Hochschule
Augsburg University of
Applied Sciences

Bachelorthesis

Fakultät für
Informatik

Studienrichtung
Informatik

Jonas Winkler

Planung und Umsetzung einer automatischen Reportlösung für Wartungs-Dienstleistungen im Bereich Software Entwicklung

Erstprüfer: Prof. Dr. Hubert Högl
Zweitprüfer: Prof. Dr.-Ing Christian Märtin
Abgabe der Arbeit am: 21. März 2022

In Kooperation mit Firma:
Wogra AG
Betreuer: Stefan Kollmann

Hochschule für angewandte
Wissenschaften Augsburg
University of Applied Sciences

An der Hochschule 1
D-86161 Augsburg

Telefon +49 821 55 86-0
Fax +49 821 55 86-3222
www.hs-augsburg.de
info@hs-augsburg.de

Fakultät für Informatik
Telefon +49 821 5586-3450
Fax +49 821 5586-3499

Verfasser der Bachelorthesis:
Jonas Winkler
Singerstraße 13 1/2
86159 Augsburg
jonas.winkler@hs-augsburg.de

Zusammenfassung

Auf dem aktuellen Stand bezüglich Software von Drittanbietern zu bleiben, gestaltet sich zuweilen durch die Vielzahl der verwendeten Software als schwierig. Deshalb ist es notwendig, Möglichkeiten zu schaffen, Entwicklern, aber auch Kunden über Neuerungen zu informieren. Aus diesem Grund befasst sich diese Abhandlung mit dem Zusammentragen nützlicher Informationen und der Implementierung einer automatischen Reportlösung. Zu Beginn werden verschieden Optionen für Reportlösungen mittels allgegenwärtigen Mechanismen zur Datensammlung erörtert. Nach einem kurzen Exkurs zur Entstehungsgeschichte der Softwareentwicklung und abweichenden Möglichkeiten, diese zu vermarkten, wird dargelegt, warum kontinuierliche Updates für Software unverzichtbar sind und wie ein solcher Built-Prozess für neue Releases innerhalb eines Entwicklerteams mithilfe von VCS und CI/CD realisiert wird. Anschließend wird mit der Open-Source-Programmiersprache Python, der GitLab-REST-API und der Verwendung verschiedener Regex-Pattern eine Implementation für eine Reportlösung implementiert. Die besagte Reportlösung schreibt vor, dass Softwareabhängigkeiten automatisch von einem Bot (Renovate) auf einem GitLab-Server upgedatet werden. Im Intervall von 30 Tagen soll das Reportsystem nach Updates, welche durch den Bot ausgeführt wurden, suchen und diese falls vorhanden ausgeben. Resultierend ergab sich ein Programm, welches den Nutzer über die Software-Updates anhand Versionsbezeichnungen belehrt. Eine Erörterung von Release-Notes, um Entwickler über wichtige Updates zu informieren, konnte jedoch nicht realisiert werden. Ein wirtschaftlicher Vorteil konnte durch die zeitliche Begrenzung der Arbeit nicht bewiesen werden.

Inhaltsverzeichnis

Inhaltsverzeichnis **III**

Abbildungsverzeichnis **IV**

Listings **V**

1 Einleitung **1**

2 Softwareentwicklung **3**

2.1 Softwarehistorie 3

2.2 Open-Source-Software 4

2.3 Version Control System 6

2.4 Gitlab 9

2.5 Docker 10

3 Automatische Reportlösung **13**

3.1 Wahl der Programmiersprache 14

3.2 Gitlab 16

3.2.1 Git-CLI 16

3.2.2 GitLab-API 19

3.3 Regex 23

3.4 Zusammenführung 27

4 Rekapitulation und Ausblick **29**

5 Erstellungserklärung **30**

Literatur **31**

Abbildungsverzeichnis

1 Verschiedene Softwarekategorien Quelle: Chao-Kuei\Free-Software-Foundation, 2021 4
2 Delta Versionsverwaltung SCCS Quelle: In Anlehnung an Rochkind, 1975, S.365 6
3 Grafische Darstellung des Konzepts eines verteilen Versionsverwaltungssystems Quelle: In Anlehnung an Chacon and Straub, 2014, S.4 8
4 Prozentuale Nutzung verschiedener Versionskontrollsysteme Quelle: Synopsys Inc., 2022 10
5 Prozentuale Nutzung von Docker Quelle: DataDog, 2018 11
6 Timings von verschiedenen Fibonacciimplementationen Quelle: Smith 2015 S. 3 15
7 Beispiel eines SSH-Fingerabdrucks Quelle: Eigene Darstellung 18
8 Beispielanwendung Zeichenklasse Quelle: In Anlehnung an López and Romero 2014 S. 11 23
9 Beispielanwendung Quantoren Quelle: In Anlehnung an López and Romero 2014 S. 19 24
10 Greedy und non-Greedy Quantoren Quelle: In Anlehnung an López and Romero 2014 S. 20 24
11 Koordinaten-Beispiel Regex Named-Groups Quelle: Eigene Darstellung . 25

Listings

1 SSH-Key Erstellen Quelle: In Anlehnung an Byrnes et al. 2005 S. 233 . . 17
2 SSH-Fingerabruck eines Servers abfragen 17
3 Git-Repository klonen 18
4 Git-Logs abrufen 18
5 Python-Multiprocessing-Pool Download Git-Repositories 19
6 URL erstellen mit Python 20
7 Python HTTP-Request mit OAuth2.0 21
8 Python HTTP-Request mit Python 22
9 Stark simplifizierter JSON-String aus GitLab-REST-API Response 22
10 QED Regex 23
11 Regex-Ausdruck für GitLab-Repository Informationen 26
12 Python-Code Abfrage von Commit-Nachrichten 27
13 REST-API Commit-Nachrichten Response eines GitLab-Server mit Beispieldaten 27
14 Regex-Pattern um Renovate Commit-Nachrichten zu finden 28

1 Einleitung

"*Det øjeblik I blev født, begyndte I at dø*[1]"(Teller 2000: 8), ist wie Janne Teller den philosophischen Kontext der menschlichen Existenz definiert. Dieses Axiom kann auch in naturwissenschaftlichen Domänen wie der progressiven Softwareentwicklung beobachtet werden. Bei dieser Kausalität spricht man jedoch nicht von Freuds Theorem der Übertragung, welche geweckte Begierden auf andere Individuen überträgt (vgl. Freud 1976: 180ff). Viel mehr entsteht die Relation durch eine Subkultur der Bionik[2], der Bioökonomie. Diese ökonomische Strategie mit der Zielsetzung ein biologisch-nachhaltiges System aufzubauen, versucht natürliche Prozesse, Sequenzen und Produkte in wirtschaftlichen Sektoren zu nutzen (vgl. Bioökonomierat 2012). Ähnlich wie bei vielen Menschen die Sehkraft mit kontinuierlicher Annäherung an das Senium[3] und der damit einhergehenden Presbyopie[4] nachlässt (vgl. Berufsverband der Augenärzte 2012), ist auch bei Software zu erkennen, dass diese mit zunehmendem Alter eine steigende Anzahl an Sicherheitslücken aufweist. Eine häufige Ursache für Fehler dieser Art sind Versehen der Entwickler. Diese ermöglichen es Angreifern beispielsweise einen Buffer-Overflow zu finden und diesen auszunutzen, was im Umkehrschluss dazu führen kann, dass ein großer Teil der Adressräume des Stacks und des Heaps ausgelesen und infiltriert werden können (vgl. Vacca 2009: 393).
Um Muster und Defekte zu manifestieren, liegt es im Interesse des Fortschritts, Daten und Werte zu analysieren. Beim menschlichen Körper ist z. B. die Sphygmoskopie[5] ein essenzieller Bestandteil der Datenerfassung. Von ähnlicher Gewichtigkeit ist beispielsweise die Last der CPU des Netzwerks wie auch der Speicherauslastung beim Monitoring von Computer-Systemen. Ungeachtet dessen, dass die Welt Anfang 2020 durch eine Pandemie (SARS-CoV-2) zum Erliegen kam und diese hierzulande 2021 laut dem Statistischen Bundesamt für 4 % der Todesfälle verantwortlich war, stieg die Anzahl der durch ischämischer Herzkrankheiten zu Tode gekommenen um 2,1 % auf 34 Prozent. Resultierend diesem seit Jahren entwickelnden Trend und dem demografischen Wandel in Deutschland sowie dem damit einhergehenden steigenden Interesse an Präventivmaßnahmen entwickelt sich in der Bundesrepublik ein stetig wachsender Gesundheitstrend(vgl. HSH-Nordbank 2017: 1ff). Nicht zuletzt aus diesen Gründen gab es in jüngsten Jahren auch Fortschritte im Bereich der Bionik und der Erfassung sphygmischer Daten. Als Jeff Holter gegen 1947 sein fast 40 kg schweres, aber tragbares EKG auf einem stationären Fahrrad testete, stellte dies einen großen technologischen Progress dar. Die Spannungsschwankungen des menschlichen Körpers waren damit nicht länger auf den inaktiven Zustand begrenzt, vielmehr konnten diese nun bei der Ausführung täglicher Aktivitäten durchgeführt werden(vgl. Museum of American History 2011). Lediglich 31 Jahre später entwickelte die finnische Firma Polar Electro die erste kommerziell vertrieben tragbare Uhr mit eingebautem Herzfrequenzmesser(vgl. Thivierge and Léger 1989: 16ff). Durch die technologische Prosperität führte dies zu Beginn des neuen Millenniums zu den

1. ÜdV: The moment you were born you began to die. (Teller 2010: 10)
2. Technische Imitation natürlicher Prozesse und Verhaltensweißen (vgl. Nachtigall 2002: 3).
3. Synonym für hohes Alter
4. Altersbedingte Abnahme an Flexibilität der Linse im Auge
5. Die Untersuchung des Pulses(Herder 1857: 283)

heutzutage allgegenwärtigen Smartwatches, ausgestattet mit Elektrokardiogrammen und weiteren Analysegeräten zur Auswertung körpereigener Attribute. Auch bei Computer-Systemen können ähnliche Entwicklungszüge beobachtet werden. Zu Beginn wurde der Begriff Monitoring lediglich mit einem Mess- und Bewertungssystem von PCs assoziiert. Während 1995 noch mit manuellen Syscalls und Memory Access Timings gearbeitet wurde, entstanden hier leistungsfähige Benchmark-Tools wie 3DMark und Prime95(vgl. Chen J. Bradley et al. 1996: 303ff). Mit der Freigabe von Berner-Lees World Wide Web 1992 stieg ebenfalls das Engagement ein- und ausgehenden Netzwerktraffic genauer zu beobachten(vgl. Grasshof 2004: 192). Hierzu zählte anfangs zunächst lediglich eine automatische Intrusion-Detektion(vgl. Burgess 2002: 196f). Heutzutage werden Netzwerkdaten in Langzeitstudien analysiert, um Fehler zu beheben, bevor diese auftreten. Dieses Vorgehen ist wichtig, um eine aufstauende Queue an Prozessen zu vermeiden sowie die Reputation einer Firma zu bewahren und damit einhergehend immens hohe Ausgaben einzusparen(vgl. Agilent Technologies 2005: 4). Wird ein Server beispielsweise durch signifikanten P2P-Traffic überlastet und verliert dadurch andere Pakete, muss der P2P-Traffic limitiert werden(Sihyung et al. 2014: 84). Wie können die beschriebenen Daten gesammelt werden und kann dies einen wirtschaftlichen Mehrwert für Unternehmen bieten? Eine Frage, mit welcher sich diese Abhandlung im Detail beschäftigt.

2 Softwareentwicklung

2.1 Softwarehistorie

Mit der im Jahre 1936 von Alan Turing publizierten Arbeit zur Zahlen- und Automatentheorie gilt dieser als Urvater der theoretischen Software. Bald darauf folgend entwickelten 1947 britische Wissenschaftler mithilfe der Braunschen Röhre[6] erste Softwareprogramme. Anfangs basierte dieses System auf der Abschätzung und Vorhersage von elektrischen Impulsen, welche in zeitlichen Abständen gemessen wurden. Später jedoch war klar, dass dieses Intervall zu ungenau sei, um zwischen 0 und 1 unterscheiden zu können. Dieses ursprüngliche Vorhaben wurde für die erfolgversprechendere Messung des positiven Pulses der Kathodenstrahlröhre verworfen. Mit nach und nach besser werdender Qualität der Röhren war es schließlich möglich, bis zu 2560 Zahlen zu speichern. Durch diese Entwicklung konnten nun Kommandos in einer Länge von 20 Ziffern abgespeichert werden. In den darauffolgenden Jahren wurde dies mithilfe verschiedener Methoden wie beispielsweise einer magnetischen Trommel weiterentwickelt (vgl. Kilburn 1990: 16ff).
Bis 1973 war der Computer ein gemeinschaftlich genutztes Instrument, welches überwiegend für Forschungszwecke verwendet wurde. Mit dem Erscheinen von Videospielen wie Spacewar! erhielt auch der private Sektor bedingt Einzug in die Welt der Rechenmaschinen (vgl. Graetz 1981: 56ff). Als 1971 der erste Computer ausgestattet mit einem Mikroprozessor (Kenbak-1) entwickelt wurde, folgte bereits 2 Jahre später das kommerzielle Model aus Frankreich (Micral) (Strimpel 1986: 2). Die bisher gemeinschaftlich implementiert und genutzten Softwareprogramme wurden jetzt dezentralisiert und vermehrt für privaten Nutzen verwendet. Bereits in den 1970ern versuchte Microsoft das geistige Eigentum, allegorisiert durch den Source-Code der eigens entwickelten Programmiersprache BASIC zu schützen. Die steigende Aspiration an Software und der Differenzierung von Programmiersprache und Maschinencode schuf einen neuen Markt für Programmiersprachen wie Fortran, BASIC und Cobol. Durch die Stufe der Abstrahierung und der damit einhergehenden Benutzerfreundlichkeit zufolge zeigte sich bald das Potenzial von BASIC. Die Programmiersprache war wie viele Programmiersprachen aus dieser Zeit kompiliert[7], Microsoft-BASIC jedoch war eine interpretierte[8] Programmiersprache. Diese bescheidene Änderung in der Realisierung der Programmiersprache hatte den entscheidenden Vorteil, dass dem Kunden Pseudo-Code anstelle des tatsächliche Programmcodes ausgehändigt werden konnte (Manes and Andrews 2013: 160ff). Diese Entwicklung führte nicht zuletzt auch in der Bundesrepublik zur gesetzlichen Bestimmung von Software. Hier wird 1987 vom Bundesgerichtshof beschlossen, dass Software nach §93 ein wesentlicher Bestandteil der Hardware sei und damit als Sache einzustufen ist (vgl. Wolters Kluwer Deutschland GmbH 2022). Präzedenzfälle wie Apple v. Franklin ein Prozess, in

6. Glaskolben, einseitig spitz zulaufend, mit einer Aluminiumdiode. Gegenseitig angebracht ist ein mit phosphoreszierender Farbe überzogener Glimmerschirm, welcher aufleuchtet, wird dieser mit negativer Ladung beschossen. Dieser Elektronenstrahl kann mithilfe einer Magnetisirungsspule abgelenkt werden (vgl. Braun 1897: 552f).
7. Bei kompilierten Programmiersprachen, wird der Programmcode von einem Compiler in Maschinencode übersetzt und kann anschließend beliebig oft ausgeführt werden. (IBM Corporation 2010).
8. Programme, die mithilfe von interpretierten Programmiersprachen erstellt wurden, müssen bei jeder Exekution neu interpretiert werden (IBM Corporation 2010).

welchem festgelegt wurde, dass der Source-Code urheberrechtlich geschützt werden könne, nicht aber die Funktionalität, die dieser bereitstelle, führten in den USA bereits 1983 zu ersten Grundbausteinen für das Entstehen komplexer Gesetzestexte, welche die Entwickler von Softwareprogrammen rechtlich schützen sollte. Durch diese Rechtsprechungen war es nun möglich Softwareprodukte zu veräußern, ohne, dass der Source-Code den Besitzer wechselte. Zusätzlich käme es einer Straftat gleich, das Programm zu analysieren und den Code zu plagiieren (vgl. Hassett 2012).

