# Inhaltsverzeichnis

# I Abkürzungsverzeichnis

| Abkürzung | Langschreibweise |
|---|---|
| **BS** | Betriebssystem |
| **API** | Application Programming Interface |
| **IDE** | Integrated Development Environment |
| **IT** | Informationstechnik |
| **JDK** | Java Development Kit |
| **JSON** | JavaScript Object Notation |
| **QA** | Quality Assurance |
| **SLES** | SUSE Linux Enterprise Server |
| **SVN** | Subversion |
| **UI** | User Interface |
| **UML** | Unified Modeling Language |
| **VM** | Virtuelle Maschine |

# 1 Einleitung

## 1.1 Problemstellung

„Wer immer tut, was er schon kann, bleibt immer das, was er schon ist.“ [1] – Henry Ford (1863-1947), Gründer des erfolgreichen Automobilherstellers Ford Motor Company[2].

Auch heute noch spiegelt dieses Zitat die Realität in vielen Bereichen der Unternehmenswelt wieder. Parallelen lassen sich beispielsweise im Bereich des Prozessmanagements erkennen. Ein Unternehmen, welches hier nur auf seine etablierten Arbeitsabläufe vertraut, lässt möglicher Weise das Potential für signifikante Zeit- und Kostenersparnisse ungenutzt. Dabei ist die Wahrnehmung eines solchen Optimierungspotentials besonders im Sektor der Informationstechnologien von Bedeutung. Aufgrund des sich technologisch schnell weiterentwickelnden Arbeitsumfeldes und des steigenden Konkurrenzdrucks in diesem Bereich ist die Durchführung von Optimierungsmaßnahmen notwendig, um am Markt erfolgreich zu bestehen. In diesem Zusammenhang kann die Senkung der eigenen Prozesskosten ein effektives Mittel zur Erreichung dieser Zielstellung sein. Um aber die Prozessstrukturen des eignen Unternehmens zu verbessern, sind eine umfassende Analyse, Modellierung und Optimierung sowie ein anschließendes Monitoring der durchgeführten Maßnahmen notwendig.[3]

Auch die dotSource GmbH (dotSource) aus Jena, welche sich auf die Bereiche E- und Social-Commerce spezialisiert hat, sieht sich den zuvor beschrieben Anforderungen gegenüber. Bereits seit ihrer Gründung im Jahr 2006 behauptet sich die dotSource mit der Konzeption, Erstellung und Erweiterung von Online-Shop-Lösungen gegenüber ihren Mitbewerbern und nimmt im Moment Platz 17 des E-Commerce Agenturrankings des Bundesverbandes digitale Wirtschaft E. V. ein.[4] Um auch in Zukunft in diesem Marksektor konkurrenzfähig zu bleiben, werden innerhalb der dotSource bestehenden Prozesse kontinuierlich auf das Potential für mögliche Optimierungen geprüft. Einen Ansatz hierfür bildet die Projektarbeit mit dem

[1] [Mon09], S. 163
[2] [Dai14]
[3] [Har14]
[4] [Bun14]

Thema „Automatisierte Erstellung und Konfiguration dynamischer virtueller Maschinen“ von Sebastian Benda (2014). In dieser wurde damit begonnen, die Prozesse für Erstellung, Verteilung und Aktualisierung virtueller Maschinen (VM) innerhalb der dotSource zu automatisieren. Zu diesem Zweck wurden verschiedene Softwarelösungen evaluiert und die Entscheidung zugunsten des Einsatzes der Open-Source-Software Vagrant in Verbindung mit dem Konfigurationsmanagementsystem Puppet getroffen. Vagrant wird dabei zur automatisierten Erzeugung virtueller Maschinen und Puppet zur Herstellung eines bestimmten Softwarezustandes auf dem erstellten Systems eingesetzt.

Dieser Prozess konnte bereits grundlegend für die Konfiguration virtueller Maschinen in der Entwicklungsabteilung für PHP-Projekte realisiert und eingeführt werden. Um die Prozessoptimierung abschließen zu können, muss die Umsetzung des erstellten Systems fortgeführt und eine Möglichkeit zur Bewertung der durchgeführten Maßnahmen geschaffen werden. Auch die Ermittlung offener Leistungspotentiale des Systems sowie möglicher Problemstellungen für den langfristigen Einsatz spielen dabei eine Rolle.

## 1.2 Ziel und Aufbau dieser Arbeit

Das Ziel dieser Arbeit ist es, ein Prozessmodell für die Verwendung virtueller Maschinen innerhalb der dotSource GmbH zu entwickeln, das bestehende Konfigurationssystem anhand dieses Modells weiterzuentwickeln und hinsichtlich seiner Effektivität zu bewerten.

Um zunächst das Verständnis für den Umgang mit den verwendeten Technologien zu verbessern, werden in Kapitel 2 die technischen Grundlagen von Vagrant und Puppet überblicksartig erläutert. Dabei wird sowohl auf deren Funktion im vorliegenden Projekt als auch die konkrete Handhabung eingegangen.

Für die Entwicklung des Modells werden grundlegende Techniken des Prozessmanagements angewendet. Aus diesem Grund wird im Kapitel 3 ein Überblick zu dessen Motivation gegeben und auf den konkreten Nutzen im vorliegenden Projekt eingegangen. Zur weiteren Vertiefung dieser Thematik werden theoretische Grundlagen hinsichtlich der Bestandteile und Darstellung von Prozessmodellen erörtert. Innerhalb der anschließenden Modellentwicklung werden alle Tätigkeit, die im Rahmen eines beliebigen Szenarios im Umgang mit virtuellen

Maschinen von Relevanz sind, abstrahiert und in den Prozessverlauf eingeordnet. Auf der Basis der zuvor ermittelten theoretischen Kenntnisse entsteht so das finale Prozessmodell.

Im nachfolgenden Kapitel 4 wird die praktische Weiterführung des Systems erläutert. Hier wird in separaten Abschnitten auf die vorliegenden und neu ermittelten Anforderungen der einzelnen Abteilungen eingegangen, der aktuelle Stand vorgestellt und auf die konkrete technische Realisierung Bezug genommen. Des Weiteren werden in diesem Zusammenhang abschließend Problemstellungen, die im Rahmen der Umsetzung festgestellt und als relevant für die Projektfortführung eingeschätzt wurden, dargelegt und erläutert.

Nachdem ein passendes Modell entwickelt und die geplanten Umsetzungen durchgeführt wurden, bildet die Auswertung der Ergebnisse den Kern des Kapitels 5. In diesem werden alle Aspekte der ermittelten Einsatzszenaren hinsichtlich des bisher etablierten Vorgehens und der durchgeführten Automatisierung betrachtet. Das erstellte Modell bildet dabei die Basis für zeitliche Messungen und eine objektive Bewertung im Rahmen der entwickelten Testszenarien. Innerhalb einer abschließenden Gegenüberstellung der erfassten Ergebnisse wird die Wirtschaftlichkeit der durchgeführten Maßnahmen bewertet und ein erster Ausblick zur Zukunft der betrachteten Prozesse gegeben.

Abschließend werden im Rahmen des Fazits noch einmal wichtige Zwischenergebnisse zusammengefasst und die daraus abgeleiteten Erkenntnisse vorgestellt.

# 2 Technische Grundlagen

Nachdem im vorangegangenen Kapitel die Ausgangssituation für diese Arbeit erläutert und auf ihre Zielstellungen eingegangen wurde, sollen nachfolgend Informationen zu technischen Aspekten des Projektes bereitgestellt werden. Eine zentrale Rolle nehmen dabei die Anwendung Vagrant und das Konfigurationsmanagementsystem Puppet ein. Die Evaluierung und Auswahl dieser Technologien aus einer Reihe von Alternativen wurde bereits in der Projektarbeit „Automatisierte Erstellung und Konfiguration dynamischer virtueller Maschinen“ durchgeführt.[5] Diese liefert außerdem eine Reihe von Informationen über technische Grundlagen dieser Arbeit. Aus diesem Grund soll im Rahmen dieses Kapitels nur ein Überblick zu den verwendeten Schlüsseltechnologien gegeben werden.

## 2.1 Vagrant

Die Open-Source-Software Vagrant nimmt eine zentrale Rolle bei der Umsetzung des Projektes ein, welches dieser Arbeit zugrunde liegt. Die Idee zu dieser Technologie stammt vom Softwareentwickler Mitchell Hashimoto, dem Gründer der Firma HashiCorp, welche seit 2012 Vagrant weiterentwickelt.[6]

Vagrant bietet die Möglichkeit, virtuelle Maschinen mittels eines gegeben Templates, der sogenannten Base Box (Box), automatisiert zu generieren und zu verwalten. Die Box stellt dabei nur eine rudimentäre virtuelle Maschine mit dem Betriebssystemkern und wenigen systemabhängigen Basiskonfigurationen zur Verfügung. Individuelle Einstellungen der VM werden über eine einfache Textdatei, welche als Vagrantfile bezeichnet wird, bereitgestellt. In diesem können Konfigurationen wie der Name und die Quelle der verwendeten Box, die Benennung der VM oder auch die Synchronisation gemeinsamer Ordner zwischen Gast- und Hostsystem definiert werden. Für die Durchführung weiterführender Hard- und Software-Konfigurationen wird im Vagrantfile außerdem das Zusammenwirken mit weiteren Technologien ermöglicht. Software-Konfigurationen werden dabei durch einen sogenannten Provisioner durchgeführt. Dieser ist neben Software auch für das Management von Benutzern, Rechten, Diensten und weiteren Funktionen des Betriebssystems zuständig.[7] Im vorliegenden

[5] [Sbe14]
[6] [Has13]
[7] [Kla14]

Projekt wird das Konfigurationsmanagementsystem Puppet als Provisioner eingesetzt, welches im nachfolgenden Abschnitt 2.2 näher erläutert wird. Die Anbindung von Puppet, inklusive der Übergabeparameter für bestimmte Funktionen des Systems, ist im folgenden Auszug des Vagrantfiles dargestellt.

```
config.vm.provision "puppet" do |puppet|
   puppet.manifests_path     = "puppet/manifests"
   puppet.module_path        = ["puppet/modules/puppetForge"]
   puppet.manifest_file      = "site.pp"
   puppet.options            = "--verbose --debug"
end
```

Die Umsetzung für Vorgaben der virtuellen Hardware wird durch die verwendete Virtualisierungssoftware, welcher im Zusammenhang mit Vagrant als Provider bezeichnet wird, durchgeführt. Im vorliegenden Projekt wird VirtualBox von Oracle als Provider eingesetzt, welches eine entsprechende Konfigurationsschnittstelle (API) bereitstellt, auf die innerhalb des Vagrantfiles zugegriffen wird.[8] Der folgende Quellcode-Auszug zeigt, wie der Zugriff auf den VirtualBox-Befehlssatz innerhalb des Vagrantfiles realisiert werden kann, um beispielsweise die Anzahl der CPUs, den verfügbaren RAM oder die 3D-Beschleunigung des Gastsystems zu konfigurieren.

```
config.vm.provider "virtualbox" do |vb|
     vb.gui  = true
     vb.name = "Vagrant_PHP"
     vb.customize ["modifyvm", :id, "--ioapic", "on",
                                    "--cpus", "4",
                                    "--memory", "8192",
                                    "--graphicscontroller", "vboxvga",
                                    "--accelerate3d", "on",
                                    "--draganddrop", "bidirectional",
                                    "--clipboard", "bidirectional",
                                    "--ioapic", "on",
                                    "--vram", "128",
                                    "--hwvirtex", "on"]
```

[8] [Mic13], S. 6

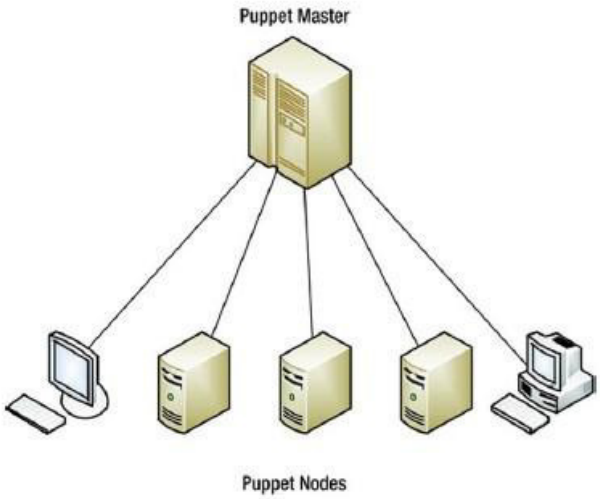
Durch die Kombination dieser unterschiedlichen Technologien ist es mit Vagrant möglich, virtuelle Maschinen auf der Basis feingranularer Konfigurationsbestandteile individuell zu erstellen. Wie im vorliegenden Projekt können so beispielsweise schnell Entwicklungsumgebungen für spezifische Anforderungen generiert werden.

## 2.2 Puppet

Wie im vorangegangenen Abschnitt bereits erwähnt, wird die softwareseitige Konfiguration der virtuellen Maschinen durch den Provisioner Puppet durchgeführt.

Zentrale Aufgaben eines Provisioners sind Installation, Konfiguration und Aktualisierung der Softwarekomponenten auf einem Zielsystem. Im Fall von Puppet wird dies durch die Verwendung von Puppet-Language, auch Puppet-Skript genannt, realisiert. Mit Hilfe dieser deskriptiven Konfigurationssprache wird der Zielzustand eines Systems beschrieben. Puppet selbst ist ein Interpreter, der die Beschreibungen liest und sie in Konfigurationsbefehle des zugrundeliegenden Betriebssystems umsetzt.[9]

**Abbildung 1: Puppet Master-Node-Struktur**




**Quelle: [Spe11], S. 3**

In der Regel wird Puppet, wie in Abbildung 1 dargestellt, in einer Client-Server-Struktur verwendet, wobei der zentrale Puppet-Master die Beschreibung der Zielzustände

[9] [Kla14]

verschiedener Systeme an eine beliebige Anzahl Puppet-Agents verteilt, die diese dann umsetzen.[10]

Im vorliegenden Projekt wird kein Puppet-Master verwendet, da mittels Vagrant auf dem Hostsystem vorhandene Puppet-Skripte auch direkt für die Konfiguration des Systems der erzeugten VM verwendet werden können.

Die Beschreibungen der Konfigurationen in Form von Puppet-Skript basiert immer auf den drei Grundelementen Ressource, Class und Node.
Die Basis aller Konfigurationen bildet die Deklaration von Ressourcen. Jede Konfigurationsoperation wird durch eine bestimmte Ressource repräsentiert. Dies kann beispielsweise die Ausführung eines Kommandozeilenbefehls durch eine Exec-Ressource oder auch das Kopieren einer Datei mittels File-Ressource sein. Neben diesen beiden stellt Puppet noch eine Vielzahl weiterer Ressourcen-Typen für spezifische Konfigurationsaufgaben bereit.[11] [12]
Im nachfolgenden Quellcode-Auszug ist eine File-Ressource mit vier Key-Value-Paaren dargestellt, welche als Attribute der Ressource bezeichnet werden. Jeder Ressourcen-Typ stellt dabei ein individuelles Set von Attributen zur Verfügung, um die jeweilige Konfiguration vollständig abbilden zu können. Funktion dieser Beispiel-Ressource ist es sicherzustellen, dass im Pfad /mnt/ ein Verzeichnis mit dem Namen network existiert, für welches die durch mode deklarierten Zugriffsrechte gelten. Die Abarbeitung der Ressource soll aber nur dann stattfinden, wenn das Paket cifs-utils erfolgreich installiert wurde.

```
#creates the folder for the network drive
file { 'networkDrive':
     path     => '/mnt/network',
     ensure   => 'directory',
     mode     => '0777',
     require  => [Package['cifs-utils']],
}
```

[10] [Jam11], S. 2
[11] [Pup14]
[12] [Dan13], S. 7 ff.

Alle weiteren Puppet-Sprachbestandteile existieren um, die Verwendung von Ressourcen für den Anwender bequemer und flexibler zu gestalten. So werden Klassen dazu genutzt, um Ressourcen in Gruppen zu organisieren, welche ein gemeinsames Ziel verfolgen, um so größere Konfigurationseinheiten zu bilden.[13] Meist werden mehrere dieser Klassen wiederum in Modulen zusammengefasst, welches dann die Installation und Konfiguration komplexer Softwarekomponenten, wie beispielsweise eines Datenbankservers, repräsentiert.

Die umfangreichste Organisationsform für Puppet-Ressourcen stellen Nodes dar. Diese bilden Systeme mit bestimmten Rollen ab, für die eine spezifische Konfiguration erforderlich ist, die wiederum durch die Zuordnung bestimmter Sets von Klassen bzw. Modulen umgesetzt wird. Sie werden bevorzugt dann eingesetzt, wenn Puppet in der eingangs beschriebenen Client-Server-Architektur betrieben wird, um so einzelne Maschinen individuell zu konfigurieren.[14]

Die Abarbeitung der mittels Ressourcen definierten Einstellungen findet bei Puppet nicht zwingend in der Reihenfolge ihrer Deklaration statt, um so auch parallele Konfigurations-schritte zu ermöglichen.[15] Dieser Umstand spielt eine große Rolle bei der Erstellung von Puppet-Skripten, da für die Definition jeder Ressource eine genaue Kenntnis über deren Einfluss auf andere Ressourcen bestehen muss, um diese mittels der entsprechenden Attribute abbilden zu können.

[13] [Lab14]
[14] [Joh13], S. 29
[15] [Lab14]

# 3 Konzeption und Modellierung

In diesem Kapitel wird das Vorgehen zur Schaffung des Prozessmodells für die Verwendung virtueller Maschinen in der dotSource GmbH erläutert. Dabei wird sowohl auf theoretische Grundlagen der Prozessmodellierung eingegangen als auch auf die konkrete Anwendung der vorgestellten Verfahren für die Entwicklung des zu schaffenden Modells.

## 3.1 Motivation für die Schaffung eines einheitlichen Modells

Die Erstellung eines Modelles für einen bestimmten Geschäftsprozess stellt ein zeitaufwändiges Verfahren dar. Welche Informationen und Maßnahmen dafür notwendig sind, wird in Abschnitt 3.2 näher erläutert. Zunächst soll aber auf die Motivation hinter der Entwicklung des Modells eingegangen werden. Diese kann, in Abhängigkeit des Einsatzzweckes, unterschiedlich sein.[16]

Eines der zentralen Ziele der Prozessmodellierung und gleichermaßen auch die Grundlage für weitere Optimierungsmaßnahmen ist die Dokumentation und Erfassung von existierenden Geschäftsprozessen. Diese bedingt die systematische Untersuchung aller im Rahmen eines Prozesses durchgeführten Aktivitäten. Durch die so gewonnenen Informationen kann die Transparenz der Arbeitsprozesse eines Unternehmens sowohl nach innen als auch nach außen verbessert werden. Nach innen, da die genauen Abläufe eines Prozesses auch nicht involvierten Mitarbeitern offen gelegt werden und nach außen, da so für Kunden erkenntlich wird, welche Aktionen mit ihren Anforderungen in Verbindung stehen.[17]

Eine besondere Rolle hat die Dokumentation und Speicherung von Informationen, wenn Prozesse modelliert werden, die bestimmten gesetzlichen Rahmenbedingungen unterliegen. In diesem Fall reduziert der Zugriff auf etablierte Vorgehensmodelle den Einarbeitungsaufwand neuer Mitarbeiter und kann vor gesetzlichen Repressalien schützen.[18]

Neben den Vorteilen der Erfassung und Dokumentation bestehender Prozesse bildet insbesondere die Möglichkeit zur Optimierung vorhandener Vorgehensweisen einen zentralen

[16] [Mic12], S. 1
[17] [Tho05], S. 92
[18] [Mic12], S. 31 ff.

Motivationsschwerpunkt. Mittels der gewonnen Informationen kann, durch das Einbringen von Erfahrungen oder den Einsatz neuer Technologien, die Wirtschaftlichkeit eines Prozesses verbessert werden.[19] Diese Verbesserungen können beispielsweise in einer Beschleunigung, der Vereinfachung oder einer verbesserten Wiederverwendbarkeit einzelner Prozessbestandteile resultieren.[20]

Obwohl im vorliegenden Projekt auch die Schaffung von Transparenz eine Rolle spielt, liegt der zentrale Motivationsaspekt in der Optimierung des zu untersuchenden Prozesses. Das Mittel zur Erreichung dieses Ziels ist dabei die Einführung neuer Technologien in Form von Vagrant und Puppet.

Neben der eigentlichen Verbesserung des Systems dient das erstellte Modell aber auch einem weiteren Zweck abseits der vorgestellten Zielstellungen. Es bildet die Grundlage für die Bewertung der Wirtschaftlichkeit der Konfigurationsautomatisierung, die im Rahmen dieser Arbeit beschrieben und weiterentwickelt wird. Wie das Modell im Detail zum Erreichen dieser Zielstellung eingesetzt wird, erläutert Kapitel 5.

## 3.2 Theoretische Grundlagen der Prozessmodellierung

Nachdem verschiedene Motivationen zur Modellierung eines Prozess vorgestellt wurden, soll nun auf theoretische Grundlagen der Prozessmodellierung eingegangen werden. Neben der allgemeinen Definition wird dabei auch auf die wesentlichen Bestandteile eines Geschäftsprozesses eingegangen und es werden Möglichkeiten der Visualisierung eines modellierten Prozesses vorgestellt. Diese Informationen bilden die Basis für die im Kapitel 3.3 durchzuführende Modellbildung des zu Grunde liegenden Projektes.

