Performanceunterschiede zwischen portablen und nativen Programmiersprachen

Projektbericht

vorgelegt am 10.03.2014

an der
Hochschule für Wirtschaft und Recht Berlin
Fachbereich Duales Studium

von	Sebastian Herrmann
Bereich:	Wirtschaft
Fachrichtung:	Wirtschaftsinformatik
Studienjahrgang:	2012/2013
Studienhalbjahr:	3
Ausbildungsbetrieb:	DB Systel GmbH
Betreuender Prüfer:	Prof. Dr. Faustmann

Inhaltsverzeichnis

1 Einleitung

Wie entwickeln wir Software? Wie entscheiden wir darüber, wie wir die eigenen Vorstellungen oder die Wünsche unserer Kunden realisieren? Welche Paradigmen und Programmiersprachen wählen wir dafür? Für Hobbyentwickler ist die Frage oftmals leicht zu beantworten, denn diese wählen schlichtweg den Weg des geringsten Widerstands und somit die Programmiersprache, die sie bereits beherrschen - die Auswahl ist von vornherein eingeschränkt und überschaubar. Unternehmen befinden sich jedoch oftmals in der luxuriösen Position, über viel Expertise zu verfügen und sich fehlendes Know-how im Notfall aneignen oder einkaufen zu können - hier zählt auch eher das Endprodukt als dessen Entstehung. Doch wonach richten sich die Entwicklerfirmen, wenn die Auswahl an verfügbaren Programmiersprachen groß und unüberschaubar ist[1]? Intelligente Unternehmer und Projektmanager werden nicht einfach drauf los programmieren lassen, sondern im Idealfall die Programmiersprache - ebenso wie die anderen Parameter der Entwicklung - genau planen und abwägen. Denn die Wechselkosten können im Falle einer Neuausrichtung enorm hoch sein.[2]

Die Wahl der Programmiersprache und entsprechenden Plattform wirkt sich oft auf wichtige Rahmenparameter wie Performance, Wartbarkeit und Unterstützungsmöglichkeiten von anderen Entwicklern und Herstellern aus. Um die Entscheidung fundiert treffen zu können, müssen die Verantwortlichen in der Lage sein, die vorhandenen Möglichkeiten zu analysieren und zu beurteilen. Dieser Bericht soll bei diesem Prozess unterstützen und konzentriert sich auf die Performanceunterschiede zwischen modernen plattformunabhängigen und nativen Programmiersprachen.

1.1 Portabilität und Nativität

Im IT-Umfeld ist Portabilität gleichzusetzen mit dem Begriff Plattformunabhängigkeit. Ein Programm gilt als portabel, wenn es mit möglichst geringem Aufwand auf verschiedenen Plattformen (z.B. Microsoft Windows und Linux) lauffähig ist.[3] Die Realisierung dieses Konzepts erfolgt in der Regel über Zwischenschichten zwischen dem Betriebssystem und der auszuführenden Software, zum Beispiel in Form einer virtuellen Maschine, welche den portablen Quelltext in die jeweiligen Betriebssystembefehle übersetzt.

---

[1] Mashey, John R. (2004), S. 34 (siehe Internetverzeichnis).
[2] Welton, David N. (2005), Abs. „Switching Costs“ (siehe Internetverzeichnis).
[3] Boehm, Barry W. u.a. (1976), S. 595; Cowell, Wayne (Hrsg.) (1977), S. 1 f.

Die Bezeichnung von Software als nativ entstand im Zuge von mobilen Anwendungen für Smartphones, die sich entweder per interaktive HTML-Seiten (sog. Webapps) oder via Entwicklung in der jeweilig unterstützten Programmiersprache (z.B. Java für Android- und Objective-C für iOS-Geräte) realisieren lassen. Letztere Vorgehensweise bringt dabei native Apps hervor, welche mehr Zugriffsmöglichkeiten auf Systemfunktionen bieten - zum Beispiel die Ansteuerung von Lagesensoren oder der Kamera.[4]

Auf der anderen Seite kann Nativität auch als Gegenteil von Portabilität angesehen werden: Software wird dabei möglichst direkt bzw. nah am Betriebssystem ausgeführt, um Performanceverluste zu vermeiden und Systemzugriffe (z.B. auf Hardwarefunktionen) zu erlauben. Dieses Verständnis von Nativität lässt sich ohne weiteres auf Desktop-Software erweitern[5] - allgemein etabliert hat sich diese Bezeichnung momentan noch nicht.

1.2 Performance und Effizienz

Bei Software wird Performance oft synonym zum Begriff Effizienz verwendet und - ebenso wie Portabilität - als wichtiger Bestandteil der Softwarequalität definiert.[6] Viele Definitionen lassen die Schlussfolgerung zu, dass Effizienz durch die Ausführungsgeschwindigkeit und damit einhergehende Laufzeit im Prozessor bestimmt wird.[7] Je weniger Prozessorzeit die Software also für die erfolgreiche Abarbeitung einer Kommandokette benötigt, desto effizienter arbeitet sie. Dabei sollte auch der Ressourcenverbrauch (meist bestimmt durch die Menge an benötigtem Arbeitsspeicher) möglichst gering ausfallen.

In der Natur der Programmierung liegt, dass sich Aufgaben auf verschiedene Weisen lösen lassen, je nach Verfügbarkeit von Werkzeugen und Mitteln der Programmiersprache. Von daher wird besonderes Augenmerk auf exklusive Softwareeigenschaften und -fähigkeiten gelegt, welche die Performance signifikant verbessern oder schmälern.