Diese Rechtsprechungen teilte das Lager der Softwareentwickler, wie in Abbildung1 dargestellt, zunächst in zwei Divisionen. Auf der einen Seite entstand die bereits erwähnte proprietäre Software, welche mithilfe von Patenten, Copyrights, Lizenzen und Closed-Source-Code Mitstreiter aus dem Markt zu drängen und damit Monopole zu kreieren versuchte (vgl. Donovan 1994: 1ff). Zu dieser Sparte gehört z.B. auch das von Adobe Inc. entwickelt und 1987 erstmalig veröffentlichte Programm Illustrator, ein Programm zum Gestalten grafischer Applikationen wie beispielsweise der Illustration von Magazinen (Adobe Systems Software Ireland Limited 2017).

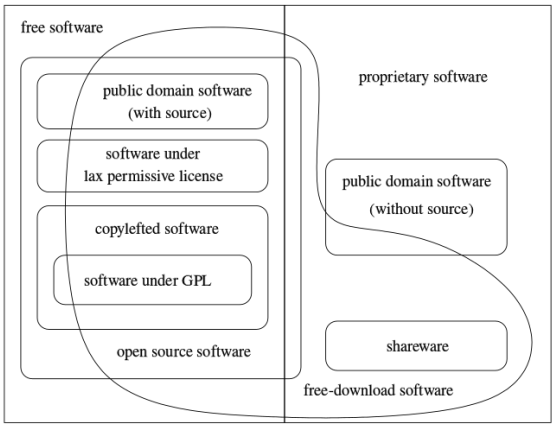


Abbildung 1: Verschiedene Softwarekategorien
Quelle: Chao-Kuei\Free-Software-Foundation, 2021

2.2 Open-Source-Software

Dem entgegen setzt sich Richard Stallman der Gründer des GNU Projekts (Gnu's Not Unix[9]) sowie der Free Software Foundation (FSF) für die Rechte von Softwarenutzern ein. Er vertritt die Ansicht, dass jegliche Software *free* zu Deutsch *frei* sein sollte. Darunter versteht er, dass der Source-Code von Softwareprogrammen nach dessen Erwerb nicht nur offen gelegt werden müsse, sondern dieser auch verändert und weiterverbreitet werden darf (vgl. Stallman 2015: 19f). Durch proprietäre Software gäben Nutzer vermehrt unbewusst Freiheiten auf. Dies geschieht oft durch den Einsatz von Backdoors, Spyware oder auch Keystroke Loggern. Firmen wie beispielsweise Microsoft und Apple können so das Verhalten von Nutzern analysieren und für eigene Interessen nutzen, ohne dass diese davon erfahren (Stallman 2015: 402). Dieses Verhalten kann durch den unzugänglichen Source-Code nur in den seltensten Fällen nachgewiesen werden. Dennoch ist es Experten vereinzelt möglich, beschriebene Fälle aufzudecken. 2017 wurde hier beispielsweise auf HP-Notebooks ein von Werk aus installierter Keylogger gefunden (vgl. BBC News 2017). Auch die Backdoor, die Microsoft zur Einsicht jeglicher Daten von Windowsnutzern verwendet, konnte aufgezeigt werden (vgl. EDRi 2015). Eine Auflistung vieler dieser Sachverhalte wird unter anderem von Zuständigen des GNU Projekts verwaltet und regel-

9. Ein 1978 von AT&T Bell Laboratories entwickeltes Betriebssystem (vgl. Ritchie and Thompson 1978: 1905ff), dessen Kernel Hauptbestandteil von BSD und damit auch heute noch Teil von macOS ist (vgl. Mckay 2019)

mäßig aktualisiert (Free Software Foundation 2022). Durch das in den 90ern zunehmende Interesse an *Free-Software* und den steigenden Nutzerzahlen wurde der Begriff *Free* auch vermehrt mit *kostenlos* fehlinterpretiert. Dieser Trugschluss führte 1998 dazu, dass Christine Peterson nach eigenen Angaben den Begriff *Open-Source* für die von Stallman initiierte Free-Software Bewegung pseudonymisierte (vgl. Peterson 2018). Während Linus Torwalds, Guido van Rossum[10] und andere Verfechter der proprietären Softwarebewegung diesen Alias mit großem Zuspruch übernahmen und weiterverbreiteten, sprach sich Stallman gegen diese Entwicklung aus. Seiner Ansicht nach entfernte sich der philosophische Kontext dessen, was die Free-Softwarebewegung verkörpere. Sie vertrete soziale Werte, die bei der Open-Source Gemeinschaft nicht länger zu vernehmen seien. Anstelle dessen werden ausschließlich praktische Nutzen mit Open-Source-Software assoziiert, verdeutlicht werde dies auch die Aussage Torwalds "Given a large enough beta-tester and co-developer base, alsmost ever problem will be characterized quickly and then fixed (Greenstein 2012: 1)"(Stallman 2015: 140f). Diese Aussage wurde später von einem der Mitgründer von OSI (Open-Source-Initiative) paraprasiert und als Linus' Law und damit Leitfaden der Open-Source-Bewegung betitelt.

≫ *Given enough eyeballs, all bugs are shallow.* ≪
Eric Raymond (2001: 30)

Anders als in der Mathematik oder Physik muss in der Informatik nicht jedes Gesetz bewiesen werden. Dieser Widerspruch führt vermehrt dazu, dass Gesetze aus der Informatik, welche oft auf empirische Analysen basieren, im Laufe der Zeit widerlegt werden. Eines der bekanntesten Beispiele hierfür ist Moores Law, welches von Gordon E. Moore (vgl. 1965: 1ff) aufgestellt wurde. Er prognostizierte, dass sich die Anzahl der Transistoren auf Mikrochips alle zwei Jahre verdoppeln würde. Dieses Ziel konnte jedoch 2010 nicht mehr erreicht werden, Huang Jensen, Nvidia's CEO verkündetet auf der CES[11], 2019 vielmehr, dass die Transistoranzahl sich höchstens alle 10 Jahre verdoppeln würde (vgl. Tibken 2019). Ähnlich verhält es sich auch mit Linus' Law. Der nicht versiegende Zyklus an Updates, welcher durch die Vielzahl an Entwicklern für Open-Source-Projekte wie Linux bereitgestellt wird, erschwert es, tiefgreifende Fehler in dessen Kernel zu implementieren, ohne dass diese von der Community erkannt würden (vgl. Raymond 2001: 31). Dennoch sind Fauxpas nicht auszuschließen, dies bewies beispielsweise der Eintrag über Margaret Thatcher in der bekannten Open-Source-Enzyklopädie Wikipedia. Über einen längeren Zeitraum wurde sie hier als fiktive Person denunziert (vgl. Greenstein 2012: 1). Während hier noch von oberflächlichen Bugs gesprochen werden kann, wurde Linus' Law spätestens mit Log4Shell widerlegt. Log4Shell ist die Bezeichnung einer Sicherheitslücke, welche von Entwicklern der chinesischen Firma Alibaba Cloud in der bekannten Javabibliothek Log4j Ende 2021 gefunden wurde (vgl. Lin and Uberti 2021).Von Experten wurde der Sicherheitsverstoß als *der schlimmste* aller Zeiten bezeichnet (Hunter and de Vynck 2021). Auch das Bundesamt für Sicherheit in der Informationstechnik stuft diesen als kritische Schwachstelle ein. Die Warnstufe wurde Ende Dezember 2021 von Rot auf Geld herabgesetzt, besteht jedoch auch 2022 weiter, da kein Fix dieses tiefgreifende Problem bislang lösen konnte (vgl. Bundesamt für Sicherheit in der Informationstechnik 2022).

10. Urheber der Programmiersprache Python (van Rossum 2009)
11. Amerikanische Technologiemesse

Aus diesen über die Zeit entwickelten Für- und Gegensprüchen der Softwareentwicklung scheint es, als bliebe nur eine Konstante. Software ist im ständigen Umbruch und kann nicht vollendet werden oder wie es Marc J. Rochkind (1975: 1) ausdrückte. *Computer programs are always changing. There are bugs to fix, enhancements to add and optimizations to make*. Um den Support von Firmen, welche eine Monopolstellung anstreben zu minimieren, ist es von zunehmender Bedeutung Free-Software bzw. Open-Source vermehrt für die Entwicklung von Software einzusetzen. Nutzerdaten sind im Informationszeitalter von exorbitantem Wert, aus diesem Grund ist es für viele Softwarefirmen, die proprietäre Software veräußern gängig Daten Ihrer User zu sammeln, analysieren und diese weiterzuverkaufen. Dennoch ist auch bei Open-Source-Software dieser Vorgang nicht ausgeschlossen, jedoch ist es für den User hier möglich zu überprüfen, welche Daten in welchem Rahmen gesammelt und zumeist auch wofür diese verwendet werden.

2.3 Version Control System

Ein VCS befähigt Entwickler, einen Versionsverlauf von Projektdateien zu etablieren. Dies ermöglicht das Zurücksetzen einzelner Dateien oder ganzer Projektstrukturen in vergangene Versionen. Eine Projektstruktur wird in diesem Kontext auch als Repository bezeichnet. Repositories sind in den meisten Fällen mit mehren Branches ausgestattet. Mittels Branches können in einem Repository verschiedene Versionen eines Projekts bestehen. Diese Funktionalität kann beispielsweise genutzt werden, um einen Haupt-/Main-Branch zu realisieren, welcher ein aktuelles stabiles und damit getestetes Release bietet. Gleichzeitig ist es jedoch möglich, einen Side-Branch aufzusetzen, auf welchem Entwickler neue Features entwickeln und testen können (vgl. Ruparelia 2010: 5). Ein weiterer wichtiger Aspekt von Versionskontrollsystemen ist die Bestimmung der kleinsten/ atomaren Einheit. Als atomare Einheit wird die kleinstmöglich veränderbare Einheit bezeichnet. Wenn ein Teil dieser Einheit verändert wird, muss festgehalten werden, dass der gesamte Datenblock geändert wurde. Da es sich bei Source-Code um Textdateien handelt, ist es bei den meisten VCS heutzutage gängig, dass diese atomare Einheit als Zeile definiert ist (vgl. Grogg and Weddle 2009: 4896ff). Historisch entwickelte Mark Rochkind 1972 das erste VCS. Damals bekannt als SCCS[12], war dies lange Zeit ein bewährtes Tool, um Source Code zu versionieren (vgl. Ruparelia 2010: 6). SCCS ging davon aus, dass jede Source-Codedatei ein Modul darstellt.

Abbildung 2: Delta Versionsverwaltung SCCS
Quelle: In Anlehnung an Rochkind, 1975, S.365

Wurden Änderungen in diesem Code-Modul vorgenommen, wurden diese als diskrete Deltas in einem dafür vorgesehen Textkörpers innerhalb des Moduls abgespeichert. Jedes Modul enthält des Weiteren ein initiales Delta, welches den Zustand der Ausgangsdatei wiedergibt, gefolgt von etwaigen Änderungen. Jedes Delta beinhaltet eine Nummer für das jeweilige Level und Release. Dieses Verhältnis kann auch in Abbildung 2 beobachtet werden. Wird die Version 1.4 benötigt, wird zunächst Version 1.0 des Moduls aufgebaut, anschließend werden alle Deltas darauf angewendet. Ist es beispielsweise notwendig Unittest zu entwickeln, kann dies ohne Release 1 zu modifizieren in derselben Datei unter

12. Source Code Control System

Release 2 vorgenommen werden. Wird das stabile Release 1 benötigt, kann dieses aufgebaut werden, indem lediglich die Deltas von 1.1 - 1.4 durchlaufen werden(Rochkind 1975: 364f). Diesem Konzept verdankt SCCS, dass es auch in den frühen 90ern noch Anklang bei der Verwaltung des Java Source-Codes und dem Solaris-Betriebssystem fand. Für einen benutzerfreundlicheren Umgang schuf Sun Microsystems 1992 Teamware, eine grafische Schnittstelle für SCCS (vgl. Smith 1991: 1ff).

Dennoch wurde bereits 1982 ein weiteres Tool zur Versionskontrolle von Walter F. Tichy entwickelt. Das Revision Control System (RCS) sollte seinem Vorgänger SCCS in nichts nachstehen, vielmehr wurde durch eine simplifizierte Benutzerschnittstelle sowie verbesserten Performance eine ernst zu nehmende Alternative entwickelt (vgl. Tichy 1982: 2). Ein Unterschied zu seinem Vorgänger ist, dass intern GNU Diffutils verwendet werden, um Versionsunterschiede festzustellen. Des Weiteren wird die Version nicht weiter von Release 1.1 aszendierent aufgebaut, viel mehr wird die letzte Version der Datei gespeichert und falls nötig kann diese mithilfe der gespeicherten Deltas auf vorherige Versionen zurückgesetzt werden (vgl. Free Software Foundation Inc. 2012). Dieses Versionskontrollsystem war unter UNIX ein beliebtes Tool für die Verwaltung von Dateien. Dem zugrunde gelegt hatte RCS bis 2013 auch immer einen festen Platz bei den Entwicklertools des auf UNIX[13] basierenden MAC OS X. Diese Verbindung wurde jedoch mit Version 5.11 des von Apple entwickelten SDKs aufgelöst. (Chacon and Straub 2014: 2) (Apple Inc. 2013).

RCS bot jedoch keine Möglichkeit für Entwickler, an unterschiedlichen Orten gleichzeitig an einem Programm zu arbeiten. Ein weiterer Makel war die Verwaltung auf Dateiebene spezifiziert und konnte nicht auf Projektebene projiziert werden. Da besonders das Arbeiten auf verteilter Ebene explizit bei der Entwicklung von Open-Source-Software eine obligatorische Funktionalität darstellt, entwickelte ein dänischer Softwareentwickler 1986 CVS[14] auf Basis eines Bourne-Shell-Scripts (vgl. Grune 1986: 1ff).Grunes CVS trug ursprünglich den Namen cmt, da es Entwickler befähigte, unabhängig voneinander Versionen zu *commiten* (vgl. Grune 2012). Dies bedeute, dass von nun an mehrere Entwickler zur selben Zeit an der gleichen Datei arbeiten konnten, ohne diese für andere zu sperren, da eines der mit CVS eingeführten Features Mergekonflikte behob bzw. diese umging (vgl. Grune 1986: 2ff).

Durch den Mangel an Innovation wurde CVS im Laufe der 2000er durch Subversion, ein von CollabNet entwickeltes Versionskontrollsystem abgelöst. Subversion brachte einige Neuerungen, darunter auch atomare Commits. Diese Art der Commits war notwendig, da Versionen vermehrt online verwaltet wurden und es häufiger zu Übertragungsfehlern oder Ausfällen kam. Wenn es zu einem Fehler kam und die Operation nur zu einem Teil ausgeführt werden konnte, wurde die Änderung zurückgesetzt und der gesamte Commit verworfen. Mit zunehmend größeren Softwareprojekten, stieg ebenfalls das Interesse an Refactoring-Optionen von Projekten, deshalb kam Subversion mit einem Feature, welches es ermöglichte Dateien und Ordner umzubenennen ohne das Verhalten des Programms zu ändern. Des Weiteren bot Subversion als erstes VCS vollständige Unterstützung für Unicode sowie Dateibezeichnungen, die den ASCII-Zeichensatz überschritten (vgl. Pilato et al. 2008: 15).

13. Betriebssystem entwickelt 1969 von AT&T Bell Laboratries (Mcllroy 1987: 1)
14. Concurrent Control Systems (Grune 2012)

Durch das nunmehr omnipräsente Internet entstand zuweilen vermehrt das Verlangen nach einer Änderung in der Softwarearchitektur für Softwareversionssysteme. Um dezentral arbeiten zu können, wurde das Repository bisher auf einem zentralen Server gespeichert. Dies sollte sich mit DVCS[15] wie Git, Mercurial, Bazaar und weiteren ändern. Das bisherige zentrale Server-System wurde in ein Client/Server Model umgearbeitet.

