Automatisierung von Systemtests im industriellen Umfeld

Continuous Integration Umgebungen werden in der Regel im Software-Entwicklungszyklus für die kontinuierliche Integration der Software eingesetzt. In diesem Artikel zeigen wir, dass sich solche Systeme auch hervorragend für die Realisierung von automatisierten Testinfrastrukturen im industriellen Umfeld eignen. Im vorliegenden Anwendungsfall wird damit eine fast vollständig automatisierte System- und Akzeptanztestumgebung von Softwareprodukten zur Überwachung und Steuerung von Messtechnik-Sensoren erreicht.

Anja Kellner, Martin Kropp | martin.kropp@fhnw.ch

Das Unternehmen, für das die Test-Automatisierungslösung entwickelt wurde, ist ein weltweit führender Hersteller von Mess- und Sensortechnik in der Schweiz. Immer wichtiger werden dabei für Kunden Komplettlösungen, welche einen zunehmenden Anteil an Softwareanwendungen zur Steuerung und Überwachung der Anlagen beinhalten. Die Software wird dabei weltweit in verschiedenen Sprachen und auf unterschiedlichen Windows-Betriebssystemen ausgeliefert.

Die hohen Qualitätsansprüche, welche das Unternehmen an seine Hardware stellt, gelten analog auch für die entwickelte Software, so dass das Testen der Software einen entsprechend hohen Stellenwert einnimmt.

Aktuell werden die Tests auf System- und Akzeptanztestniveau (Black-Box-Tests) durch manuelle Ausführung der zu testenden Applikation mittels umfangreichen schriftlichen Testspezifikationen ausgeführt. Die Testergebnisse werden ebenfalls manuell erfasst und dokumentiert. Die zu testende Applikation liegt in allen Fällen nur als Binär-Datei vor. Der Quellcode der Applikation wird bei diesen Tests nicht berücksichtigt.

Aufgrund der stetig grösser werdenden Software-Produktpalette wird das manuelle Testen immer zeitaufwendiger und damit auch kostspieliger. Aus diesem Grund sollen die Tests der Software in Zukunft weitestgehend automatisiert ausgeführt werden können. In diesem Beitrag beschreiben wir die Konzeption und den Aufbau einer kompletten Testinfrastruktur, die eine weitgehend automatisierte Durchführung von System- und Akzeptanztests ermöglicht.

Zielsetzungen

Durch eine vollständig automatisierte Testinfrastruktur soll in erster Linie der enorme Aufwand für die Systemtests gemindert werden. Zusätzlich werden damit folgende Ziele verfolgt:

- *Nightly-Tests* sollen eine automatisierte Ausführung der Tests über Nacht erlauben. Die sonst ungenutzten Rechnerressourcen können damit sinnvoll eingesetzt werden. Die Tests können manuell, über einen zeitlichen Trigger oder auch durch eine neue Version der zu testenden Applikation angestossen werden.
- *Multi-Plattform Tests* sollen es erlauben, die zu testenden Applikationen auf mehreren Betriebssystemen und in unterschiedlichen Internationalisierungen automatisiert zu testen.
- Da die Software nur in Kombination mit Hardware (Messgeräte) eingesetzt wird, müssen auch *"Real-World" Tests* durchgeführt werden. Dies bedeutet, dass die Tests immer mit realen Messgeräten durchgeführt werden. Hierfür soll die notwendige Infrastruktur aufgebaut werden.
- Die automatisierte *Archivierung der Testergebnisse* soll es erlauben, zu jeder Zeit die Ergebnisse von bereits ausgeführten Tests einzusehen. Dies ist insbesondere im Fall von Servicefällen wichtig, für die geprüft werden muss, mit welchem Ergebnis die Tests zuvor ausgeführt worden sind.
- Auch die *Reproduzierbarkeit* in Kombination mit der Verwendung von realer Hardware ist ein weiteres Ziel, mit der die Qualität der Tests verbessert werden soll.
- Schliesslich soll auch die Test-Implementierung historisiert werden, so dass die Entwicklung der Tests jederzeit rekonstruiert werden kann. Hierfür ist die Verwaltung des Testcodes in einem *Versionsverwaltungssystem* vorgesehen.