### 3.2.1 Bestandteile eines Geschäftsprozesses

Trotz unterschiedlicher Definitionsansätze in der Fachliteratur wird ein Geschäftsprozess meist als eine Abfolge logisch zusammenhängender Tätigkeiten oder Aktivitäten, die auf die Erfüllung eines übergeordneten Organisationsziels ausgerichtet sind, definiert.[21] Die

[19] [Mic12], S. 30
[20] [And11], S. 5
[21] [Jos14]

Organisationsziele können dabei, abhängig von den Tätigkeitsschwerpunkten eines Unternehmens, sehr unterschiedlich sein.

Die Modellierung eines solchen Prozessen basiert auf dem Zusammenwirken einer Reihe von Basiskomponenten. Diese sind die Aktivitäten innerhalb des Prozesses, die Kontrollflüsse, welche diese logisch verknüpfen, Ereignisse, deren Eintreten berücksichtigt werden muss sowie Ressourcen, die von Aktivitäten erzeugt oder verbraucht werden.[22]

Die Aktivitäten innerhalb eines Prozesses beschreiben jeweils einzelne und klar umrissene Arbeitsschritte des betrachteten Verfahrens. In der Regel bestehen Prozessmodelle aus mehreren solcher Aktivitäten die eine gemeinsame Zielstellung verfolgen.
Die Definition von Aktivitäten, kann innerhalb verschiedener Abstraktionsniveaus erfolgen. Bei der Definition eines Modells ist aus diesem Grund darauf zu achten, dass sich alle Aktivitäten etwa auf der gleichen Stufe der Abstraktion befinden. Eine zentrale Messgröße bei der Definition und Untersuchung von Aktivitäten ist deren Ressourcenbilanz. Diese beschreibt, welche Ressourcen von innerhalb der jeweiligen Aktivität erzeugt bzw. verbraucht werden. Dabei wird zwischen materiellen Ressourcen, das heißt physischen Gütern, und immateriellen Ressourcen, wie beispielsweise der aufgewendeten Arbeitszeit einer Person oder auch erzeugten Informationen in Form von Daten, unterschieden. Hierbei ist hervorzuheben, dass materielle Güter nach ihrem Verbrauch nicht mehr in ihrem ursprünglichen Zustand vorhanden sind, immaterielle Ressourcen aber prinzipiell immer wieder verwendet werden können.[23]
Im Arbeitsumfeld der Softwareentwicklung nehmen die immateriellen Ressourcen eine Schlüsselposition ein, da das Endprodukt eines Prozesses in der Regel kein physisches Erzeugnis, sondern beispielweise ein bestimmter Datenbestand ist.

Da ein Geschäftsprozess als Abfolge von Aktivitäten definiert ist, diese aber bisher nur isoliert betrachtet wurden, soll nun auf ihre Verknüpfung mittels Kontrollflüssen eingegangen werden. Der Kontrollfluss beschreibt die Reihenfolge der Abarbeitung verschiedener Aktivitäten, wobei die Kontrolle im Prozess von einer Aktivität auf die nächste übergeht.

[22] [Mic12], S. 31
[23] Ebenda, S. 4

Dabei werden im Rahmen der Prozessmodellierung sechs grundlegende Kontrollfluss-varianten unterschieden. Die nachfolgende Tabelle stellt diese Variante vor und erläutert kurz ihre Funktion.[24]

| **Kontrollflussvariante** | **Funktion** |
|---|---|
| Sequenz | Beginn der Abarbeitung einer Aktivität, nachdem die vorher-gehende Aktivität vollständig ausgeführt wurde. |
| Parallelisierung | Kontrollfluss wird an einer Stelle im Prozess auf verschiedene Pfade mit Aktivitäten aufgeteilt, die gleichzeitig abgearbeitet werden. |
| Synchronisation | Zusammenführung der durch Parallelisierung getrennten Kontrollpfade, wobei alle vorhergehenden Aktivitäten vollständig ausgeführt wurden sein müssen. |
| Auswahl | Kontrollfluss wird an einer Stelle im Prozess, abhängig von einer bestimmten Entscheidung oder einem Ereignis, auf eine oder mehrere Pfade aufgeteilt. |
| Verschmelzung | Zusammenführung der durch Entscheidung (Auswahl) aufgeteilten Kontrollflüsse. |
| Schleife | Punkt im Prozess, an dem eine oder mehrere Aktivitäten wiederholt ausgeführt werden. |

**Tabelle 1: Varianten von Kontrollflüssen**

Aktivitäten stellen die aktiven Bestandteile eines Prozessen dar. Doch neben diesen existieren auch passive Bestandteile, welche als Ereignisse bezeichnet werden. Die Einordnung in passive und aktive Komponenten basiert auf der Bewertung der Möglichkeit zur Einfluss-nahme auf deren Eintreten. So werden Aktivitäten gemäß dem Kontrollfluss zu definierten Zeitpunkten ausgeführt, aber Ereignisse treten in der Regel willkürlich ein, ohne dass dies beeinflusst werden kann. Dabei werden zwei verschiedene Arten von Ereignissen unterschieden. Dies sind zum einen zeitliche Ereignisse, welche zu einem bestimmten Zeitpunkt stattfinden und Mittelungsereignisse, die das Eintreffen einer konkreten

[24] [Lay10], S. 216 ff.

Information beschreiben. Da der Beginn von Aktivitäten vom Eintreten eines oder mehrerer Ereignisse abhängen kann, können diese direkten Einfluss auf den Prozessverlauf nehmen.

Eine weitere Modellierungskomponente bilden die bereits im Zusammenhang mit Aktivitäten angesprochenen Ressourcen. Diese lassen sich neben der Einteilung in physisch und nicht physisch, auch in menschliche und nicht menschliche Ressourcen, kategorisieren.
Zu den menschlichen Ressourcen einer Aktivität gehören alle Personen, die für ihre Ausführung notwendig sind. Nicht-menschliche Ressourcen hingegen sind sehr vielgestaltig und können in Form von Gebäuden, Computersystemen oder auch nur einer Tastatur vorliegen.
Eine gesonderte Form nicht menschlicher Ressourcen stellen Daten dar. Diese liegen in Form von Informationen vor und unterliegen somit nicht den konventionellen Regeln die für anderen Ressourcen Bestand haben. So können diese beispielsweise beliebig oft vervielfältigt oder gleichzeitig von verschiedenen Aktivitäten verwendet werden.

Nachdem in diesem Abschnitt alle Bestandteile, die bei der Modellierung von Prozessen berücksichtigt werden müssen, erläutert wurden, sollen im folgenden Abschnitt mögliche Darstellungsformen für das zu erstellende Model erläutert werden.

### 3.2.2 **Darstellungsformen**

Nachdem die prozessrelevanten Bestandteile innerhalb der Analyse erfasst wurden, muss eine Entscheidung zur Darstellung des Modells getroffen werden. Gegenüber der Beschreibung in Textform bietet die grafische Darstellung, aufgrund ihrer Kompaktheit und Übersichtlichkeit, eine sinnvolle Alternative. Hierbei steht eine Reihe von Modellierungssprachen mit spezifischen Notationen zur Auswahl, welche sich in der Praxis für unterschiedliche Einsatz-zwecke bewährt haben.

In den folgenden Abschnitten sollen exemplarisch drei Darstellungsformen, welche häufig in der Praxis Anwendung finden, vorgestellt werden, um eine Wahl für die Erstellung des Modells im vorliegenden Projekt treffen zu können.

Eine Form der Darstellung ist die ereignisgesteuerte Prozesskette EPK. Diese wurde 1992 von einer Arbeitsgruppe unter der Leitung von August-Wilhelm Scheer an der Universität des Saarlandes in Zusammenarbeit mit der Firma SAP entwickelt. EPK ist eine semiformale Sprache, da sie zwar eine formal definierte Syntax, aber nur eine teilweise formale Semantik besitzt. Ihr zentrales Einsatzgebiet ist die Modellierung von Geschäftsprozessen, wobei Arbeitsprozesse nach bestimmten Syntaxregeln grafisch dargestellt und entsprechend dem Prozessverlauf miteinander verknüpft werden.

Eine weitere Möglichkeit für die Darstellung des Modelles ist die Business Process Model and Notation BPMN. Diese wurde 2001 von IBM geschaffen. Genau wie EPK ist auch BPMN eine semiformale Sprache mit einer formalen Syntax, aber einer nur teilweise formalen Semantik. Der Zweck dieser Sprache ist es, Geschäftsprozesse sowie ausführbare Prozesse zu modellieren.

Den für dieses Projekt interessantesten Ansatz bietet die Unified Modelling Language (UML) in Form von Aktivitätsdiagrammen. UML wurde speziell für den Einsatz im IT-Bereich entwickelt. Sie bietet verschiedene Darstellungsformen zur Spezifikation, Konstruktion und Dokumentation von Software-Bestandteilen. Für die Zwecke der Prozessmodellierung eignet sich besonders das UML-Aktivitätsdiagramm, welches grundsätzlich der Beschreibung allgemeiner Abläufe dient und für Aktivitäten und Kontrollflüsse eine formale Semantik bereitstellt.[25]

Neben den soeben vorgestellten Verfahren existieren noch eine Reihe weiterer Darstellungsmöglichkeiten, auf welche in dieser Arbeit aber nicht weiter eingegangen werden soll, da UML bereits ein optimales Werkzeug zur Visualisierung des Modells darstellt. Beispiele weiterer Darstellungsverfahren sind unter anderem Petri Netze oder Zustandsautomaten.

[25] [Kim06], S. 62 ff.

## 3.3 Umsetzung des Prozessmodells

Nachdem im vorangegangen Abschnitt die grundlegenden Bestandteile und Vorgehensweisen der Prozessmodellierung erläutert wurden, soll nun das Vorgehen beschrieben werden, mit dem diese Bestandteile im vorliegenden Projekt ermittelt und zusammengesetzt wurden.

### 3.3.1 Erfassung der bestehenden VM-Prozesse

Um die Komponenten des Prozesses zur Verwendung virtueller Maschinen exakt modellieren zu können, ist es wichtig, alle dabei durchgeführten Arbeitsschritte zu kennen und zu verstehen. Deshalb soll an dieser Stelle noch einmal im Detail erläutert werden, welche einzelnen Aktivitäten von Mitarbeitern im Umgang mit virtuellen Maschinen durchlaufen werden. Die Ermittlung dieser Kenntnisse basiert auf den bis zu diesem Zeitpunkt realisierten Konfigurationsautomatisierungen sowie dem Informationsaustausch mit Mitarbeitern, die für diese Prozesse verantwortlich sind.

Wird in einem Projekt- oder Entwicklungsteam eine neue VM benötigt, wird diese meist durch einen Mitarbeiter des jeweiligen Teams oder der IT-Abteilung erstellt. Zu diesem Zweck wird zunächst mit Hilfe einer Virtualisierungssoftware, wie beispielsweise VirtualBox[26] oder VMWare[27], eine virtuelle Maschine erzeugt. Mit Hilfe der grafischen Oberfläche dieser Anwendungen werden dabei oder im Anschluss daran die benötigten Konfigurationen der virtuellen Hardware der VM durchgeführt. Dazu gehören unter anderem Arbeitsschritte wie das Anlegen virtueller Festplatten, die Einrichtung bestimmter Port-Weiterleitungen zwischen Gast- und Hostsystem oder die Dimensionierung des in der VM verfügbaren Arbeitsspeichers.

Anschließend folgt die Installation eines bestimmten Betriebssystems. Welches dabei verwendet wird, ist von verschiedenen Faktoren abhängig. Zu diesen können die Gewohnheiten der Teammitglieder, das verwendete E-Commerce-System oder auch die spezifischen Gegebenheiten des Zielsystems, auf dem ein Projekt betrieben werden soll, gehören. Zusätzlich zur Installation des Systems werden noch weitere Konfigurationen wie das Anlegen bestimmter Benutzer oder die Anpassung der grafischen Oberfläche durchgeführt.

---

[26] [Ora14]
[27] [Vmw14]

Nachdem das gewählte Betriebssystem installiert und entsprechend den Anforderungen konfiguriert wurde, werden die benötigten Softwarepakete installiert. Dieser Vorgang wird, abhängig vom zu installierenden Programm und verwendeten Betriebssystem, entweder manuell oder mit Hilfe eines Paketverwaltungssystems durchgeführt. Der manuelle Prozess meint in diesem Fall das Herunterladen der benötigten Dateien aus dem Internet oder internen Netzwerkquellen und die anschließende Installation mittels einer grafischen Oberfläche beziehungsweise eines Kommandozeilentools. Für unterschiedliche Betriebssysteme stehen in der Regel verschiedene Paketmanager wie das Advanced Packaging Tool (APT) für Linux oder MacPorts bei Mac OS zur Verfügung. Diese ermöglichen es, den gesamten Installationsprozess einer Vielzahl von Softwarepaketen mittels weniger Kommandozeilenbefehle durchzuführen.[28] [29]

Neben der Installation bestimmter Software-Komponenten wird ebenfalls eine Reihe von Konfigurationen durchgeführt, die notwendig sind, um zu gewährleisten, dass die VM produktiv eingesetzt werden kann. Hierzu gehören in der Regel die Einrichtung des lokalen Webservers und die damit verbundene Erstellung von virtuellen Hosts, sowie das Anlegen und Einrichten verschiedener Datenbanken. Häufig ist auch der Checkout bestimmter Projekt-Ressourcen aus dem firmeninternen oder einem externen Versionsverwaltungssystem Teil dieses Arbeitsschrittes.

Wurden alle notwendigen Komponenten installiert und Konfigurationen durchgeführt, wird die erstellte VM mittels des internen Firmennetzwerks an alle involvierten Mitarbeiter verteilt.

Im Verlauf eines Projektes oder bei fortlaufender Verwendung der virtuellen Maschine nach Projektende entsteht ein steigender Bedarf für Anpassungen dieser Umgebung. Hierfür können verschiedene Faktoren als mögliche Ursache identifiziert werden. Einer davon ist die kontinuierliche Weiterentwicklung vieler Software Produkte, auf die mit der Durchführung entsprechender Update-Vorgänge reagiert werden muss. Eine andere Art der Anpassung ist die Notwendigkeit zur Installation zusätzlicher Software, beziehungsweise die Durchführung zusätzlicher Konfigurationen. Dabei sind verschiedene Ausgangsituationen vorstellbar. So

[28] [Ell09], S. 542 ff.
[29] [Jon13], S. 47 ff.

kann es vorkommen, dass sich im Verlauf eines Projektes die Anforderungen an eine VM verändern und somit die Einrichtung weiterer Anwendungen notwendig wird. Auch der Ablauf, beziehungsweise Neukauf von Software-Lizenzen sowie die Veränderung bestimmter Prozessabläufe kann diese Form der Anpassung notwendig machen. Aus diesem Grund stellt die Anpassungen und Aktualisieren von Software einen weiteren Arbeitsschritt im Umgang mit virtuellen Maschinen dar.

Die konkreten Arbeitsabläufe für die Durchführung einer Aktualisierung sind stark von der jeweiligen Anwendung abhängig. Eine Möglichkeit ist die Verwendung programm-spezifischer Aktualisierungswerkzeuge, wie sie beispielsweise von der IDE PhpStorm bereitgestellt werden.[30] Weitere Option ist die Verwendung integrierter Update-Funktionen von Paketverwaltungssystemen. So bietet beispielweise der Paketmanager APT die Funktion Aktualisierungen von Softwarepakete durch Verwendung des Befehls apt get upgrade zu installieren. Sind keine Möglichkeiten vorgesehen, um eine Software auf eine neuere Version zu aktualisieren, kann wie im Fall von Vagrant auch die Deinstallation und anschließende Installation bestimmter Programmversionen notwendig sein. Hinsichtlich der Re-konfiguration, beziehungsweise Nachinstallation von Software, sei an dieser Stelle auf die Vorgänge im Rahmen der initialen Einrichtung der VM verwiesen.

Bei bestimmten Kombinationen von Anforderungen, die sich im Rahmen der Verwendung einer VM ergeben, kann es vorkommen, dass diese nur mit hohem Anpassungsaufwand oder auch gar nicht realisierbar sind. Dieser Fall tritt beispielsweise ein, wenn zwingend benötigte Software nicht mit dem verwendeten Betriebssystem kompatibel ist oder die Anforderungen ein anderes Betriebssystem als Basis definieren. Die Anpassung einer vorhanden virtuellen Maschine ist auch dann nicht mehr zweckmäßig, wenn der Konfigurations- und Änderungsaufwand den einer Neuerstellung der VM übersteigen würde. Gleiches gilt bei Fehlerzustand der VM in Bezug auf durchzuführende Reparaturaufwände.

### 3.3.2 **Entwicklung des Modells**

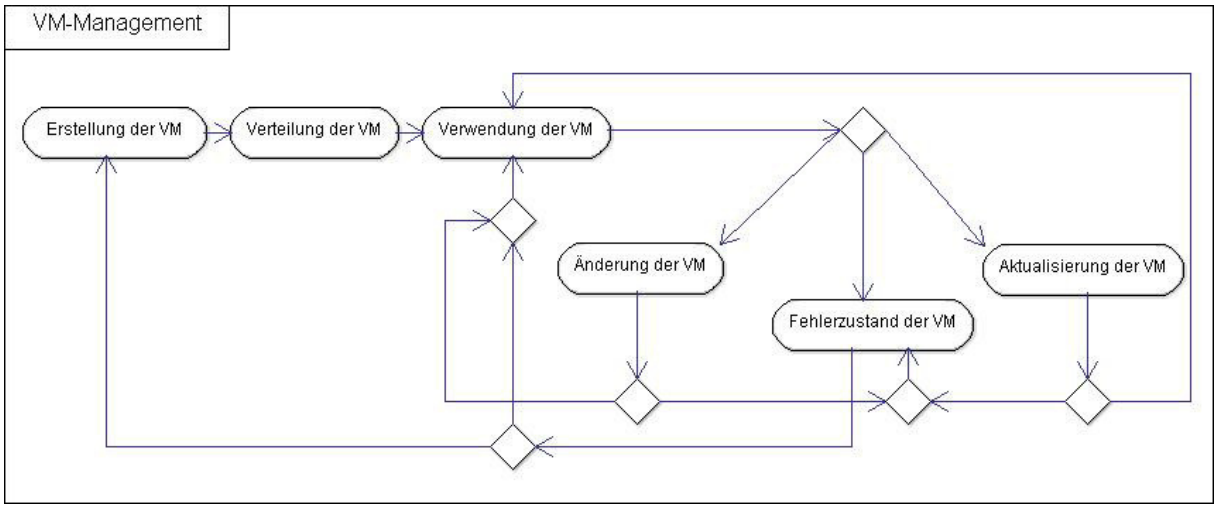
Aus den im Abschnitt 3.3.1 vorgestellten, typischen Szenarien im Umgang mit virtuellen Maschinen können unterschiedliche Teilprozesse abstrahiert werden. In diesem Kapitel sollen

[30] [Jet14]

diese Prozesse, inklusiver ihrer einzelnen Bestandteile, identifiziert und zu einem Gesamtmodell zusammengefügt werden. Das Resultat dieses Vorgehens wird dabei in Form mehrerer UML-Aktivitätsdiagramme visualisiert.

Um ein Modell zu schaffen, welches für jeden Abschnitt des VM-Lebenszyklus angewendet werden kann, erscheint es sinnvoll, den Gesamtprozess des VM-Managements zunächst in Teilprozesse zu zerlegen. Die Unterteilung erfolgt dabei nach den unterschiedlichen Situationen, in welche die virtuelle Maschine im Laufe ihrer Verwendung eintreten kann. Diese sind ihre intitiale Erstellung, das Update einzelner Komponenten, der Fehlerzustand oder die Durchführung tiefgreifender Änderungen. Jeder dieser Prozesse bedingt die Ausführung spezifischer Arbeitsschritte, welche möglicher Weise durch ein automatisches Konfigurationssystem unterstützt werden können.
Um die Zusammenhänge dieser Prozesse noch einmal zu verdeutlichen, zeigt Abbildung 2 deren Einordnung in den Gesamtprozess des VM-Managements.

**Abbildung 2: Teilprozesse VM-Management (nicht-formale Notation)**




**Quelle: eigene Darstellung**

Zunächst soll der Erstellungsprozess abstrahiert werden. Zu diesem Zweck werden alle im Laufe der Analyse ermittelten Installations- und Konfigurationsschritte hinsichtlich gleichartiger Vorgehensmuster untersucht. Die so definierten Aktivitäten werden, entsprechend ihrer Abarbeitungsreihenfolge, in das Modell eingeordnet. Sind dabei Aktivitäten wiederholt auszuführen oder erlauben eine parallele Abarbeitung, wird dies im

Kontrollfluss entsprechend abgebildet. Das Ergebnis der Modellierung des Erstellungsprozesses ist in Anlage 3 dargestellt und wird nachfolgend noch einmal im Detail erläutert.

Wie im entwickelten UML-Diagramm dargestellt ist, wurde der Hauptprozess in zwei parallel verlaufende Subprozesse unterteilt. Einer dieser Prozesse stellt dar, wie sich der Entwickler im Rahmen der durchzuführenden Installations- und Konfigurationsprozesse immer wieder in neue Technologien einarbeiten muss. Neben der kontinuierlichen Aneignung von Wissen wird in diesem Prozessstrang auch die Lösung möglicher Problemstellungen berücksichtigt. Da diese Aktivitäten nicht allgemeingültig quantifizierbar sind, aber dennoch einen großen Teil der zeitlichen Aufwände im Prozess darstellen, werden sie in dieser Form abgebildet.

Parallel zu diesen kontinuierlichen Aktivitäten verläuft der eigentliche Erstellungsprozess, welcher weitgehend sequentiell abgearbeitet wird. Dieser beginnt mit der Konfiguration der gewünschten virtuellen Hardware entsprechend der bestehenden Vorgaben. Daran schließen sich die Aktivitäten für Download, Installation und Konfiguration des Betriebssystems an. Der Übergang zwischen diesen Aktivitäten ist dabei aber fließend, da einzelne Konfigurationen auch bereits im Rahmen des Installationsvorganges durchgeführt werden können.