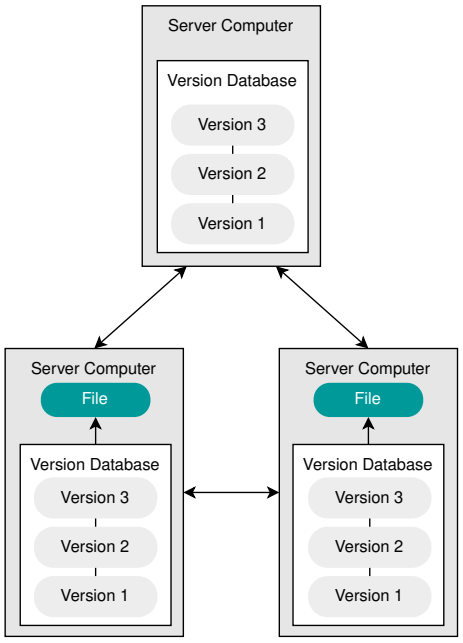


Abbildung 3: Grafische Darstellung des Konzepts eines verteilen Versionsverwaltungssystems Quelle: In Anlehnung an Chacon and Straub, 2014, S.4

Wie in Abbildung 3 abgebildet wurde das gesamte Repository dezentral auf beliebig viele Computer gespiegelt. Durch diese Änderung in der Architektur war es nun möglich, dass lokal mit dem Repository gearbeitet werden konnte. Dies ermöglichte es unter anderem frühe Entwürfe lokal zu evaluieren, ohne diese mit Teammitgliedern zu teilen. Da zuvor jeder Commit eine Netzwerkverbindung, Traffic und Zeit benötigte, konnten hier einige Ressourcen eingespart werden Unklare Versionsbezeichnungen wie Hashes oder GUIDs[16] brachten jedoch in erster Linie für den Nutzer keinerlei Vorzüge (vgl. Koc and Abdullah 2011: 5).

Larry McVoy, ein Entwickler von Sun Microsystems und Autor von Smoosh, dem Vorgänger des SCCS-Tools Teamware (vgl. McVoy 1991), entwickelte Ende der 90er das DVCS-Tool Bitkeeper, welches nachweislich einige Designfeatures von Teamware übernahm. Hierzu kam, dass zur selben Zeit Linus Torvalds, der Kreateur des Linux Kernels, ein Versionskontrollsystem benötigte. Trotz einiger Einwände anderer Entwickler bezüglich des Closed-Source Codes und des teilweise proprietären Geschäftsmodells entschied sich Torvalds 2002 für die Verwendung von Bitkeeper. Drei Jahre später versuchte Andrew Tridgell, der Entwickler von Samba[17] und rsync[18], die Closed-Source-Netzwerkprotokolle von Bitkeeper zu rekonstruieren, um eine Open-Source-Softwarealternative zur Verwendung von Bitkeeper zu kreieren. Dies führte zu einem Bruch der Geschäftsbeziehungen zwischen Torvalds und McVoy, woraufhin die freie Variante von Bitkeeper ausgemustert wurde. Daraus resultierend entwickelte Torwalds Git, ein DVCS welches zunächst nicht auf besonders viel Zustimmung stieß. Der Verzicht auf altbewährte Mechanismen wie Deltas oder Patches und der damit eingeführten Absicherung gesamter Dateiversionen stieß zu Beginn auf viel Widerstand. Ebenso erfuhren die initialen Kommandos vor deren Abstrahierung und Annäherung an gängige Befehle bekannt von VCS, Subversion u. Ä. keinen Zuspruch. Durch die Überarbeitung von

15. Distributed Version Control Systems
16. Globally Unique Identifier (Lutteroth and Weber 2008)
17. Samba ist ein Datei und Printservice für Windows, Linux und weitere Betriebssysteme (Samba Team 2022).
18. Rsync ist ein Tool, das rsh oder ssh nutzt, um Source und Destination Dateien zu synchronisieren. Dabei wird ein spezieller Update-Algorithmus verwendet, welcher es ermöglicht lediglich Änderungen zu übertragen (Tridgell and Mackerras 1996).

Git durch Hamano Junio C. sowie der manifestierten zeitlichen Einsparungen durch das sehr schnelle zusammenführen von Branches sowie Software-Patch-Erstellung erfuhr Git einen großen Aufschwung und war schon bald für viele Software-Entwickler ein Muss (vgl. Brown 2018). Wie bereits im Unterpunkt Softwarehistorie vorgestellt, gibt es aber auch hier Unterschiede. Git ist ein Open-Source-Projekt, welches unter einer GPL-2.0-only Lizenz vermarktet wird und kann deshalb solange die Lizenzvereinbarungen eingehalten werden frei verwendet, verändert, genutzt und weiter vermarktet werden. Dies haben sich sowohl Entwickler aus der Open-Source Bewegung wie auch aus der proprietären Sparte zunutze gemacht, weshalb bereits beim zentralen Abspeichern des eigens geschriebenen Source-Codes wichtig ist zu wissen, wie mit den eigenen Daten bei dem jeweiligen Service umgegangen wird. Aus diesem Grund habe ich mich bei dieser Abhandlung und dessen Source-Code-Verwaltung für die Open-Source-Plattform Gitlab entschieden.

2.4 Gitlab

GitLab ist ein Open-Source-Projekt, initiiert von zwei ukrainischen Entwicklern, Sid Sijbrandij und Dmitriy Zaporozhets. Wie viele dieser Projekte entstand auch dieses aus einem Bedürfnis heraus. Sid benötigte 2011 ein Tool, um zusammen mit seinem Team interaktiv Software zu entwickeln, aus firmenpolitischen Gründen war es ihm jedoch nicht möglich GitHub zu verwenden. Deshalb entwickelte er mit Valeriy Sizov das Tool GitLab. Für die Entwicklung entschied man sich für die Programmiersprache Ruby on Rails, da es aus Sicht der Entwickler eine abstrakte Schicht bietet, welches es ermöglicht das Projekt zu skalieren und vereinfacht neue Features einzubauen (Flowers 2018). Oftmals ist es in Softwareprojekten möglich und im Laufe der Zeit nötig Verbesserungen vorzunehmen. So wurde auch bei GitLab entschieden, dass vereinzelt die Programmiersprache GO zum Einsatz kommt. Go ist eine von Ken Thompson mitentwickelt, kompilierte jedoch plattformunabhängige Programmiersprache, welche vermehrt für Serverimplementationen verwendet wird, um höchstmögliche Leistungen zu erzielen (GitLab B.V. 2022). GitLab ist ein populäres Tool unter Entwicklern, aber auch ein geschätzter Service von Firmen. Teilweise kann dies auch dem Diagramm in Abbildung 4 entnommen werden. Git ist seit Jahren bezüglich der Versionsverwaltung von Source-Code, insbesondere im Open-Source Bereich sehr dominant. Während Subversion noch immer Anklang in bestimmten Bereichen findet, sind Projekte, die mithilfe von Bazaar, Mercuria oder auch CVS kaum noch aufzufinden. Das Diagramm in Abbildung 4 bietet jedoch keinen gesammelten Überblick über alle Softwareprojekt, vielmehr bietet es einen Überblick von Open-Source-Projekten. Dennoch kann davon ausgegangen werden, dass Git aufgrund der weiten Verbreitung und der

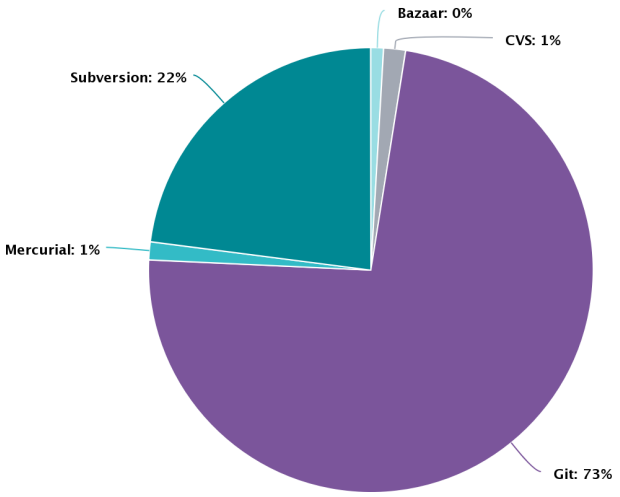


Abbildung 4: Prozentuale Nutzung verschiedener Versionskontrollsysteme
Quelle: Synopsys Inc., 2022

einfachen Handhabung auch im proprietären Softwaresektor Anklang bei den Entwicklern findet. Da Gitlab als eine der wenigen Services beinahe alle Versionen von Git unterstützt, ist eine breit aufgestellte Masse bereits mit dem Interface des Dienstes vertraut. Des Weiteren werden viele nützliche Features für die agile Softwareentwicklung von GitLab zur Verfügung gestellt und weiterentwickelt. Beispielsweise bietet GitLab die Möglichkeit Tasks zu kreieren, Releasezyklen zu integrieren sowie das Repository mit verschiedenen Analysetools zu überwachen. Große Softwarefirmen mit einem globalen Entwicklerteam wie beispielsweise Nvidia schätzt GitLab besonders, da voluminöse Projekte sehr gut skalieren und die Integration sowie Handhabung des Service sehr einfach ist (Nvidia 2022). Aber auch Firmen wie Siemens schätzen viele Features, welche GitLab mit sich bringt sehr. Insbesondere kann laut Siemens die Effizienz durch die Verwendung von GitLab enorm gesteigert werden (Siemens 2022).

2.5 Docker

Die korrekte Verteilung vorhandener Ressourcen, insbesondere bei Webanwendungen, gestaltet sich geschichtlich belegbar oft als schwierig. Webanwendungen setzen sich meist aus zwei oder mehr Bereichen zusammen: ein Webserver zur Kommunikation sowie einer Datenbank. Historisch betrachtet, entsprach es der Norm, dass jedes Rechensystem einen Prozess zum seriellen Abarbeiten anfallender Arbeiten besaß. Mussten Daten eingelesen oder geschrieben werden, gingen wichtige CPU-Zyklen verloren. Aus diesem Grund wurden Webanwendungen meist auf verschiedenen Servern betrieben. Dadurch entstanden neben höheren Anschaffungs- und Betriebskosten auch zugleich meist unausgelastete Systeme. Um Hardware besser nutzen zu können, wurden neben besseren Schedulern wie dem Linux Completely Fair Scheduler auch hypervisor basierter virtuelle Maschinen entwickelt (vgl. Ivanov 2017: 8f).

Hypervisor ein Stück Software, ursprünglich entwickelt, um Speicher, welcher üblicherweise für Programmierer unzugänglich war, zu nutzen. Namensgebend für das Tool war der „Supervisor“. Eine Term, geprägt in 60ern, jedoch heute besser bekannt als Betriebssystem (vgl. Portnoy 2016: 22). Bereits damals fand es großen Anklang und wurde in vielen großen Rechenmaschinen wie dem IBM/360 eingesetzt. Heute ist es ein wichtiger Bestandteil von Betriebssystemen und dient der Verwaltung und Abgrenzung von virtuellen Maschinen (VM). Der Hypervisor ermöglicht es multiple Betriebssysteme von-

einander **unabhängig** auf einem Host auszuführen. Durch diese Abtrennung können jedem OS eigene Rechen- wie auch Netzwerkressourcen zugeteilt werden. Die zugeteilten Ressourcen können eigens vom jeweiligen Betriebssystem verwaltet und vollständig ausgereizt werden. Des Weiteren wird der Host durch die zusätzliche Schicht des Hypervisors vor etwaigen Angreifern oder Kernel-Paniken[19] geschützt (vgl. Ivanov 2017: 9f).

Die Virtualisierung mittels Hypervisor benötigt jedoch eine nicht zu vernachlässigende Menge an Ressourcen wie CPU-Kerne, Arbeits- und Festplattenspeicher. Um Ressourcen zu limitieren und die Effizienz zu steigern, stieg das Interesse an OS-Level Virtualisierung (vgl. Ivanov 2017: 10).

chroot jail Der „**ch**ange **root**“ Befehl wurde mit der 7. Edition 1979 zu Unix hinzugefügt (vgl. Losh 2020) und gilt als Urvater des heutigen Containers. Der begleitende Gedankte war, Programme in einer sicheren Umgebung ausführen zu können. Würde eine Schadsoftware in einem solchen „Gefängnis“ ausgeführt, wäre diese lediglich imstande, die stark begrenzte Umgebung zu beeinflussen (vgl. Albing and Vossen 2017: 303f). Bereits kurze Zeit darauf wurden Exploits entwickelt, um das Environment zu verlassen und damit Zugriff auf das Host-System zu erlangen (vgl. Losh 2020). Die stabile Entwicklungsumgebung, sowie die Möglichkeit des systemunabhängigen Deployments und der minimalistischen Anforderungen sind einige der Gründe, warum chroot trotzdem heute noch in Softwareentwicklung verwendet wird (vgl. Klinger 2014).

Durch Entwicklungen am Linuxkernel und der Einführung von Namespaces und cgroups mit Version 2.6.24, wurde eine OS-Level Virtualisierungen „LXC“ entwickelt, welche ursprünglich der Grundbaustein für das heutige Docker war. Mittels Namespaces ist es möglich Prozesse vom „Default“-System abzuschotten. Wird beispielsweise der Namespace eines Bash-Prozesses mittels *unshare()* verändert und anschließend ein Verzeichnis gemountet, ist dieser Mountpunkt außerhalb des Namespaces nicht sichtbar. Isoliert durch Namespaces ist ebenfalls das UNIX-Timesharing-System, Interprocess Communication, Prozess ID, User und Netzwerk. Durch cgroups war es des Weiteren möglich einzelne Prozesse mit Ressourcen auszustatten. Damit war es möglich Feineinstellungen bezüglich CPU-Zeit, Speichermanagement und I/O je nach Container festzulegen (vgl. Ivanov 2017: 11-24).

Durch diesen fortlaufenden Entwicklungsprozess der Container, setzten sich 2008 drei Softwareentwickler (Kamel Founadi, Sebastien Pahl, Solomon Hykes) in Frankreich zum Ziel, die Verwendung von Containern in der Softwareentwicklung für immer zu verändern und gründeten *DotCloud*. Fünf Jahre darauf entstand daraus mit Docker eines der größten Open-Source-Projekte (vgl. Hykes 2018). Wie in Abbildung 5 dargestellt konnte Docker

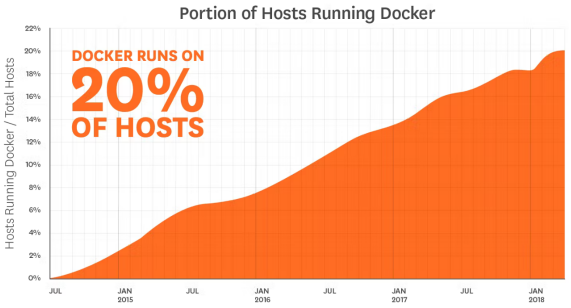


Abbildung 5: Prozentuale Nutzung von Docker
Quelle: DataDog, 2018

19. Sicherheitsmechanismus von Betriebssystemen, um größere Datenverluste bei fatalen unbehebbaren Fehlern zu vermeiden. Unter Windowsnutzern auch als „Bluescreen of Death“ bekannt.

von der Gründung bis 2018 einen nahezu linearen Anstieg der Marktanteile verzeichnen. Zumeist wird dies den minimalen Anforderungen, dem breit gefächerten Lehrmaterial und der einfachen Einstiegsmöglichkeit zugute geschrieben. In der Geschichte der virtuellen Maschinen wurden bereits einige Sicherheitslücken enttarnt, durch die zusätzliche Abschirmung des Gast-Betriebssystems wird die Hypervisor Virtualisierung jedoch als sicherer erachtet (vlg. Ivanov 2017: 9). Dennoch kann laut eigenen Angaben Docker ebenfalls eine hohe Sicherheit für virtualisierte System bieten (vgl. Docker Inc. 2022).