2 Auswahl beispielhafter Programmiersprachen

Moderne Programmiersprachen weisen viele Merkmale auf, nach denen sie sich kategorisieren lassen - beispielsweise Paradigmen (z.B. prozedural und objektorientiert), Anwendungsgebiete (z.B. für Web oder für Betriebssysteme) und Abstraktionsgrade (low-

[4] Rouse, Margaret (2013), Abs. 1 im Hauptframe (siehe Internetverzeichnis); Würstl, Daniel (2013), Abs. 1 im Hauptframe (siehe Internetverzeichnis).
[5] Google Inc. (2014a), Abs. 1 im Hauptframe (siehe Internetverzeichnis).
[6] Hoare, Charles Antony Richard (1972), S. 104; Fleming, Ian (2014), Abs. „Efficiency“ und „Portability“ (siehe Internetverzeichnis); Boehm, Barry W. u.a. (1976), S. 595.
[7] Burback, Ronald LeRoi (1998), Abs. 1 im Hauptframe (siehe Internetverzeichnis).

level und high-level).[8] In diesem Bericht werden hauptsächlich die beiden Gruppen der portablen und nativen Programmiersprachen betrachtet.

Um die Performance von Programmiersprachen konkret analysieren und vergleichen zu können, werden Java und C++ als wichtige Vertreter dieser beiden Gruppen herangezogen. Gemein ist diesen Sprachen, dass sie objektorientiert arbeiten, auf einer Vielzahl moderner Geräte (sowohl Desktop Computer als auch mobile Endgeräte) vertreten sind und daher eine hohe Relevanz für moderne Programmierung bergen.

Tabelle 1: Betrachtete Programmiersprachen, Eigene Darstellung

Sprache	**Entwickler**	**Entstehung**	**Einordnung**
Java	Sun, mittlerweile Oracle	1995	objektorientiert, portabel
C++	Bjarne Stroustrup, C++ Standards Committee	1985	objektorientiert, nativ

Quellen: Oracle Corp. (2014a); The C++ Resources Network (2013)[9]

2.1 Java

Java dient als klassisches Beispiel einer portablen Programmiersprache. „Write once, run anywhere“[10] - Java-Programme sind dank der weitreichend erhältlichen virtuellen Maschine (JVM) auf vielen Systemen von Windows über Linux, OSX bis hin zu embedded Systems (darunter Automobile und Fernseher) lauffähig.[11] Die virtuelle Maschine agiert dabei als native Anwendung und kompiliert den Java-Bytecode in Echtzeit (just in time) in ein nativ ausführbares Format.[12] Aufgrund dieser indirekten und relativ aufwändigen Ausführung des Java-Codes werden oftmals Performanceverluste und ein hoher Speicherbedarf erwartet.[13]

Mit Java vergleichbare Programmiersprachen sind u.a. Scala[14], C#[15] und Visual Basic[16].

2.2 C++

C++ ist eine der beliebtesten Compilersprachen.[17] In C++ geschriebene Programme werden pro Zielplattform einmalig vom Compiler in nativen Code übersetzt und sind danach ohne

[8] Microsoft Corp. (2014a), Abs. „Funktionale Programmierung und imperative Programmierung im Vergleich“ (siehe Internetverzeichnis); Kinnersley, Bill (2014), Abs. 1 im Hauptframe (siehe Internetverzeichnis).
[9] Oracle Corp. (2014a), Abs. 1 im Hauptframe (siehe Internetverzeichnis); The C++ Resources Network (2013), Abs. 1 im Hauptframe (siehe Internetverzeichnis).
[10] Oracle Corp. (2014b), Abs. 1 im Hauptframe (siehe Internetverzeichnis).
[11] Oracle Corp. (2014a), Abs. 1 im Hauptframe (siehe Internetverzeichnis).
[12] Lipinski, Klaus u.a. (2014), Abs. 1 im Hauptframe (siehe Internetverzeichnis).
[13] Mangione, Carmine (1998), Abs. „Performance implications: Java versus native C++ (RPI)“ (siehe Internetverzeichnis).
[14] École Polytechnique Fédérale de Lausanne (2014), Abs. 1 im Hauptframe (siehe Internetverzeichnis).
[15] Microsoft Corp. (2014b), Abs. 1 im Hauptframe (siehe Internetverzeichnis).
[16] Microsoft Corp. (2014c), Abs. 1 im Hauptframe (siehe Internetverzeichnis).

weitere Zwischenschritte ausführbar.[18] Durch diese direkte Ausführung des Maschinencodes und die relativ maschinennahen Möglichkeiten, gegeben durch die enge Verwandtschaft mit C, wird C++ als native Programmiersprache eingeordnet.

Ähnliche Sprachen sind u.a. das bei Apple-Systemen etablierte Objective-C[19], das von Google entwickelte Go (auch golang)[20] und D[21].[22]

3 Analyse der Performance

Für Boehm stehen Softwaremerkmale wie Effizienz und Portabilität im Konflikt - soll eines der Merkmale verbessert werden, müssen die anderen zwangsläufig darunter leiden.[23] Doch ist diese Ansicht für heutige Systeme mit effektiven Just-in-Time-Compilern, günstigen Mehrkernprozessoren und schnellen Frameworks noch aktuell? [24]

Nachfolgend wird analysiert, wie groß die Einbußen beim Portabilität-Performance-Kompromiss sind und welche Methoden bzw. Informationen helfen können, um die negativen Effekte einzudämmen. Dazu wird ein Blick auf Code-Benchmarks und deren Auswertungen geworfen.

3.1 Benchmarks und Ergebnisse

Im Internet finden sich zahlreiche Benchmarks zu unterschiedlichen Programmiersprachen und -techniken. Sie spiegeln das Verhalten der meist speziell dafür erstellten Testprogramme in verschiedenen Situationen und Umgebungen wider und geben Auskunft über die dabei festgestellten Laufzeiten, Speicherbedarfe und Quelltextumfänge. Dabei werden die Einzelergebnisse meist in Relation zum besten Teilergebnis gesetzt (Beispiel: die Java-Prozessorlaufzeit beträgt das 1,2-fache der Prozessorlaufzeit von C).

