Einführung in die maschinennahe, imperative, funktionale, relationale und objektorientierte Programmierung

-

EMIFROP 0.25

Markus Alpers
B.Sc. und Ausbilder f. Industriekaufleute

29. Februar 2016

Inhaltsverzeichnis

Hinweis bezüglich diskriminierender Formulierungen

In diesem Text wurde darauf geachtet Formulierungen zu vermeiden, die diskriminierend verstanden werden können. Im Sinne der Lesbarkeit wurden dabei Formulierungen wie „Informatiker und Informatikerinnen“ durch „InformatikerInnen“ (mit großem i) ersetzt. An anderen Stellen habe ich Formen wie eine/einer durch eineR zusammengefasst. Hier berufe ich mich auf den Artikel „Sprache und Ungleichheit“ der Bundeszentrale für politische Bildung, kurz BpB, vom 16. April 2014, insbesondere auf den Absatz „Zum Umgang mit diskriminierender Sprache“, online abrufbar unter:

`http://www.bpb.de/apuz/130411/sprache-und-ungleichheit?p=all`

Sollten Sie dennoch Formulierungen entdecken, die diesem Anspruch nicht entsprechen, möchte ich Sie bitten, mir eine entsprechende Nachricht zu senden, denn es ist mir wichtig, Ihnen mit diesem Buch eine wertvolle Unterstützung beim Start in die faszinierende Welt der Informatik zu bieten. Das sollte nicht durch verletzte Gefühle in Folge missverständlicher Formulierungen torpediert werden.

Sie erreichen mich unter `markus.alpers@haw-hamburg.de`.

Hinweis zur Lizenz

zenz"´. Wenn Sie einen Teil finden, in dem diese „Hinweise zur Lizenz"´ nicht zu finden ist, dann dürfen Sie für den persönlichen Gebrauch dennoch Kopien davon anfertigen und Sie dürfen diese Kopien außerhalb von kommerziellen Projekten frei verwenden.

Hinweis zur Verwendbarkeit in wissenschaftlichen Arbeiten

Bitte beachten Sie dabei aber, dass die Verwendung dieses Textes im Rahmen wissenschaftlicher Publikationen zurzeit aus anderen Gründen problematisch ist: Wie viele andere Quellen, die frei im Internet verfügbar sind, wurde auch dieser Text bislang nicht durch einen nachweislich entsprechend qualifizierten Lektor verifiziert. Damit genügen Zitate aus diesem Band streng genommen noch nicht den Ansprüchen wissenschaftlicher Arbeiten.

Bitte beachten Sie außerdem, dass dieses Buch eine Konvention nutzt, die in wissenschaftlichen Arbeiten verpönt ist: Wenn in einer wissenschaftlichen Arbeit ein Begriff hervorgehoben wird, dann wird dazu kursive Schrift verwendet. In diesem Buch verwende ich dagegen Fettdruck, da es vielen Menschen schwer fällt, einen kursiv gedruckten Begriff schnell zu finden und ich mir wünsche, dass Sie es möglichst effizient auch als Nachschlagewerk nutzen können.

Zielgruppe und Vorwort

Dieses Buch habe ich erstellt, um Studierenden der Studiengänge Media Systems (entspricht Medieninformatik an anderen Hochschulen) und Medientechnik an der HAW Hamburg den Einstieg ins Programmieren zu erleichtern. Deshalb finden sich hier teilweise Anmerkungen für die Studierenden der beiden Studiengänge, die aber in ähnlicher Form für Studierenden der Informatik und der Elektrotechnik gelten. Da es jedoch so formuliert ist, dass es für Studienanfänger ohne Programmiererfahrung geeignet ist, kann jede/r Studierende es gut nutzen, um sich in die Programmierung einzuarbeiten. Wenn die Version 1.0 abgeschlossen ist wird es eine **grundlegende Einführung** in die Konzepte (Paradigmen), der maschinennahen, der imperativen, der funktionalen, der relationalen und der objektorientierten Programmierung in den Ausprägungen protoypbasiert und klassenbasiert sein. Zusätzlich behandelt es den Einstieg in die Entwicklung verteilter Anwendungen.

Wie alle Lehrbücher für Studierende setzt es eines voraus: Wenn Sie es nutzen wollen, dann funktioniert das dann, und ausschließlich dann, wenn Sie zusätzlich zum Lesen zwei Dinge tun: Zum einen müssen Sie ständig kontrollieren, ob Sie jeden **neuen Begriff wirklich verstanden** haben und prüfen, wie er im Zusammenhang mit dem bisher Gelernten steht und zum anderen **müssen Sie tatsächlich programmieren**.

Was Sie hier nicht finden sind zum einen alle Varianten der Programmierung, die im Kern aus der Elektrotechnik entstanden sind oder für deren Verständnis Sie die Grundlagen kontinuierlicher Systeme beherrschen müssen. In der Informatik werden diese Bereich als **Technische Informatik** bezeichnet. Das schließt beispielsweise die Programmierung von Steuer- und Regelsystemen, also insbesondere **SPS**e und **FPGA**s ein.

Damit sind wir auch schon bei einem ersten Missverständnis das zwischen InformatikerInnen einerseits und NaturwissenschaftlerInnen, IngenieurInnen und TechnikerInnen (kurz **INT-Akademiker**) existiert: Was in der Informatik als **Technische Informatik** bezeichnet wird ist alles, was die übrigen drei als Informatik kennen. Diese gehen deshalb in aller Regel von der irrigen Vorstellung aus, das InformatikerInnen Programmierung meinen, wenn sie von **Praktischer Informatik** reden. Tatsächlich haben beide (Praktische Informatik und Programmierung) kaum etwas miteinander zu tun. An dieser Stelle sei deshalb (vorrangig für Informatikstudierende) betont:

Dies ist eine **Einführung ins Programmieren, nicht in die Praktische Informatik**. Es wird zwar immer wieder Hinweise auf die Praktische und

Theoretische Informatik geben, aber vorrangig ist und bleibt dies eine Einführung ins Programmieren.

Die **systemnahe Programmierung** und die Programmierung von **Parallelprozessoren** sowie die Implementierung von **Protokolle**n für die Datenübertragung über Netzwerke entfallen ebenfalls. Dennoch werden Sie in diesem Buch zumindest einen Einblick in die Grundlagen der systemnahen Programmierung erhalten, da diese Systeme die Grundlage für alle Programmieransätze darstellen, die Sie hier kennen lernen können. Hier gilt dasselbe, was schon im letzten Absatz galt: INT-Akademiker kennen in aller Regel nur die systemnahe Programmierung und alle Konzepte, die sich direkt daraus ableiten lassen und die von InformatikerInnen mit dem Oberbegriff **Technische Informatik** bezeichnet werden. Es gibt jedoch auch Programmierkonzepte, die damit nicht mehr verständlich sind und für die es nötig ist, sich wesentlich grundlegender und abstrakter mit der Programmierung zu beschäftigen. Dazu kommen wir im zweiten Teil dieses Buches.

Fragen des **Software Engineering** werden zwar angerissen und es gibt Hinweise auf typische Missverständnisse, eine grundlegende Einführung ins Software Engineering kann dieser Band jedoch nicht ersetzen. Wie der Titel dieses Buches klar ausdrückt, geht es hier ums Programmieren. An den entsprechenden Stellen werden Sie aber entsprechende Hinweise auf Bücher und Themengebiete finden, damit Sie ggf. wissen, wonach Sie für weiterführendes Wissen suchen müssen. Ob sie sich nun zunächst in die Programmierung oder ins Software Engineering stürzen wollen, bleibt Ihnen überlassen; beides hat seine Vor- und Nachteile. In der Informatik müssen Sie aber in jedem Fall beides durcharbeiten, um auch nur in Ansätzen gute Software entwickeln zu können.

Der Grund ist simpel: Bei der **Programmierung** geht es darum, Konzepte zur Lösung eines Programms in eine Sprache zu übersetzen, die ein Computer ausführen kann. Beim **Software Engineering** geht es dagegen darum, gute Konzepte zu entwickeln, die in Programmiersprachen übersetzt werden können. Wer nur eines von beidem beherrscht entwickelt häufig Programme, die niemandem nützen oder nützliche Konzepte, die niemand (in Form eines Computerprogramms) nutzen kann. Deshalb gibt es in jedem brauchbaren Kurs zur Programmierung Auszüge Teile, die eigentlich in den Bereich des Software Engineering gehören[1]. Umgekehrt enthält jeder brauchbare Kurs zum Software Engineering Teile zur Programmierung[2].

[1] Weshalb es nur wenige Kurse zur Programmierung gibt, die nach Ansicht dieses Autors brauchbar sind.

[2] Weshalb auch hierfür aus Sicht dieses Autors nur wenig Brauchbares auf dem Markt

Zusätzlich müssen Sie jedoch in jedem Fall noch die Grundlagen der Praktischen Informatik erlernen, um hochwertige Software zu erstellen. Diese können Sie im Bereich der Algorithmik (genauer **Algorithmen und Datenstrukturen**, **Algorithmendesign** sowie **Algorithmik**) erlernen.

Aufgrund der häufigen **Änderungen bei aktueller Software** kann dieses Buch nur beschränkt Unterstützung bei Installations- und Konfigurationsfragen bieten. Hier bleibt zu hoffen, dass die Entwickler der einzelnen Sprache bzw. zusätzlicher Software eine ausreichende Dokumentation auf Ihrer Webpage bereitstellen.

Nochmal in anderen Worten: Ein häufiges Missverständnis besteht darin, dass Programmierung und Informatik bzw. Programmierung und Praktische Informatik miteinander verwechselt werden. Denn **Programmierung** ist lediglich die Umsetzung einer Idee mit Hilfe einer Sprache, die einem Computer befiehlt, **was er tun soll**. Ob sie auch festlegt, **wie er das tun soll** ist eine ganz andere Frage. Einführungen in die Programmierung, die hier nicht deutlich werden sind der Grund für eine Vielzahl von Missverständnissen rund um die Programmierung.

Um überhaupt zu programmieren, müssen Sie also lediglich wissen, wie die Befehle und Befehlsstrukturen einer Sprache aussehen. Im Kern ist das also nichts anderes als das Erlernen einer gesprochenen Sprache. Doch so wie es selbst für das Erlernen nahe verwandter gesprochener Sprachen eben nicht ausreicht, nur die Übersetzung einzelner Wörter zu erlernen, genügt es für die kompetente Beherrschung von Programmiersprachen nicht, sich nur grundsätzlich damit beschäftigt zu haben: So wie Sie eine gesprochene Sprache tatsächlich in Gesprächen benutzen müssen, um sie zu erlernen, müssen Sie eine Programmiersprache benutzen, indem Sie eine Vielzahl an Programmen damit entwickeln.