3 Automatische Reportlösung

Wie bereits im Abschnitt Softwarehistorie abschließend erörtert wurde, ist Software fehleranfällig und muss kontinuierlich aufbereitet werden. Durch Softwaretests wie etwa Unittests, in welchen verschiedene Komponenten des Source-Codes einzeln getestet werden oder Code-Covering-Tests, die den Prozentsatz des durch einen Test ausgeführten Programmcodes ausgeben, um effizientere Programmstrukturen zu realisieren, können Programmierfehler oft lange vor der Erstellung neuer Builds entdeckt werden. Ein vollwertiger Build beinhaltet neben dem erfolgreichen Durchlaufen verschiedener Revisionen auch die Inspektion durch Softwareentwickler und die richtige Zusammensetzung unterschiedlicher Programmabschnitte und Funktionen durch Programmiersprachen spezifischen Build-Tools (vgl. Duvall et al. 2007: 4).

CI/CD **C**ontinuous **I**ntegration / **C**ontinuous **D**eployment ist ein automatisierter Prozess, welcher meist auf einem eigens dafür konzipierten Server stattfindet. Veränderungen, welche Entwicklern auf das VCS pushen, werden abgegriffen und in ein Integration-Build assimiliert. Um die Integrität des veränderten Programmcodes sicherzustellen, wird eine Vielzahl an Tests durchlaufen, bevor der aktuell stabile Build ersetzt wird. Kann der neue Build die benötigten Qualitätsanforderung nicht erfüllen, wird dieser verworfen. Durch die von der Entwicklung separierte Test- und Buildumgebung ist es nicht nur möglich Nightly-Builds[20] zu kreieren, vielmehr ermöglicht dies ebenfalls persistente Tests wie Fuzzing[21] auszuführen. Durch kontinuierliches Testen können viele Defekte bereits in einem frühen Stadium entdeckt und behoben werden. Ebenfalls sind durch dieses Vorgehen Zustände wie die Softwarekomplexität, über einen bestimmten Zeitraum messbar. Ein weiterer Vorteil des immer gleich ablaufenden Prozesses, der nach jedem Push auf das VCS durchgeführt wird, ist, dass diese Automation den Entwickler befähigt, mehr Zeit aufzuwenden, um substanzielle Arbeit zu verrichten (vgl. Duvall et al. 2007: 29ff). Die Antipathie gegen Continuous Integration hält sich in vielen Entwicklerteams und -studios jedoch auch heute noch stark. Oft wird der steigende Overhead, entstehend durch die Pflege von CI Systemen, bemängelt. Meist wird bei dieser Argumentation jedoch nicht bedacht, dass die Software dennoch getestet, integriert und gebuildet werden muss und eine manuelle Implementation ebenfalls pflegebedürftig und dabei fehleranfälliger ist. Der Aspekt der redundanten Investition in benötigte Hard- und Software ist ein weiterer Stereotyp, welcher oftmals für die Rechtfertigung gegen den Einsatz von CI verwendet wird. Es gilt jedoch zu beachten, dass Mängel, die im späteren Verlauf eines Softwareprojekts auftauchen, meist mit deutlich höheren Kosten verbunden sind (vgl. Duvall et al. 2007: 32ff). Sympathie hingegen kommt vermehrt durch Externe wie Klienten und Produkt-Owner. Diese äußerlich mitwirkende Kraft kann durch CI/CD zu jedem Zeitpunkt in der

20. Builds, die automatisch erzeugt und zumeist außerhalb der täglich Arbeitszeit ausgeführt werden.
21. Der Begriff Fuzz wurde 1990 durch das von Barton Miller entwickelte Tool zum Testen der Zuverlässigkeit von UNIX-Tools geprägt. Fuzz generiert zufällige Eingabewerte, welche das jeweilige Programm verarbeitet. Kommt es zu einem Fehler, gilt der Test als ungültig und das Programm als unzuverlässig (vgl. Miller et al. 1990: 34) (vgl. Oehlert 2005 : 59). Durch Log4shell geriet insbesondere Fuzzing erneut ins Licht der Öffentlichkeit. Durch diese Entwicklung entschied sich das Entwicklerteam der Programmiersprache Go kurzerhand Fuzzing in Version 1.18 in die Standardbibliothek mit aufzunehmen (vgl. The Go Projekt 2022).

Entwicklung auf eine funktionsfähige Version der Software zurückgreifen und nutzen. Dies kann unterstützenden Unternehmen gegenüber ihrer Konkurrenz einen wirtschaftlichen Vorteil ermöglichen (vgl. Duvall et al. 2007: 31).

Kontinuierlich weiterentwickelte Projekt, wie Webanwendungen, bieten zumeist eine Vielzahl an Sicherheitslücken. Diese können von Angreifern genutzt werden, um das System zu infiltrieren. Eine Webanwendung besteht aus vielen Einzelteilen: ein Webserver, eine Datenbank, JavaScript-Applikationen und Weiteren. Ein Entwicklerteam, welches beispielsweise eine Webseite entwickelt, verfügt jedoch meist nicht über die nötigen Ressourcen z. B. einen eigenen Web-Server zu entwickeln. Aus diesem Grund muss Software wie Nginx[22] aus anderen Softwareprojekten verwendet werden. Wird eine Sicherheitslücke in der Software eines Drittanbieters gefunden, wird diese geschlossen und eine neue Version veröffentlicht. Durch die meist hohe Anzahl der verwendeten Softwarepakete steigt die Komplexität stets die letzte Fassung aufzuspielen.

Renovate eine Software, entwickelt von WhiteSource. WhiteSource hat sich zum Ziel gesetzt, die Sicherheit für Open-Source-Projekte zu steigern (vgl. Software 2022b). Dabei agiert Renovate als Tool, welches Softwareabhängigkeiten in einem VCS erkennt und updatet. Dies ist wie im vorherigen Absatz erwähnt ein wichtiger Prozess bei fortlaufender Entwicklung, da Bugs und Sicherheitslücken durch Updates geschlossen werden. Renovate commited Updates automatisch auf das VCS und beschreibt den Vorgang in einer Commit-Nachricht (vgl. Software 2022a). In dem Kontext dieser Arbeit ist diese Nachricht der wichtigste Bestandteil zur Erstellung der automatisierten Reportlösung.

Programmtechnische Inhalte variieren oft je nach Betriebssystem. Oftmals handelt es sich hierbei um kleinere Differenzen wie dem Abschließen einer Zeile: (\r\n) Windows, (\n) Linux & UNIX/BSD/MAC. In anderen Situationen wie bei der Verwendung von Threads und Prozessen kann dies jedoch auch größere Auswirkungen haben. Aus diesem Grund beschränkt sich die Implementierung des Programms im Folgenden auf Linux.

3.1 Wahl der Programmiersprache

Zum Entwickeln der Software habe ich mich in diesem Projekt aus verschiedenen Gründen für die Programmiersprache Python entschieden. Python ist verschiedenen Indizes zufolge die beliebteste und am häufigsten verwendete Programmiersprache weltweit (TIOBE Software BV 2022). Diese Tatsache lässt sich durch die vielfältigen Einsatzmöglichkeiten beginnend mit der KI-Entwicklung über Big-Data-Science bis hin zu Webserveranwendungen mit Frameworks wie Django und Flask. Zusätzlich ist Python eine Open-Source-Programmiersprache und hat wie in Kapitel Softwarehistorie beschrieben entscheidende Vorteile gegenüber proprietären, aber auch vielen alten Programmiersprachen. Python verfügt nicht zuletzt auch über verschiedene Features, deren Intention es ist, die interpretierte Programmiersprache zu beschleunigen, um eine ähnliche Rechenleistung zu erhalten, wie mit etablierten und kompilierten Programmiersprachen wie C oder Fortran.

22. Oft eingesetzter Webserver, entwickelt 2004 von Igor Sysoev (vgl. F5 Network Inc. 2022)

Numba ist eine Open-Source Erweiterung für Python, mithilfe des LLVM Compilers werden Funktionen dynamisch während der Laufzeit in Maschinencode umgewandelt. Ebenfalls ist es möglich Code-Fragmente mithilfe dieses Tool nebenläufig auf der CPU, jedoch auch auf der GPU ablaufen zu lassen. Die gesamte Funktionalität von Numba wird durch Decorator im Source-Code zugeschaltet, was das Anwenden sehr benutzerfreundlich gestaltet (Anaconda 2018).

Cython ist eine weitere Möglichkeit, Python gravierend zu beschleunigen. Cython ist eine sehr nah an C orientiert und kompilierte Programmiersprache, welche nativ in Python eingebunden werden kann. In folgender Tabelle wird ersichtlich, wie viel Nutzen Cython augenscheinlich bieten kann. Hierbei wurden die Zeit gemessen, welche von den verschiedenen Programmieransätzen benötigt wurde, um *N* Fibonaccizahlen zu berechnen. Kolumne 6 (Loop body) beschreibt die Differenz der beiden Ausführungszeiten fib(90) und fib(0). Kolumne 7 drückt die relative Beschleunigung der beiden Versuche an.

Version	fib(0) [ns]	Speedup	fib(90) [ns]	Speedup	Loop body [ns]	Speedup
Pure Python	590	1	12,852	1	12,262	1
Pure C	2	295	164	78	162	76
C extension	220	3	386	33	166	74
Cython	90	7	258	50	168	73

Abbildung 6: Timings von verschiedenen Fibonacciimplementationen Quelle: Smith 2015 S. 3

Bei einer kleinen Anzahl der zu berechnenden Fibonaccizahlen kann durch den zusätzlichen Overhead kaum eine Beschleunigung festgestellt werden. Mit steigender Quantität jedoch ist auch eine erhebliche Abnahme an Rechenzeit zu verzeichnen. Wie in Abbildung 6 zu erkennen, ist es mithilfe von Cython möglich beinahe Ausführungszeiten wie in C zu erreichen. Es ist wichtig zu erwähnen, dass I/O, Netzwerkkommunikation und große Datenoperationen kaum von Cython profitieren können. Deshalb ist es wichtig, Python-Code zunächst zu profilieren, um festzustellen, ob tatsächlich Optimierungen vorgenommen werden können (Smith 2015: 7).

C-Extension ist eine weitere Möglichkeit die Performance von Python zu optimieren. Hierfür wird ein natives C-Modul in Python mithilfe einer API eingebunden. Wie in Spalte 2 der Abbildung 6 zu erkennen, ist hier ebenfalls eine deutliche Beschleunigung von Python-Programmen möglich. Python agiert hier als eine Wrapper-Programmiersprache, die es ermöglicht abstrakten Code zu schreiben und rechenintensive Programmsektoren in C-Code auszugliedern.

Die vorgestellten Python-Erweiterungen sind drei der bekanntesten Tools zur Optimierung von Python-Code. Dies liegt auch daran, dass große Firmen wie Intel und Nvidia Projekte wie Numba fördern und selbst verwenden (Anaconda 2018). Viele Entwickler, welche bisher C/C++ entwickelt haben, nutzen Python vermehrt als Wrapper für komplexere Programme und beschleunigen diese mithilfe von Cython oder C-Extensions, da sie

hiermit bereits vertraut sind. Mit zunehmendem Umweltbewusstsein in der Bevölkerung ist für viele Firmen auch deren ökologischer Fußabdruck vermehrt von Bedeutung. Einer Studie des Umweltbundesamtes zufolge war es 2020 65 % der Befragten, das Thema Umwelt und Klimaschutz '*sehr wichtig*' (Umweltbundesamt 2022). Aus diesem Grund haben Wissenschaftler der Universität Minho in Portugal die Energieeffizienz verschiedener Programmiersprachen verglichen. Anders als zunächst angenommen, konnte bewiesen werden, dass Geschwindigkeit nicht immer mit dem Energieverbrauch gleichzusetzen ist (Pereira et al. 2017: 260). Durch verschiedene Tests von Speichernutzung, Energiekonsum und Zeit konnte des Weiteren ermittelt werden, dass C, Pascal und Go die energieeffizientesten Programmiersprachen verkörpern, während Python im Mittelfeld mit Haskell, JavaScript und PHP einordnet (Pereira et al. 2017: 265). Die beliebteste Programmiersprache kann demnach mithilfe der bereits vorgestellten Erweiterungen auch zu einer der energieeffizientesten umgewandelt werden. Dieser Aspekt kann im Umkehrschluss dazu führen, dass mehr Unternehmen das Potenzial der Programmiersprache entdecken und ebenfalls zur Popularität der Sprache beitragen.

3.2 Gitlab

Für die meisten Entwickler ist GitLab ein Tool, mithilfe dessen Source-Code versioniert werden kann. Hier gibt es zwei Möglichkeiten: a) Es existiert bereits Source-Code auf dem GitLab-Server, für die Weiterentwicklung wird hier eine lokale Kopie auf den Rechner *geklont*[23] b) Es wird ein neues Projekt angelegt und hierzu wird ebenfalls ein neues Repository kreiert, welches anschließend auf den GitLab-Server gepushed wird. In diesem Projekt geht es jedoch nicht darum, Source-Code zu pushen oder zu klonen. Vielmehr müssen Informationen über Repositories[24], welche auf einem GitLab-Server liegen, abgegriffen werden.

3.2.1 Git-CLI

Die einfachste Methode, um an Informationen über ein Repository zu gelangen, ist die von Git implementierte Funktion des Klonens. Hier gibt es verschiedene Möglichkeit: Mithilfe eines Downloadbuttons über das Webinterface von GitLab und über die CLI[25] zur Verfügung gestellt. Abschließend sei erwähnt, dass es auch verschiedene grafische Implementationen von Git gibt, welche ähnliche Funktionen bieten. Auf diese Tools möchte ich an dieser Stelle jedoch nicht näher eingehen.

