

2 Grundlagen

Aufgrund der Vielzahl von technologischen Fachbegriffen und Themen, die in dieser Arbeit behandelt werden, kann das Kapitel keine ausführliche Einführung in jedes Themenfeld geben. Dieses Wissen wird weitestgehend vorausgesetzt bzw. kann der entsprechenden referenzierten Literatur entnommen werden.

Zunächst geht das Kapitel auf die Begrifflichkeiten aus den Domänen Hardware und Software ein. Anschließend werden Softwareentwicklungskonzepte nach Quasar¹⁶ (Qualitätssoftwarearchitektur) vorgestellt. Des Weiteren stellt das Kapitel verschiedene Definitionen für Portierung, Portabilität und plattformunabhängiger Entwicklung vor, um anschließend eine Ableitung für diese Arbeit zu definieren. Abschließend stellt das Kapitel einen Bezug zu bereits existierenden Arbeiten her.

2.1 Begriffsbestimmung

Nachfolgend werden die in dieser Arbeit verwendeten Hard- und Softwarekomponenten benannt. Sie dienen als Unterstützung für den Sprachgebrauch in dieser Arbeit.

2.1.1 Hardware

In dieser Arbeit werden zwei Gruppen von Rechnerklassen unterschieden:

- Windows PCs
- iOS-gestützte Endgeräte

Im Rahmen dieser Arbeit werden Desktoprechner und Laptops gleichermaßen als Windows-PCs bezeichnet. Ein Desktoprechner ist ein Personal Computer, der aufgrund seiner Bauweise auf den Schreibtisch gestellt werden kann.¹⁷ Er besitzt einen leistungsstarken Mikrocomputer und führt Anwendungen aus, die zumeist von einer Person bedient werden.¹⁸ Ein- und Ausgabegeräte sind Tastatur, Maus (oder vergleichbare Zeigergeräte) und Bildschirm.

Laptops (auch Notebook genannt) gehören ebenfalls zur Gruppe der Personal Computer. Sie besitzen die gleiche Rechnerarchitektur wie Desktoprechner und lassen sich gleichermaßen bedienen. Durch die Möglichkeit des mobilen Einsatzes grenzen

¹⁶ Vgl. Siedersleben, J.: *Moderne Softwarearchitektur*, Heidelberg 2004, S. 4

¹⁷ Vgl. Abts, D.; Müller, W.: *Grundkurs Wirtschaftsinformatik*, 5. Aufl., Wiesbaden 2004, S. 37f

¹⁸ Vgl. Abts, D.; Müller, W.: *Grundkurs Wirtschaftsinformatik*, 5. Aufl., Wiesbaden 2004, S. 55

sie sich jedoch von Desktoprechnern ab.¹⁹ Ihre Bauform ist kleiner und kompakter. Alle Hardware-Peripheriegeräte sind in einem Gehäuse zusammengefasst.

Zu den iOS-gestützten Endgeräten zählen ausnahmslos die beiden mobilen Endgeräte des Technologieunternehmens Apple²⁰. Hier ist einerseits das iPad (Tablet-PC) und andererseits das iPhone (Smartphone) zu nennen, wobei sich der Anwendungsfall auf das iPad bezieht. Auch der iPod-Touch und Apple TV sind mit diesem Betriebssystem ausgestattet, werden aber in dieser Arbeit nicht weiter betrachtet.²¹ iPhone und iPad können als mobile Endgeräte der „neuen Generation“ verstanden werden. Technisch gesehen weisen sie zwar gleiche bzw. ähnliche Eigenschaften wie ihre Vorgänger auf, heben sich aber insbesondere durch ihr Bedienkonzept und ihre leistungsstarken Mikroprozessoren von den Endgeräten der „älteren Generation“, zu denen das Mobiltelefon oder der PDA gezählt werden können, ab. Auch ein Laptop lässt sich aufgrund seiner technischen Eigenschaften den mobilen Endgeräten zuordnen, weist jedoch mehr Übereinstimmungen mit klassischen Desktop-Rechnern auf.