Entsprechen alle Einstellungen des Betriebssystems den gegebenen Spezifikationen, folgt die Einrichtung der benötigten Softwarepakete. Die dafür notwendigen Aktivitäten wurden im erstellten Modell in Form einer Schleife organisiert, deren Anzahl an Durchläufen der Anzahl zu installierenden Softwarepakete entspricht. Konfigurationen, die dabei durchzuführen sind, werden durch eine innere Schleife abgebildet, um so unterschiedliche Konfigurationsaufwände zu berücksichtigen. Innerhalb des Modells entspricht somit die Anzahl unterschiedlicher Konfigurationen eines Softwarepaketes der Zahl von Iterationen dieser Schleife.

Die darauffolgenden Aktivitäten bieten erneut die Option für eine parallele Abarbeitung. Es ist einerseits der Import bereits vorhandener Datensätze, andererseits das Kopieren von Projektressourcen, beispielsweise aus einem Versionsverwaltungssytem. Da der Import von Datensätzen, abhängig von deren Anzahl, ein zeitaufwändiger Prozess sein kann und nach Beginn des Importprozesses in der Regel keine weiteren Eingaben notwendig sind, kann diese Aktivität parallel ausgeführt werden. Beide soeben vorgestellten Arbeitsschritte müssen aber

nur dann berücksichtigt werden, wenn entsprechende Projektdaten bereits vorhanden sind. Gleiches gilt auch für die Durchführung projektspezifischer Konfigurationen, welche die abschließende Aktivität des Erstellungsprozesses bildet. Da auch hier die Anzahl von Einstellungen abhängig vom jeweiligen Projekt ist, wurde der Kontrollflusses in Form einer Schleife organisiert. Sind alle Konfigurationen durchgeführt und die dabei möglicherweise aufgetretenen Problemstellungen gelöst, werden die zu Beginn der VM-Erstellung parallelisierten Prozessstränge wieder zusammengeführt.

Um die virtuelle Maschine nach dem Prozess der Erstellung produktiv einsetzen zu können, ist ein weiter Teilprozess notwendig. Dieser wurde unter der Bezeichnung Verteilung der VM zusammengefasst und umfasst den Kopiervorgang, die Durchführung individueller Vorbereitungen sowie abschließender Konfigurationen durch den Entwickler. Abbildung 3 zeigt die korrespondierende UML-Darstellung.

**Abbildung 3: Prozessmodell - Verteilung einer virtuellen Maschine**


**Quelle: eigene Darstellung**

Obwohl der Kopiervorgang eine triviale Sequenz von Einzelaktivitäten beschreibt, ist es wichtig, diesen trotzdem im Modell zu berücksichtigen. Grund hierfür sind die zeitkritischen Komponenten des Datenaustausches bei umfangreichen virtuellen Maschinen sowie der individuelle Einrichtungsprozess des Anwenders. Bei der Untersuchung der auf dem Netzlaufwerk der dotSource befindlichen virtuellen Maschinen konnte festgestellt werden, dass deren Datenvolumen zwischen 5 und 22 GB liegt. Diese Größe unterliegt einer starken Variation, welche aus der Anzahl und dem Umfang der verwendeten Softwarepakete sowie den vorhanden Projektdaten resultiert. Wird beispielsweise angenommen, dass eine VM mit einem Volumen von 15 GB übertragen werden muss und die Übertragungsrate im lokalen Netzwerk bei durchschnittlich 35 MB/s liegt, beträgt der Zeitaufwand für einen Kopiervorgang bereits mehr als 7 Minuten. Dabei ist weiterhin zu berücksichtigen, dass dieser

Vorgang, abhängig vom Verwendungszweck der virtuellen Maschine, von einer Vielzahl von Personen auszuführen ist.

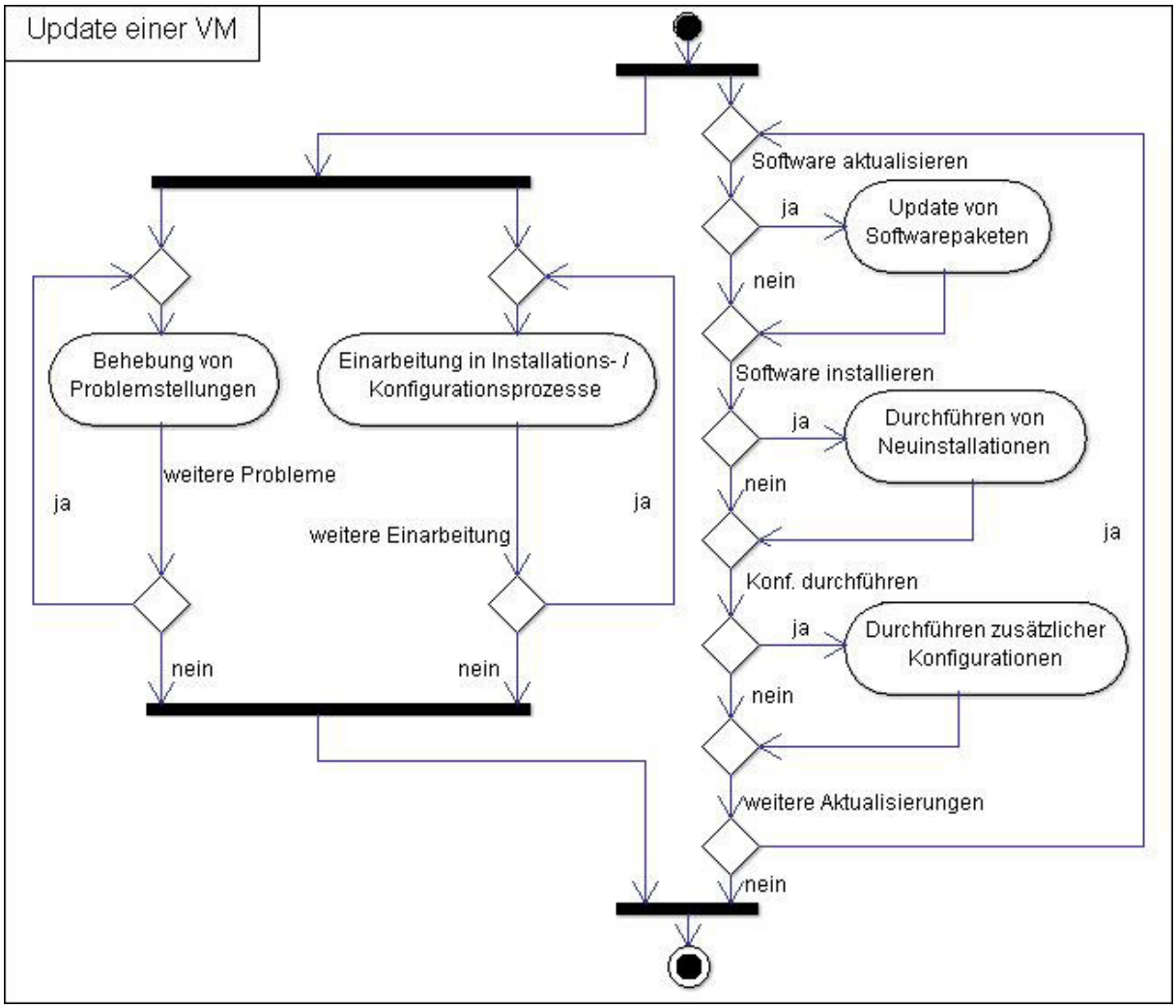
Die Durchführung individueller Vorbereitungen dient zur Erfassung von Arbeitsvorgängen wie dem lokalisieren der VM auf dem Netzlaufwerk oder der Installation beziehungsweise Aktualisierung der Virtualisierungssoftware oder deren Gasterweiterung. Auch die Installations- und Aktualisierungsprozesse von Vagrant werden durch diese Aktivität erfasst.

Trotz der angestrebten Homogenität der innerhalb eines Projektes, beziehungsweise Teams verwendeten virtuellen Maschinen, ist es jedem Anwender freigestellt in einem bestimmten Rahmen individuelle Konfigurationen an der kopierten VM durchzuführen. Anpassungen können dabei etwa die Installation persönlich bevorzugter Editoren, Konfigurationen der Oberfläche der verwendeten IDE oder des Betriebssystem sein. Um auch diese Abläufe im erstellten Modell abbilden zu können, bildet die Aktivität Durchführung individueller Konfigurationen den Abschluss des Verteilungsteilprozesses.

Nachdem die VMs initial erstellt und an die jeweiligen Anwender verteilt wurden, können im Rahmen der Verwendung grundsätzlich drei weitere Situationen eintreten, an die spezifische Prozesse gebunden sind. Eine ist die Notwendigkeit für die Aktualisierung verschiedener Komponenten der VM. Die Modellierung des damit verknüpften Prozess ist im Diagramm der Abbildung 4 dargestellt.

Wie zu erkennen ist, kann, ähnlich dem VM-Erstellungsprozess, auch die Aktualisierung grundlegend in zwei parallele Prozessstränge unterteilt werden. Einen bilden die Bearbeitung von auftretenden Problemstellungen und die notwendige Einarbeitung in bestimmte Installations- und Konfigurationsprozesse. Der Andere ist die Durchführung der geforderten Aktualisierungen. Die Reihenfolge der im Modell dargestellten Update-Aktivitäten ist dabei willkürlich gewählt und kann im Falle bestehender Abhängigkeiten auch variiert werden. Um eine beliebige Anzahl von Aktualisierungen abzubilden, wurden diese auch hier wieder in Form eines iterativen Vorgangs abgebildet.

**Abbildung 4: Prozessmodell - Update einer virtuellen Maschine**



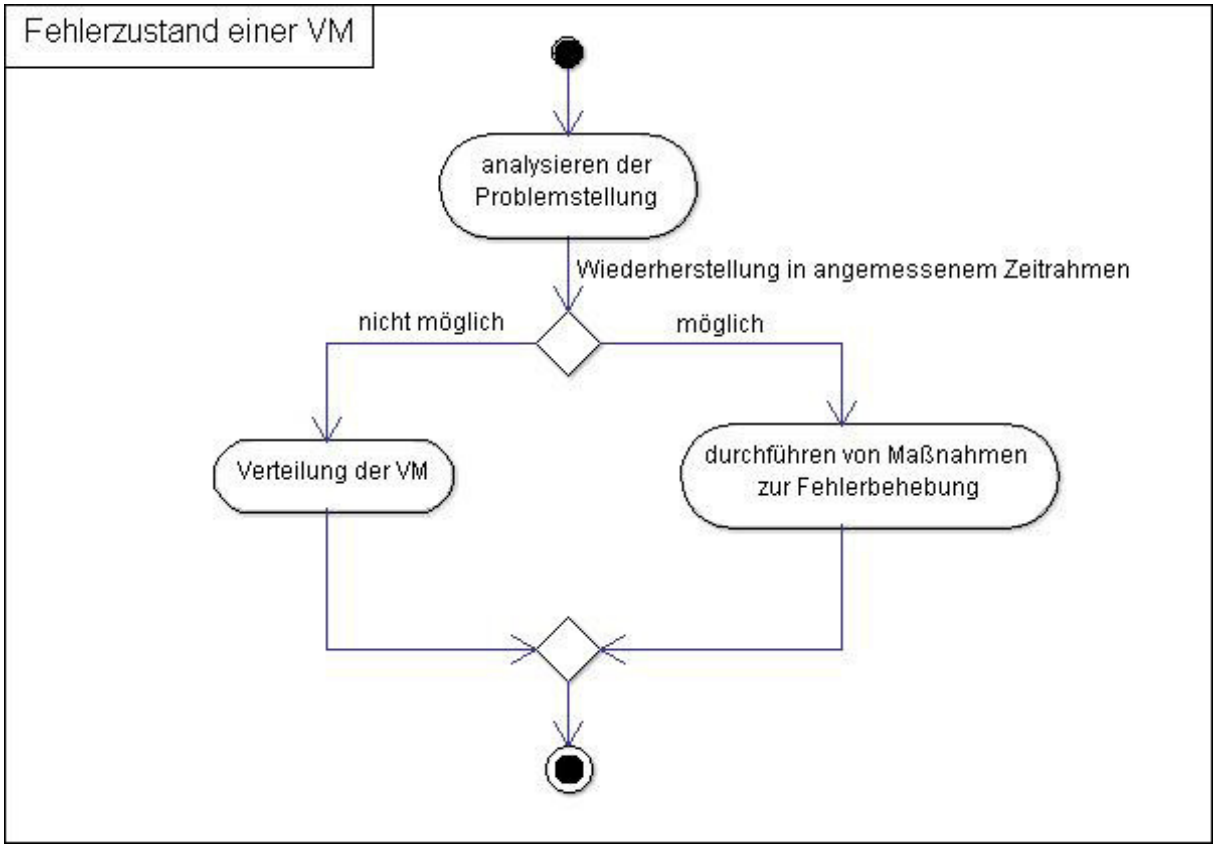

**Quelle: eigene Darstellung**

Einen weiterern Zustand, den die VM im Laufe ihrer Verwendung einnehmen kann, ist der Fehlerzustand. Die Gründe für die Einnahme dieses Zustanden können sehr unterschiedlich sein und reichen von Fehlern in der Konfiguration bis zum bereits im Aktualisierungsprozess erfassten Update-Fehler. Der modellierte Prozess, der bei Eintritt eines solchen Zustandes durchgeführt wird, kann Abbildung 5 entnommen werden.

Kommt es zu einer Problemstellung, die die Verwendung der VM nachhaltig beeinflusst, besteht die erste Aktivität darin zu ermitteln, wie sich das konkrete Problem darstellt und welche mögliche Lösungsoptionen bestehen. Im Rahmen dessen ist die Entscheidung zu treffen, ob eine Wiederherstellung des einsatzbereiten Zustands der virtuellen Maschine innerhalb eines angemessenen Zeitrahmens möglich ist oder ob die lokale Erneuerung der

VM die wirtschaftlichere Alternative darstellt. Abhängig von dieser Entscheidung werden entweder, wie im Modell dargestellt, Reparaturmaßnahmen durchgeführt oder der Verteilungsprozess für den betroffenen Mitarbeiter erneut initiiert.

**Abbildung 5: Prozessmodell - Fehlerzustand einer virtuellen Maschine**



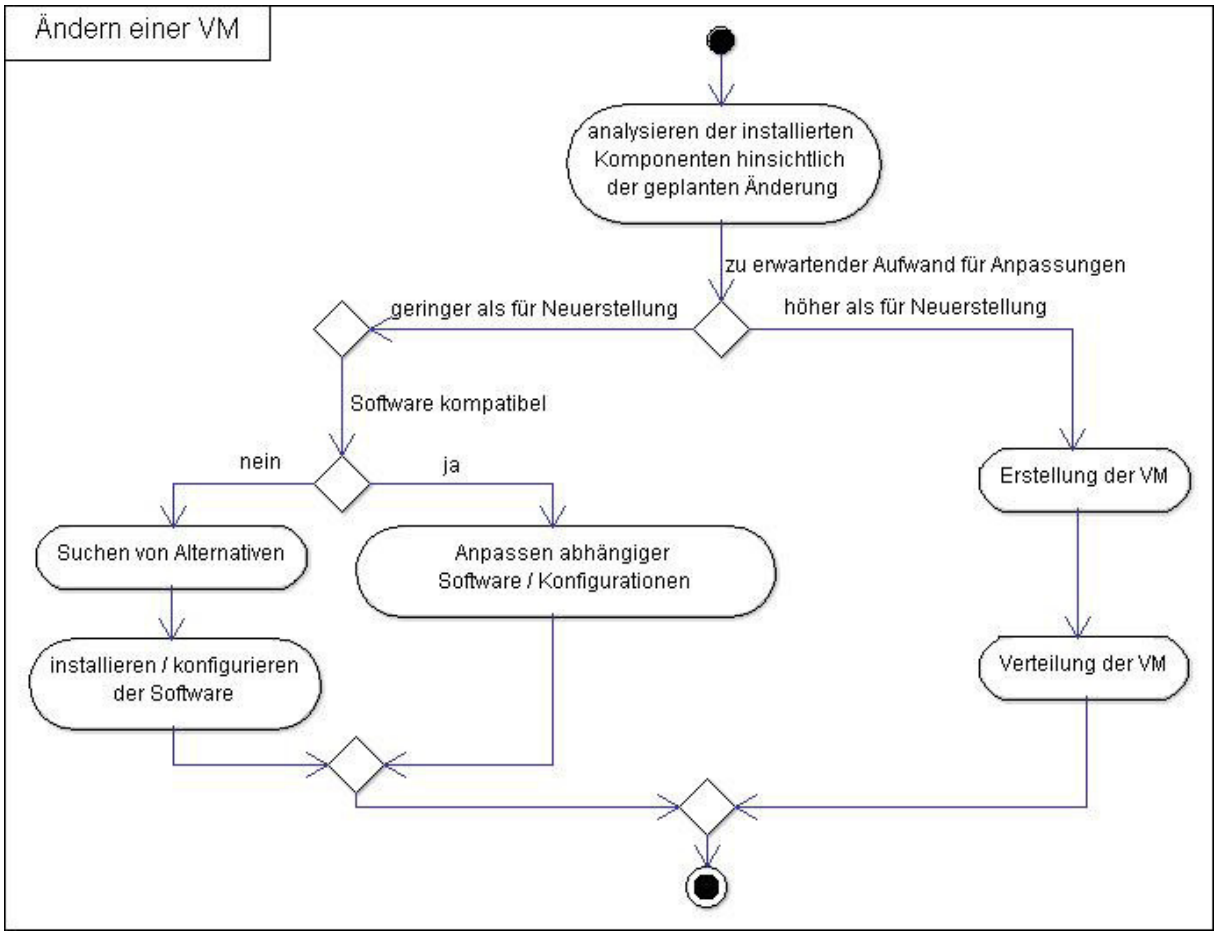

**Quelle: eigene Darstellung**

Der letzte Teilprozess des VM-Management, der an dieser Stelle untersucht wurde, ist die Änderung einer VM. Dieser beschreibt einen Update-Vorgang der tiefgreifende Änderungen im bestehenden System nach sich zieht, wie beispielsweise der Wechsel des verwendeten Betriebssystems. Der für dieses Szenario modellierte Prozess ist in Abbildung 6 dargestellt.

Den Einstieg in diesen Prozess bilden die Analyse der bereits vorhandenen Komponenten und die damit verbundene Fragestellung nach den Aufwänden für die Umsetzung der Änderungsanforderungen. Sind diese höher als der Aufwand einer zentralen Neuerstellung und Verteilung der VM, ist diese Vorgehensweise vorzuziehen und die entsprechenden Teilprozesse sind erneut zu durchlaufen. Sollte hingegen der Anpassungsaufwand der lokalen

Maschinen als geringer eingestuft werden, müssen mögliche Softwareinkompatibilitäten überprüft werden und, wie im Modell abstrahiert dargestellt, Lösungen gefunden werden.

**Abbildung 6: Prozessmodell - Ändern einer virtuellen Maschine**




**Quelle: eigene Darstellung**

Eines der Ziele, welche die Schaffung dieser Modelle verfolgt, ist die Bewertbarkeit der Wirtschaftlichkeit des eingesetzten Systems. Zu diesem Zweck mussten die zuvor erläuterten Einsatzszenarien hinsichtlich aller zeitlich relevanten Aktivitäten untersucht werden. Diese Ermittlung erfolgte auf der Basis von Erfahrungen, die innerhalb des Unternehmens im Einsatz mit virtuellen Maschinen gesammelten werden konnten.

# 4 Realisierung des Konfigurationssystems

Nachdem im vorangegangen Kapitel die Erstellung des Prozessmodells erläutert wurde, sollen im Folgenden die praktisch durchgeführten Umsetzungen erläutert werden, welche die Basis des anschließend durchzuführenden Vergleichs und dessen Auswertung bilden. In diesem Zusammenhang wird zunächst auf den Stand des Konfigurationssystems zu Beginn dieser Arbeit eingegangen. Nachfolgend werden die Maßnahmen zur Weiterentwicklung beschrieben und dabei auf Problemstellungen im Entwicklungsprozess eingegangen. Den Abschluss dieses Kapitels bildet ein Überblick über noch ausstehende Umsetzungen.

## 4.1 Aktueller Projektstand

Wie im Rahmen der Einleitung bereits geschildert, wurde im Rahmen einer Projektarbeit damit begonnen, die Prozesse des Managements virtueller Maschinen in der dotSource GmbH partiell zu automatisieren. Zum Zeitpunkt der Erstellung dieser Arbeit wurden die gewählten Softwaresysteme Vagrant und Puppet für die grundlegende Erstellung und Konfiguration einer VM in der Entwicklungsabteilung für PHP-Projekte eingeführt. Außerdem wurden die Anforderungen an ein solches System von allen anderen Entwicklungsabteilungen sowie der Abteilung für User Interfaces (UI) und Qualitätssicherung (QA) aufgenommen und dokumentiert.

Die Umsetzung besteht zum aktuellen Zeitpunkt aus einem Prototyp des Vagrant-Konfigurations-Files zur Generierung der VM und einer Reihe von Puppet-Modulen für die Installation von Software und der Durchführung von Konfigurationen. Des Weiteren existiert der Entwurf einer Verzeichnisstruktur, um die Ablage der Vagrant-Projekte im Versionsverwaltungssystem der dotSource, Apache Subversion (SVN), zu realisieren. Hier wurden auch alle bisher erstellten Konfigurationsskripte abgelegt. Die Bearbeitung von Konfigurationen durch mehrere Entwickler stellt somit kein Problem dar und auch die Aktualisierung lokaler Puppet-Skripte bleibt für den Endnutzer komfortabel.