Wichtig:
Fähige (Medien-)InformatikerInnen sind nicht automatisch guten ProgrammiererInnen. Wenn Sie verstanden haben, was der Unterschied zwischen Informatik und Programmieren ist, dann wird es sie wundern, dass es überhaupt Menschen gibt, die diese Aussage bezweifeln.

Die **Informatik** dagegen setzt sich mit der Frage auseinander, wie und ob eine bestimmte Idee besonders elegant und effizient umgesetzt werden kann. **Ob für die Umsetzung der Idee ein Computer nötig ist, ist zweitrangig.** Aber da Computer die Stärke haben, dass Sie langweilige Aufga-

---

ist.

ben mit einer für uns unfassbaren Geschwindigkeit ausführen, sind Sie das Werkzeug Nummer 1 für die Informatik. Zumindest ist nach Ansicht dieses Autors die Durchführung von 4 Milliarden Additionen pro Sekunde unfassbar schnell. Zum Vergleich: Würden alle Menschen auf dieser Welt gleichzeitig eine Addition zweier Zahlen mit bis zu 20 Stellen durchführen und für die Berechnung sowie das Aufschreiben nur zwei Sekunden brauchen, dann wären sie alle gemeinsam genauso schnell wie ein einzelner Prozessor, der in einem handelsüblichen Computer steckt.

Hier ein Beispiel, mit dem sich Informatikstudierende gegen Ende des Bachelorstudiums auseinander setzen: Sie fahren in den Skiurlaub und wollen mal das Skifahren ausprobieren. Nun könnten Sie die Skier kaufen oder mieten. Da Sie ja nicht wissen, ob Ihnen Skifahren wirklich Spaß macht, wäre es unsinnig, gleich am ersten Tag das Geld für den Kauf auszugeben. Aber auch am zweiten Tag wäre es nicht unbedingt sinnvoll, denn wer weiß, ob Sie am dritten Tag noch Lust dazu haben. Die Frage lautet also: Wann macht es Sinn, die Skier zu kaufen? Das ist ein Beispiel für einen Bereich, der in der Informatik als **online-Algorithmen** bezeichnet wird. Der zugehörige Bereich der Praktischen Informatik heißt **Algorithmendesign**.

Online-Algorithmen dienen beispielsweise dazu, die Verwaltung des Speichers einer Festplatte zu organisieren oder um die Mitarbeiter für Kassen in einem Supermarkt einzuplanen. Bei online-Algorithmen geht es immer um die Frage: Wie bereite ich mich am besten auf eine Situation vor, von der ich noch nicht genau weiß, wie sie aussehen wird? Und wie Sie sehen hat das zunächst einmal nichts mit Programmieren zu tun.

Sie möchten ein Beispiel, bei dem es um Programmierung und online-Algorithmen geht? Dann haben Sie leider noch nicht verstanden, was der Unterschied zwischen Praktischer Informatik und Programmieren ist. Also weiter mit den online-Algorithmen: Denken Sie beispielsweise daran, was bei ebay kurz vor Ende einer Versteigerung passiert. Oder wie wäre es mit einem online-Händler wie amazon, kurz vor Weihnachten: Wann wie viele Menschen versuchen werden, auf eine einzelne Seite zuzugreifen, kann niemand wissen. Aber mit fähigen Informatikern kann jeder sich darauf so gut wie möglich vorbereiten. Hier sind wir übrigens auch schon ganz nahe an einem aktuellen Forschungsgebiet der (Medien-)Informatik: Bei **Big Data** handelt es sich um einen Bereich, in dem aus gigantischen Datenmengen versucht wird, Gruppen von Personen mit gemeinsamen Eigenschaften herauszufiltern und dann Prognosen über deren zukünftiges Verhalten zu schließen. Hier sind wir ganz nahe am Gebiet des **Datenschutzes** mit ganz konkreten Auswirkungen auf das alltägliche Leben. Denn Big Data führt zum Teil zu absurden Abläufen: Wenn Sie beim Ausfüllen eines Kreditantrages online mehrfach Einträge ändern, dann wird alleine deshalb

der Zinssatz erhöht oder der Antrag abgelehnt. Die Begründung stammt direkt aus dem Bereich angewandten Big Data und lautet so: Menschen mit niedrigem Bildungsniveau vertippen sich häufiger als Menschen mit hohem Bildungsniveau. Menschen mit hohem Bildungsniveau haben in aller Regel eine höheres Einkommen, längere Arbeitsverhältnisse und eine geringere Wahrscheinlichkeit, arbeitslos zu werden. Das lässt sich verkürzen zu: Menschen mit höherem Bildungsstand sind in aller Regel zuverlässiger bei der Rückzahlung von Krediten. Wenn wir jetzt noch die erste Aussage hinzunehmen, lautet die Schlussfolgerung nach den Prinzipien des Big Data: Wer sich öfter vertippt wird häufiger Probleme haben, einen Kredit zurückzuzahlen. Also wird der Zinssatz erhöht oder der Kredit gleich ganz verweigert. Und nein, das ist kein Scherz, sondern findet so Anwendung bei online-Finanzdienstleistern.

Wieder zurück zu den online-Algorithmen: In der BWL wird dieser Teil der Informatik als **Logistik** bezeichnet, wobei der Bereich der online-Algorithmen deutlich mehr umfasst als nur Logistik. BWLern ist dabei in aller Regel leider nicht bewusst, dass es sich hier um einen Bereich handelt, der eine Kernkompetenz der Informatik ist. Das führt dazu, dass in der Forschung zur BWL teilweise Forschungen betrieben werden, um Probleme zu lösen, für die die Informatik längst eine Lösung bereit hält. Denken Sie bitte dennoch nicht in Kategorien wie Schuld: Es ist ein grundsätzliches Problem, dass zu viele Akademiker nur in den Kategorien der eigenen Disziplin denken und kaum den Austausch mit anderen Disziplinen suchen. Das ändert sich zwar in einigen wenigen Fällen, doch häufig kommt es dabei nicht zu einem vollwertigen Austausch, sondern es wird versucht, die jeweils andere Disziplin der eigenen anzupassen. Ein erster Schritt, um das nicht zu tun besteht darin, sich darüber auszutauschen, was Begriffe der jeweiligen Disziplin bedeuten. Ein „gutes"´ Beispiel haben bereits kennen gelernt: Was INT-Akademiker als Informatik bezeichnen ist für Informatiker in aller Regel nur der Teilbereich der **Technischen Informatik**. So lange solche Missverständnisse nicht ausgeräumt sind und AkademikerInnen unterschiedlicher Disziplinen die Kompetenz des/der jeweils anderen nicht anerkennen, ist ein echter Austausch nicht möglich. Und dann sind auch umfassende Projekte mit Beteiligung unterschiedlicher wissenschaftlicher Disziplinen nicht möglich.

Sie studieren **Medientechnik** und fragen sich, was das mit Ihrem Studium zu tun hat? Ganz einfach: Auch in Ihrem Bereich werden Computer eingesetzt und Sie werden nicht immer eine fertige Lösung in Form eines Computerprogramms vorfinden. Also müssen Sie im Stande sein, einfache Probleme mit einem Computer selbst zu lösen. Und wenn die Probleme zu komplex werden, dann müssen Sie im Stande sein, InformatikerInnen zu erklären, worin das Problem besteht, denn sonst können die keine Lösung

für Sie entwickeln. Wie wäre es beispielsweise mit einem Programm, mit dem Sie Ihre Schaltskizzen für das E-Technik-Labor erstellen können, die Sie außerdem online speichern und abgeben können, ohne sie als Mailanhang verschicken zu müssen? Wenn Sie die Veranstaltung „Programmieren 1“ (entspricht Teil I dieses Buches) erfolgreich absolvieren, dann werden Sie im Stande sein, solche Programme selbst zu entwickeln.

Sie studieren **Media Systems** und fragen sich, warum Sie einen Kurs mit dem Titel „Einführung ins Programmieren“ (kurz PRG) belegen sollen, wenn Sie doch schon die Veranstaltung „Programmieren 1“ (kurz P1) belegen? In der Veranstaltung P1 lernen Sie die Entwicklung von Programmen mit einer imperativen und klassenbasierten objektorientierten Programmiersprache. In PRG konzentrieren wir uns dagegen auf einen anderen Bereich: Die Entwicklung verteilter Anwendungen. Wann immer Sie eine App nutzen, nutzen Sie eine verteilte Anwendung. Wann immer Sie ein Programm nutzen, das das Internet oder ein anderes Netzwerk voraussetzt, nutzen Sie eine verteilte Anwendung. Normalerweise ist das Stoff für ein Masterstudium, aber ich habe diese Veranstaltung so konzipiert, dass sie für Einsteiger ohne Vorkenntnisse geeignet ist. Denn so bekommen Sie einen Eindruck, wie all die verschiedenen Veranstaltungen Ihres Studiums ein sinnvolles ganzes ergeben.

Hier nochmals meine Bitte: Wenn Sie in diesem Skript Fehler finden (was bei einem Dokument dieser Länge unausweichlich der Fall ist), Sie weitergehende Fragen haben oder Ergänzungsvorschläge, dann senden Sie diese bitte an mich: `markus.alpers@haw-hamburg.de`

Work in Progress

Der Begriff Work in Progress bedeutet, dass eine Arbeit noch nicht abgeschlossen ist und somit in Teilen unvollständig und in anderen Teilen unnötig detailliert ist. Das gilt zurzeit für dieses Buch: Zum einen habe ich verschiedene Passagen noch nicht abgeschlossen, zum anderen habe ich verschiedene Texte, die ich ursprünglich für unterschiedliche Kurse erstellt hatte hier zusammengefasst. Da diese Zusammenführung noch nicht abgeschlossen ist, wird es Passagen geben, in denen Sie nahezu identische Aussagen und Erklärungen erneut finden werden. Auch wenn Ihnen solche Passagen auffalen, schicken Sie mir bitte eine E-Mail.

Teil I

Einführung in die Programmierung für alle Studierenden im Bereich MINT

(Mathematik, Ingenieurswissenschaften, Naturwissenschaften, Technik)

Kapitel 1

Typische Irrtümer darüber, was Programmieren ist.

Studierende im Studiengang **Media Systems** besuchen unter anderem die Veranstaltungen Programmieren 1 und 2 sowie Informatik 3. Ziel dieser Veranstaltungen ist, dass Sie die Grundlagen zweier Arten der Programmierung erlernen. Diese werden als imperative bzw. prozdurale und klassenbasierter objektorientierte Programmierung bezeichnet.