Das Bedienkonzept der iOS-gestützten Endgeräte wird maßgeblich vom berührungsempfindlichen Display getragen, das ein neues Benutzererlebnis suggeriert. Außerdem ist die Leistungsfähigkeit der iOS-gestützten Endgeräte vergleichbar mit modernen Desktop-Rechnern, was ihre Einsatzmöglichkeiten gegenüber Mobiltelefonen und PDAs signifikant erweitert und vollkommen neue Einsatzszenarien ermöglicht. Vergleichbare Endgeräte wären Smartphones und Tablet-PCs, die u.a. mit den Betriebssystem Android²² oder auch Windows Phone²³ ausgestattet sind.

2.1.2 Software

Software lässt sich in die beiden Gruppen Systemsoftware und Anwendungssoftware gliedern.²⁴ Zur Gruppe der Anwendungssoftware gehört auch die in dieser Arbeit betrachtete Desktop-Business-Anwendung.

Nachfolgend ist die Definition einer Anwendungssoftware nach Balzert aufgeführt:

„Anwendungssoftware, auch Applikationssoftware genannt, ist Software, die Aufgaben des Anwenders mit Hilfe eines Computersystems löst. Anwendungssoftware setzt in der

¹⁹ Vgl. Abts, D.; Mülder, W.: Grundkurs Wirtschaftsinformatik, 5. Aufl., Wiesbaden 2004, S. 55

²⁰ <http://www.apple.com>, 09.10.2012

²¹ Vgl. Stäuble M.: Programmieren für iPhone und iPad, 4. Auflage, Heidelberg 2012, S. 35

²² <http://www.android.com/>, 05.11.2012

²³ <http://www.windowsphone.com/de-de>, 05.11.2012

²⁴ Vgl. Balzert, H.: Lehrbuch der Softwaretechnik, Basiskonzepte und Requirements Engineering, Heidelberg 2009, S. 4f

Regel auf der Systemsoftware, der verwendeten Hardware auf bzw. benutzt sie zur Erfüllung der eigenen Aufgaben.“²⁵

Der Begriff App ist eine Kurzform für das Wort Application, das wiederum die englische Übersetzung für Applikation ist und somit eine Anwendungssoftware bezeichnet.

Im engeren Sinne steht der Begriff App als Synonym für eine Gruppe von Anwendungen, die von einem Smartphone oder Tablet-PC heruntergeladen und auf ihm ausgeführt werden.²⁶ Geprägt wurde dieser Begriff vom Technologiekonzern Apple durch die Markteinführung des iPhones²⁷.

Mit Bekanntgabe der neuen Windows Version (Windows 8) greift auch Microsoft auf diesen Begriff zurück und bezeichnet Anwendungen, die über den Windows Store²⁸ heruntergeladen werden als Apps.²⁹

In dieser Arbeit bezeichnet der Begriff **Anwendungssoftware** sämtliche Anwendungen, die ein Desktoprechner ausführt. Mit dem Begriff **App** sind jene Software-Produkte gemeint, die ein mobiles Endgerät installiert und ausführt.

2.1.3 Native App

Native Apps bezeichnen Anwendungen die speziell für eine Plattform entwickelt wurden. Sie sind in einer plattformspezifischen Programmiersprache geschrieben und werden vom Compiler für die Plattform optimiert. Sie müssen grundsätzlich vor der ersten Ausführung auf der Zielplattform installiert werden.³⁰ Hierdurch ist es nicht möglich, die App auf einer anderen Plattform auszuführen. Eine native iOS-App kann beispielsweise nur auf einem iOS-gestützten Endgerät ausgeführt werden. Die binäre Portierung auf ein Android-gestütztes Endgerät ist nicht möglich. Somit ist eine native App nicht plattformunabhängig.