## 4.2 Abteilung PHP-Entwicklung

Bei der Erstellung der Konfigurationsskripte für die VM der PHP-Entwicklungsabteilung kann auf die bestehenden Entwicklungen des Konfigurationssystems aufgebaut werden. Ziel ist es dabei, noch offene Anforderungen umzusetzen und die bisherigen Konfigurationen so zu gestalten, dass ein nachträglicher Provisionsprozess fehlerfrei durchgeführt werden kann und eine einfachere Erweiterbarkeit gegeben ist.

### 4.2.1 Stand des vorhandenen Systems

Vor Beginn dieser Arbeit ist bereits ein, für die Anforderung des PHP-Teams angepasstes, Vagrant-Projekt erstellt worden. Dieses enthält das Vagrantfile, in welchem eine Box mit der Linux-Distribution Ubuntu in der Version 12.04 definiert ist. Neben der Definition weiterer Vorgaben an die virtuellen Hardware der VM wird an dieser Stelle auch das Konfigurations-managementsystem Puppet, wie in Abschnitt 2.1 beschrieben, als Provisioner festgelegt.
Die Konfigurationsskripte des Provisioners wurden in einem Unterverzeichnis im Pfad Puppet/Modules innerhalb des Projektes organisiert. Für die Realsierung der Anforderungen wurden sowohl eigene als auch Module aus dem Puppet Online Repository, welches auch als PuppetForge bezeichnet wird, verwendet. Mit Hilfe der verwendeten Puppet-Skripte konnte eine große Anzahl von Software-Installationen und -Konfigurationen durchgeführt werden. Hierzu gehört die Einrichtung des Apache-Webservers, die Installation des MySQL-DBMS, das Herunterladen und die Installation der IDE PhpStorm oder die Installation von Php 5.3. Eine Liste aller Konfigurationen, die durch die eingesetzten Module durchgeführt werden, kann Anlage 4 entnommen werden.

Obwohl der Projektstand zu Beginn dieser Arbeit die Anforderungen des PHP-Teams grundlegenden erfüllt, müssen weitere Anpassungen durchgeführt werden, um einen realistischen Vergleich auf Basis des entwickelten Modells zu ermöglichen. Die zu diesem Zweck durchgeführten Maßnahmen werden im nachfolgenden Abschnitt beschrieben.

### 4.2.2 Problemstellungen und Lösungen

Die durchgeführten Anpassungen der vorliegenden Versionen zielen primär darauf ab, das Projekt noch übersichtlicher zu gestalten, um so eine bessere Wart- und Erweiterbarkeit zu erzielen. Des Weiteren sollen bisher nicht realisierte Features, wie die Unterstützung der

PHP-Version 5.4 oder die parallele Verwendung mehrerer virtueller Hosts, mit dem Apache Webserver realisiert werden.

Bei genauer Analyse der Puppet-Modul-Struktur zu Beginn dieses Projektes können eine Reihe von Aspekten identifiziert werden, welche Potential für Optimierungen aufweisen. Einer ist der geringe Umfang einzelner Module, wodurch wiederum die Anzahl der Module erhöht und somit die Übersichtlichkeit verringert wird. Um hier eine Verbesserung zu erzielen, wurden die Module cs-tools und git in das Modul tools integriert. Dort wurden separate Klassen für alle Applikationen erstellt, welche zusätzliche Konfigurationen benötigen. Außerdem wurde eine separate Klasse mit dem Namen standard geschaffen, welche alle Pakete installiert, die keinem Modul direkt zugeordnet werden können und keine weiterführende Konfiguration benötigen. Der nachfolgende Auszug des verwendeten Puppet-Skriptes zeigt die konkrete Umsetzung dieser Klasse.

```
class tools::standard () {
    package { 'curl': ensure => latest }
    package { 'vim': ensure => latest }
    package { 'wget': ensure => latest }
    package { 'ant': ensure => latest }
    package { 'subversion': ensure => latest }
}
```

Dieses Vorgehen ermöglichte es, die Anzahl der Module zu reduzieren und die verbliebenen stärker auf das Ziel ihrer jeweiligen Konfiguration auszurichten, da zusätzliche Paketinstallationen entfallen.

Ein weiterer Schwerpunkt der durchgeführten Anpassungen liegt in der Verbesserung der Update-Möglichkeiten erzeugter virtueller Maschinen. Vagrant stellt hierfür das Kommando vagrant provision zur Verfügung, welches die vorhanden Puppet-Skripte sowie alle eingebundenen Shell-Provisioner erneut lädt, um so Änderungen der Einstellungen auf das vorhandene System zu übertragen. Aus diesem Grund ist es wichtig, dass jeder Befehl eines Provisioners beliebig oft hintereinander ausgeführt werden kann, ohne Fehler oder ein anderes unerwünschtes Verhalten hervorzurufen. Diese Eigenschaft von Ressourcen wird in diesem Zusammenhang auch als Indempotenz bezeichnet und beschreibt den Umstand, dass Puppet-

Ressourcen beliebig oft auf ein System angewendet werden können und dabei immer dasselbe Ergebnis erzielen.[31] Um das zu realisieren, stellt Puppet für die meisten Ressource-Typen Attribute zur Verfügung die einen Fehlerzustand bei mehrfacher Ausführung verhindern sollen. Im folgenden Quellcode-Beispiel ist mit dem ensure Attribut einer file-Ressource ein solcher Mechanismus dargestellt. In diesem Fall wird nur dann eine Datei erzeugt, wenn sich im angegebenen Pfad kein Element mit den gewünschten Spezifikationen befindet.

```
file { 'desktopIcon':
     path    => "/home/${user}/${desktopPath}/phpstorm.desktop",
     mode    => '0755',
     owner   => "${user}",
     ensure  => present,
     require => [Class["users"],File["desktopFolder"]],
}
```

Dieses Verhalten wurde in allen Puppet-Ressourcen implementiert, um ein wiederholtes fehlerfreies Provisioning zu gewährleisten.

Neben der Anpassung der einzelnen Klassen und Ressourcen innerhalb Puppet-Module wurde auch die Modulstruktur selbst verändert. Innerhalb des Modulordners wurden zu diesem Zweck zwei Unterordner mit den Bezeichnungen dotSource und puppetForge erstellt und diesen die vorhandenen Module zugeordnet. Die zugrunde liegende Problemstellung ist, dass die vorhandenen Struktur von außen keine eindeutige Unterscheidung zwischen selbst entwickelten und Modulen des Puppet Online-Repositories ermöglicht. Dies erschwert die Wartbarkeit, da für Updates immer zuerst die Herkunft innerhalb des Moduls verifiziert werden muss. Mittels der durchgeführten Unterteilung konnte die Ermittlung der Quelle eines beliebigen Moduls vereinfacht werden.

Der letzte Schritt, der bei der Aktualisierung und Erweiterung des vorhandenen Projektes durchgeführt wurde, ist die Aktualisierung der vorhandenen PuppetForge-Module. Ziel war es dabei, die Flexibilität der Konfigurationen zu verbessern und die durch diese Module verwaltete Software zu aktualisieren. Im Mittelpunkt stand dabei die Installation des PHP-

[31] [Dan13], S. 4

Modules mit der PHP-Version 5.4 oder höher. Im Rahmen dieser Aktualisierung wurde jedoch festgestellt, dass die verwendete Ubuntu-Version 12.04 PHP nur bis zur Version 5.3 unterstützt. Um Softwarepakete neuerer Ubuntu-Releases zu verwenden, wird durch den Paketmanager Apt die Möglichkeit geboten, so genannte Backports zu verwenden. Hierbei handelt es sich um Software, die unter Umständen noch nicht den Grad an Stabilität und Sicherheit erreicht hat, um Teil des vorliegenden Ubuntu-Releases zu sein. Diese Pakete bieten aber häufig neue Funktionen und können aus diesem Grund mit Hilfe des Paketmanagers dennoch installiert werden.[32] Da Backports zwar mittels des Puppet-Apt-Moduls aktiviert werden können, das verwendete PHP-Module diese aber nicht verwendet, wurde die Entscheidung zu Gunsten der Verwendung eines aktuelleren Ubuntu-Releases getroffen. Vor diesem Hintergrund wurde eine Ubuntu-Box mit der Version 14.04 erstellt, auf der alle notwendigen Modul- und Software-Updates durchgeführt werden konnten.

Durch die Weiterentwicklung des vorhandenen Konfigurationssystems konnten weitere Erkenntnisse zum grundlegenden Projekt- und Ressourcenaufbau gewonnen werden. Außerdem wurden, beispielsweise beim Update von Puppet-Modulen, mögliche Problemstellungen identifiziert und Lösungsansätze gefunden. Alle Erfahrungen, die in diesem Projektabschnitt gemacht werden konnten, werden in der Umsetzung der Konfigurationssysteme anderer Abteilungen berücksichtigt.

## 4.3 Abteilung Java-Entwicklung

Weitere Entwicklungen wurden für die Abteilungen für Java-Projekte durchgeführt. Diese werden innerhalb der dotSource nach dem E-Commerce-System unterschieden, auf dessen Basis Projekte erstellt werden. Ein Teil der Java-Entwicklungsabteilungen verwendet dabei die Enfinity Suite der Intershop Communications AG (Enfinity-Team), ein anderes das System Hybris (Hybris-Team) des gleichnamigen Unternehmens.

### 4.3.1 Anforderungen und Problemstellungen

Aus der Verwendung unterschiedlicher Shop-Systeme resultieren auch unterschiedliche Anforderungen an die zu automatisierenden Arbeitsumgebungen beider Abteilungen. Diese wurden bereits im Vorfeld dieser Arbeit grundlegend erfasst, müssen aber noch detaillierter

[32] [Can12]

betrachtet werden, um so Problemstellungen und Möglichkeiten der praktischen Umsetzungen herauszustellen. Zu diesem Zweck wurden in einem ersten Schritt Informationen von den jeweiligen Teamleitern der Abteilungen eingeholt, um so die bestehenden Anforderungen zu aktualisieren. Im Anschluss daran wurde für jede durchzuführende Konfiguration nach Möglichkeiten der Realisierung gesucht. In diesem Schritt wurden Module des Puppet-Repository hinsichtlich ihrer Eignung untersucht und der Aufwand für die Erstellung individueller Puppet-Skripte auf Basis der Komplexität der einzelnen Installations- und Konfigurationsabläufe eingeschätzt. Das Ergebnis dieser vorläufigen Betrachtungen wurde stichpunktartig festgehalten.

Bei der Untersuchung der Anforderungen des Enfinity-Teams konnte festgestellt werden, dass der Umgang mit diesem System nur die Verwendung bestimmter Betriebssysteme zulässt. Aus diesem Grund sollte für die erstellten virtuellen Maschinen SUSE Linux Enterprise Server (SLES) eingesetzt werden. Für die bisher im Rahmen der PHP-VM umgesetzten Konfigurationen bildete stets ein Ubuntu-System die Grundlage. Als Konsequenz aus der Verwendung dieser unterschiedlichen Linux-Distributionen sind, besonders im Fall der Wiederverwendung zuvor eingesetzter Konfigurationskomponenten, Kompatibilitätsaspekte zwischen beiden Systemen zu beachten. Neben skriptbasierten Installationen ohne umfangreichen Zusatzaufwand, wie der Installation des Datenbank-Tools SQL Developer oder dem Automatisierungstool Apache Ant, sind gemäß der Anforderungen auch umfangreiche Software-Komponenten einzurichten. Ein Beispiel ist das Datenbankmanagementsystem von Oracle. Auf Grund der Tatsache, dass für die Installation dieses Systems bereits Module innerhalb der PuppetForge bereitgestellt werden, sollte diese nach Einarbeitung in das entsprechende Modul keine besondere Problemstellung darstellen. Eine andere Situation ergibt sich bei Programmen, die speziell für die Entwicklung von Intershop-Systemen verwendet werden. Hierzu gehört die EnfinitySuite selbst sowie die Entwicklungsumgebung EnfinityStudio. Da diese Komponenten aufgrund ihrer Spezialisierung nur von einer begrenzten Anzahl Entwicklern genutzt werden, sind für sie häufig keine vorgefertigten Module vorhanden. Aus diesem Grund müssen diese selbst erstellt werden. Soll dabei ein hohes Maß an Stabilität und Kompatibilität erzielt werden, führt das zu potentiell sehr zeitintensiven Entwicklungsaufwänden.

Die Anforderungen des Hybris-Teams weisen in vielen Gesichtspunkten Parallelen mit den bereits umgesetzten Konfigurationen auf. So sollen alle Installationen auf Basis eines Debian-Systems durchgeführt werden, welches dem bisher verwendeten Ubuntu-System weitgehend ähnelt.[33] Die Installation vieler der benötigten Softwarekomponenten wie Vim, MySQL oder varnish wurden außerdem bereits im Rahmen der bisherigen Umsetzung in Form von entsprechenden Puppet-Modulen oder -Klassen realisiert. Für einen Großteil der übrigen Anwendungen konnten passende Puppet-Module im Repository gefunden werden. Doch auch hier bildet die Installation und Konfiguration des E-Commerce-Systems den zentralen Schwerpunkt, da auch für Hybris zum aktuellen Zeitpunkt keine Module im Puppet-Repository zur Verfügung stehen.

### 4.3.2 **Umsetzungen**

Auf Basis der ermittelten Anforderungen der Java-Entwicklungsabteilungen wurde zum Zeitpunkt der Fertigstellung dieser Arbeit damit begonnen, die Konfigurationen der Intershop-VM zu automatisieren. Da innerhalb des Vagrant Box-Repository[34] keine SLES-Box mit dem notwendigen Patchlevel 3 aufgefunden werden konnte, bildete deren Erstellung den ersten Schritt der Realisierung. Anschließend wurde das Vagrantfile entsprechend der Spezifikationen der virtuellen Hardware angepasst und die Box eingebunden. Anschließend wurde die Benutzererzeugung für das Systems automatisiert. Hierfür wurde das bereits vorhandene users Modul genutzt, wobei die Passwort-Codierung an das Verschlüsselungsverfahren des Systems angepasst wurde. Mit Hilfe des Oracle Puppet-Modules konnte anschließend die Datenbank auf dem System installiert und das integrierte Tool SQL Developer eingerichtet werden.

Aufgrund der unter 4.5 geschilderten Faktoren wurden bis zum Zeitpunkt des Abschlusses dieser Arbeit nur die soeben vorgestellten Realisierungen durchgeführt.

[33] [Can14]
[34] [Has14]

## 4.4 Abteilung Qualitätssicherung

### 4.4.1 Anforderungen

Neben den Entwicklungsabteilungen und dem UI-Team sollte auch die Abteilung für Qualitätssicherung die Möglichkeit erhalten, VMs automatisiert zu generieren. Die so erzeugten VMs sollen zwei zentrale Aufgabenstellungen erfüllen.
Die erste ist für bestimmte Aspekte des Software-Testzyklusses, wie beispielsweise die Durchführung automatisierter Browsertests, eine Umgebung mit möglichst flexibel definierbaren Charakteristika bereitzustellen. Dabei ist es wichtig, dass diese alle notwendigen Test-Tools bereits vorinstalliert und vorkonfiguriert bereitstellt. Das ist auch deshalb von Interesse, da innerhalb der Qualitätssicherung der dotSource GmbH eine flexible Aufgabenteilung herrscht und mit unterschiedlichen Arten von Tests häufig auch die Verwendung unterschiedlicher Tools verbunden ist. Eine entsprechend vorkonfigurierte Umgebung würde somit die Einarbeitungsaufwände für unterschiedliche Installations- und Konfigurationsprozesse stark reduzieren.

Die zweite Aufgabenstellung ist die Bereitstellung einer konfigurierten Entwicklungsumgebung die für bestimmte Arten von Test notwendig ist. Hierzu gehört beispielsweise die Durchführung von Update-Test. Diese werden unter anderem innerhalb der dotSource entwickelten Modulen des E-Commerce-Systems Magento durchgeführt, um das Verhalten bei Migration auf eine andere Magento-Version zu überprüfen.

### 4.4.2 Umsetzungen

Nachdem die konkreten Einsatzszenarien der Abteilung ermittelt und die entsprechenden Anforderungen abgeleitet wurden, konnte mit der Realisierung des Konfigurationssystems begonnen werden. Eine enge Zusammenarbeit mit den Mitarbeitern der Qualitätssicherung sollte dabei die zukünftig eigenverantwortliche Wartung und Weiterentwicklung des Systems ermöglichen.

Die Basis weiterer Entwicklungen stellte das Vagrant-Projekt dar, welches zuvor für das PHP-Team entwickelt wurde. Aus diesem wurden zahlreiche Module übernommen, aber auch nicht verwendete entfernt, um die Konfigurationszeit bei Generierung der VM oder dem erneuten Laden der Provisioner zu reduzieren und die Komplexität des Projektes zu reduzieren. Mit

dieser Zielstellung wurden die Module varnish, magento, n98magerun, phpsotm und monodb gelöscht und die betreffenden Einträge aus der site.pp entfernt. Auch die im PHP-Projekt entwickelte Trennung von eigenen Modulen in einem dotSource-Ordner und heruntergeladenen Modulen im puppetForge-Ordner wurde in das QA-Projekt übernommen. Um die benötigte IDE Eclipse zu installieren und einzurichten wurde ein neues Modul des Puppet-Repositories verwendet. Für die Installation einer Vielzahl weiterer QA-Tools, welche aber nur geringe Zusatzkonfigurationen notwendig machen, wurden jeweils separate Klassen innerhalb des eigens bereitgestellten Tools-Modul erstellt.

## 4.5 Einarbeitung und Problemstellungen

Nachdem in den vorangegangen Abschnitten dieses Kapitels die im einzelnen durchgeführten Arbeiten vorgestellt wurden, sollen nun die dabei aufgetretenen Problemstellungen aufgezeigt und erläutert werden. Hintergrund hierfür ist der Umstand, dass sich derartige Problemsituationen nur schwer für die Erfassung im Prozessmodell quantifizieren lassen, aber dennoch bei der Bewertung der Effizienz des Automatisierungssystems zu berücksichtigen sind. So sollen die folgenden Ausführungen das Verständnis für die im Kapitel 5 thematisierten Einarbeitungs- und Erstellungszeiten verbessern.

Ein wichtiger Faktor im Umgang mit Vagrant und Puppet sind die notwendigen Kenntnisse, die für die Entwicklung mit beiden System vorausgesetzt werden.
Zur Steuerung von Vagrant wird ein proprietärer Befehlssatz verwendet, mit dem auf Funktionen wie die Erstellung und Zerstörung von VMs, die Installation zusätzlicher Plugins und weitere Funktionen zugegriffen wird. In der Regel wird dafür ein Command-Line Interface CLI verwendet.[35] Zwar stehen zur Minimierung der notwendigen Kenntnisse online eine Reihe von Lösungen wie das Vagrant-Plugin der IDE PhpStorm[36] mit grafischer Oberfläche bereit, jedoch konnte kein Tool ermittelt werden, welches den gesamten Vagrant-Funktionsumfang unterstützt. Neben dem Befehlssatz erfolgt die Steuerung von Vagrant durch die Deklarationen innerhalb des in Abschnitt 2.2 vorgestellten Vagrantfiles. Dieses wird zwar auf Basis der Programmiersprache Ruby erstellt, verlangt diesbezüglich aber keine

[35] [Cor13]
[36] [Bra14]

speziellen Kenntnisse, da es sich nur um triviale Variablenzuweisung handelt.[37] Sind jedoch zusätzliche Funktionen, wie Beispielsweise die Überprüfung der Existenz einer bestimmten Datei auf dem Hostsystem erforderlich, werden rudimentäre Ruby-Kenntnisse benötigt. Weitere Kenntnisse setzt der Umgang mit der API der jeweiligen Virtualisierungssoftware voraus. Den Standard bildet hierbei VirtualBox von Oracle, welche auch im vorliegenden Projekt eingesetzt wird. Um gezielte Konfigurationen der virtuellen Hardware sowie des generellen Verhaltens der zu erstellenden VM zu konfigurieren, sind umfassende Kenntnisse der bereitgestellten Konfigurationspallette notwendig. [38]

Das für die Beschreibung gewünschter Konfigurationen verwendete Puppet-Skript basiert ebenfalls auf Ruby. Doch auch hier sind keine besonderen Kenntnisse dieser Sprache notwendig, da die deklarative Syntax eher an die JavaScript Object Notation (JSON) erinnert. Die Basisbestandteile des Puppet-Scripts bilden jedoch die verschiedenen Ressourcen-Typen mit ihren spezifischen Eigenschaften, welche proprietäre Sprachbestandteile darstellen.[39] Die Verwendung von Puppet setzt somit Kenntnisse zur Verwendung dieser Komponenten voraus. Jede Puppet-Ressource ist dabei an ein bestimmtes Set von Konfigurationen auf dem Zielsystem gekoppelt. Um bei ihrer Verwendung das Auftreten von Fehlersituationen zu vermeiden, ist deshalb auch umfassendes Wissen hinsichtlich möglicher Einstellungen des jeweiligen Betriebssystems und der verwendeten Softwarekomponenten notwendig.
Neben Kenntnissen zur grundsätzlichen Verwendung von Puppet benötigt auch der Umgang mit Modulen in vielen Fällen eine umfangreiche Einarbeitung. Diese kommen meist bei der Installation und Konfiguration umfangreicher Software, wie beispielsweise eines lokalen Webservers, zum Einsatz. Module werden zum einen durch das Unternehmen Puppet Labs und zum anderen durch die Puppet-Community bereitgestellt. Im Gegensatz zur generellen Struktur existieren dabei keine konkreten Vorgaben zur Dokumentation eines solchen Moduls. Dieser Umstand führt bei Community Modulen häufig zu einer nur rudimentären Beschreibung, was zu einer signifikanten Steigerung der Einarbeitungszeit führen kann. Dieses Problem wird besonders dann deutlich wenn berücksichtigt wird, dass beispielsweise das Apache-Module von Puppet Labs aus 75 einzelnen Manifesten besteht und die zentrale

[37] [Mit13], S. 20 f.
[38] [Oco14]
[39] [Pup14]

Apache-Class bereits 44 Übergabe-Parameter bereitstellt.[40] Dieser Umfang resultiert aus dem Bestreben, eine möglichst hohe Flexibilität bei der Verwendung, in Verbindung mit einer maximalen Kompatibilität zu verschieden Betriebssystemen zu bieten.[41] Komplexität und Umfang solcher Module stellen aber auch dann ein Problem dar, wenn Software eingerichtet werden soll, die nur von einer begrenzten Anzahl von Nutzern verwendet wird. In diesem Fall steht häufig kein passendes Modul zur Verfügung, was die Entwicklung einer eigenen Lösung notwendig macht. Da eigene Entwicklungen aber ebenfalls den Richtlinien vorhandener Module genügen sollten, kann ein solches Vorgehen einen sehr zeitaufwändigen Prozess darstellen.