Studierende der **Medientechnik** besuchen ebenfalls zwei Veranstaltungen mit dem Namen Programmieren 1 und 2. Die Inhalte entsprechen einer einfachen Zusammenfassung dessen, was Studierende in Media Systems in den Veranstaltungen „Einführung ins Programmieren"´, „Software Engineering"´ und „Relationale Datenbanken"´ erlernen. Sie bekommen so einen kurzen Einblick in die Bereiche, mit denen sie immer wieder zu tun haben werden, die aber eigentlich Kernbereiche der Informatik sind. Der Grund dafür ist recht simpel: Sobald elektrotechnische Systeme (also der Kernbereich der Medientechnik) zu komplex werden, um sie ohne zusätzliche Strukturierung zu nutzen, kommen wir in einen von zwei Bereichen: Nachrichtentechnik und Informatik. Beide können ohne Verständnis der Elektrotechnik nur zum Teil verstanden werden, aber das gleiche gilt auch umgekehrt.

Aber bevor wir uns ansehen, was diese beiden Arten der Programmierung ausmacht, wo Schnittpunkte und wo Unterschiede vorliegen, sollten wir eine Frage klären: Was verstehen wir eigentlich unter dem Begriff „Programmieren"´? Gerade diejenigen, die schon programmiert haben, sollten diesen Abschnitt lesen, denn Sie werden denken, dass Ihnen dieser Begriff klar ist. Einzig diejenigen, die bereits imperativ und (!) deklarativ programmiert haben, werden wissen, worin der Unterschied liegt und können ihn überspringen. (Verwechseln Sie aber bitte nicht die Deklaration einer Va-

riablen mit der deklarativen Programmierung. Beide haben soviel miteinander gemein wie Schweinezucht mit Flugzeugbau.)

1.1 Das ist an diesem Buch anders

Es gibt eine Vielzahl an Einführungen ins Programmieren. Die meisten davon gehören in eine von zwei Kategorien:

- **Variante a** richtet sich an Studierende an Universitäten und ignoriert weitgehend die konkrete Programmierung in einer Sprache. Der Fokus liegt hier vorrangig auf Aspekten der **Algorithmik**. Diese sind zwar außerordentlich wichtig, um fähigeR InformatikerIn zu werden, aber ohne eine Einführung in die konkrete Programmierung in einzelnen Sprachen ist sie kaum verständlich. Daran scheitern dann auch viele Studienanfänger. Und von denen, die nicht daran scheitern versteht nur ein Bruchteil, was Algorithmik ist. Am Ende gibts dann haufenweise Informatikabsolventen von Universitäten, die zwar ganz passabel programmieren können, deren Programme aber letztlich sehr schlecht strukturiert sind.

- **Variante b** behandelt dagegen nur die konkrete Programmierung in einer Sprache und in einer bestimmten Version, ohne dabei auf die allgemeinen Grundlagen einzugehen. Wer eine solche Einführung nutzt hat in aller Regel ein derart mangelhaftes Verständnis der grundlegenden Prinzipien, auf deren Basis die jeweilige Sprache entwickelt wurde, dass er/sie selbst mit großem Aufwand nicht im Stande ist, eine weitere Programmiersprache so zu erlernen, dass er/sie diese wirklich nutzen könnte. Ständig heißt es dann „warum macht der das denn nicht,"´ und es wird über die vermeintlich schlechte andere Sprache geflucht. Dabei ist das Problem nicht die „andere"´ Sprache, sondern die Tatsache, dass jemand mit dem Verständnis einer Programmiersprache versucht, eine andere Programmiersprache zu erlernen. Doch wenn diese andere Sprache alles genauso machen würde, wie die erste, dann wäre es komplett unsinnig, sie zu erlernen. (Medien-)informatikerInnen lernen deshalb vorrangig die Konzepte kennen, die in verschiedenen Programmiersprachen jeweils unterschiedlich eingesetzt werden.

Beide Ansätze ignorieren darüber hinaus, dass für viele Menschen sich mit der Programmierung beschäftigen wollen, das Innenleben von Rechnern unbekanntes Gebiet sind. Diese Einführung holt Leser dagegen an dem

Punkt ab, an dem keine Vorkenntnisse nötig sind und führt sie kontinuierlich in das Themengebiet ein. Der erste Teil ist dabei so aufgebaut, dass ein Überblick über einige Möglichkeiten der Programmierung vermittelt werden. Erst wenn das geschafft ist, wenn also Leser ein Grundverständnis von verschiedenen Arten der Programmierung haben, beginnt mit dem zweiten Teil die eigentliche Einführung in die Grundlagen der Programmierung. Aber auch wenn Sie schon programmieren können (egal ob in HTML, Java, C oder welcher Sprache auch immer), sollten Sie Teil I des Buches durcharbeiten, weil hier bereits einige Konzepte eingeführt werden und anhand von Beispielen in einer oder mehreren Sprachen verdeutlicht werden.

1.2 Zentrale Begriffe und Konzepte beim Programmieren

Häufig werden die Begriffe Programmieren und Informatik in einen Topf geworfen, dabei haben Sie nicht wirklich viel gemeinsam. Damit Sie also wissen, was Ihnen dieses Buch im Rahmen eines Informatikstudiums bietet und was nicht, schauen wir uns einmal an, was Programmieren eigentlich ist und was Sie von Anfang an beachten sollten.

1.2.1 Der Begriff des Programmierens

Wenn Sie beispielsweise einen HDD-Rekorder programmieren, dann reden Sie zwar vom Programmieren, gehen aber sicher nicht davon aus, dass Sie sich in einem Informatikstudium mit der Frage auseinander setzen, wie Sie einen solchen Rekorder programmieren können.

Interessanterweise können Sie diese Frage aber nach dem Besuch der Veranstaltung „Informatik 3"beantworten: Dort geht es um die Programmierung von Mikroprozessoren, also just der kleinen schwarzen Boxen, die seit Mitte der 80er Jahre praktisch jedes elektrische Gerät steuern. Na gut, die meisten Toaster noch nicht... Spätestens mit dem **IoT**, dem **Internet of Things** wird das aber kommen.

Aber was machen wir dann in Media Systems in Programmieren 1 und 2? Außerdem fehlt immer noch die Antwort auf die Frage, was Programmieren denn eigentlich ist. Von der Antwort auf die Frage, was das dann wiederum mit Informatik oder gar Medieninformatik zu tun hat, mal ganz zu schweigen.

Wenn wir (wie üblich) zunächst per deutscher Wikipedia suchen, dann erhielten wir am 27. April 2015 die Auskunft, dass es um das Erstellen von

Computerprogrammen geht, was dabei wichtig ist und wer schon etwas darüber geschrieben hat. Aber die eigentliche Antwort auf die Frage, was wir tun, wenn wir programmieren, steht nicht dort.

Dabei ist das recht simpel: Wenn wir programmieren, dann teilen wir einem Computer schlicht mit, **dass er eine Reihe von Aufgaben erfüllen soll**. Und ja, damit ist auch das Drücken der Ruftaste auf einem Telefon eine Programmierung. Nochmal: Es geht darum, dass wir dem Rechner mitteilen, dass er etwas tun soll. Die Frage in welcher Form wir das tun ist davon vollkommen unabhängig und wird unter dem Oberbegriff des **Paradigma**s geklärt.

Kontrolle
Sie sollten jetzt verstanden haben, dass wir den Begriff des Programmierens deutlich allgemeiner verwenden, als das üblicherweise von Programmierern getan wird. Wenn Sie denken, dass Programmieren beinhaltet, wie der Rechner Aufgaben ausführen soll, dann haben Sie eine zu beschränkte Vorstellung des Begriffs Programmieren.

1.2.2 Programmierparadigmen – Wie Programme entwickelt werden können

Sie wissen es jetzt bereits: Einen Computer zu programmieren bedeutet nicht, dass Sie ihm Schritt für Schritt erklärt, wie er eine Aufgabe lösen soll. Denn wenn wir über diese spezielle Art der Programmierung reden, dann nennen wir das **imperative Programmierung**: Hier erstellen Sie wie bei einem Kochrezept Zeile für Zeile eine Liste von Anweisungen, die beschreiben, wie der Rechner eine Aufgabe in Form einzelner Schritte lösen soll. Da das Programm aber nur aus den einzelnen Schritten besteht, ist später nicht mehr erkennbar, welche Aufgabe das Programm lösen soll.

Probleme tauchen hier immer dann auf, wenn ProgrammiererInnen ein Programm erstellt haben, das zu einem Ergebnis kommt, dieses Ergebnis aber nicht die gewünschte Aufgabe löst. Das liegt zum Teil an so subtilen und doch nicht trivialen Aspekten wie der Division einer Zahl durch eine andere Zahl mittels eines Computers.

Im Gegensatz zu dem, was Sie aus dem Deutschunterricht in der Schule als „den Imperativ"kennen bedeutet imperative Programmierung also nicht nur, dem Computer Befehle zu erteilen, sondern auch ihm zu befehlen, **wie** er einen Befehl auszuführen hat.

Kommen wir damit zur obersten und unumstößlichen Regel bei der Programmierung: **Nicht der Computer oder die Nutzer sind schuld, wenn et-**

was schief läuft, sondern ausschließlich die Entwickler. Wenn Entwickler beispielsweise den Eindruck bei Käufern erzeugen, dass die Nutzung ganz simpel ist, dann ist das nicht die Schuld von Menschen, die sich auf diese Aussage verlassen. Na gut: Wenn Kunden mit Kommentaren kommen wie „Das will ich nicht wissen,"dann sind sie selbst schuld, aber auch nur dann... also leider fast immer...

Wenn Sie schon einmal programmiert haben, dann werden Sie jetzt wahrscheinlich einwenden, dass man doch nur imperativ programmieren kann. Und damit liegen Sie so falsch wie jemand, der denkt, dass **Programmieren** und **Informatik** praktisch dasselbe wären, oder dass Programmieren und **Praktische Informatik** dasselbe wären. Wie im Vorwort geschrieben haben beide kaum etwas gemeinsam, sondern das eine (Informatik) kann unter anderem dazu genutzt werden, um das andere (Programmieren) gut zu machen.

Mit der Informatik und dem Programmieren ist beispielsweise so, wie mit dem Energiesparen und dem Bau eines Hauses: Es ist möglich, ein energiesparendes Haus zu bauen, aber der Hausbau an sich hat mit Energieersparnis nichts zu tun. Und umgekehrt können Sie in wesentlich mehr Bereichen Energie sparen als nur beim Hausbau: Das eine (Energiesparen) hat etwas mit der Herangehensweise an eine Vielzahl von Bereichen zu tun, das andere (Hausbau) ist eine im Vergleich dazu nur in wenigen Bereichen einsetzbare Tätigkeit, für die Sie das erste aber sehr sinnvoll einsetzen können.

