.

Da das granulare Bearbeiten von Abhängigkeiten nicht möglich ist, bestand zunächst erneut die Hypothese, das aktuelle Artefakt als XML-Dokument abzurufen, zu bearbeitet und wieder zu senden. Dieses Verfahren ist annähernd deckungsgleich mit dem des Snapshots, deswegen wurde auch diese Hypothese falsifiziert (3.3.3 Konflikt mit der REST API).

Diese Prozedur wird für jede relevante Build-Konfiguration angewandt. Der rekursive Algorithmus endet, wenn keine weiteren Artefakte mehr vorliegen.

3.3.3 Konflikt mit der REST API

An dem Punkt, an dem das Editieren bzw. die Neuerstellung von Abhängigkeiten nötig ist, kommt es zu einem Problem. In der offiziellen Dokumentation[4] gibt es keine genauen Angaben darüber, ob solch eine Operation möglich ist. Nach ausführlicher Recherche folgten Experimente, in denen getestet wurde, ob das bereits beschriebene Prinzip funktioniert (3.3.2 Abhängigkeiten). Eine Fehlermeldung des Remote-Servers mit dem „Error-Code 400“, was repräsentativ für einen „Bad Request“ steht, erschien wiederholt. Ein Blick in die Log-Dateien des TeamCity-Servers brachte keine Klarheit.

Die Lösungsansätzen:

Das bei einer „GET“-Abfrage erhaltene XML-Dokument für eine Abhängigkeit ist immer die Vorlage für eine „POST“-Operation in diesem Rahmen.

- Der erste Versuch war das kopierte, aber neu zusammengesetzte XML-Dokument mit einer „POST“-Operation an die Snapshot-/ Artefakt-Sammlung der aktuellen Build-Konfiguration zu schicken. Nach jedem Fehlversuchen wurde das XML-Dokument verändert, um diverse Fehler vorzubeugen, dennoch scheiterte dieses Experiment.
- Der folgende Versuch beinhaltete das Senden einer lokalen XML-Datei, die vorher korrekt präpariert wurde, sodass die Formatierung und Inhalt keine Fehler produzieren konnten. Erneut trat der Fehler bei der „POST“-Operation auf.
- Die Alternative zu dem Senden eines XML-Dokuments ist eine JSON-Datei im selben Stil zu konfigurieren. Dieser Versuch scheiterte erneut an der „POST“-Operation der Datei.

Das Problem war weder ausführlich noch genau beschrieben. Ausschließlich der Error-Code und eine grobe Auflistung an Ursachen waren zu entnehmen.

[4] (TeamCity, n.d.)

Ein JetBrains Mitarbeiter versicherte, dass die gewollte Operation verfügbar sei. Nachdem das Problem offengelegt und besprochen wurde, wurde keine Lösung gefunden. Alle empfohlenen Lösungsansätze wurden bereits ausreichend getestet. Die Hypothese des JetBrains-Experten besteht darin, dass die übertragenen Daten in kleinsten Teilen verfälscht werden, da die Anfrage von einem eigenen Tool ausging.

Auswirkung dieses Problems ist, dass die Abhängigkeiten korrekt vorhanden sind, jedoch nicht auf deren neue Kopie verweisen. Diese Arbeit muss im Anschluss manuell erledigt werden.

3.4 Evaluation

Die abgeschlossenen Umsetzungen können alle in Zusammenhang gebracht werden. Die Portierung bildet die Basis der Benutzung. Der PBG und der neue ConfigurationGenerator verbinden die Perforce- und TeamCity-Softwares. Diese Verbindung wurde neu geschaffen.

Bei der Realisierung der Portierung (3.13.1 Portierung) wurde bestätigt, dass eine auf .NET Framework basierenden Software zu der .NET Core-Plattform möglich ist. Die Funktionsweise des PBGs wurde mit einer Linux-Distribution geprüft und verifiziert. Der Effekt ist, dass die Nutzungsmöglichkeit und Nutzerkreis erweitert wurde.

Die Funktionalität der implementierten Automatisierungen in Perforce (3.2 Verbesserung des Branchvorgangs) ist verifiziert, indem Unit-Test aufgesetzt wurden. In den Tests wurden verschiedene Szenarien durchgespielt und deren Ausgang betrachtet. Schließlich wurde jeder aufgesetzte Test erfolgreich abgeschlossen. Die bestehenden Imports des Branches werden korrekt aktualisiert. Die Umgebungsvariablen werden überprüft und fehlerfrei für den derzeitigen Benutzer gesetzt. Die anfangs beschrieben Probleme wurden somit gelöst.

Die neu entwickelte Software erweitert den bestehenden PBG. Der ConfigurationGenerator nimmt eine XML-Datei von dem PBG entgegen und verarbeitet diese Informationen automatisch. Grundlegende Arbeit, die viel Zeit kostet, wird dem Benutzer abgenommen und automatisiert. Der Kopiervorgang der Build-Konfigurationen läuft korrekt ab und liefert ein valides Ergebnis. Der ConfigurationGenerator ist variabel programmiert und somit innerhalb von TeamCity flexibel anwendbar. Einzig das in „3.3.3 Konflikt mit der REST API“ beschriebene Problem verhindert, dass die Zielsetzung vollständig erfüllt werden kann.

4. Zusammenfassung / Ausblick

In den abschließenden Kapiteln wird die Arbeit zusammengefasst und inhaltlich bewertet. Zur Bewertung gehört ein kritischer Blick auf die Methodik und die entstandene Problematik. Enden wird das Kapitel mit einer Beschreibung für zukünftige Arbeiten innerhalb dieses Themengebiets.