SSH (**S**ecure **SH**ell) ist eine beliebte und nützliche Variante Netzwerkkommunikationen zu verschlüsseln. SSH verschlüsselt hierbei die eingehenden Daten und entschlüsselt diese erneut bei jeweiligen Empfängern. Dabei bietet SSH einige Vorzüge, wie die Legitimation von Usern sowie die Ausführung von Remote-Befehlen mithilfe aktueller Verschlüsselungstechniken (vgl. Byrnes et al. 2005: 1f). Um die Authentifizierung mit

23. clone z.D. klonen, beschreibt in diesem Kontext den Kopiervorgang von Source-Code von einem Server zu einem lokalen Computer
24. Ein Repository beschreibt alle Projektdaten, inklusive Source-Code und Metadaten.
25. **C**ommand **L**ine **I**nterface. Programm, welches auf der Kommandozeile/Terminal ausgeführt wird.

einem GitLab-Server durchzuführen, wird ein privater sowie öffentlicher Schlüssel benötigt. Dieses Schlüsselpaar kann lokal auf einem Computer wie folgt generiert werden. werden.

```
$ ssh-keygen -t ecdsa -b 521 -f GitLabKey
```

Listing 1: SSH-Key Erstellen Quelle: In Anlehnung an Byrnes et al. 2005 S. 233

Mit dem Konsolenbefehl *ssh-keygen* kann ein neues SSH-Schlüsselpaar generiert werden. Um die Art der Verschlüsselung zu wählen, ist die Option *-t* notwendig. Zu Beachten ist, dass hier auch veraltete Verschlüsselungsverfahren wählbar sind, weshalb es wichtig ist vor Erstellung den aktuellen Stand der Kryptografie abzufragen. Anschließend kann mit der Option *-b* die Größe des Schlüssels gewählt werden, hier gilt das Prinzip größer ist gleich besser. Mit der Option *-f* ist es zusätzlich noch möglich dem Schlüssel einen spezifischen Namen zu geben. Es sind weitere optionale Parameter vorhanden, diese werden jedoch nicht für die sichere Kommunikation mit einem Server benötigt. Der öffentlich Schlüssel wird anschließend auf dem zu erreichenden Server abgelegt. Bei einem Gitlab-Server wird dieser Vorgang über das Webinterface vorgenommen. Dieses Vorgehen ist zwingend erforderlich, da GitLab-Server in der Standarteinstellung keine manuelle Authentifizierung zulassen. Durch öffentliche Schlüssel entstehen sogenannte digitale Fingerabdrücke, mit welchen es möglich ist einen Server zu identifizieren. Wird versucht eine SSH-Verbindung zu einem noch unbekannten Server (repräsentiert durch dessen Fingerabdruck) aufzubauen, wird der User mittels Prompt[26] gebeten diesen zu verifizieren. Um Man-in-the-Middle-Angriffe[27] zu vermeiden und eine sichere Kommunikation mit dem Server zu gewährleisten, ist es zwingend erforderlich, diesen digitalen Abdruck mit dem Administrator des Servers abzugleichen (vgl. Alicherry and Keromytis 2009: 557). Um eine möglichst hohe Sicherheitsstufe zu realisieren, ist es möglich diese digitalen Fingerabdrücke manuell abzugleichen. Für eine Automatisierung des Prozesses gibt es die Möglichkeit in der SSH-Konfigurationsdatei die Einstellung *StrictHostKeyChecking* zu deaktivieren (vgl. OpenSSH 2019), jedoch wird hiervon das gesamte System beeinfluss und damit jede SSH-Verbindung, weshalb ich mich in diesem Projekt für eine andere Methode entschieden habe. Um den *mutmaßlichen* Fingerabdruck eines Servers abzufragen, kann folgendes Tool verwendet werden.

```
$ ssh-keyscan -H -p ServerPort -t rsa,ecdsa,ed25519 ServerIP
```

Listing 2: SSH-Fingerabruck eines Servers abfragen

Die *-H* Option generiert einen Hash für die Serverbezeichnung. In meiner Recherche konnte ich jedoch keine Begründung finden, welche dieses Vorgehen rechtfertigt, jedoch ist dies die Standardmethode Serverbezeichnungen abzuspeichern. Verwendet der zu erreichende Server nicht den Standard-SSH-Port (22), ist es erforderlich den Port mit der Option *-p* anzugeben. Falls nicht bekannt ist, welches Verschlüsselungsverfahren angewendet wird, kann mit der Option *-t* eine Vielzahl an Möglichkeiten angeben werden. Als Standard gilt jedoch auch heute noch RSA, welches ein veraltetes Verfahren darstellt und nicht als sicher eingestuft wird (vgl. SSH.com 2022b). Beendet wird der Befehl mit der IP-Adresse des zu erreichenden Servers. Anschließend erhält man einen wie in Abbildung 7 dargestellten Hash-Wert, welcher einen Fingerabdruck repräsentiert.

26. z.D. Eingabeaufforderung
27. Der Angreifer infiltriert hier die Verbindung zum Server und agiert als Proxy (vgl. SSH.com 2022a)

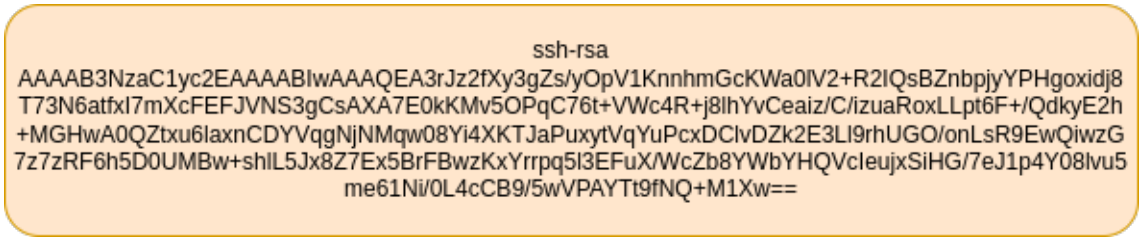


Abbildung 7: Beispiel eines SSH-Fingerabdrucks Quelle: Eigene Darstellung

Diese digitale Kennzeichnung wird in der Datei `~/.ssh/known_hosts` abgespeichert. Beim Aufbau einer SSH-Verbindung wird diese Datei, falls nicht anders angegeben, eingelesen und vorhandene digitale Fingerabdrücke mit dem des Servers abgeglichen. Kommt es zu einer Übereinstimmung, wird die Verbindung initialisiert. Ist dies nicht gegeben, kommt es zu bereits erwähntem Prompt. Um auf Repositoryinformationen zugreifen zu können, muss zunächst eine lokale Kopie angelegt werden.

```
1 $ git clone ssh://git@ServerURL:ServerPort/GitRepository.git
```

Listing 3: Git-Repository klonen

Mithilfe dieses *git clone* Befehls wird das gesamte Projekt auf den lokalen Computer heruntergeladen. Damit sind auch alle Informationen wie Commit-Nachrichten lokal zugänglich und abrufbar. Mithilfe dieser Informationen ist es nun möglich die Repositories beliebig an hand deren Daten zu sortieren.

```
1 $ git log --after='THH:MM:SS' --author='AutorName' \
2           --pretty=format:'%an;%ai;%s;%b'
```

Listing 4: Git-Logs abrufen

Unter Zuhilfenahme der Option *–pretty* ist es möglich den resultierenden String zu formatieren. Im angegebenen Beispiel soll neben dem Autor (%an), das Datum (%ai) sowie der Bezeichnung des Commits (%s) auch die Nachricht des Commits ausgegeben werden (%b). Anhand des Autornamens kann festgestellt werden, ob der Commit von dem GitLab-Runner getätigt wurde. Mithilfe des Datums kann eingegrenzt werden, wann der Commit durchgeführt wurde und ob dieser von Relevanz ist. Die Commit-Nachrichten sind wichtig, da sie Informationen über den Prozess bereitstellen, hier kann beispielsweise die Veränderung der Version einer Software nachvollzogen werden. Ein weiterer Vorteil der Pretty-Formatierung ist, dass Informationen, die nicht benötigt werden, im Verlauf des Programms keine weiteren Rechenoperationen beanspruchen und hier, wenn auch gering, Zeit und Kosten eingespart werden können.

Automatisierung Die vorgestellten Befehle können wie beschrieben in einer Shell ausgeführt werden, ebenfalls wäre es möglich diesen Prozess in einem Shell-Skript zu automatisieren. Jedoch bietet Python insbesondere bei der Nebenläufigkeit entscheidende Vorteile. Python ist bei der Implementierung von nebenläufigen Programmen um ein Vielfaches benutzerfreundlicher. Hier wird beispielsweise die Möglichkeit geboten, Fehler abzufangen, ohne das Programm direkt zu beenden. Ein Skript verhält sich in diesem Kontext ähnlich wie ein C-Programm und bietet lediglich die Möglichkeit, Signale abzufangen. Popen() und run() sind nur 2 der vielen Möglichkeiten, die Python bietet,

um Konsolenbefehle innerhalb eines Python-Programms auszuführen. Aus Kommunikationsgründen entschied ich mich in diesem Projekt für die Verwendung von Popen. Popen bietet die Möglichkeit, sogenannte Pipes zu übergeben, diese bieten die Möglichkeit den Standardoutput (stdout) sowie den Fehleroutput (stderr) in dafür vorgesehene Bereiche zu untergliedern. Um den Prozess des Downloads zu beschleunigen, entschied ich einen Pool zu erstellen und diese Operation simultan durch mehrere Prozesse auszuführen.

```
1 pool.map(dl_repo, [[key, value['repo_url']] for key, value in
2                                                repo_dict.items()])
```

Listing 5: Python-Multiprocessing-Pool Download Git-Repositories

Durch diese Zeile Code konnte im Vergleich zu der Ausführung mit einem einzigen Prozess eine 12-fache Beschleunigung erzielt werden. Die Funktion *pool.map* ist eine Funktion aus dem Multiprocessingpaket, welches zur Kernbibliothek der Pythonprogrammiersprache gehört. Bei dieser Funktion wird jedes Element, welches in der Liste vorhanden ist, auf die Funktion *dl_repo* gemappt. In dem Kontext dieser Arbeit befinden sich in der Liste *sshurls*, die auf eine Funktion gemappt werden, die diese herunterlädt bzw. klont.

Durch das Downloaden unzählig vieler Dokumente werden Netzwerkressourcen, die an einer anderen Stelle benötigt werden könnten, vergeudet. Dies liegt der Tatsache zugrunde, dass Informationen wie Commitnachrichten, Autor und Datum ein minimaler Bestandteil der Repositoryinformationen ausmacht. Source-Code, Bilder oder ähnliche Dokumente, welche für dieses Projekt nicht von Bedeutung sind hingegen bilden den größten Teil der zu herunterladenden Datenmasse. Das Netzwerk wird durch die Ausführung des Projekts nicht dauerhaft beeinflusst, dennoch fällt die Belastung zeitweise je nach Größe der Repositories deutlich ins Gewicht. Aus diesem Grund erläutere ich in folgendem Kapitel, wie ein ressourcenschonender Zugriff auf die benötigten Daten möglich ist.

3.2.2 GitLab-API

URLs **U**niform **R**esource **L**ocators sind eine Unterklasse der URIs (Uniform Resource Identifiers) und werden genutzt, um mithilfe von Browsern Daten aus dem WWW herunterzuladen. Gibt man in die Adresszeile des Browsers beispielsweise die URL `http://gitlab.com` ein, wandelt dieser die Adresse *gitlab.com* mittels des Domain-Name-Servers (DNS) in eine IP-Adresse um. Anschließend, da kein anderer Port angegeben ist, wird über eine TCP/IP Verbindung versucht der Server auf Port 80 (Standard HTTP-Port) zu erreichen. Gelingt dies, erhält der Browser das Root-Dokument /, welches anschließend interpretiert und grafisch dargestellt wird. URLs können jedoch auch diverse weitere Informationen wie Queries und Angaben zu Ports beinhalten. Ein Demonstrationsbeispiel hierfür wäre http://gitlab.com:8080/Folder%2FOtherFolder/logo?shape=square. Dabei gilt zu beachten, dass für Sonderzeichen wie (!#$&'()*+,/:;=?@[]) eine Percent-Encodierung verwendet wird, deshalb wird in der Pfadangabe *Folder/OtherFolder* das Slash-Zeichen durch *%2F* ersetzt. Mit dem Fragezeichen beginnt die Query, dabei wird der Teil vor dem Gleichheitszeichen als Parameter und der darauffolgende Teil als Value bezeichnet. Falls mehrere Queries benötigt werden, können diese zusätzlich mit einem Undzeichen unterteilt werden (vgl. Rhodes and Goerzen 2010: 138f). In Python gibt es verschiedene Kernbibliotheken, die den Prozess des URL-Erstellens vereinfacht, da das manuelle Erstellen der codierten Strings zu einer hohen Fehlerrate führen würde.

```
from urllib.parse import urlencode, urlunparse

query: str = urlencode({parameter: Value})
url: str = urlunparse((Protokol, netloc, NetzwerkPfad,
                       '', query, ''))
```

Listing 6: URL erstellen mit Python

Wie in dem Codebeispiel zu erkennen, genügt es die Funktionen urlencode und urlunparse zu importieren, um die URL-Erstellung durchzuführen. Mithilfe dieser Funktionen kann nun eine Query erstellt werden. Dies funktioniert über ein Python-Dictionary, das Value wird dabei direkt auf den Parameter gemappt. Abschließend wird mittels *urlunparse* die URL erstellt. Dazu wird in eine Python-Tuple das jeweils auszuführende Protokoll (HTTP, FTP, HTTPS usw.) gefolgt von der Netzwerkadresse wie beispielsweise „Python.org“ und falls vorhanden, der Netzwerkpfad sowie die Query. Die Funktion urlunparse erwartet eine Tuple mit 6 Elementen, auf Position 3 und 5 werden jedoch wie in dem Code-Beispiel leere Strings erwartet (vgl. Sarker and Washington 2015: 133ff).

HTTP-Methoden sind Möglichkeiten, mit Servern zu kommunizieren. GET, HEAD, PUT, POST, TRACE. OPTIONS, DELETE sind einige dieser Methoden, welche bereits seit HTTP Version 1.1 unterstützt werden. Dabei wird zusätzlich zwischen sicheren und unsicheren Methoden unterschieden. GET und HEAD sind laut Definition, die einzigen zwei Methoden, die auf dem Server keinerlei Veränderung hervorrufen. Jedoch kann dies durch die Implementation der Webseite/Server variieren. (Gourley et al. 2002: 53).

Die GET-Methode ist die wohl am häufigsten verwendete HTTP-Methode. Sie wird verwendet, um Ressourcen eines Servers anzufragen. Dabei wird eine Request-Nachricht an den Server gesendet, wird diese akzeptiert, sendet der Server eine Response-Nachricht mit HEADER und BODY (vgl. Gourley et al. 2002 : 53f). Der HEADER einer solchen HTTP-Nachricht beinhaltet Eigenschaften des Requests/Response, wie zum Beispiel die verwendete HTTP Version, einen Statuscode, welcher den Status der Nachricht impliziert. Ein solcher Statuscode variiert zwischen 100 - 599, dabei sind verschiedene Zahlenabschnitte unterschiedlichen Themen zugeordnet. Die Codes 100 - 199 stehen für Informationscodes, 200 - 299 sind erfolgreiche Statusmeldungen, 300 - 399 stehen für eine serverseitige Weiterleitung, 400 - 499 sind der Kategorie Client-Error zugeordnet und 500 - 599 der Sparte der Server-Error (Gourley et al. 2002 : 49). Des Weiteren können im HEADER die Uhrzeit für die erstelle Nachricht gespeichert sein, Größe und Typ des angefügten BODYS, der Name des verwendeten Browsers und viele weitere Informationen über den Verfasser der Nachricht (vgl. Gourley et al. 2002: 51f). Der BODY z.D. Körper umfasst das Transportelement von HTTP, welches auch ursprünglich als Transportprotokoll entwickelt wurde. In diesem Teil der Nachricht können sowohl Bilder, Videos, HTML-Dokumente oder auch Software übertragen werden (vgl. Gourley et al. 2002: 52). Die HEAD-Methode unterscheidet sich von der GET-Methode lediglich dadurch, dass die Response lediglich den HEADER beinhaltet, jedoch keinen BODY. Die POST-Methode wurde entwickelt, um Daten zum Server zu senden. Diese Methode wird allgemein für Eingabefelder auf Webseiten genutzt (Gourley et al. 2002: 54f). Weitere Einzelheiten über viele weitere HTTP-Methoden können in dem von Buch HTTP: The Definitive Guide von David Gourley & Brian Totty nachgelesen werden.