Wenn wir von imperativer Programmierung sprechen, dann fassen wir damit eine Reihe an Programmierparadigmen zusammen. Wenn Sie imperativ programmieren, dann wird das in bestimmten Fällen als **prozedurale Programmierung** oder auch als **strukturierte Programmierung** bezeichnet. Die prozedurale Programmierung ist eine Methode, die dazu gedacht ist, um schlecht lesbaren Programmcode zu vermeiden. Meist ist mit imperativer Programmierung der Spezialfall der prozeduralen und der strukturierten Programmierung gemeint.

Bei der prozeduralen Programmierung zerlegen wir eine Aufgabe so lange in immer genauer definierte Anweisungen, bis wir eine Abfolge von Zeilen haben, die direkt einer Programmzeile einer Programmiersprache entspricht.

Bei der strukturierten Programmierung müssen wir außerdem bestimmte Vorgaben beachten, die verhindern, dass unser Programm unübersichtlich wird. So sind hier Sprünge innerhalb des Programms nur dann erlaubt, wenn wir dadurch einen anderen Programmteil überspringen. Ein Rücksprung an eine beliebige frühere Stelle ist verboten. Es gibt zwar noch

die sogenannten Schleifen und Rekursionen, doch die erlauben keinen beliebigen Sprung zurück zu irgen einem früheren Teil des Programms, sondern stellen nur die Möglichkeit zur Verfügung, Teile des Programms zu wiederholen, bevor das Programm weite Zeile für Zeile abgearbeitet wird. Dabei können wir allerdings durchaus Programmteile entwickeln, bei denen unter bestimmten Bedingungen andere Teile des Programms übersprungen werden.

Leider wird häufig von der **objektorientierten Programmierung** gesprochen (auch diesem Autor passiert das immer wieder), dabei gibt es streng genommen keine objektorientierte Programmierung. Objektorientierung ist strenggenommen ein Konzept des Software Engineering, das zwar in einigen Programmiersprachen direkt angewendet werden kann, aber in aller Regel handelt es sich bei dem, was „objektorientierte"´ Sprachen anbieten nur um ein Konzept, das eher eine aktualisierte Form der strukturierten Programmierung ist: Umfangreiche Programmteile werden hier in sogenannte Module (bei Java beispielsweise als Klassen und Package bezeichnet) „verpackt"´. Dadurch werden umfangreiche Programme übersichtlicher.

Objektorientierung an sich geht einerseits weit darüber hinaus und hat andererseits mit der Lösung einer Aufgabe durch einen Computer eigentlich kaum etwas zu tun. Sie ist im Grunde ein Gegenentwurf zur Grundlage der imperativen Programmierung: Da bei dieser binäre Prozessoren mit Datenübertragungsleitungen und Speicher der konzeptionelle Ausgangspunkt sind, hat sie streckenweise starke Restriktionen, was die Umsetzung von Problemlösungen anbelangt. Die Objektorientierung ist also ein Entwurf, der die Softwareentwicklung von den strikten Restriktionen der imperativen Programmierung unabhängig macht. Sprachen, die tatsächlich die Objektorientierung integrieren sind deshalb für Entwickler mit Erfahrung in imperativer Programmierung kaum zu verstehen. Wenn wir uns der prototypbasierten Programmierung in JavaScript zuwenden, dann werden Sie verstehen, warum das so ist.

Bevor wir zu einer anderen Art der Programmierung kommen, sollten noch einige Begriffe eingeführt werden: Ein **Algorithmus** ist eine Beschreibung dafür, **wie** ein Problem gelöst werden soll. Es ist allerdings noch kein **Computerprogramm**. Wenn Sie basierend auf Algorithmen programmieren, dann haben wir es immer (!) mit **imperativer Programmierung** zu tun. Wann immer also eine Einführung in die Programmierung mit dem Begriff des Algorithmus beginnt, handelt es sich nicht um eine allgemeine Einführung, sondern um eine Einführung in die imperative Programmierung. Ein Computerprogramm ist in diesen Fällen grundsätzlich die Umsetzung eines oder mehrerer Algorithmen in eine bestimmte Programmiersprache.

Manchmal wird in diesem Zusammenhang auch von **Pseudocode** gesprochen. Das ist ein Algorithmus, der so aufgeschrieben wurde, dass er einem (imperativen) Computerprogramm ähnelt. Deshalb ist die Lösung eines Problems in Pseudo-Code sehr praktisch: Angenommen, Sie suchen im Netz nach einer effiziente Lösung für ein Problem, über das Sie bei Ihrer aktuellen Programmieraufgabe stolpern. Wenn Sie die Aufgabe in einer bestimmte Programmiersprache lösen sollen und im Netz finden Sie Code für eine andere Programmiersprache, dann müssen Sie beide Sprachen beherrschen, um die Lösung in Ihre Sprache zu übersetzen. Ist die Lösung dagegen in Pseudocode gegeben, dann können Sie sie umsetzen, so lange Sie die eigene Programmiersprache grundlegend beherrschen.

Dann wäre da noch der Unterschied zwischen Hardware und Software:

- **Hardware** ist die Gesamtheit aller Computerprogramme und sonstiger Bestandteile von Computern, die als „anfassbare"´ Komponenten vorliegen,
- **Software** ist die Gesamtheit all der Computerprogramme, die ausschließlich in Form von Daten in Rechnersystemen unterwegs sind.
- Richtig gelesen: **Hardware ist genauso sehr ein Programm bzw. Bestandteil von Programmen, wie die sogenannte Software.** Häufig werden diese Begriffe dagegen so erklärt, als wenn Hardware kein Programmteil wäre. Und das ist falsch; bis auf Dinge wie das Gehäuse eines Rechners dient praktisch die Gesamtheit der Komponenten aus denen ein Rechner zusammen gesetzt ist dazu, Daten zu speichern und zu verarbeiten. Die Speicherung und Verarbeitung eines Programms ist aber bereits ein Programmablauf. Und damit stellt auch die Hardware eine Sammlung von Programmen dar. Später werden wir uns über Server und Client oder auch über Backend und Frontend unterhalten. Genau wie die Unterteilung in Hardware und Software sind auch diese Unterteilungen für uns als (Medien-) InformatikerInnen vollkommen irrelevant.

Hier sind wir auch direkt beim Zusammenhang zwischen Elektrotechnik, Nachrichtentechnik und Informatik: Alle drei beschäftigen sich zum überwiegenden Teil mit der Nutzung von Stromflüssen, um sinnvolle Aufgaben zu erfüllen. Allerdings übernehmen biologische Moleküle und Reaktionen zwischen diesen einen immer größeren Raum in allen drei Bereichen ein oder schaffen sogar vollständig neue Anwendungs- und Forschungsgebiete. In diesem Buch werden wir uns allerdings fast ausschließlich auf die Bereiche konzentrieren, in denen es um Anwendungen auf Basis fließender

Ströme geht:

- Die **Elektrotechnik** setzt einfache Schaltelemente ein, verbindet diese zu zum Teil hochkomplexen Schaltungen und findet vorrangig in der Peripherie von IT-Systemen Anwendungen.
- Die **Nachrichtentechnik** beschäftigt sich mit der Frage, wie beliebige Daten über verschiedene Medien und Distanzen oder Zeiträume transportiert werden können.
- Die **Informatik** beschäftigt sich dagegen mit der Frage, wie Daten zur Erzeugung von Daten genutzt werden können. Sie setzt die Nachrichtentechnik also zur Datenübertragung von Ort zu Ort und zur Speicherung von Daten ein. Die Elektrotechnik kommt hier insbesondere bei der Interaktion von Informatik-Systemen mit der Umgebung ein.
- Wenn wir Informatik mit Nachrichtentechnik oder Elektrotechnik verbinden, landen wir direkt bei der **Technischen Informatik**.

Aber zurück zum eigentlichen Thema dieses Abschnitts und damit zu einer anderen Art der Programmierung:

Es gibt auch Programmiersprachen, in denen man die **Prämissen** der Aufgabe beschreibt und den Rechner dann auffordert, eine mögliche Lösung zu nennen. Eine Prämisse ist eine Voraussetzung für etwas bzw. bei der logischen Programmierung so etwas wie eine Bedingung, die in irgend einer Form Auswirkung darauf hat, welche möglichen Lösungen für unser Problem existieren. Dieser Ansatz der Programmierung wird als **logische Programmierung** bezeichnet und gehört in den Bereich der **deklarativen Programmierung**. Zur Erklärung:

Und vermutlich ploppen genau jetzt vor Ihrem inneren Auge die Fragezeichen auf: Wie soll das denn gehen?! Da wir uns zunächst mit verschiedenen Formen der imperativen Programmierung beschäftigen werden, sei hier nur ein Beispiel angeführt:

Jemand fragt Sie, ob Sie ihm den Weg zu einem Restaurant beschreiben können. Hier gibt es natürlich mehrere Möglichkeiten zu antworten. Wie sinnvoll eine Antwort ist, das hängt von den Prämissen ab, die für den Fragenden gelten, z.B.: Welches Verkehrsmittel will er nutzen? Welche Teile des Stadtplan kennen Sie? Usw. usf.

Eine Form deklarativer Programmierung besteht nun darin, dass Sie all diese bekannten Fakten (Straßennetz, Bedingungen des Fragenden, usw.)

einprogrammieren. In der logischen Programmiersprache **PROLOG** geschieht das in Form sogenannter **Klausel**n. Diese Klauseln ähneln sehr den Relationen, die Sie in mathematischen Vorlesungen kennen lernen. Abschließend geben Sie eine Klausel ein, die z.B. die Frage repräsentiert, ob es einen Weg zum Ziel (hier dem Restaurant) gibt, ob es einen Weg einer bestimmten Länge dorthin gibt usw. Im Gegensatz zur imperativen Programmierung müssen Sie dem Computer dagegen nicht einprogrammieren, wie er nach diesem Weg suchen soll. Das erfolgt nach Regeln der Aussagenlogik und ist bereits als Teil der Programmiersprache festgelegt. Das Programm gibt dann eine mögliche Lösung an oder es gibt an, dass es keine solche Lösung gibt. Wenn Sie sich also bislang mit der Planung des Einsatzes von Mitarbeitern herumgeschlagen haben, dann erlernen Sie doch stattdessen die Programmierung in PROLOG, dann können Sie dieses Problem durch einen Computer lösen lassen.

Solche unterschiedlichen Ansätze der Programmierung werden auch als Paradigmen bzw. Programmierparadigmen bezeichnet. Langfristig werden Sie eine Vielzahl weiterer Paradigmen kennen lernen. Wenn Sie später einen Master in Informatik machen wollen, müssen Sie sich damit bereits während des Bachelorstudiums beschäftigen.