3.1.1 The Computer Language Benchmarks Game

Eine der populärsten Benchmarkquellen bildet das von Brent Fulgham ins Leben gerufene Computer Language Benchmarks Game, welches 33 verschiedene Programmiersprachen

---

[17] O'Grady, Stephen (2014), Abs. 5 im Hauptframe (siehe Internetverzeichnis); LangPop (2013), Abs. „Normalized Comparison" (siehe Internetverzeichnis).
[18] Schnabel, Patrick (2012), Abs. „Compiler" (siehe Internetverzeichnis).
[19] Apple Inc. (2010), Abs. 1 im Hauptframe (siehe Internetverzeichnis).
[20] Google Inc. (2014b), Abs. 1 im Hauptframe (siehe Internetverzeichnis).
[21] Digital Mars (2014), Abs. 1 im Hauptframe (siehe Internetverzeichnis).
[22] Binstock, Andrew (2012), Abs. 2 im Hauptframe (siehe Internetverzeichnis).
[23] Boehm, Barry W. u.a. (1976), S. 594 (siehe Internetverzeichnis).
[24] Bezanson, Jeff u.a. (2014), Abs. 1 im Hauptframe (siehe Internetverzeichnis).

miteinander vergleicht.[25] Hierbei wird Wert darauf gelegt, dass stets möglichst gleichartige Algorithmen unter gleichen Systemvoraussetzungen ausgeführt werden, um sinnvolle Messergebnisse zu erzielen.[26]

Um das tendenzielle Verhalten der relevanten Programmiersprachen Java und C++ zu ermitteln, wurden die Mittelwerte aus allen Messergebnissen gebildet. Diese zeigen, dass Java 7 im Durschnitt deutlich gegen C++ verliert, was Prozessorzeit und Speicherverbrauch der Programme angeht, aber eine ähnliche Quelltextgröße produziert.

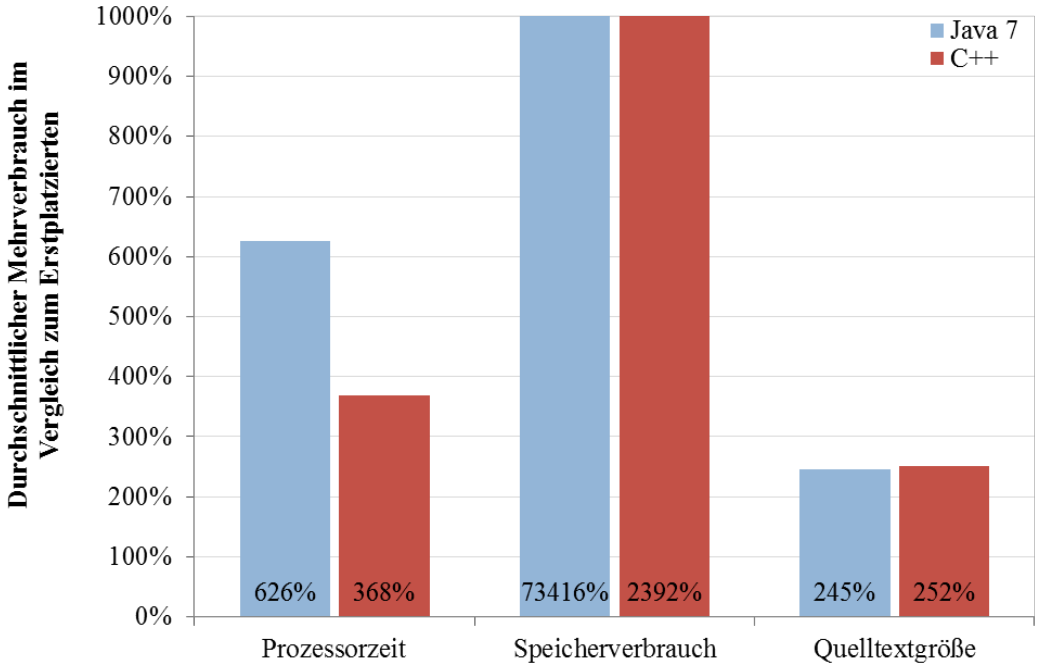


Abbildung 1: Mittelwerte der Messergebnisse für Java und C++

Quelle: Eigene Darstellung nach Fulgham, Brent (2014a)[25]

Der im Vergleich zu den Kontrahenten meist extrem hohe Speicherbedarf von Java scheint die in Kapitel 1.3.1 genannten Vorurteile diesbezüglich zu bestätigen. Dies hängt in den meisten Fällen vermutlich mit der Abhängigkeit von der virtuellen Java-Maschine zusammen. Um den Einfluss der JVM auf die Performance zu ermitteln, führte Fulgham auch hierzu Tests durch (das genaue Verfahren lässt sich auf der Website nachlesen).[27]

[25] Fulgham, Brent (2014a), Abs. 1 im Hauptframe (siehe Internetverzeichnis).
[26] Fulgham, Brent (2014b), Abs. „Why don't you accept every program that gives the correct result?" (siehe Internetverzeichnis).
[27] Ebenda, Abs. „What about Java® VM warm–up?".