Das gesamte Testkonzept lässt sich in vier Teilbereiche unterteilen:

- Konzept der Testautomatisierung
- Definition der Infrastruktur
- Definition des Testprozesses
- Umsetzung des Testkonzeptes

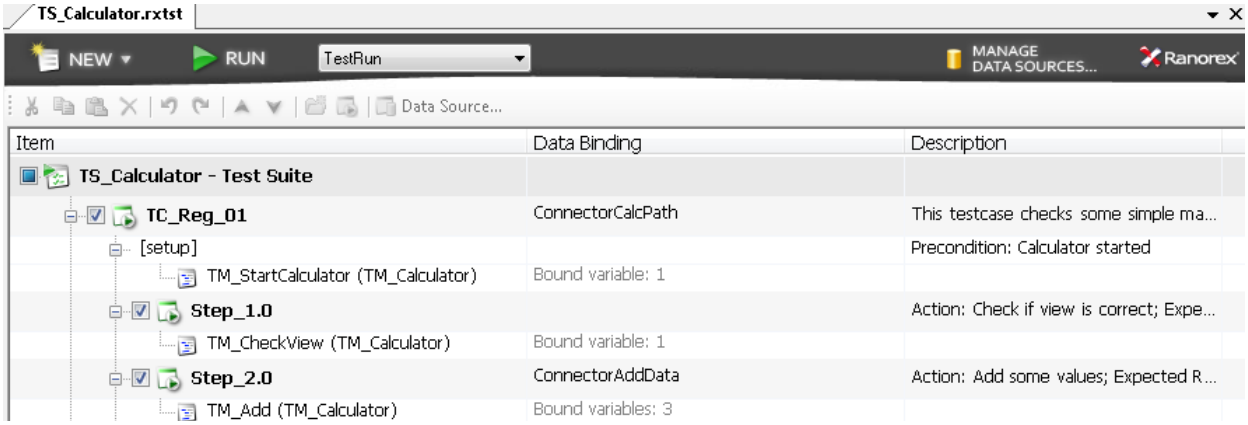

Abbildung 1: Screenshot: Ausschnitt Testsuite-View (Beispiel: Windows Calculator)

Testautomatisierung

Bei den durchzuführenden Tests handelt es sich um Systemtests, d.h. reine Black-Box-Tests, bei welchen die zu testende Applikation zur Ausführung kommt. Für die Automatisierung dieser Tests bieten sich idealerweise GUI-Testautomatisierungs-Werkzeuge an. Diese ermöglichen es, auf dem GUI der zu testenden Applikation Aktionen auszuführen und danach den Zustand zu prüfen. Die meisten dieser Werkzeuge verfügen auch über einen Capture-Replay Mechanismus, der es ermöglicht, Bedienabläufe aufzuzeichnen, als Skript anzupassen und ablaufen zu lassen.

Bei der Evaluation verschiedener GUI-Testwerkzeuge hat sich das Tool Ranorex [Ran] als das am besten geeignete durchgesetzt. Dieses Produkt ist insbesondere in Bezug auf die Objekterkennung der graphischen Steuerelemente auf der GUI der zu testenden Applikationen den anderen evaluierten Produkten überlegen.

Die Objekterkennung von Ranorex basiert auf XPath[1]-ähnlichen Pfaden. Jedes Objekt auf der GUI der zu testenden Applikation ist über einen solchen Pfad eindeutig identifizierbar. Diese Pfade werden über das sogenannte *Object-Repository* den entsprechenden GUI-Objekten zugeordnet.

Ranorex *Testmodule* bilden die möglichen Aktionen ab und werden in C# implementiert. Die neueste Version Ranorex 3.x verfügt zudem über eine sogenannte *Testsuite-View* (siehe Abb. 1), die es erlaubt mittels Drag&Drop Testmodule in beliebiger Reihenfolge auszuführen. Die Testmodule sind in der linken „Item" Spalte in einer Baumstruktur mit dem Präfix „TM_" referenziert.

Infrastruktur

Die Infrastruktur für die Testautomatisierung soll möglichst vor äusseren nicht-beeinflussbaren Einflüssen geschützt sein. Aus diesem Grund empfiehlt sich ein separates Netzwerk für die Testautomatisierung. Die Abbildung 2 zeigt das im konkreten Anwendungsfall realisierte Testnetzwerk:

- *File-Exchange-Server*: Mit diesem können Daten zwischen dem Firmennetzwerk und dem Automatisierungsnetzwerk ausgetauscht werden. Insbesondere dient dieser Server dazu, die Installationspakete der verschiedenen Produkte und Produktversionen zu verwalten.
- *Proxy*: Dieser dient als Schnittstelle nach aussen (Zugriff auf Lizenzserver, Internet und das Versionsverwaltungssystem).
- *Rack mit Automation-PC*: Die so genannten Racks erfüllen zwei zentrale Funktionen:
- Zum einen enthalten sie die Hardware (Messgeräte, Bussysteme,...), für die Ausführung der *„Real-World"-Tests*. Ausserdem sind sie mit einem weiteren Computer ausgerüstet, auf dem die Tests effektive zur Ausführung kommen. Dieser Rechner verfügt über zwei Netzwerkkarten: die eine ist mit dem Automatisierungsnetzwerk verbunden und die andere mit den Geräten im Rack (privates Netzwerk des Racks). Der Automation-PC ist als sogenannter *headless* Computer aufgesetzt, verfügt also weder über einen Bildschirm, noch Maus oder Tastatur.
- *Remote-PC*: Dieser PC dient dazu, per Remote-Verbindung auf die Automation-PCs zuzugreifen. Dies kann nötig sein, um den Status des Rechners zu prüfen, aber auch für die Entwicklung und Ausführung von Testskripten.

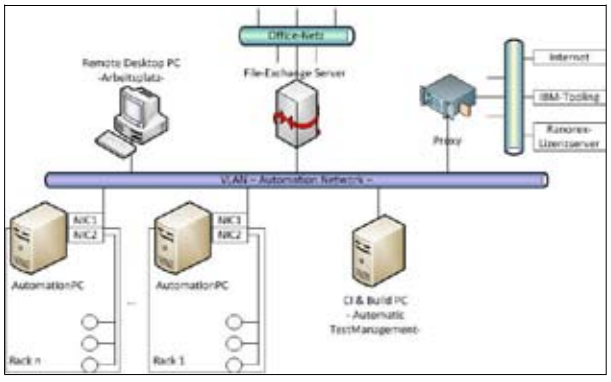

Abbildung 2: Netzwerk für Testautomatisierung

1 http://www.w3.org/TR/xpath/

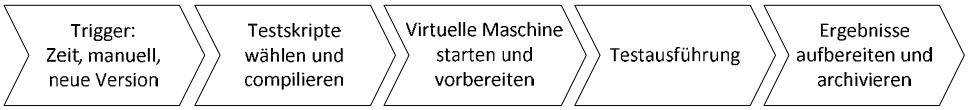

Abbildung 3: Realisierter Testprozess. Nur der erste Schritt wird auf dem CI-Server ausgeführt, die nachfolgenden auf einem Automations-PC.

- *CI- & Build-PC*: Auf diesem Computer ist die Continuous Integration Umgebung installiert. Er ist somit der Master, der die Ausführung der automatisierten Tests auf den Automation-PCs koordiniert und die zu testenden Applikationen auf die Automation-PCs verteilt.

Für die Testcode-Verwaltung wird das Versionsverwaltungssystem *ClearCase UCM* von IBM eingesetzt [IBM-CC], welches bei dem Unternehmen schon seit längerem im Einsatz ist. Es kann somit auf den Testcode zu jeder beliebigen Produktversion der zu testenden Applikation zugegriffen werden.

Um die Software-Produkte des Unternehmens in verschiedenen Sprachen und Betriebssysteme testen zu können, kommen bereits vorkonfigurierte virtuelle Maschinen zum Einsatz. Dies ermöglicht die Simulation der unterschiedlichsten System-Setups *(Multi-Platform-Tests)*. Zusätzlich bietet dieser Ansatz die Möglichkeit, die virtuelle Maschine gemeinsam mit den Testergebnissen so zu archivieren, dass damit auch die Umgebung der Tests selbst reproduzierbar ist. Die virtuellen Maschinen werden automatisiert auf die Automation-PCs der Racks geladen und gestartet.

Testprozess

Der gesamte Testprozess, vom Holen des benötigten Test-Quellcodes bis zur Archivierung der Testreports, soll vollständig automatisiert werden. Dabei soll die automatisierte Ausführung sowohl manuell als auch ereignisgesteuert ausgelöst werden können. Ereignisse können z.B. zu einer bestimmten Uhrzeit (Nightly-Build), oder durch das Vorliegen einer neuen Produktversion auf dem File-Exchange-Server ausgelöst werden. Der Prozess soll sowohl für alle vorhandenen, aktuellen Produkte als auch für eine ausgewählte Produktversion angestossen werden können.