²⁵ Balzert, H.: Lehrbuch der Softwaretechnik, Basiskonzepte und Requirements Engineering, Heidelberg 2009, S. 5

²⁶ Vgl. Franko, O. I.; Tirrell, T. F.: Smartphone App Use Among Medical Providers in ACGME Training Programs, in: Journal of Medical Systems, 36. Jg., 2012, H. 5

²⁷ Vgl. o.V.: Apple erfindet mit dem iPhone das Mobiltelefon neu, <http://www.apple.com/de/pr/library/2007/01/09Apple-Reinvents-the-Phone-with-iPhone.html>, 09.10.2012

²⁸ Vgl. Kolokythas, P.: Microsoft bestätigt Windows Store für Windows 8 <http://www.pcwelt.de/news/Windows-8-besitzt-Windows-Apps-3418928.html>, 09.10.2012

²⁹ Vgl. o.V.: Erste Schritte mit Apps im Metro-Stil, <http://msdn.microsoft.com/library/windows/apps/br211386>, 09.10.2012

³⁰ Vgl. Bertram, A.: Nativ versus Web - Verschiedene Ansätze bei der App-Entwicklung, <http://stud.fh-wedel.de/~inf7337/native-app-entwicklung.html>, 05.11.2012

Native Apps werden typischerweise über eine App Store vertrieben und stehen dadurch einer breiten Nutzergemeinde zur Verfügung. Der Entwickler braucht sich nicht um den Vertriebsweg der App kümmern, muss allerdings bei Kostenpflichtigen Apps einen festgelegten Prozentsatz vom Verkaufspreis der App an den Store-Betreiber abführen. Bei Apple muss der Entwickler außerdem den Source Code offenlegen, damit Apple ein Review vornehmen kann. Das fördert zwar die Qualität der Apps und vermeidet eventuellen Schadcode, führt aber dazu, dass sich der Release-Zeitpunkt heraus zögert. Auch die Verteilung innerhalb von Unternehmen ist möglich. Dabei muss ebenfalls ein zentraler Server eingesetzt werden, der die Verteilung steuert. Die Kontrolle über das Verteilungssystem obliegt dem Unternehmen.

Native Apps haben direkten Zugriff auf sämtliche Hardwareschnittstellen, die durch das Betriebssystem abstrahiert werden. Das führt zu einer erhöhten Performance, da keine zwischengeschalteten Wrapper-APIs benötigt werden. Außerdem hat der Entwickler Zugriff auf die Frameworks zur Gestaltung der grafischen Benutzerschnittstellen. Hier durch können Apps im nativen Look & Feel der Hersteller erzeugt werden.

2.2 Softwarekomponenten nach Quasar

Das Akronym Quasar steht für Qualitätssoftwarearchitektur³¹ und wurde vom deutschen Softwareunternehmen sd&m³² entwickelt. Quasar versucht auf vier Ebenen (Ideen und Konzept, Begriffe, Standardarchitektur/-schnittstellen, Standardkomponenten) besonders wichtige Regeln und Mechanismen der Softwaretechnik verständlich zu beschreiben, zu präzisieren und zum Standard zu erklären.³³ Letztendlich soll Quasar dazu beitragen, die Frage zu beantworten, wie sich gegebene Anforderungen so einfach wie möglich in Software umsetzen lassen.

Die nachfolgende Abhandlung soll nicht den Aufbau und die Inhalte von Quasar beschreiben. Vielmehr geht das Kapitel im Sinne der Portierung auf die Kategorisierung von Komponenten ein, die zum Zeitpunkt der Portierung eine wichtige Rolle spielt. Quasar hat hier einen leichten und nachvollziehbaren Weg gefunden, Komponenten in Kategorien einzuordnen. So lässt sich im Rahmen der

³¹ Vgl. Siedersleben, J.: Moderne Softwarearchitektur, Heidelberg 2004, S. 4

³² Vgl. o.V.: Capgemini sd&m tritt ab sofort unter der Marke Capgemini auf, <http://www.de.capgemini.com/news-events/news/capgemini-sdm/>, 08.10.2012

³³ Vgl. Siedersleben, J.: Moderne Softwarearchitektur, Heidelberg 2004, S. 4

Portierung leichter identifizieren, welche Komponente unbedingt und welche Komponente vielleicht portiert werden sollte.