Doch auch bei adäquater Einarbeitung ist das Auftreten von Problemstellungen häufig nicht vermeidbar. Ein Grund dafür ist, dass Vagrant mit zwei Jahren Entwicklungszeit, seit der Gründung des Unternehmens HashiCorp, ein noch junges Projekt ist. Dass dabei der Entwicklungsprozess noch nicht abgeschlossen ist, zeigt die hohe Zahl von Versions-Updates innerhalb eines kurzen Zeitraumes, welche auch in der Release-Historie des Entwicklungs-repository[42] zu erkennen sind. Ein Nebeneffekt dieses Entwicklungsprozesses kann aber auch das Auftreten von Bugs sein. Dabei arbeitet Hashicorp mit der Vagrant-Community zusammen um so neue Problemstellungen mittels eines Bug-Trackers[43] kontinuierlich erfassen und bearbeiten zu können. Doch trotz umfangreicher Maßnahmen, die Anzahl von Bugs zu minimieren, stellen das Auftreten und die Behebung solcher Fehlersituationen einen zeitlichen Einflussfaktor auf die Bearbeitung von Vagrant-Projekten dar.
Neben der Weiterentwicklung von Vagrant durch HashiCorp werden viele Zusatzfunktionen durch sogenannte Plugins realisiert. Diese werden meist durch die Community erstellt und erweitern beispielsweise das Spektrum unterstützter Virtualisierungssoftware oder den Funktionsumfang bei der Verwendung bestimmter Provisioner.[44] Aufgrund der Tatsache, dass die Weiterentwicklung solcher Plugins in der Verantwortung des jeweiligen Erstellers liegt, ist nicht immer sicher gestellt, dass notwendige Anpassungen bei Weiterentwicklung angrenzender Systeme wie Vagrant oder dem jeweiligen Provisioner, zeitnah durchgeführt

[40] [Pul14]
[41] [Pla14]
[42] [Mit01]
[43] [Mit02]
[44] [Mit03]

werden. Somit entsteht auch an dieser Stelle das Potential für notwendige Nachbearbeitung im Projektverlauf.

Die Grundlagen der soeben vorgestellten zeitlichen Einflussfaktoren sind die Erfahrungen, die im Rahmen der Weiterentwicklung des Projektes, das dieser Arbeit zu Grunde liegt, gesammelt werden konnten. Aus diesem Grund soll an dieser Stelle noch einmal ihre Relevanz für die Bewertung der Effizienz des erstellten Systems hervorgehoben werden.

### 4.6 Offene Umsetzungen und Ausblick

Im Rahmen der praktischen Realisierungen dieser Arbeit wurde als zentraler Schritt das Konfigurationssystem des PHP-Teams fertiggestellt. Dieses Vorgehen diente zum einen dazu, für alle weiteren Entwicklungen eine gut strukturierte Projektvorlage bereitzustellen und zum anderen, um alle im Prozessmodell erfassten Aktivitäten abzubilden zu können und so eine stabile Vergleichsbasis bereitzustellen. Neben den Weiterentwicklungen im PHP-Team wurden außerdem verschiedene Konfigurationen der Java-VM für Intershop-Projekte umgesetzt und die Fertigstellung der VM der Qualitätssicherung unterstützt.

Aufgrund der im vorangegangenen Abschnitt beschriebenen Einflüsse auf die Umsetzung des Konfigurationssystems, sind auch zum Zeitpunkt der Fertigstellung dieser Arbeit noch eine Reihe von Features zu realisieren. Zu diesen gehören die Erweiterung des Java-Projektes hinsichtlich der noch offenen Konfigurationsanforderungen sowie die vollständige Realisierung der Projekte des Hybris- und des User Interface-Teams der dotSource.

Neben den konkreten Anforderungen, die für weitere Entwicklungen bestehen, soll in den nachfolgenden Abschnitten erläutert werden, welche Funktionen für einen langfristigen Einsatz des Systems noch denkbar wären.

Um die Einarbeitungsaufwände für die Erweiterung von Konfigurationsskripte zu minimieren, sollten für diese klare Styling-Regeln existieren. Die Software Puppet-Lint gibt zu diesem Zweck eine Reihe von Best-Practices vor, die sich für den Umgang mit Puppet bewährt haben. Um die Einhaltung dieser Regeln zu gewährleisten und den personellen Aufwand dabei so gering wie möglich zu halten, kann Puppet-Lint auch in Jenkins integriert

werden. Eine solche Integration ermöglicht es, die Qualität der im Versionsverwaltungssystem der dotSource abgelegten Skripte automatisiert zu prüfen.

Ein weiter Ansatzpunkt für die Weiterentwicklung des Systems ist Anwenderfreundlichkeit. Innerhalb der aktuellen Umsetzung müssen alle Konfigurationsanpassungen der zu erstellenden VM direkt in den verantwortlichen Puppet-Skripten durchgeführt werden. Dieses Vorgehen weist eine Reihe von Nachteilen auf. So können Einstellungen nur von erfahrenen Entwicklern durchgeführt werden oder diese benötigen umfangreiche Einarbeitungszeiten. Außerdem besteht durch den uneingeschränkten Eingriff in die Konfiguration des Systems ein erhöhtes Risiko zur Schaffung von Fehlersituationen. Einen Ansatz zur Minimierung dieser Risiken stellt die Integration einer grafischen Benutzeroberfläche dar. Beispiele für ein solches Vorgehen stellen die Systeme Protobox[45] und Rove dar, welche eine Weboberfläche mit einer Vielzahl vorgegebener Einstellungen bereitstellen. Zur Verdeutlichung ist in Anlage 2 die Oberfläche von Rove dargestellt. Aus den Daten, die vom Nutzer in dieses Web-Interface eingegeben werden, werden automatisch das entsprechende Vagrantfile sowie die Skripte des vorgegebenen Provisioners erstellt. Durch das breite Spektrum von verwendbarer Software und der Zahl der damit verbundenen Einstellungen können solche Oberflächen aber nur einen Teil der möglichen Konfigurationen abdecken. Um also die einfache Bedienung einer grafischen Oberfläche mit der Flexibilität der individuellen Konfiguration zu verbinden, könnte die Entwicklung einer firmeneigenen GUI eine adäquate, wenn auch aufwändige Lösung, darstellen.

Auch die Zentralisierung des potentiell zeitaufwändigen automatisierten Erstellungsprozesses der virtuellen Maschinen stellt eine Möglichkeit zur weiteren Verbesserung des Systems dar. Wird dieser Prozess auf einem Server, optimaler Weise außerhalb der regulären Arbeitszeit, durchgeführt, können auch zeitintensive Operationen wie der Import umfangreicher Datenbanken automatisch ablaufen, ohne den Projektverlauf zu beeinflussen. Die so generierten VMs müssen dann nur noch von den Projektmitgliedern kopiert werden. Ein solches Vorgehen würde zwar den Kopieraufwand erhöhen, doch da die betreffenden Daten auch lokal importiert werden müssten, wird der zeitliche Aufwand nur verlagert. Die soeben

[45] [Pro14]

beschriebene Technik ist jedoch nicht für den Einsatz mit jeder VM geeignet, da bei häufigen Änderungen und einem nur geringen Volumen von Projektdaten eine lokale Generierung effizienter scheint.

Durch die beiden soeben vorgestellten Erweiterungsmöglichkeiten könnte das Management-system für virtuelle Maschinen möglicher Weise noch effizienter gestaltet werden. Vor einer konkreten Umsetzung müssen diese aber noch einer detaillierten Evaluierung unterzogen werden. Diese Überlegungen würden aber an dieser Stelle zu weit führen und somit den Rahmen dieser Arbeit überschreiten.

# 5 Messung und Auswertung der Ergebnisse

In den vorangegangenen Kapiteln wurde erläutert welche theoretischen Grundlagen notwendig waren, um ein Prozessmodell für die Verwendung virtueller Maschinen in der dotSource GmbH zu entwickeln und wie dieses konkret umgesetzt wurde. Im Anschluss daran wurde auf die technische Weiterentwicklung des bestehen Konfigurationssystems zur Realisierung der ermittelten Anforderungen sowie damit verbundene Problemstellungen eingegangen.

Dieses Kapitel beschäftigt sich mit der Analyse des wirtschaftlichen Nutzens des Systems hinsichtlich des weiteren Einsatzes im Unternehmen. Zwar wurde bereits in der Praxisabreit „Automatisierte Erstellung und Konfiguration dynamischer virtueller Maschinen“ umrissen, wie die Effizienz im Unternehmen mit Hilfe des beschriebenen Systems gesteigert werden könnte, doch es wurden bisher keine empirischen Daten vorgelegt, die diese These verifizieren. Nachfolgend werden die dafür notwendigen Informationen mittels des erstellten Modells erfasst und ausgewertet.

## 5.1 Szenario

Um das in Kapitel 3.3 entwickelte Prozessmodell für den Vergleich der Effizienz des praktizierten VM-Managements mit der entwickelten Automatisierungslösung einsetzen zu können, werden definierte Testszenarien vorausgesetzt. Diese bilden den Rahmen für die Erfassung der zeitlichen Ressourcen, die beim Durchlaufen der Aktivitäten des Modells aufgewendet werden. Alle dabei gesammelten Daten werden dokumentiert und nachfolgend ausgewertet.

Obwohl sich die Softwarelandschaften innerhalb der unterschiedlichen Abteilungen, in denen das entwickelte System zum Einsatz kommen soll, im Einzelnen stark unterscheiden, sind gemeinsame Strukturen erkennbar. So benötigt in der Regel jedes Entwickler-Team neben einer Reihe von Entwicklungswerkzeugen ein Datenbankmanagementsystem, das jeweils eingesetzte E-Commerce-System und die Daten des zu bearbeitenden Projektes. Aufgrund dieser Parallelen und des unterschiedlichen Projektfortschrittes in den einzelnen Abteilungen wurde die Entscheidung getroffen, das Vagrant-Projekt einer Abteilung repräsentativ für die Datenerfassung in den einzelnen Testszenarien einzusetzen. Da die VM-Automatisierung des

PHP-Teams alle bis zu diesem Zeitpunkt gestellten Anforderungen erfüllt, bildet dieses Projekt die Basis für die durchzuführenden Tests.

Um die Praxisnähe des Vergleiches zu maximieren, werden alle Messungen anhand fiktiver Projektsituation durchgeführt, deren Eckdaten im Folgenden erläutert werden.
Es wird angenommen, dass für vier Projekte der dotSource GmbH jeweils eine neue virtuelle Maschine als Arbeitsumgebung für das Projekt-Team erstellt werden soll. Die einzelnen Projekt-Teams sind dabei unabhängig voneinander. Außerdem besteht jedes Team aus einer anderen Zahl von Projektteilnehmern, die wie folgt verteilt sind.

- **Projekt 1**: 4 Personen
- **Projekt 2**: 6 Personen
- **Projekt 3**: 12 Personen
- **Projekt 4**: 8 Personen

Eines der Teammitglieder erstellt die VM, welche dann von den übrigen Mitarbeitern über das Firmennetzwerk kopiert wird.

Tabelle 2 zeigt die Anforderungen an die virtuelle Hardware der VMs, die bei der Erstellung zu berücksichtigen sind. Hierbei bestehen keine Unterschiede zwischen den Projekten.

**Tabelle 2: Anforderung - virtuelle Hardware**

| **Einstellung** | **Wert** |
|---|---|
| Arbeitsspeicher | 8192 MB |
| Prozessorkerne | 4 |
| 3D-Beschleunigung | aktiviert |
| Portweiterleitung | Host 8080 auf Gast 80<br>Host 443 auf Gast 443 |
| Gemeinsamer Ordner | Host: C:\Users\[Benutzername]\data\<br>Gast: /home/data/ |
| Drag and Drop | Bidirektional zwischen Gast und Host |
| Gemeinsame Zwischenablage | Bidirektional zwischen Gast und Host |

Als Betriebssystem für alle virtuellen Maschinen wird Ubuntu in der Version 14.04 LTS eingesetzt, welches unter dem Namen Trusty Tahr entwickelt wird. Dieses wird mit der Standardoberfläche Ubuntu GNOME bereitgestellt und bietet durch entsprechende Verknüpfungen auf der Arbeitsoberfläche direkten Zugriff auf die installierte IDE und das Kommandozeilen-Tool Bash.

Neben den Hardware-Spezifikationen und dem Betriebssystem sollen den Entwicklern eine Reihe vorkonfigurierter Software-Pakete bereitgestellt werden, um einen möglichst schnellen Produktiveinsatz der VM zu gewährleisten. Die nachfolgende Tabelle fasst die konkreten Software-Anforderungen der einzelnen Projekt-Teams zusammen. Die Markierung einer Tabellenzeile bedeutet dabei, dass die jeweilige Software für das Projekt installiert bzw. die betreffenden Daten kopiert werden müssen.

**Tabelle 3: Anforderungen - Software / Daten**

| Software / Daten | Projekt 1 | Projekt 2 | Projekt 3 | Projekt 4 |
|---|---|---|---|---|
| Apache Webserver | ■ | ■ | ■ | ■ |
| MySQL | ■ | | ■ | ■ |
| PhpMyAdmin | | | ■ | ■ |
| PhpStrom | ■ | ■ | | ■ |
| Codestyle-Tools[46] | | ■ | ■ | |
| PHP | ■ | ■ | ■ | ■ |
| Varnisch | | ■ | ■ | ■ |
| MongoDB | | ■ | | ■ |
| JDK | | | | |
| Apache Ant | ■ | ■ | ■ | ■ |
| PostFix[47] | | | | |
| SVN Projektdaten[48] | ■ | ■ | | ■ |
| Datensätze[49] | ■ | | | ■ |

[46] Installation des PHP CodeSniffers und PHP Mess Detectors zur Überwachung der Code-Qualität
[47] Einrichten der E-Mail-Weiterleitung auf der VM mittels des Mail Servers Postfix
[48] Checkout der vorhandenen Projektdaten aus dem SVN auf die VM
[49] Einspielen der vorhandenen Datensätze in die installierte MySQL-Datenbank

Die Anforderungen der einzelnen Projekt-Teams wurden unterschiedlich gestaltet, um die Installations- und Konfigurationsaufwände der Einzelkomponenten zu relativieren.

**Neben den benötigten Software-Komponenten, ist ebenfalls der Zugriff auf die vorhandenen Projektdaten zu gewährleisten. In Projektdaten zu gewährleisten. In**

Tabelle 1 sind diese Arbeitsschritte in Form der Zeilen SVN Projektdaten und Datensätze aufgeführt. Das Datenvolumen der Projektdaten im Testszenario umfasst dabei 30,1 MB und die Größe der Datensätze 10,4 MB.

Nachdem der Erstellungsprozess der VM abgeschlossen wurde, soll das Update der einzelnen VMs abgebildet werden. Hierfür wird angenommen, dass diese bereits auf die einzelnen Projektteilnehmer verteilt wurden. Um jede der Aktivitäten des Modells abzubilden, wird das Update einer Komponente jeweils eine De- bzw. Neuinstallation und eine Beispiel-konfiguration durchgeführt. Welche Maßnahmen dabei im Einzelnen durchgeführt werden, ist der folgenden Übersicht zu entnehmen.

**Tabelle 4: Anforderungen - Updateprozess**

| Aktivität | Maßnahme |
|---|---|
| Update | IDE PhpStorm von Version 6 auf 7 aktualisieren |
| De- / Neuinstallation | Entfernen des Tools Apache Ant / Installation des JDK |
| Konfiguration | Einrichtung einer E-Mail-Weiterleitung mittels PostFix |

Nachdem alle Aktualisierungen durchgeführt wurden soll die VM wieder einen stabilen Zustand übergehen, der weiterhin eine reguläre Verwendung ermöglicht.

Des Weiteren wird angenommen, dass sich im Rahmen der Projektlaufzeit die Spezifikationen des Zielsystems, auf dem das Projekt betrieben werden soll, geändert hat und somit das Betriebssystem der VM ebenfalls auf die Ubuntu-Version 13.10 angepasst werden muss. Dieses Vorgehen repräsentiert den Änderungsprozess einer virtuellen Maschine. Da eine solche Anpassung der einzelnen VMs zu zeitaufwändig ist wird angenommen, dass ein

Mitarbeiter die virtuelle Maschine für die gegebenen Anforderungen neu erstellt und sie wie bisher im Projekt-Team verteilt wird.

Ein weiterer VM-Zustand, der innerhalb des Test-Szenarios berücksichtigt werden soll, ist der Fehlerzustand. Zu diesem Zweck wird angenommen, dass jeweils einer der Projektmitarbeiter seine VM nicht mehr starten kann. Eine erste Fehleranalyse ergab dabei, dass die Wiederherstellung eines stabilen Zustandes mehr Zeit in Anspruch nehmen würde als das erneute Kopieren und Einrichten der VM.

Mit Hilfe der in diesem Abschnitt beschriebenen Test-Szenarien können alle Zustände, die im Rahmen der Prozessmodellentwicklung als relevant eingeschätzt wurden, abgebildet werden. Somit wird eine objektive Vergleichbarkeit der beiden Vorgehensweisen ermöglicht. Da aber verschiedene Faktoren, wie die Einarbeitung in bestimmte Technologien, nur unzureichend innerhalb solcher Testszenarien erfasst werden können, wird auf diese noch einmal explizit im Rahmen der Auswertung eingegangen werden.

## 5.2 Manuelles Vorgehen

Im ersten Schritt des Vergleiches soll herausgestellt werden, welche Aufwände im Fall des Vorgehens ohne Konfigurationssystem, nachfolgend als manuelles Vorgehen bezeichnet, im Rahmen der beschriebenen Testszenarien anfallen. Zu diesem Zweck wurde der Prozess der manuellen VM-Erstellung und Verteilung im Falle vier verschiedener VMs verfolgt und dokumentiert. Die dabei ermittelten Zeitaufwände resultieren aus den Angaben der Mitarbeiter der Entwicklungsabteilung, die mit der Erstellung betraut waren, sowie aus unabhängig durchgeführten Messungen. Die installierte Software sowie vorgenommene Konfigurationen auf den einzelnen virtuellen Maschinen bildete dabei die Vorlage für das erstellte Testszenario.

Der Ablauf der Vorgänge, die für die manuelle Konfiguration durchgeführt wurden, folgte den Vorgaben des entwickelten Prozessmodels. Aufgrund unterschiedlicher Einarbeitungszeiten in verschiedene Konfigurationsprozesse und die Zeit, die zur Lösung aufgetretener Problemstellungen notwendig war, können für die meisten Aktivitäten nur Richtwerte angesetzt werden. Im Prozessmodell werden solche Aufwände, wie bereits beschrieben, durch

die entsprechenden Aktivitäten abgebildet, welche parallel zu allen Installations- und Konfigurationsprozessen ablaufen. Da das zeitliche Volumen dieser Aktivitäten vom Kenntnisstand des jeweiligen Entwicklers und anderen nicht zu pauschalisierenden Faktoren abhängt, lassen sich diese nur schwer bemessen. Die Erfassung von Aufwänden dient letztendlich dem Vergleich zweier unterschiedlicher Verfahren, wobei für beide gleichermaßen Einarbeitungs- und Problemlösungsaufwände zu berücksichtigen sind. Da bei einer Berücksichtigung dieser Aufwände für einen Vergleich kein Mehrwert an Information entsteht, wird dieser im Rahmen der Einzelbetrachtung nicht berücksichtigt und erst in der Auswertung näher erläutert.