Kontrolle
Was Sie sich an dieser Stelle merken sollten, sind zunächst zwei Punkte:

- dass es verschiedene Paradigmen gibt,
- dass Sie Programmiersprachen danach auswählen sollten, ob sie für das Paradigma passend sind, mit dem Sie gerade zu tun haben.

Wenn Sie das verstanden haben, dann haben Sie auch verstanden, warum Diskussionen sinnlos sind, in denen es darum geht, dass gewisse Sprachen nichts taugen. Für Betriebssysteme gilt ähnliches. Es ist allerdings durchaus möglich, dass frühere Versionen von Sprachen (genau wie Betriebssystemen) schlichtweg überholt sind und damit tatsächlich nicht mehr sinnvoll einsetzbar sind. Vor allem zeichnet es (Medien-)InformatikerInnen aus, dass sie im Stande sind, ein Paradigma auszuwählen, mit dem ein bestimmtes Problem effizient gelöst werden kann.

Das zweite, was Sie jetzt verstanden haben sollten ist, dass das was die meisten Menschen allgemein unter Programmierung verstehen die sogenannte prozedurale Programmierung ist, die zur Obergruppe der imperativen Programmierung gehört.

1.2.3 Man muss doch nicht immer das Rad neu erfinden – Middleware, Framework und Bibliothek

Nahe verwandt mit Programmiersprachen sind die Begriffe **Middleware**, **Framework** und **Bibliothek**. In der Urzeit der Programmierung entwickelte jeder Programmierer alles selbst. Dann wurden Softwarepakete entwickelt, die wie der Teil einer Programmiersprache verwendet werden können, aber im Grunde vollständige oder fast vollständige Programme sind. Je nachdem, wie umfangreich diese Pakete sind und was sie an funktionalem Umfang bieten, unterscheidet man zwischen den drei genannten Arten.

Der Begriff Middleware hat eine besondere Bedeutung, die Sie dann kennen lernen werden, wenn Sie eine Veranstaltung zur Nachrichten- oder Kommunikationstechnik besuchen. Dort steht er in aller Regel weniger für etwas, das direkt mit Programmierung zu tun hätte, sondern vielmehr für bestimmte Teile von Strukturen, Aufgaben und Hierarchien innerhalb eines Netzwerkes.

Im Rahmen dieses Buches werden wir zwischen den dreien nicht unterscheiden, die Begriffe werden hier nur eingeführt, damit Sie wissen, dass es da um Programmteile geht, die Sie in Ihrer Software nutzen können, ohne sie selbst entwickelt zu haben.

Wichtig
Java beinhaltet zwar auch einen Teil, mit dem Sie imperativ programmieren können, aber **zum Großteil ist Java eine Middleware**.

Kontrolle
Es sollte Ihnen bewusst sein, dass Sie sich viel Zeit sparen können, indem Sie auf Middlewares, Frameworks und Bibliotheken zurückgreifen. Wie genau das geht, ist Teil aller Veranstaltungen, in denen Sie programmieren. Aber wie schon eingangs erläutert ist das Programmierung und nicht Praktische Informatik.

1.2.4 Damit Sie sich aufs Entwickeln konzentrieren können – IDEs / Entwicklungsumgebungen

Auch **IDE**s (Integrated Development Environments bzw. Entwicklungsumgebungen) sind ein Mittel, mit dem Sie sich viel Arbeit sparen können. Eine IDE ist eine Zusammenstellung von Programmen, die Sie beim Programmieren unterstützen. Anfangs werden wir ohne eine IDE arbeiten, da Einsteiger häufig von den vielen Komfortfunktionen eher verwirrt als unterstützt werden.

Kontrolle
Sie sollten den Begriff IDE in Zukunft so selbstverständlich benutzen, wie andere Menschen den Begriff Brot.

1.2.5 Dokumentation

Nachdem Sie jetzt also einen ersten Eindruck davon haben, was Programmieren ist und wie Sie es sich erleichtern können, kommen wir zu einem entscheidenden Unterschied zwischen dem Alltag von professionellen ProgrammiererInnen und von HobbyprogrammiererInnen: Die Arbeit im Team.

Sie wissen wahrscheinlich, dass professionelle Software üblicherweise nicht von einzelnen Entwicklern in der Garage, sondern zum Teil von Hunderten von Mitarbeitern entwickelt wird. Und die müssen irgendwie miteinander arbeiten. Bevor wir hier auf die Details eingehen, folgt die zweite wichtige Regel fürs Programmieren:

Ein einsamer Wolf kam eine gute Software initiieren, aber dauerhaft können nur Teams daraus eine gute Software entwickeln.

Der erste Schritt, um im Team erfolgreich Software zu entwickeln, ist die Dokumentation. Dokumentationen sind gleichzeitig der
aufwändigste und vermeintlich wertloseste Teil der Arbeit im Team. Doch langfristig ist eine Softwareentwicklung ohne Dokumentation zum Scheitern verurteilt.

Stellen Sie sich einfach folgende Situation vor: Ein Kollege hat (irgendwann) einen Programmteil entwickelt, die in einem Spezialfall fehlerhaft funktioniert, der bislang nie auftrat. Zum Glück ist Ihnen das aufgefallen. Aber leider kann sich niemand mehr daran erinnern, wo genau sie in den 5000 Zeilen steckt und wer das damals eigentlich programmiert hat. Hätten Sie eine gute Dokumentation, dann würden Sie es jetzt nachschlagen. So können Sie nur beten, dass der Fehler niemals auffällt... Was natürlich bei der Steuerung eines AKWs keine gute Lösung ist.

Kontrolle
Wenn Sie in ein bestehendes Team einsteigen wollen, fragen Sie nach der Dokumentation. Gibt es keine oder ist sie sehr dünn, dann wird das Projekt scheitern, weil am Ende niemand mehr versteht, welche Funktion wo erfüllt wird. Allerdings werden Dokumentationen in aller Regel nicht ausgedruckt.

1.2.6 SCM / Versionskontrolle

Damit Sie nun mit anderen gemeinsam an einer Software arbeiten können, gibt es Programme, die als Versionskontrollsysteme bezeichnet werden. Daneben ist auch die Bezeichnung **Software Control Management (SCM)** üblich. Bei diesen wird ihr Programm in einem sogenannten **Repository** gespeichert, das auf einem Server platziert wird, den Sie über ein Netzwerk erreichen können.

Im Gegensatz zu dem, was sie wahrscheinlich bislang kennen gelernt haben, werden im Repository auch alle Änderungen (**Delta**s) gespeichert, sodass Sie mittels einzelner Befehle unterschiedliche Versionen der Software einsehen und bearbeiten können.

Sie werden aktuell vorrangig auf zwei SCMs treffen: **Git** und **SVN** (kurz für **Subversion**). Ohne auf die Details einzugehen: Bei Git haben Sie den Vorteil, dass Sie auch dann problemlos weiterarbeiten können, wenn Sie keinen Zugriff auf das Repository haben. Das ist bei SVN nur sehr beschränkt möglich.

Übrigens ist Git ein SCM, das von Linus Torvalds, dem Initiator und höchsten Entwickler von **Linux** angestoßen wurde. Mehr dazu auf `linux.com` und `git-scm.com`

Kontrolle
Ja, Sie können im stillen Kämmerlein vor sich hin arbeiten, aber das Thema Versionskontrolle müssen Sie im Hinterkopf haben. (Und das Thema Dokumentation natürlich erst recht.)

1.2.7 Software Engineering / Softwareentwicklung

Und wieder folgt ein Begriff, der Ihnen in Fleisch und Blut übergehen muss: **Software Engineering**, was als **Softwareentwicklung** übersetzt wird. Die Idee hier besteht im Gegensatz zu den Paradigmen weniger darin, dass Sie bestimmte Konzepte nutzen, um eine Aufgabenstellung zu lösen, sondern es geht um die Arbeitsteilung bei der Entwicklung eines großen Projekts. Die ursprüngliche Bedeutung des Begriffs lässt sich mit „Sicherstellung hoher Qualität von Softwareßusammenfassen, doch wenn wir uns damit beschäftigen, sind wir wieder einmal im Bereich der **Praktischen Informatik**.

Wo es bei den Paradigmen um die Frage geht, wie wir eine möglichst optimale Lösung für unser Problem erreichen, stehen beim Software Engineering entsprechende Fragen im Mittelpunkt: Was will der Kunde eigentlich?

Wie lange brauchen wir dafür? Können wir das überhaupt anbieten? Was wollen wir dafür berechnen? Wie oft müssen wir mit dem Kunden Besprechungstermine vereinbaren? Usw. usf.

Streng genommen handelt es sich hier also um eine Kernkompetenz der Betriebswirtschaftslehre (BWL), die als Projektmanagement bezeichnet wird. Aber im Gegensatz zum **Projektmanagement** der BWL haben wir Werkzeuge zur Verfügung, mit denen wir räumlich getrennt gemeinsam an einer Software arbeiten können.

Einer der neuesten Vertreter dieser Spezies wird als **agil**e Softwareentwicklung bezeichnet. Ältere Vertreter laufen unter Namen wie **Wasserfallmodell** und **V-Modell**. Aber keine Sorge, die Feinheiten des Software Engineering lernen Sie erst später im Studium kennen.

Kontrolle
Sie sollten wissen, dass es beim Software Engineering um Methoden geht, um Teamarbeit bei Softwareprojekten zu koordinieren. Dadurch wird eine hohe Qualität von Software in großen Projekten sichergestellt. Um hohe Qualität von Software geht es zwar auch bei der Praktischen Informatik, doch was Sie dort lernen können nützt Ihnen individuell bei der Entwicklung hochwertiger Software. Als letztes werfen wir nochmal kurz einen Blick auf die Programmierung: Um qualitativ hochwertige Software zu entwickeln müssen Sie einige Grundlagen verschiedener Programmierparadigmen lernen. Wenn Sie sehr gut in der Praktischen Informatik sind, ist es weitgehend belanglos, wie gut Sie eine bestimmte Programmiersprache beherrschen: Sie werden gute Software entwickln können. Wenn Sie dagegen kaum etwas von Praktischer Informatik verstehen, dann werden Sie selbst in Sprachen, die Sie sehr gut beherrschen bestenfalls mittelmäßige Software entwickeln.

1.2.8 Für diejenigen, die die Programmierung von Apps erlernen wollen

Der Begriff **App** ist im Grunde absurd. Eine App ist nichts anderes als ein Kunstwort, das geschaffen wurde, um Menschen, die nichts von Informatik oder Programmierung verstehen den Eindruck vorzugaukeln, dass es sich hier um eine besonders moderne oder hochwertige Anwendung handelt. Tatsächlich ist das Gegenteil der Fall: Der Begriff App ist vom Begriff Application abgeleitet, was nichts anderes als **Anwendung** heißt. Eine Anwendung ist ein Programm, das für die Nutzung durch Menschen gedacht ist. Zwar wird in Bezug auf Anwendungen für mobile Endgeräte regelmäßig von Apps gesprochen, aber es gibt nichts und zwar überhaupt rein gar nichts, was an der Entwicklung solcher Anwendungen anders wäre als bei

der Entwicklung von Anwendungen für andere Endgeräte.