Für die Automatisierung der Systemtests eignet sich eine Continuous Integration (CI) Umgebung wie sie üblicherweise für die Software-Integration zum Einsatz kommt, da beide Prozesse einige Gemeinsamkeiten aufweisen:

- Der benötigte Testcode muss in beiden Fällen aus einem Versionsverwaltungssystem geholt werden.
- Das Kompilieren des Testcodes ist beiden gemein.
- In beiden Fällen werden die Software-Tests ausgeführt. Während dies bei Software-Entwicklungsprojekten jedoch in der Regel Unit- und Integrationstests sind, werden im gegebenen Fall die Systems-Tests auf den bereits fertigen und automatisiert installierten Applikationen ausgeführt.
- Als Resultat wird in beiden Fällen ein Testreport erstellt.

Die Ausführung des gesamten Testprozesses (gemäss Abb. 3) wird über ein Continuous Integrati-

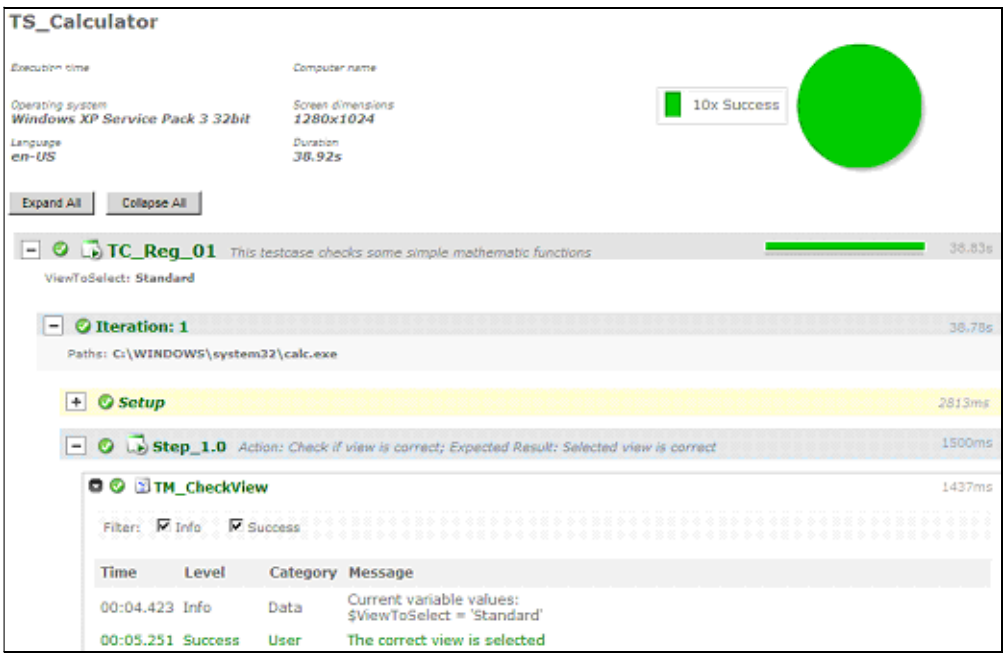

Abbildung 4: Screenshot: Ausschnitt eines erfolgreichen Testreports (Beispiel: Windows Calculator)

Abbildung 5: Jenkins Projektübersicht

on Werkzeug gesteuert. Die konkrete Umsetzung wird nachfolgend beschrieben:

1. Trigger: Die Test-Ausführung wird auf dem CI-Server entweder durch einen zeitlichen Trigger (*Nightly-Tests*), manuellen Trigger oder durch eine neue Version der zu testenden Software auf dem File-Exchange-Server angestossen.
2. Testskripte: Der Testcode wird aus dem Versionsverwaltungssystem geholt. Mittels MS-Build wird der Testcode kompiliert und zu ausführbaren Tests.
3. Virtuelle Maschine: Die benötigte virtuelle Maschine wird im gewünschten Zustand gestartet, so dass die Voraussetzungen für die Tests erfüllt sind.
4. Ausführen der Tests: Die Ausführung der Tests findet in einer virtuellen Maschine statt, auf welche über ein Remote-Excecution-Werkzeug zugegriffen wird.
5. Ergebnis Aufbereitung: Nach Ablauf aller Tests werden die Testergebnisse für ein geeignetes Reporting aufbereitet. Die Abbildung 4 zeigt am Beispiel von Ranorex, wie die Testergebnisse innerhalb von Jenkins verlinkt und mittels Dashboard angezeigt werden können. Zusätzlich werden die Testergebnisse im Versions-Verwaltungssystem abgelegt.