**Tabelle 5: Zeitmessung - manuelle Konfiguration (Angaben in Minuten)**

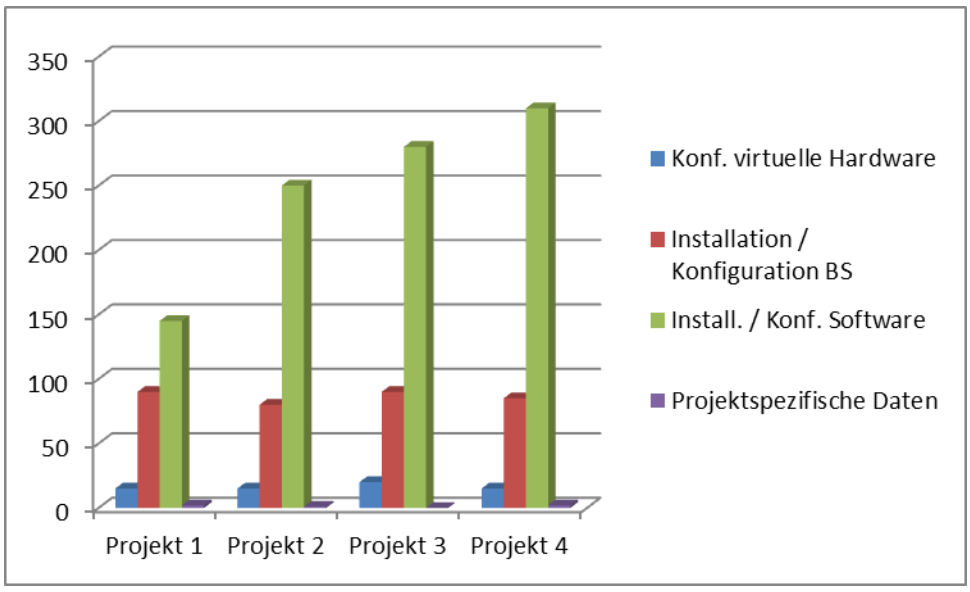
| | **Projekt 1** | **Projekt 2** | **Projekt 3** | **Projekt 4** |
|---|---|---|---|---|
| Konf. virtuelle Hardware | 15 | 15 | 20 | 15 |
| Installation BS | 35 | 35 | 40 | 35 |
| Konfiguration BS | 55 | 45 | 50 | 50 |
| Install. / Konf. Apache Webserver | 60 | 70 | 65 | 60 |
| Installation PHP | 30 | 40 | 25 | 35 |
| Install. / Konf. MySQL Percona | 35 | - | 40 | 45 |
| Install. / Konf. phpMyAdmin | - | - | 40 | 40 |
| Installation PhpStorm | 15 | 15 | - | 15 |
| Installation Codestyle-Tools | - | 30 | 40 | - |
| Install. / Konf. Varnish | - | 60 | 65 | 70 |
| Install. / Konf. MongoDB | - | 30 | - | 40 |
| Installation Apache Ant | 5 | 5 | 5 | 5 |
| Install. / Konf. Postfix | - | - | - | - |
| Projektdaten Clonen (SVN) | 1 | 1 | - | 1 |
| Datensätze in DB importieren | 1 | - | - | 1 |
| **Gesamtaufwand** | 252 | 346 | 390 | 412 |

Tabelle 5 zeigt die erfassten Zeiten der verschiedenen Aktivitäten im Rahmen der einzelnen VM-Erstellungen. Die dabei erfassten Abweichungen im Gesamtaufwand, resultieren aus den

unterschiedlichen Softwareinstallationen sowie den individuellen Fähigkeiten des ausführenden Mitarbeiters.

Nach dem die Einzelmessungen im Rahmen der VM-Erstellung durchgeführt wurden, sollen diese nun so aufbereitet werden, dass eine optimale Vergleichbarkeit mit dem automatisierten Vorgehen gewährleistet werden kann. Dabei ist es wichtig zu berücksichtigen, dass bei der Durchführung der automatisierten Konfigurationen durch den Provisioner Puppet Einstellungsressourcen zu großen Teilen parallel abgearbeitet werden. Somit ist eine separate Zeitmessung der Einzelkonfigurationen nur schwer zu realisieren. Da für einen Vergleich jedoch nicht die einzeln durchgeführten Konfigurationen, sondern vielmehr der Zeitaufwand zur Herstellung des gewünschten Zielzustandes des System von Interesse ist, können diese Aufwände zusammengefasst werden. Abbildung 7 zeigt das Resultat der beschriebenen Abstraktion. Um die Vergleichbarkeit dabei weiterhin sicherzustellen ist zu beachten, dass die Softwarebestandteile des manuell konfigurierten Systems kongruent mit denen des automatisierten sein müssen.

**Abbildung 7: Zeiterfassung – Zfg. der Einzelkonfigurationen (Angaben in Minuten)**

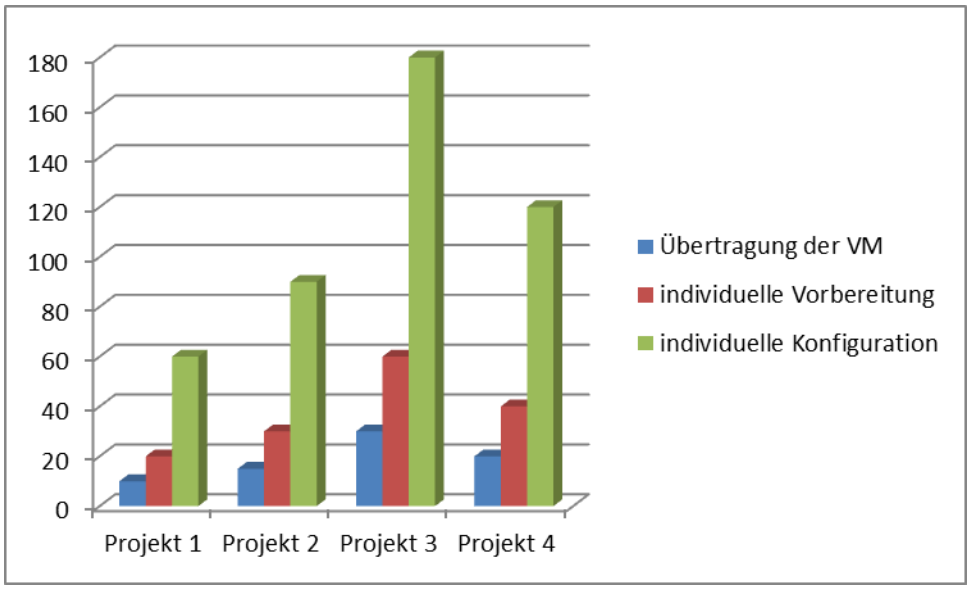



**Quelle: eigene Darstellung**

Nach der Erfassung der Konfigurationsaufwände für die Erstellung der VM soll im Folgenden deren Verteilung an die einzelnen Projektmitglieder betrachtet werden. Das Gesamtdatenvolumen der erzeugen VM beträgt durchschnittlich 9,1 GB, wodurch die Übertragung im

internen Firmennetzwerk rund 2,5 Minuten in Anspruch nimmt. Dabei wird vorausgesetzt, dass das die Kopiervorgänge nicht exakt zur selben Zeit stattfinden und dass die Übertragungsleistung des Netzwerkes durch keine äußeren Faktoren beeinflusst wird. Neben dem Kopieren der VM werden im Modell auch die Durchführung individueller Vorbereitungen sowie nachträglicher Konfigurationen berücksichtigt. Da diese Aufwände von den einzelnen Projektteilnehmern abhängig sind, werden hierfür Richtwerte verwendet die auf Erfahrungswerten verschiedener Mitarbeiter der Entwicklungsabteilung beruhen. Somit werden für individuelle Vorbereitungen 5 Minuten und für individuelle Konfigurationen 15 Minuten angesetzt. Da nun alle Aufwände des Verteilungsprozesses erfasst wurden, werden diese für die jeweilige Anzahl der Projektmitglieder summiert. Das Ergebnis ist im nachfolgenden Diagramm in Abbildung 8 dargestellt.

**Abbildung 8: Zeiterfassung – manuelle Verteilung (Angaben in Minuten)**

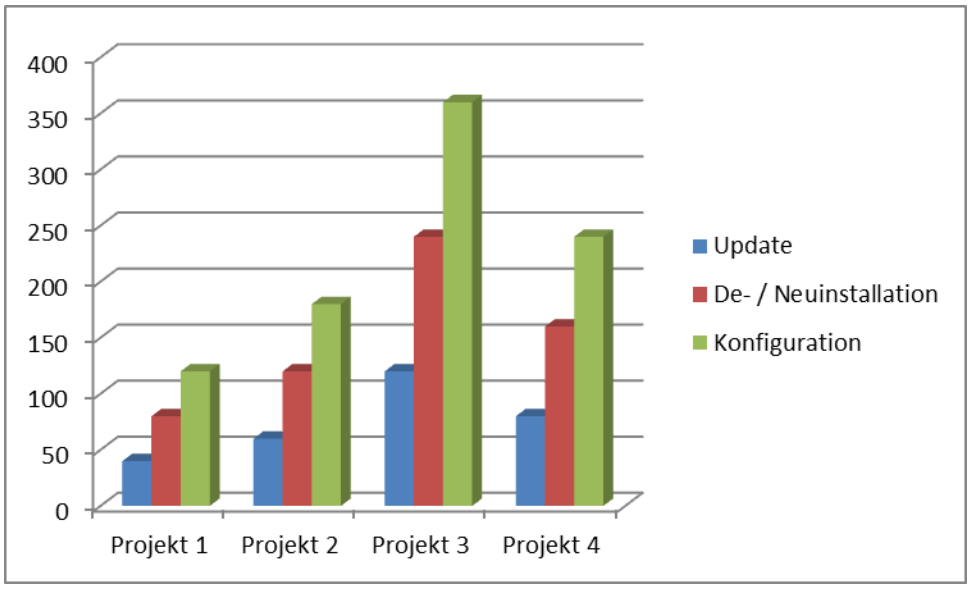



**Quelle: eigene Darstellung**

Die nächste im Testszenario beschriebene Situation ist die Aktualisierung von virtuellen Maschinen. Hierbei wurden die beschrieben Aktionen für Update, Installation und Konfiguration durchgeführt. Da für die Durchführung dieser Prozesse auf den verschiedenen virtuellen Maschinen keine signifikanten Abweichungen festgestellt werden konnten, wurden die entsprechenden Messungen für Projekt 1 als Basiswert angenommen. Dabei wurde für das Update von PhpStrom 10 Minuten, für die Installation des JDK und Desinstallation von Ant insgesamt 20 Minuten und die Einrichtung der E-Mail-Weiterleitung 30 Minuten benötigt. Da

diese Arbeiten bei einem lokalen Updatevorgang von allen Projektteilnehmern durchzuführen sind, zeigt Abbildung 9 die resultierenden Gesamtaufwände für die einzelnen Projektteams.

**Abbildung 9: Zeiterfassung - manuelles Update (Angaben in Minuten)**




**Quelle: eigene Darstellung**

Die Änderung der VM im Testszenario beschreibt den Prozess des Wechsels des Betriebssystems. Da hierfür die komplette Neuerstellung der VM und die anschließende Verteilung im Projektteam notwendig sind, werden die für diese Prozesse erfassten Aufwände in diesem Fall erneut angesetzt. Analog wird auch im Fehlerzustand der VM verfahren, allerdings wird in diesem Fall keine Neuerstellung sondern nur der Verteilungsaufwand an das jeweilige Projektmitglied berücksichtigt.

## 5.3 Verwendung des Konfigurationssystems

Nachdem im vorangegangen Kapitel die aufgewendeten Zeiten, für die Prozesse des Testszenarios beim manuellen Vorgehen dokumentiert wurden, wird diese Erfassung nun auch für das automatisierte System durchgeführt. Die Messungen wurden dabei wie folgt realisiert.

Wird im Rahmen der Messungen ein Vagrant-Kommando ausgeführt, wird dessen Ausgabe in eine Log-Datei geschrieben. Wie im Auszug eines solchen Logs in Abbildung 10 zu erkennen ist, wird zu Beginn der Ausführung des Kommandos ein Zeitstempel erzeugt und in das Log

geschrieben. Durch innerhalb des Vagrantfiles eingebundene Shell-Skripte werden weitere Zeitstempel vor und nach der Ausführung des Provisioners gesetzt. Durch die Zeitdifferenz zwischen den verschiedenen Zeitstempeln, lässt sich die Zeit für die Erzeugung und Einrichtung der VM sowie für die Durchführungen der Konfigurationen am System separat bestimmen.

**Abbildung 10: Auszug Vagrant-Log**

```
12:03:14,70
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'vagrant-ubuntu-trusty'...
←[KProgress: 30%←[KProgress: 40%←[KProgress: 50%←[KProgress: 60%←[KProgress: 70%←
[KProgress: 90%←[K==> default: Matching MAC address for NAT networking...
==> default: Setting the name of the VM: Vagrant_PHP
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: bridged
==> default: You are trying to forward to privileged ports (ports <= 1024). Most
==> default: operating systems restrict this to only privileged process (typically
==> default: processes running as an administrative user). This is a warning in case
==> default: the port forwarding doesn't work. If any problems occur, please try a
==> default: port higher than 1024.
==> default: Forwarding ports...
    default: 80 => 8080 (adapter 1)
    default: 443 => 443 (adapter 1)
    default: 22 => 2222 (adapter 1)
==> default: Running 'pre-boot' VM customizations...
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
```

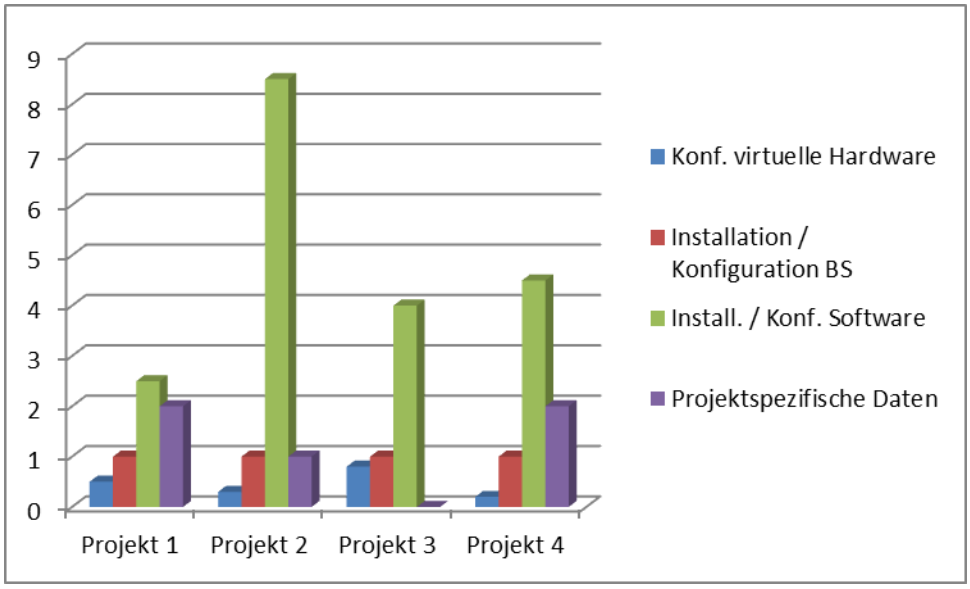
**Quelle: eigene Abbildung**

Wie bereits in den vorangegangenen Kapiteln erläutert, bilden das Vagrantfile und die notwendigen Puppet-Skripte die Grundlage für die automatisierte Erstellung einer VM. Diese werden nach ihrer Erstellung für unterschiedliche Projekte wiederverwendet und dabei nur um bestimmte Konfigurationen erweitert. Aus diesem Grund wird der Aufwand für ihre Erstellung zwar im Rahmen der folgenden Auswertung berücksichtigt, fließt jedoch nicht direkt in die Zeitmessung ein. Für das vorliegende Szenario wird somit angenommen, dass bereits für alle zu installierenden Software-Komponenten passende Puppet-Skripte vorhanden sind.

Den ersten Teil der Messung bildet wie bereits beim manuellen Vorgehen die Zeit für die Erstellung einer VM. Da dieser Schritt hier aber lokal auf dem Rechner jedes Nutzers durchgeführt wird, sind die ermittelten Zeiten mit der Anzahl der Projektteilnehmer zu multiplizieren. Der erste Schritt im Erstellungsprozess ist die Generierung der VM aus der

bereitgestellten Box und die anschließende Konfiguration der virtuellen Hardware. Da alle Projekte dieselbe Ubuntu-Box als Basis verwenden und identische Hardwareanforderungen aufweisen, kann angenommen werden, dass dabei nahezu deckungsgleiche Ergebnisse erzielt werden. Die Ergebnisse der Messungen für die unterschiedlichen Konfigurationen der einzelnen Testprojekte wurden, in Abbildung 11 zusammengefasst. Sie bestätigen die Annahme ähnlicher Konfigurationszeiten der virtuellen Hardware und zeigen, dass die Installation und Konfiguration der Softwarekomponenten den größten Aufwand im Rahmen der VM-Erstellung darstellt.

**Abbildung 11: Zeitmessung – automatisierte Erstellung (Angaben in Minuten)**




**Quelle: eigene Darstellung**

Nach der Erstellung der VM wurde im Rahmen der Messungen am manuellen Vorgehen die Verteilung auf die einzelnen Teammitglieder untersucht. Da im Falle der automatisierten Erstellung der notwendige Prozess auf dem System des Endnutzers abläuft, ist ein nachträglicher Datentransfer nicht mehr notwendig. Allerdings setzt die Verwendung des Konfigurationssystems bestimmte Bedingungen auf dem lokalen System voraus, deren Erfüllung Parallelen zur manuellen Verteilung aufweist. Hierzu gehören der Checkout der Vagrant-Projektressourcen aus dem Versionsverwaltungssystem, die Installation der aktuellen Vagrant-Version inklusive notwendiger Plugins und auch der aktuellen Version von virtualBox. Außerdem können auch bei diesem Vorgehen nachträgliche Individualisierungen an der generierten VM vorgenommen werden. Da das Datenvolumen der Projektressourcen

zum aktuellen Zeitpunkt nur 7 MB beträgt, ist der Kopieraufwand zu vernachlässigen. Um jedoch die Zeit für die Einrichtung des Projektes und die Durchführung des Checkout-Vorganges zu berücksichtigen, werden 5 Minuten als notwendiger Aufwand angenommen. Aufgrund des geringfügig erhöhten Vorbereitungsaufwandes gegenüber dem manuellen Vorgehen werden für diesen Schritt 10 Minuten und für die individuellen Konfigurationen ebenfalls 15 Minuten berechnet.

Für das Update der VM, entsprechend der im Testszenario festgelegten Spezifikationen, wird der Befehl vagrant provision ausgeführt, der die eingebundenen Puppet-Skripte erneut lädt. Diese müssen zuvor von jedem Projektmitglied lokal mit den Projektressourcen im SVN synchronisiert werden, wofür nochmal jeweils ein Aufwand von 5 Minuten angesetzt wird. Bei der Ausführung des Vagrant-Befehls werden noch einmal alle Konfigurationen des Systems der VM mit dem durch Puppet beschriebenen Zielstand abgeglichen und wenn nötig Änderungen durchgeführt.

Im Fall der Änderung einer virtuellen Maschine kann dieser Vorgang, anders als bei der manuellen Durchführung, auf den lokalen Systemen der Projektmitglieder durchgeführt werden. Hierbei muss nur der Pfad der verwendeten Base Box im Vagrantfile getauscht werden und mittels vagrant up eine neue VM erzeugt werden. Der zeitliche Aufwand entspricht somit den Messungen im Rahmen des Erstellungsprozesses. Ist jedoch keine Box mit dem benötigten Betriebssystem vorhanden, muss diese von Hand erstellt werden. Der hierfür notwendige Prozess ist in Anlage 5 abgebildet und wird hinsichtlich der zeitlichen Aufwände im Rahmen der Auswertung noch einmal näher betrachtet.

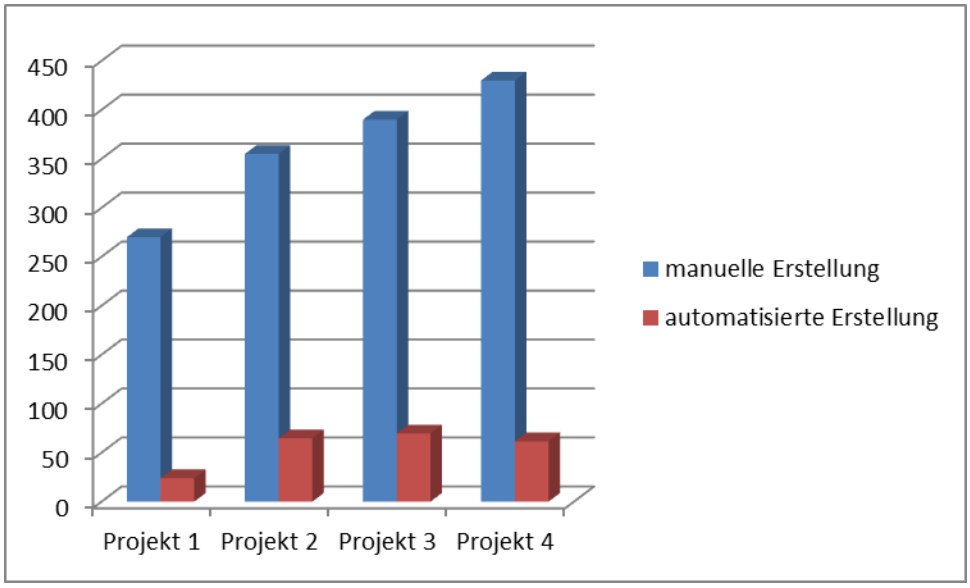
Wird wie im Testszenario beschrieben ein Fehlerzustand einzelner VMs angenommen, muss für die Wiederherstellung der Funktionsfähigkeit, der Zeitaufwand der im Rahmen der Erstellung erfasst wurde berücksichtigt werden. Um den Zustand des Systems vor Eintreten der Fehlersituation wieder herzustellen muss außerdem, der Aufwand für die Durchführung individueller Konfigurationen berücksichtigt werden.

## 5.4 Vergleich des manuellen und automatisierten Vorgehens

Nachdem in den vorangegangen Abschnitten ein konkretes Testszenario beschrieben und die ermittelten Zeiten des manuellen sowie automatisierten VM-Managements zusammengetragen wurden, werden die erfassten Daten im Folgenden ausgewertet. Ziel ist es dabei die Effizienz der Automatisierung durch Vagrant und Puppet zu bestimmen und Handlungsempfehlungen hinsichtlich der weiteren Verwendung des Systems zu geben.