Im Gegensatz zu anderen Anwendungen sind Apps jedoch immer sytemabhängig; Sie können also keine App entwickeln, die auf iPhone und Android unverändert lauffähig ist: Für jedes System sind umfangreiche Anpassungen nötig, die einzig deshalb nötig sind, weil die Entwickler der Geräte die Softwareentwickler davon abhalten wollen, für mehrere Systeme zu entwickeln. Gerade wenn Sie sich die Entstehungsgeschichte von Java ansehen werden Sie feststellen, dass die Systemabhängigkeit bei der App-Entwicklung mit Java (beispielsweise bei Android) geradezu absurd ist.

Wer die Entwicklung von Apps lernen will zeigt damit also im Regelfall, dass er/sie so viel von Softwareentwicklung versteht wie jemand, der glaubt, dass das Rauchen von Zigaretten, der Konsum von Alkohol bzw. Drogen oder der Kauf von Waren auf Ratenzahlung etwas mit Freiheit zu tun hätte: Er/Sie ist auf einen Werbeslogan hereingefallen und bedankt sich noch dafür, dass hohe Kosten für etwas zu zahlen sind, das das Versprechen nicht einhält und meist noch andere Kosten nach sich zieht.

Leider müssen wir diesen Begriff aber im Regelfall nutzen, weil Konsumenten und nicht-Informatiker auf genau dieses Marketingkonstrukt hereinfallen und gar nicht begreifen, wie unsinnig der Begriff App ist. Da diese Menschen uns für unsere Arbeit bezahlen müssen wir uns hier quasi dumm stellen und die „asiatische Lösung“ wählen: Immer schön nicken und lächeln. Womit wir bei einem der Gründe wären, warum viele IT'ler und NT'ler im Servicebereich regelmäßig schlechte Laune haben, wenn sie mit nicht-IT'lern über IT reden (müssen): Sie werden von Nutzern damit konfrontiert, dass die ach so tollen Apps eben längst nicht das leisten, was die NutzerInnen sich aufgrund der Marketingbotschaften einbilden.

Schauen Sie sich dazu die Geschichte von Apple in den letzten zwanzig Jahren an, denn die ist direkt mit der Einführung des Begriffs der App verbunden: Gegen Ende der 90er stand Apple kurz vor dem Konkurs. Also entwickelten Jobbs & Co. mobile Endgeräte, für die im Ein- bis Zweijahresrhythmus Nachfolger mit geringfügigen Verbesserungen bezüglich der Funktionalität erschienen. Vielmehr wurden hier jeweils signifikante ästhetische Änderungen durchgeführt, die Käufern den Eindruck vermittelten, die Nutzung dieser Geräte sei gleichbedeutend damit, zur technologisch-gesellschaftlichen Elite zu gehören. Gleichzeitig wurde wie in der Mode der Eindruck vermittelt, dass nur die jeweils aktuelle Baureihe genügen würde, um diesen vermeintlichen Status beizubehalten. Als dann nach einigen Baureihen absehbar war, dass dieses Vorgehen beim iPod nicht mehr die nötige Kundenbindung erzeugen würde, wurde mit

dem iPhone bzw. dem iPad eine Kombination von iPod, Mac und Handy entwickelt, bei der die gleiche Strategie erfolgreich verfolgt wurde.

Wenn Sie über den Begriff **mobiles Endgerät** irritiert sind: Damit werden vorrangig in der Nachrichtentechnik all die Geräte bezeichnet, die an ein Netzwerk angeschlossen sind, die aber relativ frei bewegt werden können. Es geht hier also um Handys, Smartphones, Laptops mit WLAN, usw. Denn auch wenn die Nutzung eines Smartphones mit „Internet"-zugang für Sie genauso selbstverständlich sein dürfte wie die Nutzung eines Rechners, der per Kabel ans Internet angeschlosse ist, ist die Technik und Technologie, durch die insbesondere kabllose Anschlüsse realisiert werden alles andere als simpel. LTE ist dabei der erste erfolgreiche Versuch, verschiedene Arten von Vernetzung zu kombinieren.

Mittlerweile ist aber auch im Bereich von Smartphones die maximale Ausreizung des Marktes erreicht. Deshalb wurden inzwischen die **Wearables** entwickelt: Kleine Geräte, die nur in Verbindung mit der aktuellsten Baureihe eines bestimmten Smartphones nutzbar sind. Während bislang die Ästhetik der Geräte selbst im Vordergrund stand, um eine vermeintlich elitäre gesellschaftliche Position und modisches Bewusstsein der BesitzerInnen zu vermitteln, steht bei den Wearables das Konzept im Vordergrund, KäuferInnen würden durch das sichtbare Tragen dieser Geräte zusätzlich sportliche Fitness, Flexibilität und Belastbarkeit beweisen. Mittlerweile steht Apple hier allerdings in direkter Konkurrenz zu Samsung. Die Koreaner haben dabei den großen Vorteil, dass Sie aus dem Land kommen, das weltweit seit Jahren die modernste Internetinfrastruktur besitzt. Deutschland ist in dieser Hinsicht selbst im EU-weiten Vergleich bestenfalls Mittelmaß, was ein essentieller Grund dafür sein dürfte, dass Siemens schon vor Jahren aus der Entwicklung von mobilen Endgeräten ausgestiegen ist.

Es sei an dieser Stelle allerdings noch angemerkt, dass Geräte von Apple häufig tatsächlich einen funktionalen Mehrwert gegenüber denen der Konkurrenz besitzen. Nur wird dieser Mehrwert eben von vielen Kunden gar nicht genutzt: Beispielsweise sind die Schnittstellen derart hochwertig, dass die Verbindung mehrerer Geräte im Regelfall problemlos ohne Zutun von Nutzern funktioniert. Microsoft versucht seit einigen Jahren ebenfalls in diesen Marktbereich einzudringen, um zur Konkurrenz im Bereich mobiler Endgeräte aufzuschließen. Das neueste Surface und Nokias neuste Modelle der Lumia-Reihe stellen in Verbindung mit dem Marketingkonzept rund um Windows 10 und dem augmented reality Sytem HoloLens den ersten erfolgreichen Versuch von Microsoft dar, die Grenzen zwischen verschiedenen Systemen aufzubrechen und neue Systemtypen ins Angebot zu integrieren. Wenn Sie sich dann allerdings die Situation des Herstellers von Blackberry Smartphones ansehen oder die Gründe für Apples wirt-

schaftlichen Einbruch in den 90er Jahren, werden Sie feststellen, dass eine Konzentration auf hochwertige Geräte für den Einsatz im professionellen Bereich wahrscheinlich keine Basis für dauerhaften wirtschaftlichen Erfolg darstellt.

Nun fragen Sie sich vielleicht, was all das mit einem Hochschulkurs zur Programmierung zu tun hat. Die Antwort ist simpel: Nach Ihrem Studium (egal ob Sie nach dem Bachelor noch einen Master und ggf. eine Doktorarbeit anfertigen oder direkt ins Berufsleben bzw. die Forschung starten) werden Sie entweder für einen Arbeitgeber tätig werden oder selbständig sein. Und wo auch immer das sein wird, gilt eine grundsätzliche Tatsache: Wenn Sie Software (oder auch Hardware) entwickeln, dann hängt Ihre Zukunft davon ab, ob diese Software tatsächlich deutlich mehr Geld einbringt, als Ihre Entwicklung gekostet hat. Schließlich müssen kontinuierlich neue Hardware angeschafft, Lizenzen erworben, Miete gezahlt werden usw. Das soll kein Appell gegen Indie Games oder Open Source sein, sondern es geht hier um Faktoren, die Ihre Zukunft bestimmen werden. Denn egal wohin Sie im Anschluss gehen, müssen Sie sich bewusst sein, dass Geld nicht einfach so entsteht, wie beispielsweise die Planer von Projekten wie der HafenCity, der Elbphilharmonie, von Stuttgard 21, Flughafen BER usw. usf. denken, sondern dass Sie nur dann dauerhaft ein zumindest ausreichendes Einkommen erreichen, wenn Sie Software entwickeln, die sowohl bestimmte Qualitätsstandards erreicht als auch von einer gewissen Anzahl von Kunden gekauft wird. Sollten Sie also in einem Unternehmen tätig werden, in dem entweder die Qualität der Software oder die Änderungen im Umfeld ignoriert werden, dann sollten Sie sich schnell nach einer neuen Aufgabe umsehen. Diese ignorante Strategie des Managements kann einige Jahre funktionieren, aber früher oder später wird sie das nicht mehr tun. Und ja, das gilt auch für den öffentlichen Dienst wie beispielsweise in öffentlichen Hochschulen, auch wenn wir hier nicht über wenige Jahre sprechen, sondern über Zeiträume, die eher in Richtung von zehn bis zwanzig Jahren gehen.

Zwei konkrete Beispiele möchte ich an dieser Stelle nennen: Zynga galt in den ersten Jahren, in denen Facebook international erfolgreich wurde als das erfolgreichste Unternehmen im Bereich der Browsergames. Werfen Sie dazu einen Blick in die Making Games-Ausgaben von 2012. Beispielsweise stammt das Spiel Farmville von diesem Entwickler. Zynga existiert zwar noch, aber mittlerweile hat das Unternehmen vier Fünftel seines Wertes verloren und viele Mitarbeiter wurden entlassen. Noch schlimmer sieht es beim ursprünglich größten Konkurrenten von Blizzard-Activision im Bereich MMORPGs aus: Sony Online Entertainment (kurz SOE) war u.a. mit Everquest II einer der erfolgreichste Entwickler von MMORPGs bis World of Warcraft erschien. Bei nachfolgenden Titeln versuchte SOE dann stets be-

sonders hohe Qualitätsmaßstäbe zu setzen, was regelmäßig misslang. Im Februar 2015 wurde das Unternehmen an einen Finanzdienstleister verkauft und existiert heute unter dem Namen Daybreak Game Company. In beiden Fällen haben Sie es mit Unternehmen zu tun, die eine Zeitlang sehr erfolgreich waren, die es aber nicht geschafft haben, sich den Änderungen des Marktes anzupassen. Und die Auswirkung bestand darin, dass viele Mitarbeiter arbeitslos wurden.