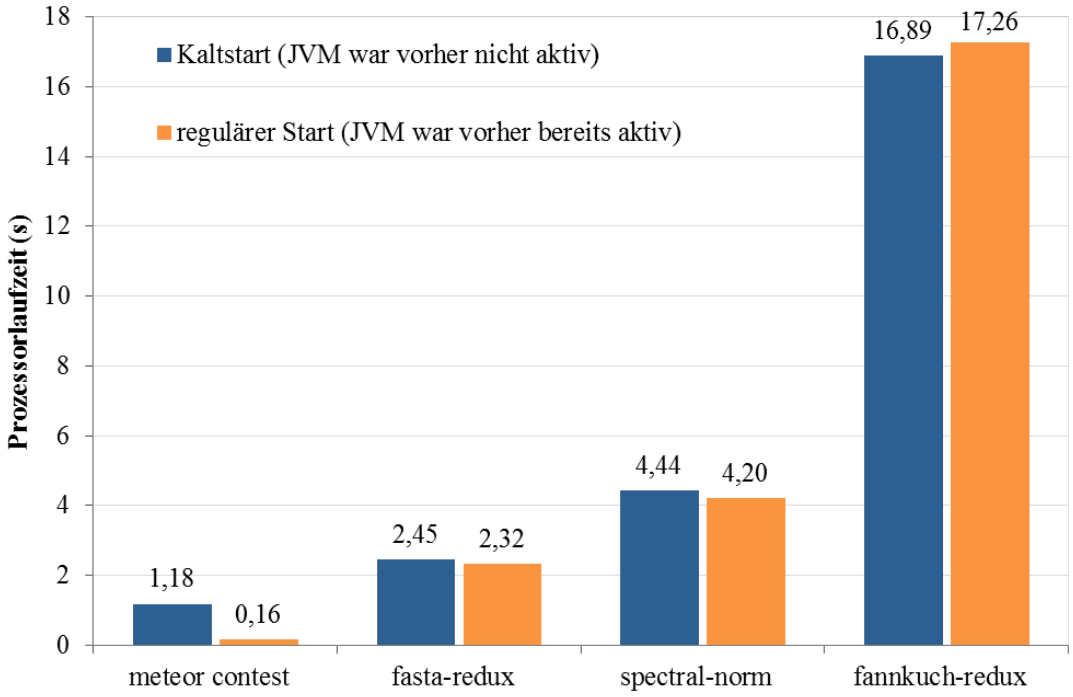


Abbildung 2: Vergleich der Programmlaufzeiten mit verschiedenen JVM-Zuständen und Algorithmen, die Werte von „meteor contest“ werden zur besseren Übersicht mit dem Faktor 100 dargestellt
Quelle: Eigene Darstellung nach Fulgham, Brent (2014b)[28]

Die Resultate zeigen, dass der Zustand der virtuellen Java-Maschine durchaus Auswirkungen auf die Performance der Java-Anwendung hat. Im Durchschnitt liegt die Leistungssteigerung bei 12%, in den Extremfällen bei 86% (meteor contest) Leistungszuwachs und im schlechtesten Fall tritt sogar ein Leistungsverlust von 2% (fannkuch-redux) ein. Im Regelfall sorgt eine „aufgewärmte“, also bereits vorher aktiv gewesene JVM dafür, dass der Code weniger Prozessorzeit beansprucht. Zurückführen lässt sich dies auf die dynamischen Optimierungsverfahren, welche neben der eigentlichen Ausführung des Codes von der virtuellen Maschine durchgeführt werden. Nichtsdestotrotz warnt die relativ große Ergebnisstreuung vor einer Generalisierung der Ergebnisse - der Grad der Auswirkung hängt stark von den verwendeten Strukturen und Algorithmen ab.

3.1.2 Bioinformatics Language Benchmark

Fourment und Gillings der Macquarie University in Sydney widmeten sich einem etwas praxisnäheren Benchmark, bei dem sie typische Algorithmen und Datensätze aus der Bioinformatik verwendeten. Dabei wurden sechs verschiedene Programmiersprachen auf den Plattformen Fedora (Linux) und Windows XP getestet, darunter auch C, C++ und Java.[29]

[28] Ebenda, Abs. „What about Java® VM warm–up?“.
[29] Fourment, Mathieu und Gillings, Michael R (2008b), S. 2 (siehe Internetverzeichnis).

Tabelle 2: Ergebnisse der Bioinformatics-Benchmarks unter Windows XP

	C	C++	Java
Prozessorlaufzeit (s)			
- Global Alignment	0,12	0,28	0,46
- Neighbor-Joining	0,07	0,15	2,23
- BLAST Parser	6,63	8,50	7,76
Speicherverbrauch (kB)			
- Global Alignment	40.892	41.376	52.948
- Neighbor-Joining	1.148	1.680	20.036
- BLAST Parser	-	-	-
Lines of Code			
- Global Alignment	119	76	65
- Neighbor-Joining	240	194	140
- BLAST Parser	82	81	33

Quelle: Fourment, Mathieu und Gillings, Michael R (2008)[30]

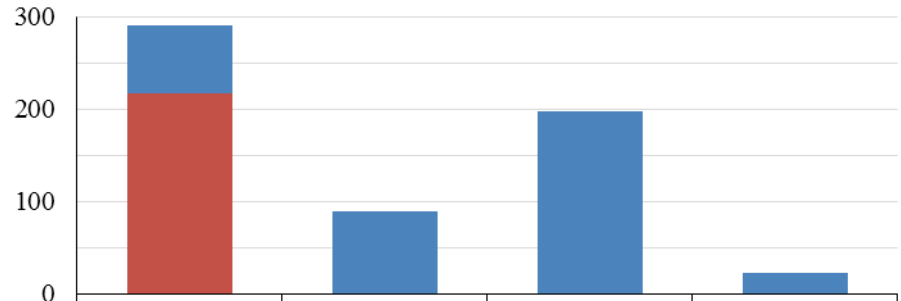
Auch hier wartet Java mit längeren Laufzeiten und größeren Ressourcenverbräuchen auf, wobei die Java-Programme teils deutlich weniger Lines of Code verwenden, was für eine komfortablere Programmierung spricht.

Die Autoren des Benchmarks sehen Javas Nachteil hauptsächlich im hohen Speicherverbrauch. Sie beschreiben die beiden „semi-kompilierten" Sprachen Java und C# als guten Kompromiss zwischen maschinennahen, performanten Sprachen wie C/C++ und einfach zu nutzenden Sprachen wie Perl und Python. Zudem empfehlen sie die Integration von C-Programmteilen in Java mittels Java Native Interface (JNI)-Erweiterungen, um leistungskritische Programmteile zu unterstützen. [31]

3.1.3 Loop Recognition in C++/Java/Go/Scala

Der von Google-Mitarbeiter Robert Hundt initiierte Benchmark vergleicht die vier Sprachen C++, Java, Go und Scala mit einem kombinierten Algorithmus zur Mustererkennung. Nach dem initialen Testlauf wurden Google-intern Code-Verbesserungen und -Anpassungen durch Spezialisten der jeweiligen Programmiersprachen durchgeführt, um Verbesserungspotenziale durch sprachspezifische Features und Einstellungen aufzudecken. [32]

[30] Fourment, Mathieu und Gillings, Michael R (2008), Abs. 1 im Hauptframe (siehe Internetverzeichnis).
[31] Fourment, Mathieu und Gillings, Michael R (2008b), S. 4 (siehe Internetverzeichnis).
[32] Hundt, Robert (2011), S. 2 (siehe Internetverzeichnis).