Im ersten Teil des Testszenarios wurde die Erstellung einer virtuellen Maschine mit vier verschiedenen Konfigurationssets untersucht. Wird dabei der Mittelwert der jeweils gemessenen Zeiten betrachtet, ist die automatisierte VM-Generierung deutlich schneller als das manuelle Vorgehen. Es ist jedoch zu beachten, dass die automatisierte Erzeugung auf den Systemen der jeweiligen Endnutzer durchgeführt wird und somit der Gesamtzeitaufwand proportional zur Größe des Projektteams steigt. Abbildung 12 verdeutlicht diesen Umstand. Hierbei ist allerdings beachten, dass beispielsweise in Projekt 2 erst ab einer hypothetischen Teamgröße von 33 Mitarbeitern der Vorteil der schnelleren Erstellung nicht mehr relevant wäre.

**Abbildung 12: Vergleich – Erstellung der virtuellen Maschine (Angaben in Minuten)**



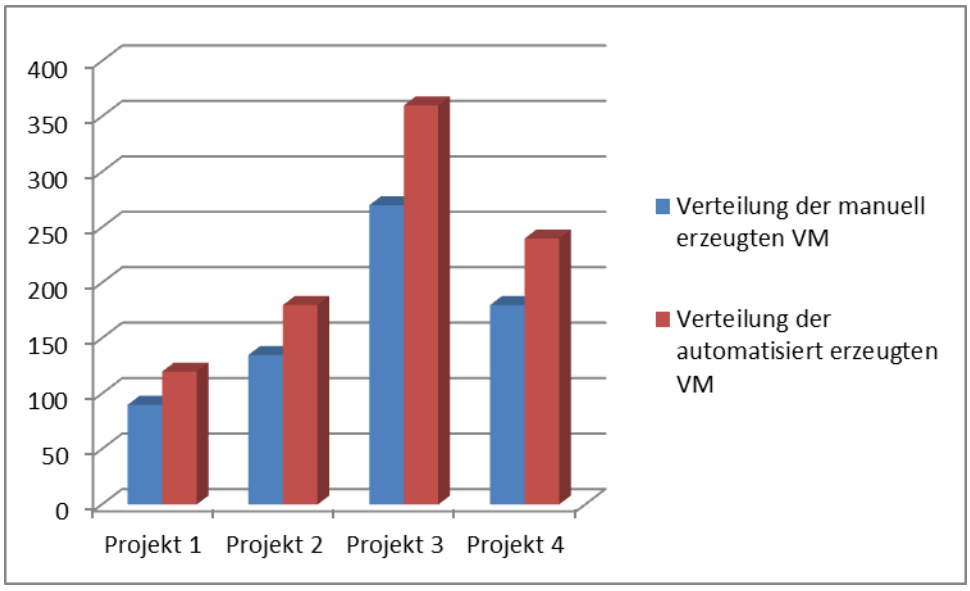

**Quelle: eigene Darstellung**

Dabei ist außerdem zu berücksichtigen, dass die Zahl von Nutzern für die eine Beschleunigung erzielt werden kann abnimmt, je größer der statische Anteil der Konfiguration

ist. Damit sind Konfigurationen gemeint, wie z.B. das Kopieren großer Datenmengen oder Import von Datensätzen in eine Datenbank, die nicht durch eine Automatisierung beschleunigt werden können.

Der Vergleich der Zeitaufwände für die Verteilung der virtuellen Maschienen ergab, dass weder das manuelle noch das automatisierte Vorgehen signifikante Vorteile gegenüber dem anderen ausweist. Die nachfolgende Abbildung 13 verdeutlicht diesen Umstand. Eine mögliche Ursache für dieses Ergebnis liegt in den einzelnen Aktivitäten, welche diesem Teilprozess zugeordnet wurden. Sowohl das Durchführen individueller Vorbereitungen, als auch von nachträglichen individuellen Konfigurationen sind Arbeitsschritte deren zeitlicher Aufwand weitgehend unabhängig von einer Automatisierung sind. Große Unterschiede bestehen hingegen beim Kopieren der VM-Daten, da hier beim Einsatz von Vagrant nur wenigen Megabyte Daten in Form von Konfigurationsfiles das mehrere Gigabyte umfassende VM-File gegenüber steht. Bei genauer Analyse kann jedoch festgestellt werden, dass der Großteil der Daten nicht auf dem Rechner des jeweiligen Nutzers generiert, sondern im Rahmen des Erstellungsprozesses von unterschiedlichen Quellen kopiert wird. Aus diesem Grund erhöht sich bei Zunahme statischer Konfigurationen die Zeit für die Erzeugung der virtuellen Maschine im gleichen Maß wie die Dauer des Kopiervorgangs.

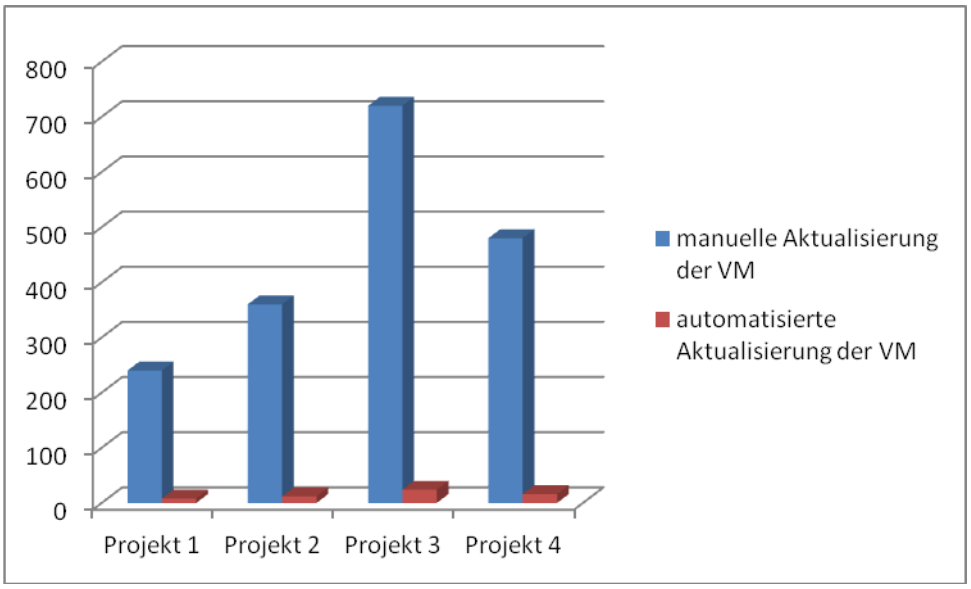
**Abbildung 13: Vergleich – Verteilung der virtuellen Maschinen (Angaben in Minuten)**




**Quelle: eigene Darstellung**

Ein weiterer Bestandteil des konstruierten Testszenarios stellte das Anpassen von Software und Konfigurationen einer bereits verteilten VM in Form von Updates dar. In diesem Fall stellt die Verwendung von Vagrant und Puppet einen entscheidenden Vorteil dar. Die Zeiteinsparung erhöht sich dabei mit steigender Teamgröße. Diese Tendenz ist deutlich in Abbildung 14 zu erkennen, welche die Mittelwerte beider Verfahren in den einzelnen Projektteams gegenüberstellt. Aufgrund der Gleichartigkeit der Arbeitsschritte, die jeder einzelne Benutzer innerhalb des gewählten Update-Szenarios manuell durchführt, kann durch die Erhöhung der Abarbeitung dieser Konfigurationsschritte, in Form einer Automatisierung, der Gesamtaufwand stark verringert werden.

**Abbildung 14: Vergleich – Update der virtuellen Maschinen (Angaben in Minuten)**




**Quelle: eigene Darstellung**

Des Weiteren wurde die Änderung einer VM in Form eines Betriebssystemwechsels untersucht. Die Bewertung dieses Szenarios gleicht dabei der grundlegenden Erstellung einer virtuellen Maschine, da auch hier der Erzeugungsprozess auf Basis des neuen Systems durchlaufen werden muss. Ein wesentlicher Unterschied dabei ergibt sich aber hinsichtlich des notwendigen Kenntnisstandes des Mitarbeiters, der die VM erstellt. Bei einer manuellen Erstellung sind umfangreiche Kenntnisse über die Beschaffenheit des neuen Systems notwendig, um alle Installationen und Konfigurationen durchzuführen. Wird das System hingegen automatisiert eingerichtet, sollte es bei flexibel konstruierten Modulen keinen Unterschied machen, welches Betriebssystem verwendet wird. Um allerding das gewünschte

System installieren zu können, wird beim Einsatz von Vagrant die passende Box benötigt. Ist diese nicht im öffentlichen online Repository oder anderen Quellen vorhanden, geht dem VM-Erstellungsprozess die Erzeugung der entsprechenden Base Box voraus. Im Fall der im Rahmen des Intershop-VM-Projektes erstellten Box eines SUSE Linux Enterprise Server Systems benötigte dieser Vorgang 180 Minuten. Die dabei durchgeführten Aktivitäten, inklusive des gemessenen Zeitaufwanden können Anlage 6 entnommen werden.

Abschließend wurde betrachtet, ob beim Eintreten einer Fehlersituation die Wiederherstellung eines einsatzbereiten Zustandes durch eine Automatisierung unterstützt werden kann. Auch hierfür dienen die Messungen der Erstellung einer virtuellen Maschine als Basis. Jedoch wird dabei die Zeit zur Erzeugung einer VM mit Vagrant dem Kopiervorgang der bereits vorhandenen VM des manuellen Vorgehens gegenübergestellt. Im Testszenario ist dabei das Kopieren mit 2,5 Minuten schneller als die Erzeugung, welche durchschnittlich nach 7,5 Minuten beendet war. Da der Aufwand für nachträgliche individuelle Konfigurationen in beiden Fällen durchgeführt werden muss, kann dieser im Vergleich vernachlässigt werden.

### 5.5 Auswertung

Der direkte Vergleich der Vorgehensweisen auf Basis des Testszenarios ergab, dass bei der Erstellung einer VM die Automatisierung eine signifikante Reduzierung des Aufwandes bedeuten kann. Dieses Ergebnis kann aber nur dann erzielt werden, wenn die initiale Erstellung der Konfigurationsskripte nicht berücksichtigt wird und eine kritische Anzahl von Nutzern nicht überschritten wird. Die Verteilung der notwendigen Datenmengen auf die einzelnen Nutzer wird in beiden Fällen ähnlich effizient realisiert, auch wenn die Aufwände dabei auf unterschiedlichen Aktivitäten verteilt sind. Wurden die im Updateprozess notwendigen Änderungen der VM bereits in Form von Konfigurationsskripten umgesetzt, ist auch hier eine deutliche Effizienzsteigerung durch die Automatisierung zu erkennen. Ähnlich verhält es sich bei der Änderung der VM. Sind die vorhandenen Konfigurationsskripte kompatibel zum gewählten Betriebssystem und ist dieses in Form einer Box bereits vorhanden, erzielt Vagrant bessere Resultate. Hingegen kann das einfache Ersetzen einzelner fehlerhafter VMs durch Kopieren schneller realisiert werden als durch eine erneute Generierung.

Im Rahmen des Vergleiches des manuellen und automatisierten VM-Managements konnte wiederholt festgestellt werden, dass für eine fundierte Auswertung mehr Informationen notwendig sind als ausschließlich die erfassten Messwerte. So ist für eine Bewertung der Effizienz des Systems immer wieder wichtig, ob die notwendigen Konfigurationsskripte bereits vorhanden sind oder ob diese für ein konkretes Szenario neu erstellt werden müssen. Die Aufwände zur Erstellung dieser Skripte lassen sich dabei aber nicht eindeutig quantifizieren, da sie von verschiedenen, zum Teil subjektiven Faktoren, abhängig sind. Hierzu gehören der Kenntnisstand des Entwicklers und dessen Fähigkeiten, sich in dieses Thema einzuarbeiten. Auch die Verfügbarkeit von Puppet-Modulen innerhalb des Online-Repository, durch die Konfigurationsaufgaben teilweise oder ganz übernommen werden können, ist dabei von Bedeutung.

Weiterhin ist die Art der durchzuführenden Konfigurationen von Bedeutung, da nur bestimmte Operationen von einer Automatisierung profitieren. Bilden Dateioperationen wie das Kopieren oder Verschieben großer Datenmengen den Schwerpunkt der zeitlichen Aufwände, kann der betreffende Konfigurationsprozess auch durch eine Automatisierung nur geringfügig beschleunigt werden. Um diesem Umstand bei der Erzeugung virtueller Maschinen entgegenzuwirken, besteht die Möglichkeit, diese auf einem Server zu generieren und wie im Rahmen des manuellen Vorgehens auf die einzelnen Projektmitglieder zu verteilen.

Langfristig bietet die Verwendung von Vagrant und Puppet besonders hinsichtlich der Dokumentation von Konfigurationsprozessen Vorteile. Werden virtuelle Maschinen wie bisher manuell erstellt, wird der Prozess der Einrichtung einzelner Softwarepakete häufig schlecht oder unvollständig dokumentiert, folglich geht das dabei aufgebaute Wissen verloren, wenn der verantwortliche Mitarbeiter das Unternehmen verlässt. Wird dagegen ein skriptbasiertes Konfigurationssystem wie Puppet verwendet, sind die Informationen über die Bewältigung einer Konfigurationsaufgabe in den jeweiligen Puppet-Manifesten persistiert. Mitarbeiter, die mit den grundlegenden Prinzipien dieses Systems vertraut sind, können so Konfigurationsschritte nachvollziehen oder bereits erstellte Komponenten wiederverwenden.

Die Wirtschaftlichkeit des Einsatzes von Vagrant und Puppet kann nur unter der Einbeziehung aller in diesem Kapitel dargelegten Gesichtspunkte sicher beurteilt werden. Der wohl wichtigste Aspekt bleibt dabei jedoch der zeitliche Aufwand. Kann dieser verringert werden, ist das mit der Freisetzung personeller Ressourcen und somit auf Projektsicht durch Laufzeitverringerung, mit der Reduzierung von Kosten verbunden. Hier erzielte das Konfigurationssystem gute Ergebnisse bei der Erstellung und dem Update von VMs. Maßgeblicher Faktor beim Einsatz ist aber in beiden Szenarien die Dauer des Einsatzes im Unternehmen. Dieser bestimmt in wie weit auf die Kenntnisse und Erfahrungen von Mitarbeitern sowie auf bereits erstellte Konfigurationskomponenten zugegriffen werden kann. Für die Erstellung von virtuellen Maschinen sind außerdem, wie zuvor erläutert, die Art der Konfiguration sowie die Größe des Projektteams von Bedeutung. Der Einfluss beider Faktoren kann jedoch, wie ebenfalls dargelegt durch eine Zentralisierung des Generierungsprozesses verringert werden. Werden diese Hinweise berücksichtigt, ist der Einsatz von Vagrant hinsichtlich der Aufwandsreduzierung des VM-Managements innerhalb der dotSource GmbH zu empfehlen.

# 6 Fazit

Nur das Unternehmen, welches bereit ist, seine etablierten Prozesse hinsichtlich des Potentials für Optimierungen zu untersuchen, kann dieses auch zur Verbesserung seiner Arbeitsabläufe nutzen. Vor diesem Hintergrund bestand das Ziel dieser Arbeit darin, das System zur automatischen Generierung virtueller Maschinen innerhalb der dotSource GmbH auf Basis einer durchzuführenden Prozessmodellierung weiterzuentwickeln und dessen weiteren Einsatz zu bewerten. Dieses Ziel konnte zum Abschluss dieser Arbeit realisiert werden. Obwohl noch nicht alle Teile des Systems praktisch realisiert werden konnten, war es mit Hilfe des erstellten Modells möglich alle notwendige Messungen am bereits umgesetzten Konfigurationssystem durchzuführen und so Rückschlüsse für weitere Maßnahmen zu ziehen.

Nachdem grundlegende technische Aspekte der Umsetzung in Kapitel 2 vorgestellt und erläutert wurden, beschäftigte sich Kapitel 3 mit der Untersuchung und Modellierung der Arbeitsprozesse des VM-Managements der dotSource. Im Zuge dessen wurde festgestellt, dass neben dem zentralen Erstellungsprozess einer virtuellen Maschine auch deren Aktualisierung, Änderung und das Auftreten von Fehlerzuständen für die Erstellung eines Modells zu berücksichtigen sind. Die Erfassung und allgemeingültige Formulierung der Aktivitäten, die im Rahmen dieser Einzelprozesse durchgeführt werden, sowie die Abbildung der dabei ablaufenden Kontrollflüsse stellte den Schwerpunkt im Rahmen der Modellentwicklung dar.

Der Schwerpunkt des nachfolgenden Kapitels 4 bildeten die im Rahmen dieser Arbeit durchgeführten praktischen Umsetzungen. Dabei wurden, ausgehend vom Stand des Projektes zu Beginn der Realisierungen, alle Neu- und Weiterentwicklungen des Konfigurationssystems auf Abteilungsebene erläutert. Ein Großteil dieser Umsetzungen wurde dabei von den, im Rahmen der Modellentwicklung ermittelten Prozesskenntnissen motiviert. Beispielhaft sei an dieser Stelle die Überarbeitung einer großen Anzahl von Puppet-Ressourcen zur Unterstützung des Update-Prozesses genannt. Am Ende dieses Kapitels wurden noch einmal alle Erfahrungen, die im Rahmen der Erstellung des Konfigurationssystems gesammelt werden konnten, zusammengefasst. Dabei wurden besonders notwendige Einarbeitungsprozesse und aufgetretene Problemstellungen näher erläutert und so ein grober Leitfaden für die Fort-

führung des Projektes geschaffen. Eine ähnliche Zielstellung verfolgte die Beschreibung noch offener Umsetzungen, wobei auch auf die Vision des finalen Projektzustandes eingegangen wurde.

Das abschließende Kapitel diente der Zusammenführung der zuvor entwickelten Komponenten Modell und praktische Realisierung. Zu diesem Zweck wurden Testszenarien entwickelt, welche den Einsatz von virtuellen Maschinen innerhalb einer exemplarischen Projektsituation abbilden. Anschließend wurde der zeitliche Aufwand, für die im Modell definierten Aktivitäten, sowohl für das manuelle, als auch das automatisierte VM-Management erfasst. Nachfolgend wurden die resultierenden Werte gegenüber gestellt und dabei nach möglichen Einflussgrößen für die ermittelten Werte gesucht. Im Rahmen der anschließenden Auswertung wurden so die Dauer des Einsatzes im Unternehmen und die Art der durchzuführenden Konfiguration als zentrale Einflussfaktoren herausgestellt. Dabei wurde nochmals dargelegt, wie das erstellte Konfigurationssystem eingesetzt werden könnte, um eine maximale Steigerung der Effizienz des Managements virtueller Maschinen zu erzielen. Zum einen wurde in diesem Zusammenhang auf die Möglichkeit der Zentralisierung des Erstellungsprozess virtueller Maschinen eingegangen, um so zeitaufwändige Datenbankimportprozesse nicht auf jedem einzelnen Entwicklersystem durchführen zu müssen. Zum anderen wurde erläutert, dass eine kontinuierliche Weiterentwicklung der Konfigurationsskripte des Provisioners den Aufwand für die Erstellung weiterer Automatisierungen in Zukunft sukzessive verringert und somit der gemesseneGeschwindigkeitsvorteil der automatischen Erstellung und Aktualisierung von VMs im Unternehmen an Bedeutung gewinnt.

Innerhalb der vorliegenden Bachelorarbeit konnten alle zu Beginn definierten Zielstellungen erreicht und so das System zur automatisierten Generierung virtueller Maschinen in der dotSource GmbH weiterentwickelt werden. Das in diesem Rahmen entwickelte Modell des VM-Managements trägt als Richtlinie für zukünftige Weiterentwicklungen dazu bei, die Effizienz der untersuchten Prozesse nachhaltig zu verbessern. Somit konnte sichergestellt werden, dass auch die dotSource GmbH ihr Potential zu Prozessoptimierung in Form neuer Technologien nutzt und dadurch ihre Stellung am Markt möglicher Weise verbessert.