Ausführliche Erläuterungen über Quasar liefern die Bücher von Siedersleben³⁴ sowie Engels³⁵ et al.

2.2.1 Komponente

Laut Quasar definiert sich eine Komponente aus sechs Merkmalen. Wobei Quasar die ersten fünf Merkmale aus etablierten Definitionen extrahiert und das sechste Merkmal ergänzt. Eine oft zitierte Quelle ist Szyperski:

*„A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.“*³⁶

Quasar betrachtet eine Komponente aus verschiedenen Blickwinkeln, bei denen die Technik eine eher untergeordnete Rolle spielt.

Nachfolgend sind Merkmale nach Quasar aufgelistet:³⁷

1. Eine Komponente **exportiert** eine oder mehrere Schnittstellen. Die Schnittstelle kann als **Vertrag** zwischen exportierenden und importierenden Komponente gesehen werden.
2. Eine Komponente **importiert** Schnittstellen anderer Komponenten und nutzt somit die Methoden der importierten Schnittstelle.
3. Eine Komponente **versteckt** die Implementierung und kann daher gegen eine andere Komponente ersetzt werden, die die gleiche Schnittstelle exportiert.
4. Eine Komponente ist **wiederverwendbar**, da sie ihre Umgebung, in der sie läuft, nicht kennt, sondern nur minimale Annahmen über sie macht.
5. Komponenten lassen sich über beliebig viele Stufen aus vorhandenen Komponenten zusammensetzen.

Siedersleben definiert als sechstes Merkmal und als zentrale Rolle der Komponente, dass eine Komponente neben der Schnittstelle als wesentliche Einheit des **Entwurfs**, der **Implementierung** und damit der **Planung** gilt.

³⁴ Siedersleben, J.: Moderne Softwarearchitektur, Heidelberg 2004

³⁵ Engels, G. et al.: Quasar Enterprise: Anwendungslandschaften serviceorientiert gestalten, Heidelberg 2008

³⁶ Szyperski, C. et al.: Component Software: Beyond Object-Oriented Programming, London 2002, S. 195

³⁷ Vgl. Siedersleben, J.: Moderne Softwarearchitektur, Heidelberg 2004, S. 42f

Damit ist gemeint, dass das Denken in Komponenten schon mit der Planung einer Softwareanwendung beginnt und über die Schritte Entwurf und Implementierung konkretisiert wird.

2.2.2 Kategorisierung

Die Kategorisierung von Komponenten befasst sich mit der Trennung von Zuständigkeiten innerhalb eines Software-Systems, die im Nachhinein über Abhängigkeiten wieder zusammengefasst werden. Dabei ist es erstrebenswert, technische Problemstellungen (z.B. Datenbank- oder Betriebssystemzugriff) von Anwendungsproblemen (z.B. kann die Buchung noch storniert werden) zu trennen. Quasar formalisiert diese Idee folgendermaßen:

„Jedes Software-System befasst sich mit der fachlichen Anwendung, denn dafür wurde es gebaut. Und es verwendet eine technische Basis (Betriebssystem, Datenbank, Verbindungssoftware), denn im luftleeren Raum kann es nicht laufen.“³⁸

Jede Komponente einer Anwendung lässt sich einer der fünf folgenden Softwarekategorien zuordnen:³⁹

- **0-Software:** Komponenten der 0-Software sind unabhängig von Anwendung und Technik. Sie stellen die Basis jeder Anwendung dar und haben sich in der Softwareentwicklung etabliert. Sie sind wiederwendbar, für sich alleine aber ohne Nutzen. Zur 0-Software gehören beispielsweise Komponenten, die sich mit Datentypen oder Container befassen. In .NET gehören dazu z.B. Komponenten des System-Namensraums⁴⁰. Auch die Implementierung eines Designpatterns lässt sich zur Kategorie der 0-Software zuordnen.
- **A-Software:** Komponenten der A-Software werden bestimmt durch den Anwendungskontext. Sie sind damit stark abhängig von der Anwendung aber unabhängig von der Technik. A-Software befasst sich mit Begriffen wie Bankkunde, Sparbuch, Girokonto oder Buchung und nicht mit Technologien zum Datenbankzugriff oder der Serialisierung von Daten. A-Software wird zumeist speziell für einen Anwendungsfall entwickelt. Ihre Funktionalität bestimmt in der Regel der Wert einer Anwendung.
- **T-Software:** Komponenten der T-Software kennen Technologien und sind unabhängig von der Anwendung. Diese Komponenten lassen sich in vielen