	Java	Java (optimiert)	C++	C++ (optimiert)
Prozessorlaufzeit (s)	73,95	89	197	23
Garbage Collection (s)	216,05	.	.	.
Prozessorlaufzeit (%)	100%	31%	68%	8%

Abbildung 3: Ergebnisse des Loop Recognition-Benchmarks

Quelle: Eigene Darstellung nach Hundt, Robert (2011)[33]

Hierbei zeigte sich unter anderem, dass der bei Java und Scala aktive Garbage Collector einen Großteil (74,5%) der Laufzeit beansprucht und somit große Auswirkungen auf die Performance hat. Durch Codeoptimierungen und Anpassungen von JVM-Startparametern gelang es dem Google-Team, die Laufzeit von ursprünglich 290 Sekunden auf 89 Sekunden zu senken und somit deutlich näher an die Laufzeit von C++ (23 Sekunden) zu bringen.[33,34]

Optimierungen fanden auch bei den anderen Programmiersprachen statt und umfassten u.a. die Verwendung von möglichst kleinen Datentypen (z.B. Verwendung der empty()-Methode mit boolescher Rückgabe statt size(), welche zuerst einen Integer berechnet[35]) und die Vermeidung von totem, also nicht aufzurufendem Code.

Ein IBM-Mitarbeiter hat es nicht nehmen lassen, die Ergebnisse weiter zu betrachten und eigene Codeanpassungen vorzunehmen, mit dem Fokus auf der Verbesserung der Java-Performance. Die Ergebnisse wurden im IBM developerWorks-Blog veröffentlicht und zeigen, dass bestimmte Compilereinstellungen und Sprachkonstrukte (z.B. „for (int i = 0 ; i < nodeList.size(); i++)“ statt „for (UnionFindNode iter : nodeList)“) die Java-Performance nicht nur weiter verbessern, sondern sogar C++ überholen lassen können. [36]

3.2 Weitere Quellen

Im Rahmen dieses Projektberichts können keine weiteren Benchmarks im Detail betrachtet werden, allerdings erfolgt die Empfehlung, die Augen auch auf Mitbewerber der IT-Branche zu richten. Interessante Einblicke ergeben sich z.B. bei der Betrachtung der verwendeten

[33] Ebenda, S. 8, Abb. 8.
[34] Ebenda, S. 8, Abs. „C. Tuning Garbage Collection“.
[35] Ebenda, S. 9, Abs. „D. C++ Tunings“.
[36] IBM Corp. (2011), Abs. 1 im Hauptframe (siehe Internetverzeichnis).

Programmiersprachen in etablierten Unternehmen. Einzelne Geschichten können hier Einblicke in das Verhalten von Programmen in großen Maßstäben geben. So schreibt eBay seit 2002 alle Anwendungen in Java um (komplettes Re-Engineering), um eine saubere wartbare Basis zu schaffen und die Systemanforderungen gegenüber C++ zu senken.[37] Ähnliche Beobachtungen sind natürlich auch für den umgekehrten Fall bekannt, z.B. bei der Entwicklung der Notizanwendung Evernote, welche seit 2010 auf C++ basiert[38].

4 Kritik an Benchmarks

Trotz vieler wertvoller Erkenntnisse, welche durch Benchmark-Verfahren gewonnen werden können, bergen diese auch immer die Gefahr, bestimmte Potenziale und Chancen unentdeckt zu lassen oder zu missinterpretieren, bedingt durch die Generalisierung der Ergebnisse. Eine weitere Gefahr liegt bei der schlussendlichen Verwendung bzw. Wissensgenerierung aus den Benchmarkergebnissen und Statistiken - ein synthetischer Test, welcher vielleicht hauptsächlich das Verhalten mittels Schleifenkonstrukten misst, kann zum Beispiel keine Aussagen zum Verhalten des Programms bei Datenzugriffen (Lese- und Schreibvorgänge) geben. Oftmals sind bereits die gemessenen Zeiten nicht mit einer realistischen Anwendersituation vergleichbar, in der nicht nur das eine relevante Programm auf dem Computersystem aktiv ist, sondern zusätzliche Prozesse von anderen Anwendungen gestartet werden. Um dem entgegenzuwirken, wird unter anderem die Größe „wall clock time“ beim Benchmarking verwendet[39]. Zudem können Benchmarks unmöglich alle verwendeten Systemkombinationen und -konfigurationen abdecken, welche sich im Alltag finden lassen. Oft warnen die Ersteller der Benchmarks zu Recht selbst vor der Zweckentfremdung der Ergebnisse.[40]

Schon während des Entwicklungsprozesses wirken sich viele Umstände auf die letztliche Programmperformance aus - beispielhaft seien hier verschiedene Programmierstile[41], verwendete externe Bibliotheken und Compiler genannt.

Nichtsdestotrotz zeigen die u.a. in Kapitel 3.1 gewonnenen Erkenntnisse, wie schnell die gezielte Betrachtung von Benchmarks zu einer Verbesserung der Programmperformance beitragen kann, indem sprachspezifische ineffiziente Konstrukte identifiziert werden können.

---

[37] Shoup, Randy und Pritchett, Dan (2006), S. 16 (siehe Internetverzeichnis).
[38] Constantinou, Philip (2010), Abs. „Starting from scratch“ (siehe Internetverzeichnis).
[39] IBM Corp. (2011), Abs. „Benchmarking methodology“ (siehe Internetverzeichnis).
[40] Fulgham, Brent (2013), Abs. „And please don’t jump to conclusions!“ (siehe Internetverzeichnis).
[41] Hundt, Robert (2011), S. 2 (siehe Internetverzeichnis).