Umsetzung des Testkonzeptes mittels Continuous Integration

Zur Steuerung und Ausführung des gesamten Test-Prozesses wird ein Continuous Integration Werkzeug eingesetzt. Dabei kommt das Open Source Produkt Jenkins zum Einsatz [Jen]. Ausschlaggebend für dessen Wahl sind die grosse Funktionalität, die sehr gute Integration mit dem verwendeten Versionsverwaltungssystem, sowie die sehr gute Erweiterbarkeit mittels Plugins. Es existiert eine grosse Entwicklergemeinde, die sich sehr aktiv an der Weiterentwicklung des Systems beteiligt.

Jenkins selbst wird typischerweise auf einem eigenen Server installiert und betrieben. Jedes Projekt wird in Jenkins als separater *Job* definiert. Jeder Job kann separat konfiguriert werden und so für jedes Projekt der individuelle Build-Prozess definiert werden. Hierfür steht in Jenkins ein sehr komfortables Web Interface zur Verfügung.

Die Abbildung 5 zeigt einen Ausschnitt einer Projektübersicht von Jenkins für eine Umgebung mit 5 Projekten. Die Übersicht zeigt durch entsprechend eingefärbte Ballons für jedes Projekt an, ob der letzte Build-Durchlauf erfolgreich war, sowie die Stabilität des Build-Prozesses über die letzten

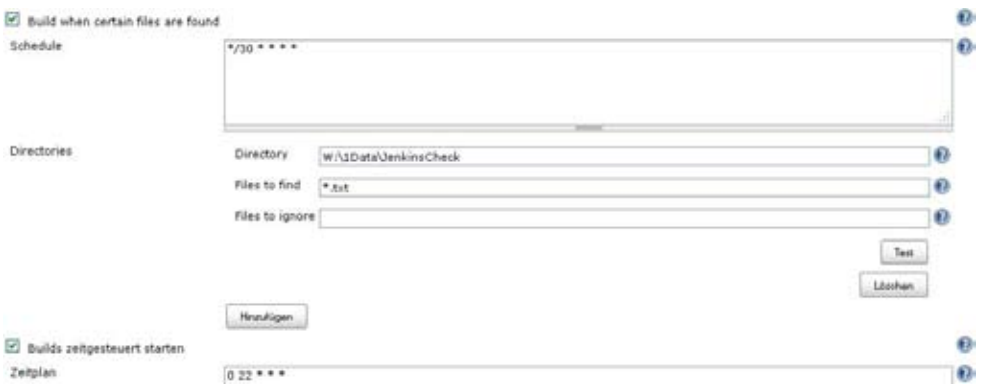

Abbildung 6: Build Trigger Konfiguration

Abbildung 7: Konfiguration des Build- und Test-Prozesses

5 Durchläufe mit Hilfe eines „Wetter"-Symbols in der Kolonne „W".

Die Abbildungen 6 bis 8 zeigen exemplarisch wie mittels Konfiguration in Jenkins der zuvor beschriebene Testprozess schrittweise umgesetzt wird.

Abbildung 6 zeigt die beiden Optionen zum Auslösen des Build-Prozesses. Der dateibasierte Trigger prüft alle 30 Minuten, ob eine bestimmte Datei (in diesem Fall ein beliebigesTextdokument) in einem ebenfalls spezifizierten Verzeichnis vorhanden ist (Hinweis: dies kann durch das zusätzlich installierte Plugin „Files Found Trigger" erreicht werden). Der zeitgesteuerte Trigger sorgt dafür, dass das Projekt mindestens jede Nacht um 0:22 Uhr gebildet wird.

Die Abbildung 7 zeigt die Konfiguration zur Ausführung des eigentlichen Build- und Test-Prozesses. Jenkins erlaubt die Konfiguration von beliebig vielen Einzelschritten. In Abbildung 7 sehen wir, dass zunächst die MSBuild-Datei zur Kompilation des Projektes ausgeführt wird. Anschliessend wird die vorkonfigurierte virtuelle Maschine auf dem Automation-PC gestartet und danach mittels Batch-Skript die Testausführung angestossen.

Die im Testprozess beschriebene Archivierung und Veröffentlichung der Testergebnisse wird in Jenkins mit Hilfe sogenannter „Post-Build" Aktionen konfiguriert wie dies in Abbildung 8 dargestellt ist.

Um die vollständige Testausführung während der Nacht überhaupt erst zu ermöglichen, sollte die Testausführung möglichst parallel und gleichzeitig auf den verschiedenen Automation-PCs durchgeführt werden. Für die gleichzeitige parallele Ausführung der Tests bietet Jenkins das Master/Slave-Konzept [Jen2].