³⁸ Vgl. Siedersleben, J.: Quasar: Die sd&m Standardarchitektur Teil 1, München 2003, S. 3

³⁹ Vgl. Siedersleben, J.: Quasar: Die sd&m Standardarchitektur Teil 1, München 2003, S. 3

⁴⁰ Vgl. o.V.: System Namespace, <http://msdn.microsoft.com/en-us/library/yxcx7skw.aspx>, 08.10.2012

verschiedenen Anwendungen einsetzen und erfüllen meist eine ähnliche Aufgabe. Hierzu zählen beispielsweise Technologien wie Ole DB⁴¹ oder Xaml⁴².

- **AT-Software:** Die Kombination von A- und T-Software führt zur ungewünschten **AT-Software**. AT-Software ist stets zu vermeiden, da sie eine hohe Abhängigkeit zwischen Anwendung und Technologie erzeugt.⁴³ Durch diese Abhängigkeit ist diese Kategorie schwer zu warten und lässt sich kaum wiederverwenden.⁴⁴
- **R-Software:** „R-Software transformiert fachliche Objekte in andere Repräsentationen und wieder zurück.“⁴⁵ Sie übernimmt keine fachlichen Aufgaben. So zählt z.B. die Komponente der Serialisierung zur R-Software, da sie das fachliche Objekt in ein anderes Format, wie z.B. XML, übersetzt. Das Erzeugen des XML-Formats übernimmt wiederum eine technische Komponente.

Respektive zu den postfixen der Softwarekategorien (0-, A-, T-, AT- und R) lassen sich die Komponenten als 0-, A-, T-, AT- und R-Komponenten bezeichnen.

2.3 Softwarearchitekturen nach Quasar

Quasar sieht für die Gestaltung der Softwarearchitektur drei Ebenen vor, die sich wie folgt definieren:⁴⁶

- **TI-Architektur:** Sie beschreibt die technische Infrastruktur des Informationssystems. Dazu gehören die technischen Geräte wie Rechner, Netzleitungen und Drucker, die installierte Systemsoftware wie z.B. das Betriebssystem und das Zusammenspiel von Hardware und Systemsoftware. Auch die verwendete Programmiersprache zählt Quasar zur TI-Architektur
- **A-Architektur:** Sie beschreibt die Architektur der Anwendung und somit die anwendungsspezifischen Bestandteile des Systems. Laut Quasar ist es ohne weiteres möglich, „ohne Annahme über die Technik und nur in Kenntnis der fachlichen Anforderungen die Anwendung in sinnvolle Komponenten zu

⁴¹ Vgl. o.V.: Übersicht über die Ole DB-Programmierung, <http://msdn.microsoft.com/de-de/library/5d8sd9we%28v=vs.80%29.aspx>, 08.10.2012

⁴² Vgl. o.V.: Übersicht über Xaml, <http://msdn.microsoft.com/de-de/library/vstudio/ms752059.aspx>, 08.10.2012

⁴³ Vgl. Siedersleben, J.: Quasar: Die sd&m Standardarchitektur Teil 1, München 2003, S. 3

⁴⁴ Vgl. Siedersleben, J.: Moderne Softwarearchitektur, Heidelberg 2004, S. 90

⁴⁵ Vgl. Siedersleben, J.: Quasar: Die sd&m Standardarchitektur Teil 1, München 2003, S. 3

⁴⁶ Vgl. Siedersleben, J.: Moderne Softwarearchitektur, Heidelberg 2004, S. 145f

zerlegen und diese Komponenten ändern sich im Entwicklungsprozess nicht oder nur wenig⁴⁷.