Um Nutzern, die Webseiten mittels Browser aufsuchen, wird das Resultat der POST-Methode zumeist mit einem Redirect abgeschlossen. Dies hat den Vorteil, dass wenn die URL geteilt oder schlicht erneut aufgerufen werden die Form nicht erneut ausgefüllt werden muss. Des Weiteren ist für Programme, welche Informationen von Webseiten nutzen oder diese sammeln, ein anderes Datenformat wie JSON oder XML oft von Vorteil. Diese Weiterleitungen, welche unausweichlich eine erhöhte Netzwerkauslastung herbeiführen sowie der Vereinfachung von Informationsaustausch durch andere Formate sind zwei der vielen Gründe, weshalb APIs für Webseiten entwickelt wurden. Aufgangs wurden für viele APIs lediglich die Methoden GET und POST verwendet, dies erwies sich als unzureichend, da es auf diese Art schwer nachzuvollziehen ist, ob Ressourcen erstellt, abgefragt oder entfernt werden, deshalb entwickelte Roy Fielding in seiner Dissertation zur Jahrtausendwende das Designpattern *„Representational State Transfer“*, welches schon bald als REST bekannt wurde und auf vielen Webservern zum Einsatz kam. Das Designpattern spezifiziert, dass die Methoden PUT, GET, POST und DELETE namensgebend zum Erstellen, Abfragen, Modifizieren und Entfernen von Dokumenten verwendet werden (vgl. Rhodes and Goerzen 2010: 150ff). Gegenüber vielen anderen APIs bietet REST viele Vorteile. Ein wichtiger Aspekt ist, dass keine Zustände gespeichert werden, jeder API-Aufruf verfügt über die notwendigen Daten, um die Abfrage abzuschließen. Durch diese Zustandslosigkeit müssen viele Abfragen gecacht werden, was im Umkehrschluss zu einer beschleunigten Anwendung führt. Ein weiterer wichtiger Faktor, welcher die Beliebtheit von REST weiter vorantreibt, ist die Möglichkeit der Authentifikation. Hierfür werden drei Möglichkeiten geboten, alle 3 beinhalten, dass der Nutzer der API in der Regel zunächst einen Account auf dem Server anlegt. Ist ein Benutzerkonto vorhanden gibt, es die Möglichkeit die Zugangsdaten mittels HEADER zu übertragen. Bei der Übertragung mittels HTTP, wird das Passwort in Base64 dekodiert, jedoch bietet diese „Codierung“ in etwa so viel Schutz wie die Caesar-Verschlüsslung[28]. Aus diesem Grund ist es wichtig, bei dieser Art der Authentifizierung, auch genannt **BasicAuth** darauf zu achten, dass HTTPS als Protokoll verwendet wird. GitLab verwendet wie viele andere Webservices (Twitter, Twitch usw.) die Authentifizierungsmethode **OAuth2.0**. Bei dieser Methode erhält der User einen Token, welcher Passwort und Benutzername ersetzt. Ein Vorteil ist, dass das Ändern von Passwörtern den Token nicht verändert und damit die Software auch weiterhin ohne Kompromisse nutzbar ist (Amundsen et al. 2013: 250ff). Das folgende Codebeispiel, veranschaulicht, wie diese Authentifizierung mit Python durchgeführt werden kann.

```
1 import requests
2
3 header = {'PRIVATE-TOKEN': f'{Token}'}
4 response = requests.get(url, headers=header)
```

Listing 7: Python HTTP-Request mit OAuth2.0

Zunächst wird eine URL benötigt, diese kann wie in Listing 6 dargestellt erstellt werden. Anschließend wird der Header erstellt, dieser beinhaltet den Token, welcher im Web-Interface auf dem jeweiligen GitLab-Server angefordert werden kann. Anschließend kann die HTTP-GET-Methode den erstellten Strings ausgeführt werden, um Daten vom Server

28. Zeichen wird durch ein im Alphabet folgendes Zeichen ersetzt. Ist der Verschlüsselungsgrad beispielsweise 2, wird ein „A“ durch ein „C“, ein „B“ durch ein „D“ und ein „Z“ durch ein „B“ ersetzt

abzufragen. Mit der Kombination dieser Funktionen sowie einer weiteren Scrollfunktion, wie sie GitLab nutzt, ist es nun möglich, die benötigten Daten von einem GitLab-Server abzufragen, ohne dass dies das Downloaden von Repositories beinhaltet.

```
1  idx: int = 0
2  projects: List[str] = []
3  while True:
4      query: str = urlencode({'per_page': f'100', 'page': f'{idx}'})
5      url: str = urlunparse((protocol, netloc, f'{api/v4/}projects/', '',
       query, ''))
6      header = {'PRIVATE-TOKEN': f'{Token}'}
7      response = requests.get(url, headers=header)
8      data: List[str] = [str(line) for line in response.json()]
9      if not data:
10         break
11     projects.extend(data)
12     idx += 1
```

Listing 8: Python HTTP-Request mit Python

GitLab ermöglicht lediglich 100 Einträge pro „Seite“, sind für die Abfrage jedoch mehr Daten erforderlich, kann mittels der Scrollfunktion eine mehrseitige Anfrage erstellt werden. Dies kann wie in Listing 8 in Zeile 2 durch eine Query angegeben werden. Durch das Inkrementieren der Seitenzahl, wie hier im Beispiel Variable *idx* kann so auf alle Einträge zugegriffen werden. Durch die Funktion *response.json()* wird der BODY welcher einen JSON-String enthält direkt in eine Python-Liste umgewandelt. Diese Funktion fungiert als Parser und kann in manchen Situationen von Nutzen sein.

JSON **J**ava**S**cript **O**bject **N**otation ist ein weit verbreitetes Format für den digitalen Datenaustausch und wurde 1997 von Douglas Crockford entwickelt. Der Erfolg des Formates ist dem Erfolg von JavaScript zugute zu schreiben. Da die Syntax von JSON des eines JavaScript-Objekts beinahe identisch ist, die Wahl des Kommunikationsformats bei der Webentwicklung meist obligatorisch (Safris 2019). In Listing 9 ist eine stark reduzierte Darstellung der Responsenachricht eines GitLab-Servers. Dieser String wurde wie in Listing 8 geschildert abgerufen. Für die Vereinfachung wurde der String stark gekürzt, da viele übermittelte Informationen nicht von Relevanz für dieses Projekt sind.

```
1  [
2    {
3       "id":1,
4       "description":"Die ist das erste Projekt",
5       "name":"ProjektName01",
6       "name_with_namespace":"Vorname Nachname",
7       "ssh_url_to_repo":"ssh://git@GitLabServer.com/Vorname.
     Nachname/ProjektName01.git",
8       "last_activity_at":"2015-07-22T16:17:09.000Z"
9    },
10   {
11      "id":2,
12      "description":"Dies ist das zweite Projekt",
```

```
13      "name":"ProjektName02",
14      "name_with_namespace":"Vorname Nachname",
15      "ssh_url_to_repo":"ssh://git@GitLabServer.com/Vorname.
    Nachname/ProjektName02.git",
16      "last_activity_at":"2015-07-22T16:17:10.000Z"
17    }
18 ]
```

Listing 9: Stark simplifizierter JSON-String aus GitLab-REST-API Response

3.3 Regex

Der Mathematiker Stephen Kleene entwickelte 1956 erste theoretische Ansätze zu Regular-Expressions. Einige Zeit später, übersetzte Ken Thompson die Theorie in die Praxis und entwickelte die erste Implementation von Regex für die Programmiersprache QED. In QED wurde Regex wie folgt ausgeführt.

```
1 g/<regular expression>/p
```

Listing 10: QED Regex

Verwendet man anstelle *regular expression* die Abkürzung *re*, ergibt sich aus diesem Programmcode *g/re/p*. Dadurch erhielt auch das später entwickelte UNIX-Tool *grep* seinen Namen. Bekannt wurde Regex jedoch erst durch die von Larry Wall entwickelte Programmiersprache Perl. Die Perl Entwickler fügten viele nützliche Features zu dem bisherigen Regex hinzu, was zu einer Abspaltung des Originals und zu verschiedenen Varianten führte. Heute gibt es 2 verbreitete Varianten: Perl und POSIX (vgl. López and Romero 2014: 6ff). Regex ist heute in vielen Tools und Programmiersprachen enthalten und wird verwendet, um Teilstrings zu finden und/oder zu ersetzen. Dabei setzt Regex auf sogenannte Pattern, um den gesuchten String zu finden. Ein String, welcher zu einem Pattern passt, wird auch als Match bezeichnet. Pattern können aus verschiedenen Token bestehen. Ein solches Token ist beispielsweise die **Zeichenklasse**, diese besteht aus Zeichen welche von eckigen Klammern eingeschlossen sind.

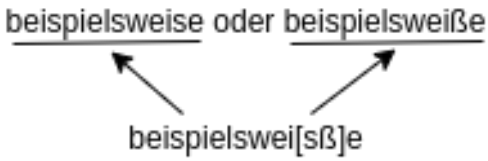


Abbildung 8: Beispielanwendung Zeichenklasse
Quelle: In Anlehnung an López and Romero 2014 S. 11

In der deutschen Sprache findet eine kontinuierliche Rechtschreibreform statt. Dadurch hat sich die Schreibweise einiger Wörter drastisch verändert. Das „scharfe S“ (ß) wird in den neuen Reformen mehr und mehr aus dem deutschen Wortschatz entfernt. Veränderungen dieser Art sind für einen Parser besonders wichtig und müssen berücksichtigt werden. Die Zeichenklasse ermöglicht wie in Abbildung 8 dargestellt eine Option, dadurch können beide Wörter trotz unterschiedlicher Schreibweise gefunden werden. In der Zeichenklasse ist es ebenfalls möglich eine Reihe zu nutzen. Reihen können mittel eines Bindestrichs realisiert werden. *[a-zA-Z]* kann beispielsweise genutzt werden, um alle Buchstaben, die in der

ASCII-Tabelle vorkommen abzubilden. *[0-9]* bietet die Möglichkeit, alle Zahlen darzustellen (vgl. López and Romero 2014: 11). Zusätzlich bietet Regex-Engines vorgefertigte Zeichenklassen, wie *\w ([a-zA-Z0-9_]), \d ([0-9]) und \s (Jegliche Whitespacezeichen)*. Um eine Zeichenklasse zu negieren, genügt es die Klammer mit einem Carot-Zeichen zu beginnen [^0-9]. Der Punkt (.) gilt bei Regex als besonderes Zeichen, er symbolisiert alle, außer dem Newline-Zeichen *\n* (vgl. López and Romero 2014: 13). Ein weiterer wichtiger Bestandteil von Regex sind **Quantoren**. Regex beinhaltet vier verschiedene Möglichkeiten, um Wiederholung auszudrücken. Das Fragezeichen (?) symbolisiert eine oder keine Wiederholung des vorangehenden Terms, der Stern (*) steht für keine oder unendlich viele Repitionen, das Pluszeichen (+) hingegen für eine oder unendlich. Des Weiteren gibt es die Option einen exakten Quantor mittel geschweifter Klammer anzuwenden {4} (viel Wiederholungen, {4,5} vier - 5 Wiederholungen, {4,} vier bis Unendlich (vgl. López and Romero 2014: 17).

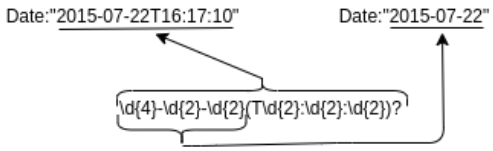


Abbildung 9: Beispielanwendung Quantoren
Quelle: In Anlehnung an López and Romero 2014 S. 19

Der Fragezeichenquantor wird auch oft als Option bezeichnet. Untersucht man beispielsweise einen String, der ein Datum und eventuell zusätzlich eine Uhrzeit beinhaltet, ist es möglich, den Teil mit der Zeit wie in Abbildung 9 dargestellt einzuklammern und anschließend mit einem Fragezeichenquantor zu versehen. Dadurch ist es möglich, das Datum (im angegebenen Format) in einer Zeichenkette zu finden, auch wenn diese wie auf der rechten Seite keine Uhrzeit enthält. Für eine lange Zeit war XML das beliebteste Format Daten zu übertragen, dieser Rang wird wie bereits erwähnt derweil von JSON besetzt, dennoch ist die Kommunikation mittels XML weiterhin stark vertreten. XML besitzt sogenannte Elemente oder Tags, welche wie bei HTML mit größer/kleiner Zeichen eingeklammert werden.

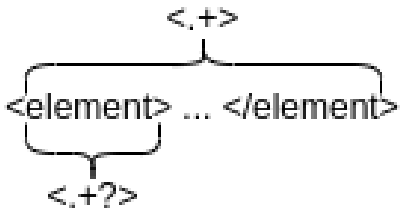


Abbildung 10: Greedy und non-Greedy Quantoren
Quelle: In Anlehnung an López and Romero 2014 S. 20

Die Elementnamen sind generisch und können nicht statisch angenommen werden. Wird ein bestimmtes Element benötigt, kann dieses mit Regex ausfindig gemacht werden. Um einen Elementnamen zu finden, bietet es sich an, für Zeichen zwischen größer/kleiner Klammern zu suchen, da diese in XML reservierte Zeichen sind. Der Elementname kann auch beinahe jedem Zeichen bestehen, weshalb es sich anbietet, den Punkt (.) zu verwenden, da dieser wie in Zeichenketten erläutert fast alle Zeichen darstellt. Wird demnach eine unbestimmt lange Zeichenkette, welche von <> eingegrenzt wird gesucht, könnte man wie in Abbildung 10 oben abgebildet, versuchen dies mit <.+> zu probieren. Das Pattern sucht augenscheinlich eingangs nach einem Kleiner-Symbol, anschließend nach mindestens einem oder unendlich vielen Zeichen, bis es auf ein Größer-Symbol stößt. Die Regex-Engine jedoch löst dieses Pattern „Greedy“, dies bedeutet, dass zunächst nach einem Kleiner-Zeichen gesucht wird, anschließend nach einem oder unendlich vielen Zeichen, was in diesem fall dem Ende der Zeile (Der Punkt ersetzt standardgemäß keine

Newline-Zeichen) entspricht und dann rückläufig nach einem Größer-Zeichen, was dem letzten Zeichen entspricht. Wird dem Quantor ein Fragezeichen wie im unteren Teil der Abbildung 10, verfolgt die Engine das Ziel die kleinstmögliche Zeichenkette, die dem Pattern entspricht zu finden. Dieses Vorgehen bezeichnet man als „Non-Greedy“ oder auch „Reluctant“. Der vermutlich wichtigste Bestandteil von Regex ist die Funktion der Gruppierung. Gruppen können für unterschiedliche Zwecke eingesetzt werden. Eine Option ist das Erstellen von Unterabfragen, welche mit Quantoren versehen werden können. Ein Beispiel hierfür wurde in Abbildung bereits vorgestellt. Die Unterabfrage ist das Pattern der Uhrzeit, welches mit einem optionalen Quantor (?) ausgestattet ist. Die runden Klammern bilden dabei die Grenze der Unterabfrage bzw. der Gruppe. Ein alternativer Anwendungszweck für Gruppen ist das Extrahieren von Teilstrings eines Matches. Dabei ist in Python zusätzlich möglich, diese Gruppen mit einem Namen zu versehen. Benannte Gruppen können komplexe Regex-Ausdrücke und den dazugehörigen Code insbesondere für das menschliche Auge deutlich vereinfachen. Im folgenden Beispiel werden X und Y Koordinaten in einem XML-Dokument gesucht.