Dieses Konzept erlaubt es mittels Jenkins-Konfiguration weitere Rechner als sogenannte Knoten in das System aufzunehmen. Dazu wird auf dem gewünschten Rechner ein Jenkins-Service gestartet, der es erlaubt diesen als Knoten in die CI-Umgebung aufzunehmen. Nun kann für jedes Projekt dezidiert angegeben werden, auf welchem Knoten das Projekt gebildet und ausgeführt werden soll. Wird für ein bestimmtes Projekt der Trigger aktiv und stellt fest, dass das Projekt ausgeführt werden soll, so wird die komplette Ausführung an den entsprechenden Knoten delegiert wie dies in Abbildung 9 dargestellt ist. Im gegebenen Fall werden die Automation-PCs aus den Racks als Knoten registriert, sodass diese parallel Tests ausführen können.

Abbildung 8: Archivierung und Veröffentlichung

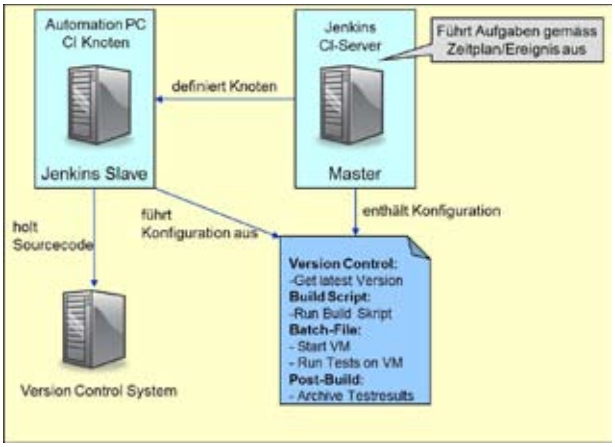

Abbildung 9: Jenkins Master/Slave Konzept

Erste Erfahrungen und Ausblick

Mit dem beschriebenen Testkonzept ist es möglich, sehr viele der bisher manuell ausgeführten Tests automatisiert auszuführen. Durch Nightly-Tests können die sonst ungenutzten Rechnerressourcen über Nacht sinnvoll genutzt werden. Die Analyse der Testergebnisse muss allerdings noch manuell vorgenommen werden, da sonst eventuell auch Fehler in ein Bug-Tracking-System eingetragen werden würden, die nichts mit den Testergebnissen, sondern mit dem gesamten Prozess zu tun haben. Beispiele solcher Fehler sind das Fehlschlagen eines Builds, eine falsche oder fehlerhafte virtuelle Maschine. Der Aufwand der Analyse ist jedoch vertretbar, wenn er mit dem Aufwand manueller Tests verglichen wird.

CI-Umgebungen scheinen auch den Anforderungen für automatisierte System- und Akzeptanztests gerecht werden zu können. Eine Umsetzung mit diesen Systemen ermöglicht es, dass die Test-Quellcodes immer zur Laufzeit kompiliert werden können. Dies stellt sicher, dass immer die aktuellste Version des Testcodes verwendet wird. Zusätzlich ist es möglich, eine Umgebung für die Tests zu schaffen, in der auch Hardware und unterschiedliche Betriebssysteme verwendet werden können

Der bisherige Einsatz hat gezeigt, dass die Ausführung der Tests mittels Remote-Commandline Werkzeugen in der virtuellen Maschine verbessert werden kann. Nach Beendigung der Tests liefert die Remote-Anwendung immer eine erfolgreiche Testausführung zurück, auch im Falle eines Test-Failures. Das tatsächliche Testergebnis kann nur durch Analyse des Test-Reports ausgelesen werden. Hier wäre es denkbar, auch die virtuelle Maschine an sich als Slave zu registrieren und das Build-Projekt in mehrere Teile aufzuteilen, die nacheinander auf jeweils unterschiedlichen Slaves ausgeführt werden. Dies bedeutet allerdings, dass die virtuelle Maschine entsprechend vorbereitet werden müsste.

Referenzen

[IBM-CC] IBM Homepage: http://www-01.ibm.com/software/awdtools/clearcase/, 15.08.2011

[Jen] Jenkins, Continuous Integration: http://www.jenkins-ci.org/, 15.08.2011

[Jen2] Kohsuke Kawaguchi, Distributed Builds: https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds, 15.08.2011

[Ran] Ranorex Homepage: http://www.Ranorex.com, 15.08.2011