Umgekehrt gibt es aber auch Unternehmen, die eine ausgewählte Kundengruppe bedienen, damit kontinuierlichen Erfolg haben aber nur gelegentlich in Spieletestzeitschriften auftauchen. Der isländische Entwickler CCP-Games beispielsweise betreibt seit mehr als 10 Jahren das MMORPG Eve online, das seit Jahren kontinuierlich von mehr als 300.000 Kunden monatlich bezahlt wird. Zwar gab es hier durchaus Einbrüche bei den Nutzerzahlen, doch das Unternehmen reagierte auf Kritik und konnte so weiterhin bestehen.

1.3 Informatik versus Programmierung, Studium und Arbeit

Nachdem Sie jetzt einen ersten Einblick in einige Bereich bekommen haben, die bei der Programmierung eine Rolle spielen, schauen wir uns jetzt an, wo Informatik und Programmierung zusammenhängen.

1.3.1 Informatik und Programmierung

InformatikerInnen oder der Informatik nahe stehende Akademiker sind im Stande, ein Problem unabhängig von einer bestimmten Programmiersprache auszuformulieren und sich für ein Paradigma entscheiden, mit dem sie dann das Problem lösen können.

Der Bereich der **Praktischen Informatik** widmet sich u.a. der Frage, wie Sie das Problem effizient lösen können und hat mit der **Programmierung**, also der Umsetzung dieser Lösung in einer Programmiersprache nichts zu tun. Wenn Sie zusätzlich die Grundlagen der **Theoretischen Informatik** gemeistert haben, können sie außerdem erkennen, ob ein Problem überhaupt lösbar ist. Quereinsteiger versuchen dagegen häufig Programme zu entwickeln, die gar nicht lösbar sind. Und erst wenn sie wissen, dass und mit welchen Mitteln ein Problem idealerweise lösbar ist, entwickeln sie eine Lösung und setzen diese dann in einer gut geeigneten Sprache um. Außerdem sind sie im Stande, mit Dutzenden oder Hunderten anderer InformatikerInnen und ProgrammiererInnen gemeinsam Software zu entwickeln.

Auch hier nochmal der Hinweis: NaturwissenschaftlerInnen, TechnikerInnen und IngenieurInnen kennen diese Unterteilung in aller Regel nicht. Sie erlernen das Programmieren in einer oder mehrerer Sprachen, die für Ihren Bereich voll und ganz ausreichen. Dazu lernen Sie meist noch die Grundlagen dessen, was InformatikerInnen als **Technische Informatik** zusammenfassen, gehen aber davon aus, dass es sich hier um Informatik als ganzes handelt. Der Austausch mit diesen INT-AkademikerInnen ist deshalb meist schwierig: Sie begreifen oftmals nicht, dass sie die beiden Bereiche der Praktischen und Theoretischen Informatik mit Ihren Kenntnissen nicht erfassen können und gehen zusätzlich davon aus, dass Praktische Informatik dasselbe wie Programmieren ist. Und das ist nicht einmal ansatzweise richtig.

ProgrammiererInnen können all das nicht oder nur zu einem geringen Teil. Sie beherrschen eine oder mehrere Programmiersprachen und quetschen eine Lösung in diese Sprachen, egal ob das Ergebnis überhaupt brauchbar ist oder nicht. Manches, was Media Systems Studierende schon im Studium kennen lernen erkennen reine Programmierer erst nach Jahren oder Jahrzehnten. Aber unterschätzen Sie sie nicht: Dafür kennen ProgrammiererInnen häufig Kniffe, die man eben erst dann kennen lernt, wenn man eine Programmiersprache in- und auswendig gelernt hat. Und das ist immer wieder sehr wertvoll.

Leider werden InformatikerInnen und ProgrammiererInnen auch häufig deshalb in einen Topf geworfen, weil es den Ausbildungsberuf zum/zur „FachinformatikerIn“ gibt. Dort liegt der Fokus im Gegensatz zum Informatikstudium klar auf der Programmierung. Im Anglo-Amerikanischen Raum gibt es noch den Studiengang Computer Science, der einem FH-Studium der Informatik hierzulande ähnelt.

Sie können sich den Unterschied zwischen Informatik und Programmierung auch dadurch veranschaulichen, dass Sie an Bauarbeiter und Bauingenieure denken: Ein Bauingenieur weiß, wie ein von ihm geplantes Gebäude errichtet werden muss, damit es z.B. nicht unter dem eigenen Gewicht zusammen bricht. Aber er wird im Regelfall viele Arbeitsschritte nicht selbst durchführen können. Der Bauarbeiter dagegen wird sehr genau wissen, wie die Arbeitsschritte richtig auszuführen sind. Dafür kennt er sich beispielsweise nicht mit den Kräften aus, die bei einem modernen Hochhaus auftreten. Sie beide haben unterschiedliche Fähigkeiten, aber nur gemeinsam können sie großes erreichen.

1.3.2 Informatik: Uni versus HAW (FH)

Im Gegensatz zu Ihnen beschäftigen sich Universitätsstudierende in der Informatik praktisch durchgehend mit der Frage, wie eine bestimmte Aussage zu beweisen ist. Diese Fragestellung ist spannend, aber es geht hier um rein abstrakte Konzepte, die bei höchst komplexen Projekten massive Effizienzsteigerung bedeutet.

Deshalb sollten Sie sich bewusst machen: Gute UniversitätsabsolventInnen sind Ihnen im Regelfall deutlich überlegen, wenn Sie es bei einer Aufgabenstellung mit sehr großen Datenmengen zu tun haben.

1.3.3 Informatik und Programmieren im Beruf

Schon beim Studienbeginn fragen viele Studierende, wie denn Ihre Aussicht auf dem Arbeitsmarkt ist. Hier gibt es zwei grundsätzliche Varianten: Zum einen können Sie nach dem Bachelor-Abschluss ein Masterstudium anstreben. Das ist gerade in der Elektrotechnik und der Informatik kein allzu großes Problem, weil hier viele Studienabsolventen bereits mit einem Bachelor-Abschluss spannende Stellen bekommen. Die Bewerbungssituation ist also entspannter, als beispielsweise bei den Sozialwissenschaften. Wenn Sie diesen Weg verfolgen, können Sie langfristig (ein Doktortitel ist da leider immer noch eine übliche Voraussetzung, wenn es Ihnen nicht genügt, als Assistent zu arbeiten) eine akademische Forschungskarriere anstreben.

Wenn Sie dagegen das Studium aufgenommen haben, um anschließend direkt in den Arbeitsmarkt zu starten, dann sollten Sie die Zeit neben dem Studium nutzen, um eigene Projekte zu realisieren. Denn das ist bei vielen Arbeitgebern eine gute Möglichkeit, um die eigenen Fähigkeiten nachzuweisen.

Doch das beantwortet natürlich nicht die Frage, welche Kenntnisse Arbeitgeber eigentlich erwarten, bzw. was Sie (neben dem Studium) an Kenntnissen erwerben müssen, um für eine bestimmte Tätigkeit gut vorbereitet zu sein. Dieser Abschnitt soll Ihnen da eine kleine Unterstützung bieten. Natürlich treffen die folgenden Aussagen nicht auf alle Arbeitgeber zu und wie die Situation in drei Jahren aussieht (also zu dem Zeitpunkt, zu dem Sie das Ende Ihres Studiums anstreben), ist noch eine andere Frage. Als Grundlage für die folgenden Aussagen dienten zwei Quellen: Zum einen aktuelle Stellenausschreibungen (August 2015), zum anderen Gespräche mit HR'lern im Hamburger Raum.

Die einfachste Antwort darauf, was von Ihnen häufig erwartet wird ist die

am wenigsten hilfreichste: Es wird von Ihnen erwartet, dass Sie **backend** developer oder **frontend** developer sind oder sonst eine Stelle ausfüllen können, die in Unternehmen zu besetzen ist. Das hat aber mit einem Studium (z.B. der Informatik) nichts zu tun, denn was Sie hier lernen (können) sind grundlegende Techniken und Methoden, die in beiden Bereichen angewendet werden. Leider gibt es immer wieder HR'ler (ja selbst in IT-Unternehmen), denen das nicht klar ist und die Sie dann mit Fragen konfrontieren, die Sie kaum beantworten können. Machen Sie sich in diesen Fällen nichts draus, wenn's mit der Stelle nichts wird; da gibt es bessere Angebote.

Die nächste Antwort hilft auch nicht weiter: Selbstverständlich sollen Sie... naja, so ziemlich alles können. Ums kurz zu machen, auch von solchen Unternehmen sollten Sie Abstand halten, denn dort wird von Ihnen im Grunde erwartet, dass Sie die Kenntnisse mitbringen, die Ihrem Vorgesetzten fehlen. Und glauben Sie mir, das wollen Sie nicht.

Kommen wir jetzt also zu hilfreichen Antworten.

Zunächst sollten Sie, nachdem Sie alle (!) Leistungsnachweise der ersten drei Semester erworben haben (ja, damit sind insbesondere Mathe 1 und 2 gemeint), sich ein wenig Zeit nehmen und prüfen, welche Bereiche Ihnen besonders liegen, bzw. mit welchen Konzepten der verschiedenen Veranstaltungen Sie am meisten anfangen konnten.

Hier ein Einwurf: Viele Studierende, die in Mathe 1 (oder einem ähnlich anspruchsvollen Fach) durchfallen, versuchen im nächsten Semester dennoch alle Veranstaltungen zu bestehen. Manche „schiebenäuch das, was noch nachzuholen ist in ein späteres Semester, weil Sie denken, dass das der passende Ansatz wäre. Beides hat wenig Aussicht auf Erfolg: Erledigen Sie zuerst das, was zuerst im Studienplan vorgesehen ist. Und wenn Sie dann zwei Veranstaltungen haben, zwischen denen Sie sich entscheiden müssen, dann entscheiden Sie sich für diejenige, die Sie für schwerer halten. Wenn Sie dann merken, dass Sie nirgends richtig vorankommen, dann streichen Sie die jeweils leichteste Veranstaltung, damit Sie für die übrigen Veranstaltungen mehr Zeit haben. Denn wenn Sie so agieren, haben Sie gute Aussicht darauf, mit nur einem zusätzlichen Semester einen guten Abschluss zu erlangen.

Jetzt also wieder zurück zur Situation, wenn Sie alles aus den ersten drei Semestern bestanden haben. Sie haben dann ein grundlegendes Verständnis für zwei Bereiche der Softwareentwicklung: FPGA-basiert und imperativ bzw. objektorientiert.

Wenn Sie sich jetzt im Bereich der maschinennahen Softwareentwicklung bzw. bei der Entwicklung von **FPGA**s wohler fühlen, dann sprechen Sie die Kollegen Edeler und Behrens an; diese können Ihnen die spannenden Möglichkeiten dieser Bereiche aufzeigen.