- **T-Architektur:** Die T-Architektur ist das Bindeglied zwischen der A- und der TI-Architektur. Sie stellt Schnittstellen bereit, gegen die die A-Komponenten auf die technische Infrastruktur zugreifen können. Sie definieren insbesondere den Umgang mit dem Betriebssystem und anderen systemnahen, technischen Komponenten.⁴⁸

Die A-Architektur beinhaltet die A-Komponenten und die T-Architektur die T-Komponenten.

2.4 Portierung und Portabilität

Im Kern befasst sich die Arbeit mit einer Portierung. Aus diesem Grund werden die Begriffe Portierung und Portabilität nachfolgend definiert und erläutert.

2.4.1 Portierung

Der folgende Abschnitt soll einen Konsens für den Terminus Portierung schaffen. Indirekt lässt sich aus dem Terminus Portierung der Fachausdruck Portabilität ableiten, für den die Arbeit ebenfalls eine gemeingültige Definition ableitet.

Der Abschnitt gliedert sich in zwei Teile. Im ersten Teil werden Definitionen aus einschlägiger Literatur vorgestellt und bewertet. Darauf aufbauend leitet der Autor im zweiten Teil eine für diese Arbeit geltende Beschreibung der genannten Fachausdrücke ab. Hilfestellung bietet die veröffentlichte Diplomarbeit von Michael Müller, die in Kapitel 2.1.1⁴⁹ eine umfassende Sammlung der Definitionen liefert. Zusätzlich hat der Autor aktuelle Definitionen ergänzt.

Die Portierung an sich beschreibt den Prozess, der notwendig ist, um ein Softwaresysteme auf eine andere Systemumgebung zu übertragen. Auch in der einschlägigen Literatur lassen sich hierzu einige Definitionen finden. So beschreibt Mooney (1997), in seinem Technical Report, die Portierung als ein Verfahren, um ein bereits existierendes Software-System auf einer neuen Umgebung zu erstellen.

⁴⁷ Siedersleben, J.: Moderne Softwarearchitektur, Heidelberg 2004, S. 151

⁴⁸ Vgl. Siedersleben, J.: Moderne Softwarearchitektur, Heidelberg 2004, S. 156

⁴⁹ Müller, M.: Metriken zur Portabilitätsanalyse Windows basierter Software-System, Diplomarbeit, Humboldt-Universität zu Berlin, Berlin 2002, S. 15-20

„Porting is the act of producing an executable version of a software unit or system in a new environment, based on an existing version.“⁵⁰

In seiner Definition vereinigt er unter dem Begriff **software unit** Anwendungsprogramme, Systemprogramme und Teilprogramme. Unter einem **software system** versteht er eine Sammlung von **software units**. Ein **Environment** bezeichnet er als eine Gruppe von Elementen, die mit der Software kommuniziert. Dazu zählt er Prozessoren, Betriebssysteme, Ein- und Ausgabegeräte, Netzwerke und sogar den Menschen.

In der weiteren Ausführung seines Reports betont er, dass eine Portierung nicht mit einer Neuentwicklung (engl. Redevlopment) gleichzusetzen ist, sondern als Alternative angesehen werden sollte. Da eine Portierung aus seiner Sicht auf eine bereits existierende Implementierung basiert, wohin die Neuentwicklung von einer existierenden Spezifikation ausgeht.⁵¹

Kaindl geht in seinem Artikel *Portability of Software* noch einen Schritt weiter, indem er den Ausdruck der Portierung anhand einer Formel beschreibt.