4.1 Bewertungskriterien abseits der Performance

Dieser Bericht konzentriert sich auf den Vergleich von Programmiersprachen in Hinblick auf dessen Performance, allerdings sollten die anderen Softwaremerkmale dabei nicht völlig außer Acht gelassen werden, zumal einige davon die Qualität der Entwicklung und des Codes bedeutend beeinflussen.

Jede Programmiersprache befindet sich in einem Ökosystem, bestehend aus Anwendern und Entwicklern, welche zu dessen Fortschritt beitragen. Je größer und breiter gefächert die Anwendergemeinschaft der Programmiersprache ist, desto wahrscheinlicher werden Dokumentationen, Schnittstellen, Frameworks und Hilfestellungen verfügbar sein und das alltägliche Programmieren erleichtern. Je offener und zugänglicher die Sprache gestaltet ist, desto einflussreicher wird dieser Netzwerkeffekt ausfallen.[42]

Neben dieser sozialen und politischen Komponente spielen auch weitere technische Aspekte eine wichtige Rolle. Generell sollte im Voraus geprüft werden, wie gut sich die Programmiersprache in ggf. bestehende Systeme integrieren lässt, hierzu zählt die vorhandene IT-Infrastruktur, die Entwicklungsumgebung und die Offenheit der Programmierer für neue Herausforderungen. Diese Kennzahlen lassen sich nicht durch Benchmarks, sondern eher durch entsprechende Recherchen und Analysen ermitteln.

Davon abgesehen bestehen noch weitere Qualitätsmerkmale, aus welchen sich Anforderungen an das Softwareergebnis und somit auch an die Programmiersprache ableiten lassen. Diese sollen aber nicht mehr Gegenstand dieses Berichts sein.

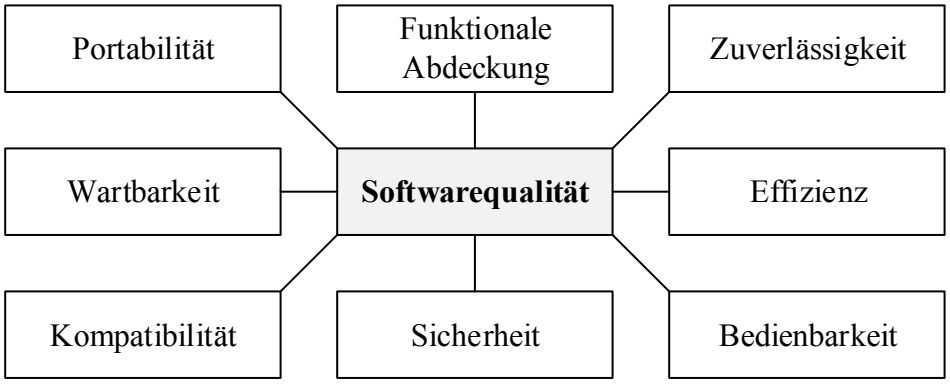


Abbildung 4: Softwarequalitätsmerkmale, Quelle: Eigene Darstellung nach ISO/IEC 25010[43]

[42] Welton, David N. (2005), Abs. „Positive Network Externalities“ (siehe Internetverzeichnis).
[43] ISO (2011), Abs. „Software product quality model“ (siehe Internetverzeichnis).

5 Zusammenfassung

Da die Performance von Entwicklern und Anwendern der Programmiersprachen als wichtiges Merkmal angesehen wird, entfachen immer wieder lange Diskussionen zu dem Thema.[44] Wer das Ziel verfolgt, eine bestimmte Sprache als performanteste oder geeignetste Lösung für eine Vielzahl an Lösungen zu vermarkten, wird jedoch früher oder später daran scheitern.

Auf Grundlage der ermittelten Ergebnisse lässt sich nur eines eindeutig festhalten: Generell schlägt keine Programmiersprache die andere in allen Aspekten. Native Sprachen wie C und C++ agieren näher am Betriebssystem und führen oft die Benchmarklisten an, zahlen dafür aber häufig den Preis einer aufwändigeren Handhabung. Portable High-Level-Sprachen wie Java können ähnlich effizient arbeiten, wenn sie für das Zielsystem optimiert werden und zur Not sogar durch C ergänzt werden. Sowohl native, als auch portable Sprachen sind heutzutage weit verbreitet und dienen als Grundlage für eine große Vielfalt an Software.[45]

Um eine optimale Programmperformance zu erreichen, können sich die Entwickler jedoch durch die Betrachtung von Benchmarks ein Bild von der zu erwartenden Performance machen und Inspirationen bezüglich der Optimierung sammeln. Viele der erwähnten Techniken lassen sich relativ schnell umsetzen und bergen enormes Potenzial, die eigene Programmeffizienz zu steigern.

Fakt bleibt jedoch auch, dass Programmiersprachen oft für einen bestimmten Zweck entwickelt werden und sich nicht alle Sprachen für alle Zwecke äquivalent gut eignen. Einige Sprachen haben sich in bestimmten Bereichen etabliert (z.B. PHP für dynamische Webentwicklung[46]), doch jedes Jahr verändert sich die IT-Landschaft und oftmals lohnen sich innovative Experimente mit jungen Technologien. Belohnt werden die neugierigen Entwickler, welche sich mit den Ursprüngen und Zielstellungen der Sprachen auseinandersetzen und sich neue Techniken aneignen. Von daher steht hinter jedem Programm nicht nur eine Programmiersprache, sondern vor allem ein Team von Entwicklern, welches die Sprache als Werkzeug nutzt, um effiziente Software zu schreiben.

---

[44] Vgl. http://stackoverflow.com/questions/9668709/objective-c-vs-c-speed oder http://stackoverflow.com/questions/926728/will-my-iphone-app-take-a-performance-hit-if-i-use-objective-c-for-low-level-cod

[45] Oracle Corp. (2014c), Liste im Hauptframe (siehe Internetverzeichnis); Stroustrup, Bjarne (2014), Liste im Hauptframe (siehe Internetverzeichnis).