4.1 Summary

The three mentioned topics (porting, improvement and extension) and their tasks were worked on and a valid solution was found for each issue. The basic idea was to incorporate the term of “automation” more into the principle of “continuous integration”. The Perforce- and TeamCity-Software represented this principle in the current context. Depending on the existing limits of the used software and the defined problem in 1.5, the solutions and ideas were implemented.

By porting the software to the .NET-Core platform, a higher compatibility is achieved and thus allows more users to access the current software. The problem, that e.g. Linux or MacOS users could not access the tool is fixed. In general, this implementation is useful and helpful.

The implementations in the PBG ensure that errors are automatically and foresightedly avoided. The use of the software is made easier for the user by independently checking and creating the variables which are required. This also prevents the tool from running into an invalid state. The chosen solution is implemented logically and correctly.

The new ConfigurationGenerator automates the creation of a project in TeamCity. The tool takes over the process of copying all required components and a large part of the necessary adjustments of the copied content. However, based on the resulting problem of the TeamCity REST API, changing snapshot or artifact dependencies within individual build-configurations is not yet working. An update of the current REST API could improve the needed functionality and finally solve the problem. Nevertheless, the process was fundamentally automated as far as possible. The whole project can be rated as a success.

4.2 Kritische Bewertung

Der Fokus der Methodik dieser Arbeit lag eindeutig auf der experimentellen Arbeitsweise. Zur Gewinnung quantitativer Informationen waren Befragungen bzw. Besprechungen mit Mitarbeitern essenziell.

Der experimentelle Ansatz hat klare Stärken bei einem Software-Projekt, da direkt geprüft werden kann, wie das Programm sich verhält oder welche Auswirkungen die Ausführung hat. Generell wird somit ein effektives Troubleshooting ermöglicht. Als Schwäche hat sich herausgestellt, dass ein hoher Aufwand nötig ist, um eine Testumgebung zu planen und umzusetzen. Obwohl eine künstliche Umgebung geschaffen wurde, die der echten Umgebung sehr ähnelt, lässt dies nicht repräsentativ auf die Realität schließen.

Durch diese Experimente entsteht eine geplante und zu jeder Zeit kontrollierte Situation, was dazu führt, dass die interne Validität hoch ist. Systematische Fehler werden vermieden und die gelieferten Ergebnisse sind glaubwürdig.

Die Methodik der Mitarbeiterbefragung hat Schwächen gezeigt Das Projektthema bzw. das Problem musste dem Gegenüber immer genau erklärt werden, damit verstanden wurde, worum es sich handelt. Die Informationen waren dennoch qualitativ hochwertig, jedoch mussten diese gewonnene Informationen stellenweise noch weiter aufbereitet oder tiefer analysiert werden, was Zeit kostete. Ein Beispiel hierfür ist die Problematik aus 3.3.3 Konflikt mit der REST API, da dort neue Informationen von einem JetBrains-Mitarbeiter benötigt wurden. Der Austausch mit dieser Person war schwerfällig und langsam, da die genannten Kritikpunkte eintraten.

Trotzdem war die Befragung von Experten eine der Hauptquellen, um gute Informationen oder Ansätze zur Problemlösung zu erhalten.

4.3 Ausblick

Mit der Nutzung der TeamCity-Schnittstelle ist eine Grundlage für künftige Problemlösungen oder Automationen geschaffen worden. Die nächsten Schritte des Projekts können weitere Erweiterungen bzw. Neuerungen sein.

An erster Stelle steht das Lösen des Problems aus 3.3.3 Konflikt mit der REST API.

Vor allem muss ein Ziel sein, dass die Benutzerfreundlichkeit verbessert wird, sodass die Nutzung attraktiver wird. Reine Funktionalität bringt nichts, wenn der Nutzungsaufwand zu groß bzw. zu umständlich ist. Dazu könnte eine grafische Benutzeroberfläche mit jeglichen Einstellungsmöglichkeiten nützlich sein. Eine Möglichkeit, um dies zu verwirklichen ist das Grafik-Framework „Windows-Presentation-Foundation“, welches auch auf der .NET Core-Plattform anwendbar ist.

Bei dem PBG kann die Erstellung von Dev-Lines auch auf Release-Branches ausgeweitet werden. Diese Idee wäre nützlich und in der Umsetzung sehr gut machbar. Zudem können mehr Informationen aus Perforce für TeamCity genutzt werden. Beispielsweise kann darauf geachtet werden, ob in Perforce die Imports auch gebrancht werden. Auf diese Art und Weise kann der ConfigurationGenerator verbessert werden, um noch intelligenter zu handeln.

4.4 Vision

Das Ziel des Projekts bleibt es, dass das Tool regelmäßig benutzt werden soll. Um dieses große Ziel zu erreichen, müssen die in 4.3 Ausblick möglichen Teilziele noch realisiert werden. Schließlich soll das Projekt so weit gebracht werden, dass es in der täglichen Entwicklung zum Einsatz kommt. Ein klares Ende ist schwer zu definieren, weil viele Möglichkeiten für eventuelle Erweiterungen bzw. Neuerungen bestehen.

Kommentiertes Literaturverzeichnis

Dotnet-bot. „NET-Dokumentation“ [WWW Document]. URL https://docs.microsoft.com/de-de/dotnet/standard/ (Zugegriffen 26. August 2020).

Duvall, Paul M., Steve Matyas, und Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Pearson Education, 2007.

Humble, Jez, und David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education, 2010.

TeamCity. „REST API - Help | TeamCity“ [WWW Document]. URL https://www.jetbrains.com/help/teamcity/rest-api.html (Zugegriffen 27. August 2020).

Wingerd, Laura. *Practical Perforce*. O’Reilly Media, Inc., 2005.