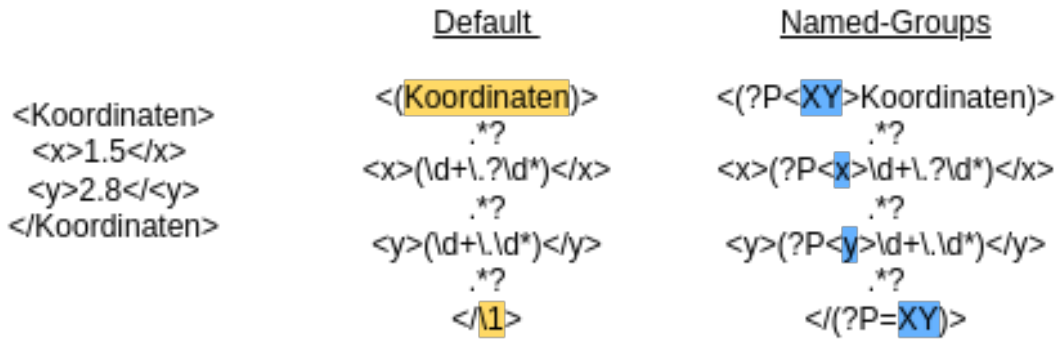


Abbildung 11: Koordinaten-Beispiel Regex Named-Groups Quelle: Eigene Darstellung

In dem „Default“-Regex-Pattern in Abbildung 11 werden die Gruppen lediglich mit runden Klammern umschlossen. Wird eine sogenannte „Backreference[29]“ wie im Beispiel benötigt, kann der Index der Gruppe hierfür verwendet werden. Der Indexwert Null ist eine reservierte Gruppe, die stets das gesamte Match beinhaltet. Aus diesem Grund beginnt die erste Gruppe mit dem Index Eins. Werden hingegen „Named-Groups“ verwendet, kann auf diese Gruppen mit ihrem jeweiligen Namen (Blau markiert) zugegriffen werden, sowohl in Python wie auch in der Abbildung 11 in der Backreferenz. Ein letztes wichtiges Instrument, um Regex-Ausdrücke korrekt zu verwenden, sind Compiler-Flags. Insgesamt gibt es acht verschiedene Flags: IGNORECASE wird verwendet, um Groß- und Kleinschreibung zu ignorieren, MULTILINE ermöglicht das Durchsuchen mehrerer Zeilen unterteilt durch Newline-Zeichen (\n), mit dem DOTALL Flag ist jedes Zeichen, auch das Newline-Zeichen durch einen Punkt ersetzbar, LOCALE wird benötigt, wenn

29. Als Backreference wird in Regex eine Gruppe bezeichnet, deren Match erneut in dem gleichen Pattern auftaucht. Beim Parsen von XML-Elementen ist diese Funktionalität sehr hilfreich, da geöffnete Elemente mit gleichem Namen geschlossen werden müssen.

das lokale Alphabet den ASCII-Zeichensatz überschreitet und dennoch Zeichenklassen wie (\w) verwendet werden, UNICODE wird verwendet, um den Zeichensatz auf UTF-8 zu erweitern, VERBOSE wird verwendet, um Regex-Ausdrücke wie in Abbildung 11 zu kommentieren und DEBUG kann verwendet werden, um Pattern zu debuggen (vgl. López and Romero 2014: 43-47).
Python ist eine der wenigen Programmiersprachen, welche die Möglichkeit bietet sowohl die Perl- als auch die POSIX-Variante zu verwenden. Der größte Unterschied zwischen den beiden Varianten ist die Möglichkeit, dynamische Look-Behinds[30] zu realisieren. Obwohl im folgenden Beispiel kein solcher Look-Behind verwendet wurde, entschied ich für dieses Projekt die POSIX-Variante zu verwenden, da diese Funktionalität in vielen Situationen äußerst hilfreich ist. In Listing 11 wird der JSON-String dargestellt, in Listing 9 mittels Regex-Pattern dargestellt. Im ersten Abschnitt wird die *ID* des Repositories abgefragt. Diese befindet sich immer nach dem String *ïd":* und vor dem darauffolgenden Komma. Die Zeichen (") und (:) können mit der Zeichenklasse (\W), was dem Gegenteil von (\w) entspricht, beschrieben werden. Die *ID* selbst kann lediglich aus einer ganzzahligen Ziffer bestehen, weshalb es genügt diese mit + zu deuten. Zu beachten gilt, dass die Abbildung in Listing 9 nicht vollständig ist, deshalb ist ein „Non-GreedyQuantor nach der *ID* notwendig. Anschließend werden Name, URL und Datum mittels Pattern beschrieben.

```
import regex as re

repo_pattern = re.compile(
"""
    id
    \W+                  # tag and value seperated by none 'norm'
   characters
        (?P<id>
            \d+?         # capture id consisting of one or more numbers
        )
    \,.*?name\W+
        (?P<name>
            [\w\-\s]+?  # capture name consisting of [a-zA-Z_- and \s]
        )\'             # non-greedy ends with apostrophe
    .*?ssh_url_to_repo\W+
        (?P<url>
            .*?                 # capture url non-greedy
        )
    \'.*?last_activity_at\W+    # group capture ends with apostrophe
        (?P<date>
            \d{4}-\d{2}-\d{2}   # capture date format YYYY-MM-DD
        )
""", flags=re.MULITILINE | re.DOTALL | re.VERBOSE)
```

Listing 11: Regex-Ausdruck für GitLab-Repository Informationen

Um multiple Repository-Informationen auf diese Weiße zu parsen, ist es sinnvoll, das Pattern zu kompilieren, da dadurch eine deutliche Beschleunigung erzielt werden kann. Diese Methode des Parsen von Strings ist herkömmlichen Parsen, welche einen weit größeren Overhead besitzen, deutlich überlegen. Ein einfaches Beispiel hierfür bietet der Python-JSON-Parser, er benötigt für 2500 der in Listing 9 beschriebenen JSON-Elemente durch-

30. Ein Look-Behind ist eine Unterabfrage, die dem Pattern vorangestellt ist. Ein Beispiel hierfür wäre '(?<=(John|Jonathan)\s)McLane)' (López and Romero 2014: 72)

schnittlich 12 ms. Das kompilierte Regex-Pattern hingegen im Schnitt lediglich 2 ms. Dies entspricht einer 6-fachen Beschleunigung und kann nicht vernachlässigt werden.

3.4 Zusammenführung

Mit dem gesammelten Wissen ist es möglich Informationen über zugängliche Repositories eines GitLab-Servers mittels der REST-API abzugreifen. In diesem Projekt für automatische Reportlösung sind lediglich Repositories von Interesse, falls diese in den letzten 30 editiert wurden. Sind die Projekte gefiltert, kann mittels Projekt-ID, welche auf einem GitLab-Server als UID zu betrachten ist, sowie eines weiteren API-Aufrufs auf die Commit-Nachrichten der Repositories zugegriffen werden.

```
1 def __get_commit_messages(self, project_id, since):
2     query: str = urlencode({'all': 'True'})
3     path: str = f'api/v4/projects/{project_id}/repository/commits'
4     url: str = urlunparse((protocol, netloc, path, '', query, ''))
5     header = {'PRIVATE-TOKEN': f'{p_token}'}
6     response = requests.get(url, headers=header)
7     return response.json()
```

Listing 12: Python-Code Abfrage von Commit-Nachrichten

Die Query "*all=True*" ermöglicht es, alle Nachrichten seit Erstellung des Repositories abzugreifen. Mit Python Version 3.6 und der Einführung sogenannter f-String wie in Listing 12 in Zeile 3 abgebildet, kann die ID des Repositories mittels geschweiften Klammern und dem Variablennamen in einen String eingebaut werden. Eine Response mit Beispieldaten dieser Anfrage ist in Listing 13 abgebildet.


```
1  {
2   "id":"b9cc1f95116343c15453d4adcc8d6f76b0bcb866",
3   "short_id":"b9cc1f95",
4   "created_at":"2013-02-21T16:41:22.000+01:00",
5   "parent_ids":[
6
7   ],
8   "title":"Update",
9   "message":"Nginx V. 1.21.6",
10  "author_name":"Renovate",
11  "author_email":"Renovate@revovate.com",
12  "authored_date":"2013-02-21T16:41:22.000+01:00",
13  "committer_name":"Renovate",
14  "committer_email":"Renovate@renovate.com",
15  "committed_date":"2013-02-21T16:41:22.000+01:00",
16  "trailers":{
17
18  },
19  "web_url":"http://gitlab.com/example"
20 }
```


Listing 13: REST-API Commit-Nachrichten Response eines GitLab-Server mit Beispieldaten

Dieser JSON-String kann wie in Listing 12 mit dem JSON-Parser in ein Python-Objekt gewandelt werden. Jedoch verursacht eine oft hohe Anzahl an Commit-Nachrichten auch hier Leistungseinbußen. Aus diesem Grund bietet es sich erneut an, Regex zu verwenden, um festzustellen, ob Renovate den Commit getätigt hat und wenn die enthaltene Commit-Nachricht als **Report-Nachricht** zu werten.

```
1 "message\W+"(?P<c_message>.+?)".*?"author_name\W+"Renovate"
```

Listing 14: Regex-Pattern um Renovate Commit-Nachrichten zu finden

Ist der Name des Autors wie in Listing 13 “Renovate” ist davon auszugehen, dass es sich bei dem Commit um ein Software-Update handelt. Mit dem in Listing 14 abgebildeten Regex-Pattern kann die Commit-Nachricht und damit die Software sowie die aufgespielte Version gruppiert werden.
Leider konnte in dem Umfang des Projekts keine Realisierung bezüglich Versionen und Release-Notes stattfinden. Für die Implementierung einer solchen Weiterführung wäre es notwendig zu bestimmen, welche Software von Renovate upgedatet werden kann und welche Informationen signifikant für die Reportlösung sind.

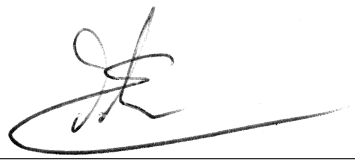
4 Rekapitulation und Ausblick

In dieser Abhandlung wurde die Möglichkeit untersucht, automatische Reports für projektbezogene, automatisch durch einen Bot aktualisierte Software und Dockerimages zu generieren. Um etwaigen Nutzen hervorzuheben, wurde zunächst geprüft, ob Adaptionen in der Komposition des technischen Bestrebens und des täglichen Lebens dokumentierbar sind. Ein kurzer Exkurs in die Geschichte der Softwareentwicklung konnte neben deren Entstehungsgeschichte auch verschiedenste Arten aufzeigen, welche im Laufe der Zeit entwickelt wurden, um dieser zu veräußern. Ob philosophischer Aspekt oder der des Machtmissbrauchs, die Open-Source Bewegung hat viele Verfechter, dennoch konnte dargelegt werden, dass die Argumente der Advokaten prädominieren. Durch den unermüdlichen Fluss an Updates und Patches, um Sicherheitslücken zu schließen und Features zu komplettieren, kann Software keinen Zustand der Persistenz erreichen. Weiterführend konnten die Vorteile von Version-Control-Systems und deren Einsatzgebiete aufgezeigt werden. Mithilfe verschiedener Optionen zur Realisation von Virtualisierungen konnte dargelegt werden, warum die Sicherheit insbesondere bei Servern mit einer sehr hohen Priorität behandelt wird. Um die automatische Reportlösung zu implementieren, wurde die Programmiersprache Python gewählt, da diese einen merklichen Abstraktionsgrad sowie eine Featurevielfalt für Webabfragen, Prozesserstellung und Beschleunigung bereitstellt. Für das Projekt mussten Commit-Nachrichten der letzten 30 Tage, erstellt von Renovate, einem Bot, welcher Softwareabhängigkeiten innerhalb Repositories entdeckt und automatisch updatet, erkannt werden. Nachdem ein ressourcenlastiger Versuch alle Repositories zu klonen, um die Git-Datensätze nach etwaigen Commit-Nachrichten durch eine zu hohe Netzwerkbelastung fehlschlug, wurde die GitLab-REST-API weiterführend verwendet, um auf Informationen und Datensätze der Repositories zuzugreifen. Unter Zuhilfenahme von Regex konnten die übermittelten Daten aufgegliedert und zur Reportlösung verarbeitet werden. Durch diese Lösung ist es nun möglich, Kunden und Teammitgliedern ein detailliertes Bild über veränderte Software von Drittanbietern zu vermitteln. Durch die zeitliche Begrenzung des Projekts konnte jedoch kein wirtschaftlicher Vorteil gemessen werden.
Für zukünftige Implementierungen wäre es ebenfalls von Interesse, detaillierte Informationen über die verschiedenen Softwarereleases in den Report mit aufzunehmen. Für dieses Vorhaben wäre es jedoch wie bereits abschließend in Abschnitt Zusammenführung erwähnt, erforderlich, weitere Auskünfte über verwendete Software zu besitzen sowie die Möglichkeit, den Kontext der oftmals sehr umfangreichen Release-Notes zusammenzufassen. Der erste Punkt könnte beispielsweise mithilfe eines Parsers, welcher Repositories nach Softwareabhängigkeiten durchsucht, ermöglicht werden. Eine ressourcenfreundlichere Option bietet das Editieren des Open-Source-Codes von Renovate. Dieser könnte so verändert werden, dass der Name der veränderten Software aus der Commit-Nachricht ersichtlich ist. Der zweite Punkt stellt eine größere Hürde dar und kann vermutlich nur durch den Einsatz von KI und äußert komplexen Implementierungen realisiert werden. Aus diesem Grund wird wohl auch in Zukunft die manuelle Einsicht der Release-Notes ein wichtiger Bestandteil für die Informationsfindung bleiben.

5 Erstellungserklärung

Erklärung zur Abschlussarbeit

Hiermit versichere ich, die eingereichte Abschlussarbeit selbständig verfasst und keine andere als die von mir angegebenen Quellen und Hilfsmittel benutzt zu haben. Wörtlich oder inhaltlich verwendete Quellen wurden entsprechend den anerkannten Regeln wissenschaftlichen Arbeitens zitiert. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht anderweitig als Abschlussarbeit eingereicht wurde.
Das Merkblatt zum Täuschungsverbot im Prüfungsverfahren der Hochschule Augsburg habe ich gelesen und zur Kenntnis genommen. Ich versichere, dass die von mir abgegebene Arbeit keinerlei Plagiate, Texte oder Bilder umfasst, die durch von mir beauftragte Dritte erstellt wurden.



Augsburg, 21.03.2022

Ort, Datum

Unterschrift des/der Studierenden

Literatur

Adobe Systems Software Ireland Limited. Adobe-newsroom, 2017. https://www.adobe-newsroom.de/2017/12/04/ [Accessed:22.02.2022]. 2.1

Carl Albing and JP Vossen. *Bash Cookbook*. O'Reilly Media, Inc, 2 edition, 2017. ISBN 9781491975336. 2.5

Mansoor Alicherry and Angelos D. Keromytis. Doublecheck: Multi-path verification against man-in-the-middle attacks. *2009 IEEE Symposium on Computers and Communications*, pages 557–563, 7 2009. doi: 10.1109/ISCC.2009.5202224. 3.2.1

Mike Amundsen, Sam Ruby, and Leonard Richardson. *RESTful Web APIs*. O'Reilly Media, Inc., 2013. ISBN 9781449358068. 3.2.2

Anaconda. About numba, 2018. https://numba.pydata.org/ [Accessed:08.03.2022]. 3.1, 3.1

Apple Inc. Xcode release notes 5.11968433, 2013. https://developer.apple.com/library/archive/releasenotes/DeveloperTools/RN-Xcode/Chapters/Introduction.html [Accessed:09.02.2022]. 2.3

BBC News. Hp laptops found to have hidden keylogger, 2017. https://www.bbc.com/news/technology-42309371 [Access:02.03.2022]. 2.2

Berufsverband der Augenärzte. Alterssichtigkeit (presbyopie), 11 2012. http://cms.augeninfo.de/hauptmenu/augenheilkunde/fehlsichtigkeit/alterssichtigkeit-presbyopie.html [Accessed: 05/11/2021]. 1

Bioökonomierat. Bioökonomie, 4 2012. http://www.biooekonomierat.de/biooekonomie/ [Accessed:16.02.2022]. 1

Ferdinand Braun. Ueber ein verfahren zur demonstration und zum studium des zeitlichen verlaufes variabler ströme. *Annalen der Physik und Chemie*, 296: 552–559, 1897. ISSN 00033804. doi: 10.1002/andp.18972960313. 6

Zach Brown. Git origin story, 7 2018. https://www.linuxjournal.com/content/git-origin-story [Accessed:10.02.2022]. 2.3

Bundesamt für Sicherheit in der Informationstechnik. Kritische schwachstelle in java-bibliothek log4j, 2022. https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Empfehlungen-nach-Angriffszielen/Webanwendungen/log4j/log4j_node.html [Accessed:03.03.2022]. 2.2

Mark Burgess. *Management Technologies for E-Commerce and E-Business Applications*, volume 2506. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-00080-8. doi: 10.1007/3-540-36110-3. 1

Robert G. Byrnes, Richard E. Silverman, and Daniel J. Barrett. *SSH, the Secure Shell The Definitive Guide*. O'Reilly Media, 2005. ISBN 9780596008956. (document), 3.2.1, 2

Scott Chacon and Ben Straub. Pro git. *Pro Git*, 2014. doi: 10.1007/978-1-4842-0076-6. https://git-scm.com/book/en/v2 [Accessed:09.02.2022]. (document), 2.3, 3

Chao-Kuei\Free-Software-Foundation. Categories of free software, 11 2021. https://www.gnu.org/philosophy/categories.html#ProprietarySoftware [Accessed:22.02.2022]. (document), 1

J. Bradley Chen, Yasuhiro Endo, Kee Chan, David Mazières, Antonio Dias, Margo Seltzer, and Michael D. Smith. The measured performance of personal computer operating systems. *ACM Transactions on Computer Systems*, 14:3–40, 2 1996. ISSN 0734-2071. doi: 10.1145/225535.225536. https://www.scs.stanford.edu/~dm/home/papers/chen:p5-sosp.pdf [Accessed:]. 1