[46] The PHP Group (2014), Abs. 1 im Hauptframe (siehe Internetverzeichnis).

Literaturverzeichnis

Bog, Anja (2014): Benchmarking Transaction and Analytical Processing Systems, 2014.

Clausing, Achim (2011): Programmiersprachen - Konzepte, Strukturen und Implementierung in Java, 2011.

Cowell, Wayne (Hrsg.) (1977): Portability of Numerical Software, 1977.

Hoare, Charles Antony Richard (1972): Software: Practice and Experience, 1972.

International Organization for Standardization (2011): ISO/IEC 25010:2011 - Systems and software Quality Requirements and Evaluation (SQuaRE), 2011.

Internetverzeichnis

Apple Inc. (2010): Why Objective-C?, 15.11.2010, abgerufen am 01.03.2014, https://developer.apple.com/library/ios/documentation/cocoa/conceptual/OOP_ObjC/Articles/ooWhy.html.

Apple Inc. (2012): C++ Versus Objective C, 08.02.2012, abgerufen am 07.03.2014, http://support.apple.com/kb/TA45902?viewlocale=en_US#.

Apple Inc. (2014): Xcode IDE, 2014, abgerufen am 02.03.2014, https://developer.apple.com/xcode/ide/.

Bezanson, Jeff u.a. (2014a): Julia Language 0.2.0 documentation - Introduction, 2014, abgerufen am 06.03.2014, http://docs.julialang.org/en/release-0.2/manual/introduction/.

Bezanson, Jeff u.a. (2014b): The Julia Language, 2014, abgerufen am 05.03.2014, http://julialang.org/.

Binstock, Andrew (2012): The New Native Languages, 08.05.2012, abgerufen am 13.02.2014, http://www.drdobbs.com/architecture-and-design/the-new-native-languages/232901652.

Boehm, Barry W. u.a. (1976): Quantitative Evaluation of Software Quality, 1976, abgerufen am 03.03.2014, http://csse.usc.edu/csse/TECHRPTS/1976/usccse76-501/usccse76-501s.pdf.

Burback, Ronald LeRoi (1998): Towards the Definition of Performance, 14.12.1998, abgerufen am 02.03.2014, http://infolab.stanford.edu/~burback/watersluice/node76.html.

Constantinou, Philip (2010): Evernote 4 for Windows is here!, 2010, abgerufen am 09.03.2014, http://blog.evernote.com/blog/2010/10/26/evernote-4-for-windows-is-here/.

Digital Mars (2014): D Programming Language, 24.02.2014, abgerufen am 08.03.2014, http://dlang.org/.

École Polytechnique Fédérale de Lausanne (2014): The Scala Programming Language, 2014, abgerufen am 08.03.2014, http://www.scala-lang.org/.

Fleming, Ian (2014): ISO9126 - Software Quality Characteristics, 2014, abgerufen am 02.03.2014, http://www.sqa.net/iso9126.html.

Fourment, Mathieu und Gillings, Michael R (2008a): Language benchmark - Results, 2008, abgerufen am 07.03.2014, http://www.bioinformatics.org/benchmark/results.html.

Fourment, Mathieu und Gillings, Michael R (2008b): A comparison of common programming languages used in bioinformatics, 05.02.2008, abgerufen am 14.02.2014, http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2267699/.

Fulgham, Brent (2013): And please don't jump to conclusions!, 01.12.2013, abgerufen am 05.03.2014, http://benchmarksgame.alioth.debian.org/dont-jump-to-conclusions.php.

Fulgham, Brent (2014a): Computer Language Benchmarks Game, 2014, abgerufen am 14.02.2014, http://benchmarksgame.alioth.debian.org/.

Fulgham, Brent (2014b): Computer Language Benchmarks Game - Play, 15.01.2014, abgerufen am 05.03.2014, http://benchmarksgame.alioth.debian.org/play.php.

Fulgham, Brent (2014c): Computer Language Benchmarks Game - Java 7 vs C++, 28.02.2014, abgerufen am 06.03.2014, http://benchmarksgame.alioth.debian.org/u64q/benchmark.php?test=all&lang=java&lang2=gpp&data=u64q.

Google Inc. (2014a): Android, the world's most popular mobile platform, 2014, abgerufen am 01.03.2014, http://developer.android.com/about/index.html.

Google Inc. (2014b): The Go Programming Language, 2014, abgerufen am 08.03.2014, http://golang.org/.

Google Inc. (2014c): What Are Chrome Apps?, 2014, abgerufen am 09.03.2014, https://developer.chrome.com/apps/about_apps.

Hundt, Robert (2011): Loop Recognition in C++/Java/Go/Scala, 2011, abgerufen am 14.02.2014, http://static.googleusercontent.com/media/research.google.com/de//pubs/archive/37122.pdf.

IBM Corp. (2011): Getting C++ like Performance from your Java Application, 30.09.2011, abgerufen am 08.03.2014, https://www.ibm.com/developerworks/community/blogs/738b7897-cd38-4f24-9f05-48dd69116837/entry/getting_c_like_performance_from_your_java_application33?lang=en.

Isensee, Pete (2008): C++ Optimization Strategies and Techniques, 29.12.2008, abgerufen am 02.03.2014, http://www.tantalon.com/pete/cppopt/main.htm.

Janssen, Cory (2014): What is Code Efficiency?, 2014, abgerufen am 02.03.2014, http://www.techopedia.com/definition/27151/code-efficiency.

Kinnersley, Bill (2014a): The Language List, 2014, abgerufen am 28.02.2014, http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm.

Kinnersley, Bill (2014b): The Language List - Language Categories, 2014, abgerufen am 05.03.2014, http://people.ku.edu/~nkinners/LangList/Extras/classif.htm.

LangPop (2013): Programming Language Popularity, 25.10.2013, abgerufen am 11.02.2014, http://www.langpop.com/.