„A port is a mapping $(P, E(\{l_1, \dots, l_i\}, o, m)) \rightarrow (P', E'(\{l'_1, \dots, l'_i\}, o', m'))$, where P is the program to be ported, P' the resulting program after the port, E the original and E' the target environment. The following condition must be hold: $(\{l_1, \dots, l_i\} \neq \{l'_1, \dots, l'_i\}) \vee (o \neq o') \vee (m \neq m')$.“⁵²

Bestandteil dieser Formel ist eine klare Definition der Umgebung (E), die wie folgt definiert wird:

„An Environment E is a triple $(\{l_1, \dots, l_i\}, o, m)$, where $\{l_1, \dots, l_i\}$, $i \in \mathbb{N}$, is a set of language processors, o is an operating system and m is a machine“

Nach oben genannter Regel ist von einer Portierung die Rede, sobald mindestens

- eine Implementierungssprache,
- das Betriebssystem,
- die Maschine oder
- mehrere Umgebungsmerkmale gleichzeitig

verändert werden.

⁵⁰ Mooney, J.D.: Bringing Portability to the Software Process, Technical Report TR 97-1, West Virginia University, Morgantown WV 1997, S. 2

⁵¹ Vgl. Mooney, J.D.: Bringing Portability to the Software Process, Technical Report TR 97-1, West Virginia University, Morgantown WV 1997, S. 2

⁵² Kaindl, H.: Portability of Software, in: SIGPLAN Notices, 23. Jg., 1988, H. 6, S.

Ebenso werden in der Literatur die Begriffe Transport und Adaptierung erwähnt. Mooney nennt sie im Zusammenhang mit der Portierung:

*„The porting process has two principal components which may be called adaption and transportation. Adaption is any modification that must be performed on the original version. Transportation is physical movement, with low-level adaptation.“*⁵³

Unter Adaptierung versteht Mooney die Anpassung des Software-Systems an die neue Umgebung. Der Transport beschreibt die Übertragung auf das neue System. Mit *low-level adaptation* meint er die Anpassung von unterschiedlichen Datenformaten auf die neue Umgebung.⁵⁴

2.4.2 Portabilität

Nun stellt sich aber die Frage, ob eine Portierung überhaupt lohnenswert ist. Dafür hat sich der Begriff der Portabilität etabliert. Er drückt aus, wie leicht ein Softwaresystem in eine andere Umgebung zu übertragen ist. Aus Sicht von Hoffmann beschreibt sie den Grad der **Plattformunabhängigkeit** eines Software-Systems.⁵⁵

Auch Mooney befasst sich in seiner bereits erwähnten Ausführung mit diesem Ausdruck:

*„A software unit is portable (exhibits portability) across a class of environments to the degree that the cost to transport and adapt it to a new environment in the class is less than the cost of redevelopment.“*⁵⁶

Er beschreibt die Portabilität als Vergleich für den Aufwand einer Neuentwicklung und den Aufwand für eine Portierung. Dabei wird der Aufwand nicht als zeitliche sondern als monetäre Kennziffer gesehen. Andere Autoren gehen da nicht soweit und sehen die Portabilität eher als Eigenschaft oder Fähigkeit. So definiert Hommel die Portabilität folgendermaßen:⁵⁷

⁵³ Mooney, J.D.: Strategies for Supporting Application Portability, in: IEEE Computer, 23. Jg., 1990, H. 11, S. 59ff

⁵⁴ Müller, M.: Metriken zur Portabilitätsanalyse Windows basierter Software-System, Diplomarbeit, Humboldt-Universität zu Berlin, Berlin 2002, S. 18

⁵⁵ Vgl. Hoffmann, Dirk W.: Software-Qualität, Berlin, Heidelberg 2008, S. 107

⁵⁶ Mooney, J.D.: Bringing Portability to the Software Process, Technical Report TR 97-1, West Virginia University, Morgantown WV 1997

⁵⁷ Hommel, G.: Portabilität von Software, in: Schneider, H.G. (Hrsg.): Portable Software, Erlangen 1980, S. 10

Strategie für die Portierung von
Desktop-Business-Anwendungen auf iOS-gestützte
Endgeräte

Schmitz, M.

2014, XIX, 135 S. 43 Abb., Softcover

ISBN: 978-3-658-04768-9