DataDog. Docker adoption, 6 2018. https://www.datadoghq.com/docker-adoption/ [Accessed:19.03.2022]. (document), 5

Docker Inc. Docker security, 2022. https://docs.docker.com/engine/security/ [Accessed:19.03.2022]. 2.5

S. Donovan. Patent, copyright and trade secret protection for software. *IEEE Potentials*, 13:20–24, 8 1994. ISSN 0278-6648. doi: 10.1109/45.310923. 2.1

Paul M. Duvall, Steve Matyas, and Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Professional, 2007. ISBN 978-0-321-33638-5. 3, 3

EDRi. Microsoft's new small print – how your personal data is (ab)used, 2015. https://edri.org/our-work/microsofts-new-small-print-how-your-personal-data-abused/ [Access:02.03.2022]. 2.2

F5 Network Inc. Nginx history, 2022. https://www.nginx.com/careers/#history-nginx [Accessed:19.03.2022]. 22

Aricka Flowers. Why we use ruby on rails to build gitlab, 2018. https://about.gitlab.com/blog/2018/10/29/why-we-use-rails-to-build-gitlab/ [Accessed:06.03.2022]. 2.4

Free Software Foundation. Proprietary surveillance, 2022. https://www.gnu.org/proprietary/proprietary-surveillance.html#SpywareInWindows [Accessed:02.03.2022]. 2.2

Free Software Foundation Inc. Gnu rcs, 11 2012. https://www.gnu.org/software/rcs/ [Accessed:09.02.2022]. 2.3

Sigmund Freud. *Hysterie und Angst by Sigmund Freud*, volume 6. Fischer Taschenbuch Verlag GmbH, 3 u. erg. edition, 1 1976. ISBN 3-596–27306-4. 1

GitLab B.V. Go guide, 2022. https://docs.gitlab.com/ee/development/go_guide/ [Accessed:06.03.2022]. 2.4

David Gourley, Brian Totty, Marjorie Sayer, Anshu Aggarwal, and Sailu Reddy. *HTTP: The Definitve Guide*. O'Reilly Media, Inc., 2002. ISBN 9781565925090. 3.2.2

J.M. Graetz. The origin of spacewar. *Creative Computing*, 7:56–67, 1981. https://archive.org/details/creativecomputing-1981-08/page/n70/mode/1up?view=theater [Accessed: 23.02.2022]. 2.1

Voker Grasshof. *Freie Software. Zwischen Privat- und Gemeineigentum*, volume 2. Bundeszentrale für politische Bildung, 11 2004. ISBN 978-3-89331-569-7. https://freie-software.bpb.de/Grassmuck.pdf [Accessed:07.02.2022]. 1

Shane Greenstein. The range of linus' law. *IEEE Micro*, 32:72–72, 1 2012. ISSN 0272-1732. doi: 10.1109/MM.2012.10. 2.2

Jilll E. Grogg and Jeff Weddle. *Encyclopedia of Library and Information Sciences*, volume 1. CRC Press, 3 edition, 12 2009. ISBN 9780203757635. doi: 10.1081/E-ELIS3. 2.3

Dick Grune. Concurrent version system, a method for independent cooperation, 1986. https://dickgrune.com/Books/Publications/Concurrent_Versions_System,_a_method_for_independent_cooperation.pdf [Accessed:09.02.2022]. 2.3

Dick Grune. Concurrent version system cvs, 11 2012. https://dickgrune.com/Programs/CVS.orig/ [Accessed:09.02.2022]. 2.3, 14

Rob Hassett. Impact of apple vs. franklin decision, 12 2012. http://internetlegal.com/impact-of-apple-vs-franklin-decision/ [Accessed:24.02.2022]. 2.1

Herder. Herders konversations-lexikon, 1857. https://www.deutschestextarchiv.de/book/view/nn_conversationslexikon05_1857?p=1 [Accessed:25.11.2022]. 5

HSH-Nordbank. Gesundheitswirtschaft und prävention - ein jahr jünger bringt 10 milliarden euro, 8 2017. https://www.hcob-bank.de/media/pdf_3/marktberichte/branchenstudien/gesundheitswirtschaft/201708_standpunkt_hshnordbank_praevention.pdf [Accessed:07.02.2022]. 1

Tatum Hunter and Gerrit de Vynck. The 'most serious' security breach ever is unfolding right now. here's what you need to know., 2021. https://www.washingtonpost.com/technology/2021/12/20/log4j-hack-vulnerability-java/ [Accessed:03.03.2022]. 2.2

Solomon Hykes. Au revoir, 3 2018. https://www.docker.com/blog/au-revoir/ [Accessed:17.03.2022]. 2.5

IBM Corporation. Compiled versus interpreted languages, 2010. https://www.ibm.com/docs/en/zos-basic-skills?topic=zos-compiled-versus-interpreted-languages [Accessed:23.02.2022]. 7, 8

Konstatin Ivanov. *Containerization with LXC*. Packt, 2017. ISBN 9781785888946. 2.5, 2.5, 2.5, 2.5

Tom Kilburn. From cathode ray tube to ferranti mark i. *Resurrection: The Bulletin of the Computer Conservation Society*, 1:16–20, 1990. ISSN 0958-7403. https://www.computerconservationsociety.org/resurrection/pdfs/res02.pdf [Accessed: 23.02.2022]. 2.1

Gernot Klinger. How to use a chroot-jail for software development, 2014. https://gernotklingler.com/blog/use-chroot-jail-software-development/ [Accessed:18.03.2022]. 2.5

Ali Koc and Tansel Abdullah. A survey of version control systems, 2011. https://www.iiis.org/cds2011/cd2011imc/iceme_2011/paperspdf/fb394vz.pdf [Accessed:09.02.2022]. 2.3

Liza Lin and David Uberti. Alibaba employee first spotted log4j software flaw but now the company is in hot water with beijing, 2021. https://www.wsj.com/articles/china-halts-alibaba-cybersecurity-cooperation-for-slow-reporting-of-threat-state-media-says-11640184511 [Accessed:03.03.2022]. 2.2

Warner Losh. Whither chroot?, 6 2020. https://bsdimp.blogspot.com/2020/06/whither-chroot.html [Accessed:18.03.2022]. 2.5

Christof Lutteroth and Gerald Weber. Efficient use of guids. *2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 115–120, 2008. doi: 10.1109/PDCAT.2008.67. 16

Félix López and Victor Romero. *Mastering Python Regular Expressions*. Packt, 2014. ISBN 978-1-78328-315-6. (document), 3.3, 8, 3.3, 9, 10, 3.3, 30

Stephen Manes and Paul Andrews. *Gates: How Microsoft's Mogul Reinvented an Industry – and Made Himself the Richest Man in America*. Cadwallader & Stern, 4 2013. ISBN 0671880748. 2.1

Dave Mckay. Is macosx unix and what does that mean, 2019. https://www.howtogeek.com/441599/is-macos-unix-and-what-does-that-mean/ [Accessed:02.03.2022]. 9

M. Douglas Mcllroy. A research unix reader: Annotated excerpts from the programmer's manual, 1971-1986, 6 1987. https://www.cs.dartmouth.edu/~doug/reader.pdf [Accessed:09.02.2022]. 13

Larry McVoy. Smoosh - a tool for merging related sccss files, 10 1991. http://www.mcvoy.com/lm/bitmover/lm/papers/smoosh.pdf [Accessed:10.02.2022]. 2.3

Barton P. Miller, Louis Fredriksen, and Bryan So. An empirical study of the reliability of unix utilities. *Communications of the ACM*, 33:32–44, 12 1990. ISSN 0001-0782. doi: 10.1145/96267.96279. 21

Gordon E. Moore. Cramming more components onto integrated circuits. *IEEE*, 38: 30–35, 1965. doi: 10.1109/N-SSC.2006.4785860. URL https://newsroom.intel.com/wp-content/uploads/sites/11/2018/05/moores-law-electronics.pdf. 2.2

Werner Nachtigall. *Bionik*. Springer Berlin Heidelberg, 1 edition, 2002. ISBN 978-3-642-62399-8. doi: 10.1007/978-3-642-18996-8. http://link.springer.com/10.1007/978-3-642-18996-8 [Accessed:19.03.2022]. 2

Nvidia. How gitlab geo supports nvidia's innovation, 2022. https://about.gitlab.com/customers/Nvidia/ [Accessed:06.03.2022]. 2.4

P. Oehlert. Violating assumptions with fuzzing. *IEEE Security and Privacy Magazine*, 3: 58–62, 3 2005. ISSN 1540-7993. doi: 10.1109/MSP.2005.55. 21

National Museum of American History. At the heart of the invention: The development of the holter monitor, 11 2011. https://americanhistory.si.edu/blog/2011/11/at-the-heart-of-the-invention-the-development-of-the-holter-monitor-1.html [Accessed:04.02.2022]. 1

OpenSSH. Ssh config man5, 2019. http://manpages.ubuntu.com/manpages/bionic/en/man5/ssh_config.5.html [Accessed:10.03.2022]. 3.2.1

Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. Energy efficiency across programming languages: how do energy, time, and memory relate? *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*, pages 256–267, 10 2017. doi: 10.1145/3136014.3136031. 3.1

Christine Peterson. How i coined the term 'open source', 2018. https://opensource.com/article/18/2/coining-term-open-source-software [Access:02.03.2022]. 2.2

C. Michael Pilato, Ben Collins-Sussman, and Brian W. Fitzpatrick. *Version Control with Subversion*. O'Reilly Media, Inc, 2 edition, 9 2008. ISBN 9780596510336. 2.3

Matthew Portnoy. *Virtualization Essentials*. Sybex, 2 edition, 2016. ISBN 978-1-119-26772-0. 2.5

Eric S. Raymond. *The Cathedral and the Bazaar Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly and Associates, 1 edition, 2001. ISBN 978-0596001087. 2.2

Brandon Rhodes and John Goerzen. *Foundations of Python Network Programming*. Apress, 2010. ISBN 978-1-4302-3003-8. doi: 10.1007/978-1-4302-3004-5. 3.2.2, 3.2.2

D.M. Ritchie and K. Thompson. Unix-belllabs. *The Bell System Technical Journal*, 57: 1905–1929, 1978. URL https://ia802709.us.archive.org/21/items/bstj57-6-1905/bstj57-6-1905.pdf. 9

Marc J. Rochkind. The source code control system. *IEEE Transactions on Software Engineering*, SE-1:364–370, 12 1975. ISSN 0098-5589. doi: 10.1109/TSE.1975.6312866. (document), 2.2, 2, 2.3

Nayan B. Ruparelia. The history of version control. *ACM SIGSOFT Software Engineering Notes*, 35:5–9, 1 2010. ISSN 0163-5948. doi: 10.1145/1668862.1668876. 2.3

Seva Safris. A deep look at json vs. xml, part 1: The history of each standard, 2019. https://www.toptal.com/web/json-vs-xml-part-1 [Accessed:15.03.2022]. 3.2.2

Samba Team. Was ist samba?, 2 2022. https://www.samba.org/samba/what_is_samba.html [Accessed:10.02.2022]. 17

M. O. Faruque Sarker and Sam Washington. *Learning Python Network Programming*. Packt, 2015. ISBN 9781784396008. 3.2.2

Siemens. How siemens created an open source devops cutlutre with gitlab, 2022. https://about.gitlab.com/customers/siemens/ [Accessed:06.03.2022]. 2.4

Lee Sihyung, Levanti Kyriaki, and S. Kim Hyong. Network monitoring: Present and future. *Computer Networks*, 65:84–98, 6 2014. ISSN 1389-1286. doi: 10.1016/J.COMNET.2014.03.007. 1

Kurt Smith. *Cython*. Published by O'Reilly Media, Inc., 1 edition, 2015. ISBN 978-1-491-90155-7. (document), 6, 3.1

Larry Smith. Smoosh - a tool for merging related sccs s-files, 10 1991. http://www.mcvoy.com/lm/bitmover/lm/papers/smoosh.pdf [Accessed:19.03.2022]. 2.3

WhiteSource Software. Renovate documentaion, 2022a. https://docs.renovatebot.com/ [Accessed:19.03.2022]. 3

WhiteSource Software. Whitesource about, 2022b. https://www.whitesourcesoftware.com/about-us/ [Accessed:19.03.2022]. 3

SSH.com. Man in the middle, 2022a. https://www.ssh.com/academy/attack/man-in-the-middle [Accessed:10.03.2022]. 27

SSH.com. Ssh-keygen, 2022b. https://www.ssh.com/academy/ssh/keygen [Accessed:10.03.2022]. 3.2.1

Richard M. Stallman. *Free Software, Free Society Selected Essays of Richard M. Stallman by Richard M. Stallman*. Free Software Foundation Inc., 2015. ISBN 978-0983159254. 2.2

Statistischen Bundesamt. Todesursachenstatistik 2020, 11 2021. https://www.destatis.de/DE/Presse/Pressemitteilungen/2021/11/PD21_505_23211.html [Accessed:25.11.2022]. 1

Oliver Strimpel. The catalog of personal computers. *The Computer Museum report*, 17, 1986. http://ed-thelen.org/comp-hist/TCMR-V17.pdf [Accessed:23.02.2022]. 2.1

Synopsys Inc. Compare repositories, 2022. https://www.openhub.net/repositories/compare [Accessed:06.03.2022]. (document), 4

Agilent Technologies. Evaluating high availability mechanisms, 2005. http://literature.cdn.keysight.com/litweb/pdf/5989-4388EN.pdf [Accessed:07.02.2022]. 1

Janne Teller. *Intet*. Dansklærerforeningens, 1 edition, 12 2000. ISBN 9788777046254. http://xn--mns-ula.dk/sky/apps/files_sharing/get.php?token=718d96e1066b3653105895af71d2c5f532fe42a7&path=/Intet.pdf [Accessed:03.11.2022]. 1

Janne Teller. *Nothing*. Atheneum Books for Young Readers, 1 edition, 2010. ISBN 9781416985792. 1

The Go Projekt. Go fuzzing, 2022. https://go.dev/project [Accessed:16.03.2022]. 21

Madeleine Thivierge and Luc Léger. Critical review of heart rate monitors. *ACSEPL*, 55: 16–31, 5 1989. https://www.researchgate.net/figure/Heart-rate-monitors-and-their-equivalent-models-MANUFACTURER-Sport-Tester-PE-2000-et-PE_tbl1_293555770 [Accessed:04.02.2022]. 1

Shara Tibken. Ces 2019: Moore's law is dead, sys nvidia's ceo, 1 2019. https://www.cnet.com/tech/computing/moores-law-is-dead-nvidias-ceo-jensen-huang-says-at-ces-2019/ [Accessed:02.03.2022]. 2.2

Walter Tichy. Design, implementation, and evaluation of a revision control system, 3 1982. https://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1322&context=cstech [Accessed:09.02.2022]. 2.3

TIOBE Software BV. Tiobe index, 2022. https://www.tiobe.com/tiobe-index/ [Accessed:08.03.2022]. 3.1

Andrew Tridgell and Paul Mackerras. Fist release of rsync, 1996. 18

A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, pages 230–265, 11 1936. ISSN 00246115. doi: 10.1112/plms/s2-42.1.230. https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf [Accessed:23.02.2022]. 2.1

Umweltbundesamt. Umweltbewusstsein und umweltverhalten, 2022. https://www.umweltbundesamt.de/daten/private-haushalte-konsum/umweltbewusstsein-umweltverhalten#stellenwert-des-umwelt-und-klimaschutzes [Accessed:09.03.2022]. 3.1

John R Vacca. *Computer and Information Security Handbook*. Morgan Kaufmann, 2 edition, 7 2009. ISBN 978-0-12-374354-1. 1

Guido van Rossum. A brief timeline of python, 2009. https://python-history.blogspot.com/2009/01/brief-timeline-of-python.html [Accessed:02.03.2022]. 10

Wolters Kluwer Deutschland GmbH. Bgh beschl. 03.12.1987 az.: Viii zr 314/86, 2022. https://research.wolterskluwer-online.de/document/74f346f5-9401-4ad2-a43c-12ae97def7b1 [Accessed:23.02.2022]. 2.1