Leibniz-Institut für Wissensmedien (IWM) (2011): Plattformunabhängigkeit, 21.02.2011, abgerufen am 13.02.2014, http://www.e-teaching.org/projekt/nachhaltigkeit/plattform/.

Lévénez, Éric (2013): Computer Languages History, 16.09.2013, abgerufen am 28.02.2014, http://www.levenez.com/lang/.

Lextrait, Vincent (2013): The Programming Languages Beacon, 13.12.2013, abgerufen am 14.02.2014, http://www.lextrait.com/vincent/implementations.html.

Lipinski, Klaus u.a. (2014): JVM - Java virtual machine, 2014, abgerufen am 01.03.2014, http://www.itwissen.info/definition/lexikon/Java-virtual-machine-JVM-Java-virtuelle-Maschine.html.

Mangione, Carmine (1998): Performance tests show Java as fast as C++, 01.02.1998, abgerufen am 02.03.2014, http://www.javaworld.com/article/2076593/performance-tests-show-java-as-fast-as-c--.html.

Marceau, Guillaume (2009): The speed, size and dependability of programming languages, 30.05.2009, abgerufen am 14.02.2014, http://blog.gmarceau.qc.ca/2009/05/speed-size-and-dependability-of.html.

Marcus, Ryan (2012): Objective C v C Speed / Benchmarks, 10.09.2012, abgerufen am 05.03.2014, http://rmarcus.info/?p=488.

Mashey, John R. (2004): Languages, Levels, Libraries, and Longevity, 27.12.2004, abgerufen am 11.02.2014, http://queue.acm.org/detail.cfm?id=1039532.

Microsoft Corp. (2014a): Funktionale Programmierung oder imperative Programmierung?, 2014, abgerufen am 05.03.2014, http://msdn.microsoft.com/de-de/library/bb669144.aspx.

Microsoft Corp. (2014b): Visual Basic Resources, 2014, abgerufen am 08.03.2014, http://msdn.microsoft.com/en-us/vstudio/hh388573.

Microsoft Corp. (2014c): Visual C# Resources, 2014, abgerufen am 08.03.2014, http://msdn.microsoft.com/en-us/vstudio/hh341490.aspx.

Mooney, James D. (1997): Bringing Portability to the Software Process, 1997, abgerufen am 13.02.2014, http://www.cs.wvu.edu/~jdm/research/portability/reports/TR_97-1.pdf.

Nadata, Shmuel (2013): Java vs. C++ for High Performance Systems, 09.09.2013, abgerufen am 02.03.2014, http://www.trdpnt.com/java-vs-c-for-high-performance-systems/.

o.V. (2011): Programming Languages Benchmarks, 2011, abgerufen am 14.02.2014, http://attractivechaos.github.io/plb/.

O'Grady, Stephen (2014): The RedMonk Programming Language Rankings: January 2014, 01.2014, abgerufen am 11.02.2014, http://redmonk.com/sograd y/2014/01/22/language-rankings-1-14/.

Oracle Corp. (1999): Code Conventions for the Java Programming Language, 20.04.1999, abgerufen am 08.03.2014, http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-139411.html#16712.

Oracle Corp. (2014a): How Will Java Technology Change My Life?, 2014, abgerufen am 05.03.2014, http://docs.oracle.com/javase/tutorial/getStarted/intro/changemylife.html.

Oracle Corp. (2014b): NetBeans Platform Showcase, 2014, abgerufen am 09.03.2014, https://platform.netbeans.org/screenshots.html.

Oracle Corp. (2014c): Was ist Java und wozu wird es benötigt?, 2014, abgerufen am 01.03.2014, http://www.java.com/de/download/faq/whatis_java.xml.

Rouse, Margaret (2013): What is a native application?, 14.02.2013, abgerufen am 28.02.2014, http://searchsoftwarequality.techtarget.com/definition/native-application-native-app.

S., Amaya und cplusplus.com (2014): C++ - A Brief Description, 2014, abgerufen am 14.02.2014, http://www.cplusplus.com/info/description/.

Schnabel, Patrick (2012): Compiler und Interpreter, 09.2012, abgerufen am 14.02.2014, http://www.elektronik-kompendium.de/sites/com/1705231.htm.

Shoup, Randy und Pritchett, Dan (2006): The eBay Architecture, 29.11.2006, abgerufen am 09.03.2014, http://www.addsimplicity.com/downloads/eBaySDForum2006-11-29.pdf.

Siepermann, Markus und Lackes, Richard (2014): Definition „Portabilität“, 2014, abgerufen am 13.02.2014, http://wirtschaftslexikon.gabler.de/Archiv/55014/portabilitaet-v7.html.

Stroustrup, Bjarne (2014): C++ Applications, 17.02.2014, abgerufen am 09.03.2014, http://www.stroustrup.com/applications.html.

The C++ Resources Network (2013): History of C++, 2013, abgerufen am 01.03.2014, http://www.cplusplus.com/info/history/.

The PHP Group (2014): PHP: Hypertext Preprocessor, 07.03.2014, abgerufen am 09.03.2014, http://www.php.net/.

TIOBE Software BV (2014): TIOBE Index for February 2014, 02.2014, abgerufen am 11.02.2014, http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

Vervaet, Erwin und De Cock, Maarten (2002): Java optimization techniques, 06.2002, http://www.dsc.ufcg.edu.br/~jacques/cursos/2004.2/gr/recursos/j-javaopt.pdf.

Vogel, Lars (2010): Java Performance - Memory and Runtime Analysis, 12.12.2010, abgerufen am 14.02.2014, http://www.vogella.com/tutorials/JavaPerformance/article.html.

Welton, David N. (2005): The Economics of Programming Languages, 18.07.2005, abgerufen am 11.02.2014, http://www.welton.it/articles/programming_language_economics.html.

Würstl, Daniel (2013): Unterschiede und Vergleich native Apps vs. Web Apps, 21.02.2013, abgerufen am 13.02.2014, http://www.app-entwickler-verzeichnis.de/faq-app-entwicklung/11-definitionen/107-unterschiede-und-vergleich-native-apps-vs-web-apps.