# II Tabellenverzeichnis

# III Abbildungsverzeichnis

# IV Literaturverzeichnis


[And11] Andreas Hoheisel: „Prozessmodellierung“, Vortrag – Prozessmodellierung, Fraunhofer-Institut für Rechnerarchitektur und Softwaretechnik FIRST, o.O., 2011

[Bra14] JetBrains: „Plugins - Vagrant“, o.O., o.J., http://plugins.jetbrains.com/plugin/7379?pr= Abruf: 19.06.14

[Bun14] Bundesverband Digitale Wirtschaft e.V.: „E-Commerce Ranking 2014“, o.O., 2014, http://www.agenturranking.de/rankings/2014/e-commerce.html Abruf: 05.07.2014

[Can12] Canonical Ltd.: „UbuntuBackports“, o.O., 2012, https://help.ubuntu.com/community/UbuntuBackports Abruf: 19.07.2014

[Can14] Canonical Ltd.: „Ubuntu and Debian“, o.O., 2014, http://www.ubuntu.com/about/about-ubuntu/ubuntu-and-debian Abruf: 19.07.2014

[Cor13] HashiCorp: „Command-Line Interface“, o.O., 2013, http://docs.vagrantup.com/v2/cli/ Abruf: 08.06.2014

[Dai14] Daimler: „ Daimler 2013 auf Erfolgskurs: Bestwerte bei Absatz, Umsatz, EBIT und Konzernergebnis “, Stuttgart, 2014, http://www.daimler.com/dccom/0-5-7171-49-1671030-1-0-0-0-0-0-87-7164-0-0-0-0-0-0-0.html Abruf: 18.07.2014

[Dan13] Dan Bode, Nan Liu: „Puppet Types and Providers“, 1. Auflage, O’Reilly Media Inc., Sebastopol, 2013

[Ell09] Ellen Siever, Stephen Figgins, Robert Love, Arnold Robbins: „Linux in a nutshell“, 6. Auflage, O’Reilly Media Inc., Sebastopol, 2009

[Har14] Prof. Dr.-Ing. Hartmut F. Binner, Dipl.-Ing. Paul Rieckmann: „Geschäftsprozesse – analysieren, optimieren und modellieren“, o.O., 2005, http://www.compliancedigital.de/ce/geschaeftsprozesse-analysieren-optimieren-und-modellieren/detail.html
Abruf: 18.07.2014

[Has13] HashiCorp: „About Vagrant“, o.O., 2013,
http://www.vagrantup.com/about.html
Abruf: 10.07.2014

[Has14] HashiCorp: „Vagrantbox.es“, o.O., 2014,
http://www.vagrantbox.es/
Abruf: 06.06.2014

[Jam11] James Loope: „Managing Infrastructure with Puppet“, 1. Auflage, O’Reilly Media Inc., Sebastopol, 2011

[Jet14] JetBrains: „IntelliJ IDEA 13.1.0 Web Help“, o.O., 2014,
http://www.jetbrains.com/idea/webhelp/updates.html
Abruf: 19.07.2014

[Joh13] John Arundel: „Puppet 3 Beginner's Guide“, 1. Auflage, Packt Publishing Ltd., Birmingham, 2013

[Jon13] Jonathan Levin: „Mac OS X and iOS Internals“, John Wiley & Sons Inc., Indianapolis, 2013

[Jos14] Prof. Dr. Josef L. Staud: „Ereignisgesteuerte Prozessketten: Das Werkzeug für die Modellierung von Geschäftsprozessen“, Josef L. Staud, o.O., 2014

[Kim06] Kim Hamilton, Russell Miles: „Learning UML 2.0“, O’Reilly Media Inc., Sebastopol, 2006

[Kla14] Dipl.-Ing. Klaus Lipinski: „Provisioning“, o.O., 2014, http://www.itwissen.info/definition/lexikon/provisioning-Bereitstellung.html Abruf: 09.07.2014

[Lab14] Puppet Labs: „Type Reference“, Portland, 2014, http://docs.puppetlabs.com/puppet/latest/reference/lang_summary.html Abruf: 25.06.2014

[Lay10] Layna Fischer: „2010 BPM and Workflow Handbook“, Future Strategies Inc., Lighthouse Point, 2010

[Mic12] Michael Becker: „Prozessmodellierung“, Seminar - Prozesse der Leipziger Kreativwirtschaft, Abteilung Betriebliche Informationssysteme, Universität Leipzig, 2012

[Mic13] Michael Peacock: „Creating Development Environments with Vagrant“, Packt Publishing Ltd., Birmingham, 2013

[Mit01] Mitchell Hashimoto: „Vagrant Releases“, o.O., o.J., https://github.com/mitchellh/vagrant/releases Abruf: 15.07.2014

[Mit02] Mitchell Hashimoto: „Vagrant Releases“, o.O., o.J., https://github.com/mitchellh/vagrant/issues Abruf: 15.07.2014

[Mit03] Mitchell Hashimoto: „Vagrant Releases“, o.O., o.J., https://github.com/mitchellh/vagrant/wiki/Available-Vagrant-Plugins Abruf: 15.07.2014

[Mit13] Mitchell Hashimoto: „Vagrant: Up and Running“, 1. Auflage, O’Reilly Media, Sebastopol, 2013

[Mon09] Monika Mörtenhummer, Harald Mörtenhummer: „Zitate im Management“, Wien, 2009

[Oco14] Oracle Corporation: „VBoxManage“, Redwood Shores, o.J.,
https://www.virtualbox.org/manual/ch08.html
Abruf: 20.06.2014

[Ora14] Oracle Corporation: „VirtualBox“, Redwood Shores, 2014,
https://www.virtualbox.org/
Abruf: 20.06.2014

[Pla14] Puppet Labs: „Introduction to Puppet“, Portland, 2014,
http://docs.puppetlabs.com/guides/introduction.html
Abruf: 14.07.2014

[Pro14] Protobox: „Getting Started“, o.O., 2014,
http://getprotobox.com/
Abruf: 18.07.2014

[Pul14] Puppet Labs: „PuppetForge“, Portland, 2014,
https://forge.puppetlabs.com/puppetlabs/apache
Abruf: 14.07.2014

[Pup14] Puppet Labs: „Type Reference“, Portland, 2014,
http://docs.puppetlabs.com/references/latest/type.html
Abruf: 19.06.2014

[Sbe14] Sebastian Benda: „Automatisierte Erstellung und Konfiguration dynamischer virtueller Maschinen“, unveröffentlichte Projektarbeit, 2014

[Spe11] Spencer Krum, William Van Hevelingen, Ben Hero, James Turnbull, Jeffrey McCune: „Pro Puppet“, 2. Auflage, Apress Media LLC, o.O., 2011

[Tho05] Prof. Dr. Thomas Allweyer: „Geschäftsprozessmanagment“, buch bücher dd ag, Birkach, 2005

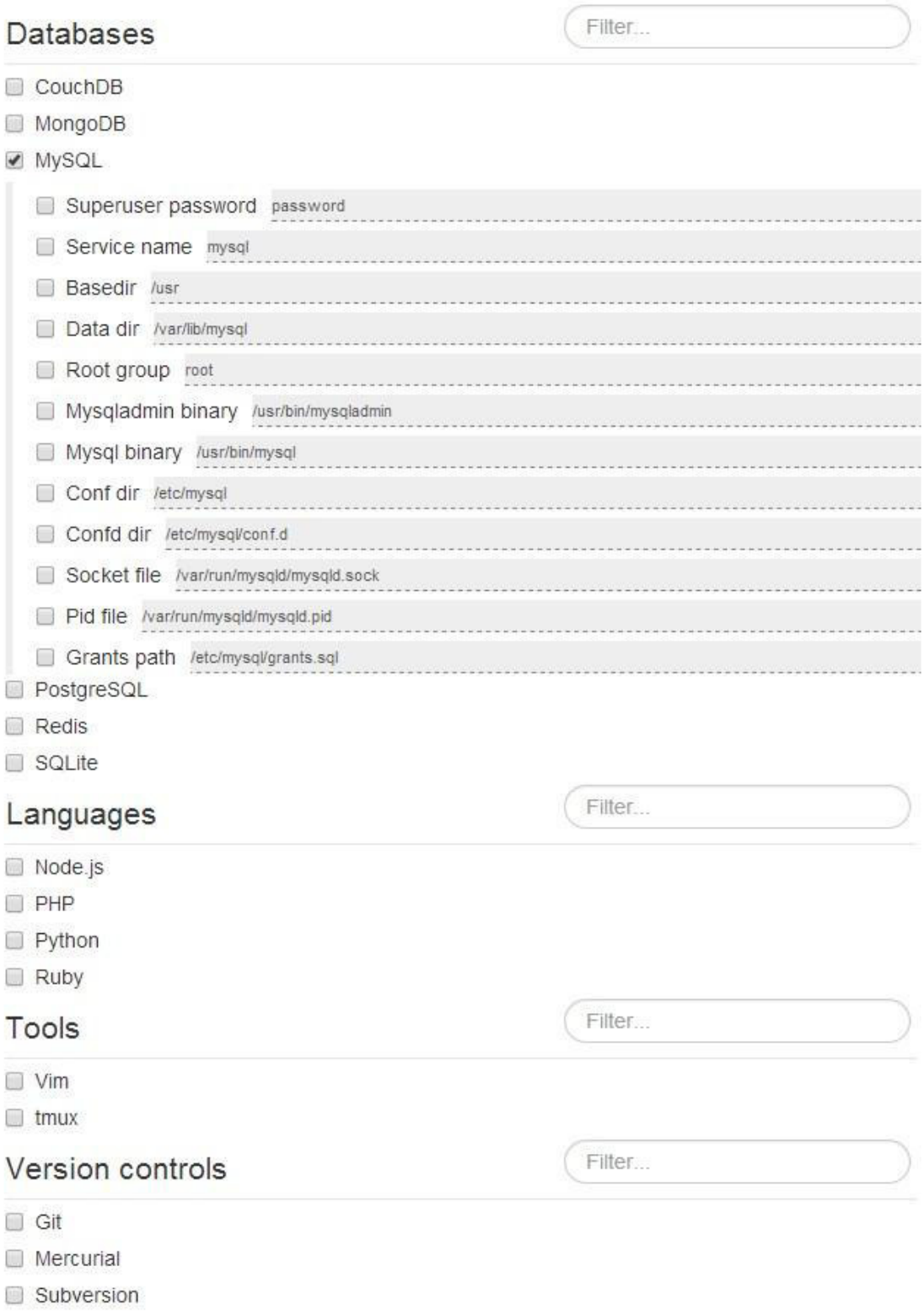
[Vmw14] VMware Inc.: „VMware Player“, o.O., 2014,
http://www.vmware.com/de/products/player/
Abruf: 01.07.2014

# V Anlagenverzeichnis

**Anlage 1: Szenarien und Aktivitäten**

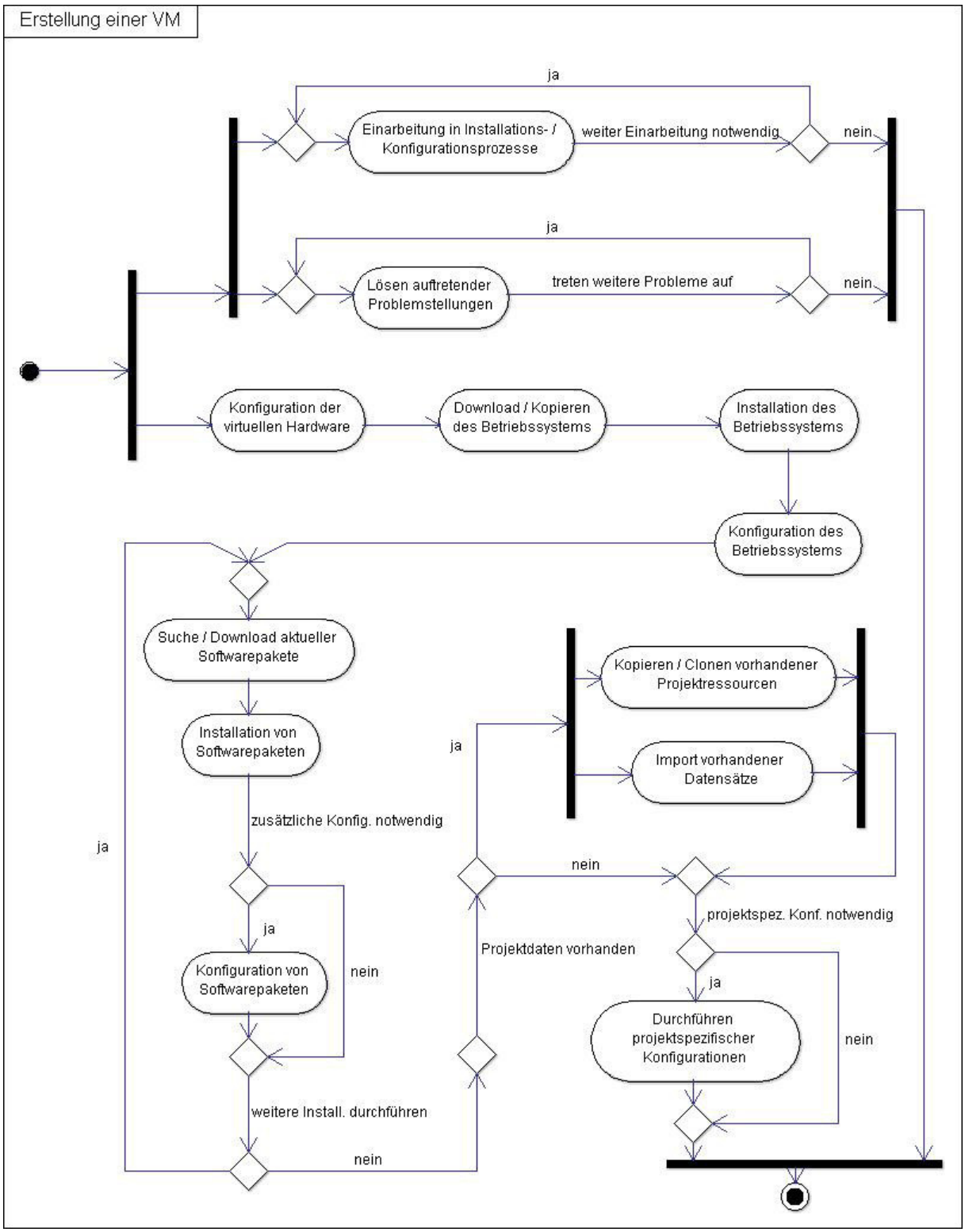
| Einsatzszenario | Aktivitäten |
|---|---|
| Erstellung der VM | - Ermittlung von Softwareanforderungen<br>- Download/Beschaffung des OS / Erstellung Vagrant-Box<br>- Beschaffung von Lizenzen kostenpflichtiger Software<br>- Download von Softwarepaketen/Updates<br>- Einarbeitung in neue Technologien (z.B. Puppet)<br>- Einarbeitung in besondere Installations- / Konfigurationsprozesse<br>- Konfiguration der VM (RAM, Prozessor, etc.)<br>- Installation des gewählten OS<br>- Konfiguration des OS (z.B. Tastenlayout, Usermanagement)<br>- Installation von Softwarepaketen<br>- Konfiguration von Anwendungen<br>- Dateiarbeit (Erstellen/Kopieren/Anpassen von Konfigurationsdateien, Symlinking, etc.)<br>- Import Datenbank-Dumb<br>- Individuelle Konfiguration (z.B. Installation persönlich bevorzugter Tools)<br>- Verteilung der VM |
| Update der VM | - Update jeder einzelnen VM (evtl. unterschiedliche Vers. im Team)<br>- Update durch jeden einzelnen Entwickler<br>▪ Behebung von individuellen Problemen beim Updateprozess<br>▪ Behebung individueller Problemstellungen durch unterschiedliche Software-Versionen<br>o Update der VM und erneutes Verteilen<br>▪ Verteilung der aktualisierten VM<br>▪ Update durch einen Entwickler<br>▪ Individuelle Konfiguration<br>▪ Erneutes Einrichten nicht initialer Projekte |
| Änderung der VM | - Erneuter VM-Erstellungsprozess<br>- Anpassung anderer Software/Konfigurationen<br>- Wechsel von Software durch Inkompatibilität |

## Anlage 2: Web-Konfigurator Rove




**Quelle: Screenshot von http://www.rove.io/**

## Anlage 3: Prozessmodell – Erstellung der virtuellen Maschine

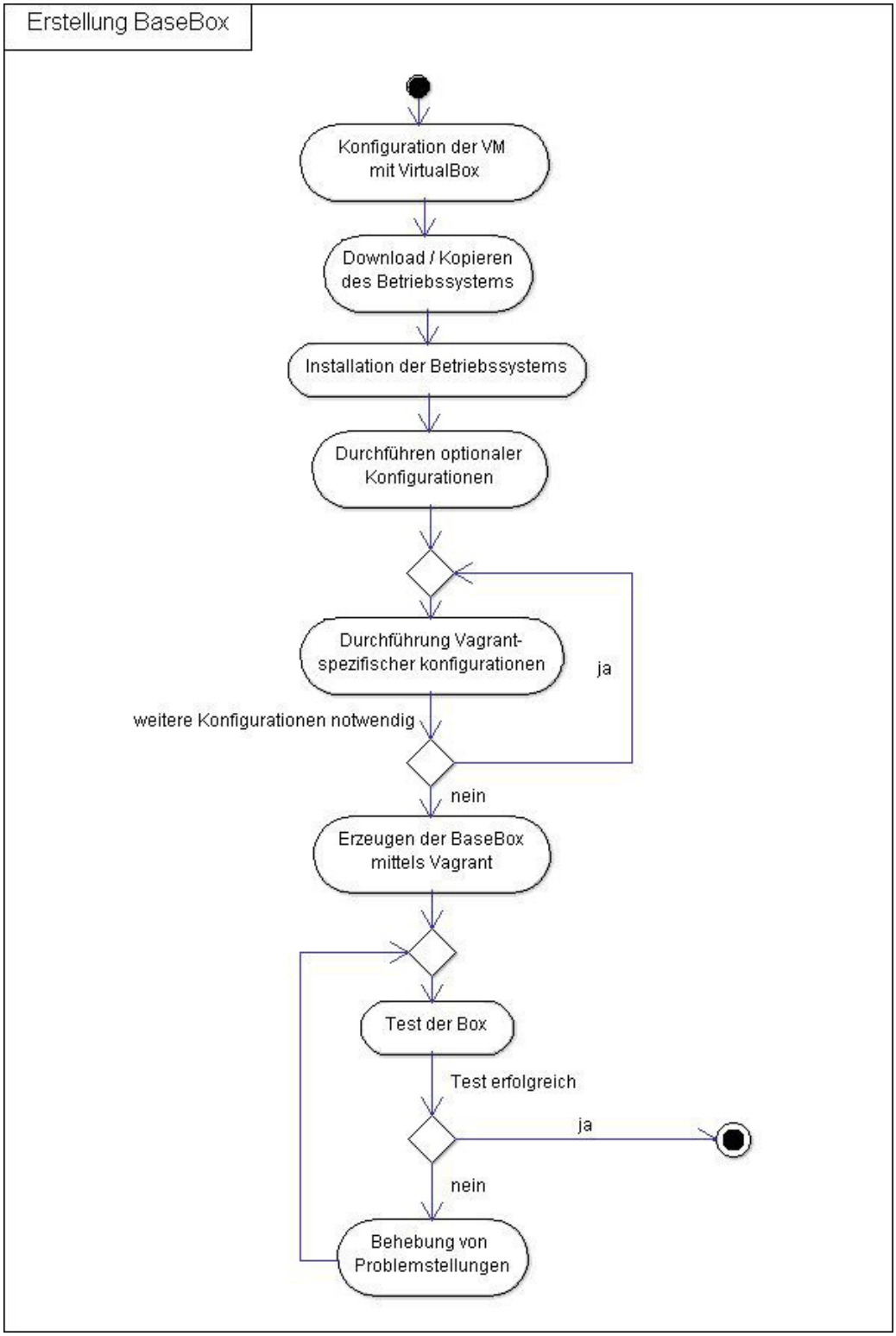


**Anlage 4: Übersicht Puppet-Module / Konfiguration**

| Modul-Name | Konfigurationen |
|---|---|
| ant | - Installation Ant-Build-Tool |
| apache | - Installation Apache-Webserver<br>- Installation Multi-Processing-Modul *mpm_prefork_module*<br>- Installation *rewrite*-Modul<br>- Installation *vhost_alias*-Modul<br>- Installation *php5*-Modul |
| augeas | - Installation *Augeas* (Puppet-Management-Tool für Konfigurationsdateien und andere Dateioperationen) |
| concat | - Verwendung durch andere Modulen um Dateifragmente zusammenzusetzen |
| config | - Enthält Konfigurationsdatei für die zu erzeugende VM (Angabe von Projektname, Versionsnummern bestimmter Softwarepakete, etc.) |
| consol_data | - Einstellung des deutschen Tastatur-Layouts für GUI und Konsole |
| cs-tools | - Installation *PHP CodeSniffer*<br>- Installation *PHP Mess Detector* |
| dev-tools | - Download, Installation *PhpStorm*<br>- Kopieren von Lizenzdaten / Programmkonfigurationen |
| git | - Installation git-Client (Versionsverwaltungssystem)<br>- Projektdaten aus SVN auschecken |
| gui | - Installation / Starten von KDE |
| magento | - Kopieren der Magento-Daten von der angegebenen Position<br>- Installation Magento (mittels n98magerun-Modul)<br>- Ausführen der angegebenen *build.xml* |
| mysql | - Installation *MySql*-Datenbank<br>- Installation *php*-Modul<br>- Einrichtung *MySql*-Server mit Root-Passwort |
| n98magerun | - Installation *n98magerun* |
| php | - Installation PHP 5.3 |

| | |
|---|---|
| | - Installation der Extensions *mysqli, mcrypt, pdo, pdo_mysql* |
| phpmyadmin | - Installation *phpmyadmin*<br>- Einrichtung virtual Host |
| phpunit | - Installiert *phpunit* |
| stdlib | - Puppet-Standard-Bibliotheken die von anderen Modulen verwendet werden |
| tools | - Installation *curl*<br>- Installation *vim*<br>- Installation *wget* |
| tasks | - Führt Aufgabenstellungen durch die nicht direkt der Installation / Konfiguration eines bestimmten Softwarepaketes zugeordnet werden können<br>z.B.: Einspielen von Datenbank-Dumps |
| users | - Erstellung / Konfiguration Standard-Benutzer (Betriebssystem) |

## Anlage 5: Prozessmodell – Erstellung Base Box




**Quelle: eigene Darstellung**

**Anlage 6: Zeitmessung – Erstellung Base Box (SUSE Linux Enterprise Server)**

| **Installationen / Konfigurationen** | **Zeitaufwand in Minuten** |
|---|---|
| SLES Installation | 35 |
| Einrichtung passwortloses sudo | 10 |
| Anpassung Hostname | 10 |
| Behebung bekannter Probleme von virtualBox beim Klonen von SUSE-VMs | 30 |
| Aktualisierung der Software-Paketliste | 10 |
| Installation wichtiger Updates | 10 |
| Installation der virtualBox-GuestAddition | 20 |
| Installation Puppet | 30 |
| Entfernen nicht mehr benötigter Software-Pakete | 10 |
| Erzeugung der Box mittels Vagrant | 15 |
| **Gesamtaufwand** | 180 |

# Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich,

1. dass ich meine Studienarbeit mit dem Thema:

   „Weiterentwicklung des Systems zur automatisierten Generierung von virtuellen Maschinen in der dotSource GmbH“

   ohne fremde Hilfe angefertigt habe,
2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe und
3. dass ich meine Studienarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

_____ _____

Ort, Datum Name