Wenn Sie sich dagegen eher bei **Desktoprechner**n und **Smartphone**s zu Hause fühlen, dann sind Sie im Bereich der Web- bzw. der Anwendungsentwicklung richtig. Sie können nun Spezialkenntnisse erwerben, um z.B. eine guter Entwickler für Android- oder iPhone-Apps zu werden. In dem Fall wäre Herr Plaß der Ansprechpartner Ihrer Wahl.

Um Software für mobile und/oder vernetzte Systeme entwickeln zu können, nutzen Sie die Kenntnisse aus PRG, P1 und P2 sowie Software Engineering, RDB, NWI und Kryptologie für Media Systems aus und setzen Sie diese Kenntnisse in Projekten um.

Sie wollen es genauer wissen? Gut: Aktuell (Herbst 2015) gibt es vorrangig Stellengesuche nach Leuten mit Kenntnissen in den folgenden Bereichen:

- Programmierung in **C** bzw. **C++**
  Diese Stellen sind meist eher etwas für reine Informatiker, da hier umfangreiche Kenntnisse im Bereich der Algorithmik nötig sind, die in Ihrem Studienplan leider vollständig fehlen, obwohl sie ein der Grundlagen jeder Informatik und damit auch der Medieninformatik sind.

- Programmierung in **.NET** und angrenzenden Bereichen
  Das sind Spezialisten, die sich in die Softwareentwicklung mit Sprachen vertieft haben, die von Microsoft entwickelt wurden und ausschließlich auf Windows-Rechnern und –Smartphones genutzt werden können.

- Programmierung in **Objective-C** oder **C#**
  Hier wird's schon interessanter, weil Sie zwar diese Sprachen in Ihrem Studium nicht kennen lernen, diese Sprachen aber eine sehr große Ähnlichkeit mit Java aufweisen.

- Programmierung in **Java für Server**
  Einerseits lernen Sie im Studium zwar Java, andererseits belegen Sie keine Kurse zu Betriebssystemen bzw. zur Serverwartung. Und genau das bräuchten Sie dafür.

- Programmierung in **HTML**, **CSS** und **PHP** oder **JavaScript**
  Hier sind Sie genau richtig, wenn Sie im Department Medientechnik der HAW Hamburg studieren, denn die Grundlagen erlernen Sie

im ersten Semester im Kurs "´Einführung ins Programmieren„ bzw. „Programmieren 1". Deshalb können Sie sie leicht ausbauen.

Bei den aktuellen Stellenanzeigen wird dann noch nach weiteren Kenntnissen gefragt. Damit Sie nachvollziehen können, was darunter verstanden wird und Sie in den noch folgenden Semestern die Möglichkeit haben, sich jeweils einzuarbeiten, folgen ein paar Erklärungen:

- **HTML5** und **CSS3**
  Zunächst sind das die beiden Sprachen, in denen Sie Webpages programmieren (HTML) und das Design festlegen (CSS). Allerdings ist noch hinzuzufügen, dass die Änderungen in HTML5 (gegenüber der Version 4) so umfangreich sind, dass es eigentlich eine komplett neue Programmiersprache ist. So ist vieles, was früher mit Hilfe von PHP oder JavaScript programmiert werden musste Teil von HTML5. Aber das ist nur ein kleiner Teil von HTML5, der im Grunde auch als HTML 4.1 hätte veröffentlicht werden können. Dagegen beinhaltet die Version 5 mit den sogenannten **Microdata** eine umfangreiche und standardisierte Semantik, womit es eine der ersten vollwertigen semantischen Programmiersprachen sein dürfte. Das ist gleichbedeutend mit derart fundamentalen Änderungen bei der Entwicklung von Anwendungen, das die Auswirkungen ähnlich wie beim WWW erst in zehn bis zwanzig Jahren spürbar werden dürften. Aber sie dürften ähnlich weite Kreise ziehen wie die Entwicklung des WWW selbst.

  Im Gegensatz dazu ist CSS nichts anderes als eine Sammlung von Befehlen, mit denen sich die Anzeige von Elementen innerhalb einer Webanwendung anpassen lässt. Es gibt dort sicher einiges, was Sie sich ansehen könnten, aber für MedieninformatikerInnen ist es ungefähr so wichtig, die Feinheiten der CSS-Programmierung zu beherrschen wie die genaue Bestimmung der Farbe der eigenen Tastatur. Sie halten das für vollkommen überflüssig und fragen sich, warum CSS dann überhaupt erwähnt wird? Na dann willkommen im Club: Der einzige Grund, sich als MedieninformatikerIn mit CSS zu beschäftigen ist der, dass es MediendesignerInnen gibt, die nicht einmal im Stande sind, eine Definition für einen Farbwert z.B. RGB-Werte in einen Computer einzugeben. Und dann wird diese Aufgabe eben an MedieninformatikerInnen delegiert...

  Wichtig
  Wenn ein Arbeitgeber explizit nach HTML und CSS fragt, aber sonst kaum nach Kenntnissen, die in dieser Aufstellung auftauchen, dann geht es in aller Regel um eine Stelle als Webdesigner und nicht um eine Stelle als Webdeveloper. (Letzteres wäre Ihr Gebiet, für ersteres sind Sie im falschen Studiengang.)

- **CSS Präprozessoren**
  Dieses Schlüsselwort weist eindeutig auf eine Stelle für Webdesigner hin.

  Beispiele: **Twig**, **Gulp**, **SASS**, **LESS**.

- **XHTML**
  ist eine Kombinaion aus HTML und XML. Diese Kombination ist im Grunde durch HTML5 und Microdata oder andere Formen des semantic web in den meisten (aber nicht allen Bereichen) überflüssig geworden. Über das semantic web reden wir gleich in der ersten Veranstaltung des Kurses „Einführung in die Programmierung"´.

- **JavaScript** oder **PHP**
  Für die Einarbeitung in PHP sollten Sie nicht allzu lange brauchen, wenn Sie eine andere imperative Sprache (wie Java) beherrschen, weil PHP nur wenige Konzepte der imperativen Programmierung umsetzt.

  Bei JavaScript sieht das anders aus: Der Einstieg ist zwar nicht schwer, aber um gute Software zu entwickeln (was in PHP eher nicht möglich ist), werden Sie schon ein Jahr Übungszeit brauchen. Das liegt auch daran, dass JavaScript u.a. das funktionale Programmierparadigma umsetzt. Entwickler, die nur die klassenbasierte Objektorientierung (z.B. aus Java) kennen sind damit in aller Regel überfordert, ohne es zu merken. Aufgrund der Möglichkeiten, die JavaScript bietet und sein Status als Standard-Programmiersprache für dynamische HTML5-Seiten, dürfte PHP in zehn Jahren kaum mehr von Belang sein, wenn die Entwickler der Sprache hier keine fundamentalen Änderungen einführen. Leider sieht es zurzeit nicht so aus, als wenn das bei PHP7 der Fall wäre. `https://github.com/php/php-src/blob/php-7.0.0RC1/UPGRADING` Es scheint so, als wenn hier lediglich Neuerungen eingeführt werden würden, die in anderen Sprachen wie Ruby längst üblich sind oder die einfach selbst programmiert werden können. (Stichworte: UTF-8-Unterstützung oder die Einführung eines expliziten Spaceship-Operators)

- **ECMAScript**
  ist eine standardisierte Version u.a. von JavaScript. Wenn Sie JavaScript beherrschen, sollten Sie sich hier relativ schnell zurecht finden. Umgekehrt gilt dasselbe.

- **JavaScript ... frameworks**
  Beispiele: **AngularJS**, **Backbone**, **EmberJS**, **Grunt**, **jQuery**, **requireJS**, **AJAX**.

- **PHP ... frameworks**
  Im Falle des **Zend** framework handelt es sich bei Zend nicht um eine kleine Erweiterung von PHP, sondern um ein sehr mächtiges Werkzeug, das eine längere Einarbeitungszeit benötigt. Dieses setzt u.a. voraus, dass Sie PHP objektorientiert programmieren können.

- **Flash** und **ActionScript**
  sind praktisch dasselbe. ActionScript ist eine Konkurrenzsprachen zu JavaScript, deren Rechte bei Adobe liegen. Da JavaScript als Standardsprache für dynamische Webpages in HTML5 festgelegt wurde und es im Gegensatz zu ActionScript kein Unternehmen gibt, das hier Rechte beanspruchen würde, dürfte ActionScript ähnlich wie PHP in zehn Jahren nur mehr eine Nischensprache sein.

  Die Abkürzung AS wird teilweise für ActionScript verwendet, aber nicht nur dafür: Im Bereich der Programmierung von **SPS**en gibt es eine Programmiersprache namens Ablaufsprache, die ebenfalls mit **AS** abgekürzt wird. Im Englischen wird diese Sprache als SFC / Sequencial Function Chart bezeichnet. ActionScript und Ablaufsprache haben so viel miteinander gemein wie Michael Schuhmacher mit einem Chinesen, der Tai Chi macht.

- **Frontend** und **BackendBackend**
  stehen für zwei Bereiche, die bei PHP noch strikt getrennt waren. (Auf die Details kommen wir bei der Einführung in PHP zu sprechen.) Meist ist mit Frontend die Programmierung des View gemeint (auch dazu kommen wir bald) und damit geht es dann in aller Regel um eine Aufgabe für Webdesigner. Das muss aber nicht so sein; sehen Sie sich ggf. die Stellenausschreibung genau an und fragen Sie beim Arbeitgeber nach. Beim Backenddevelopment haben Sie dagegen nie etwas mit Gestaltung zu tun. Als Studierende der Medieninformatik wäre das also der Bereich, in dem Sie vorrangig tätig werden. Für die Entwicklung des Frontends brauchen Sie immer zumindest eineN MediendesignerIn mit Grundkenntnissen der Typographie, da nur diese ein wenigstens ausreichendes Verständnis für die Darstellung medialer Inhalte haben. Hier nochmal der entsprechende Hinweis: Medieninformatik und Mediendesign arbeiten zwar beide im weitesten Sinne mit Medien, aber ersteres ist eine Spezialisierung der Informatik, letzteres eine Spezialisierung des Designs. Und wer aus dem einen Bereich kommt, kann bestenfalls verstehen, worüber die SpezialistInnen des anderen Bereichs reden. Mehr nicht!

  Grundsätzlich ist die Trennung in Front- und Backend aber recht

willkürlich, weil damit suggeriert wird, es gäbe eine klare Trennung, wo eigentlich keine ist. Denn an welcher Stelle (auf dem Rechner eines Nutzers oder auf einem Server im Netz) Sie bei einem Softwareprojekt welche Funktionalitäten und Komponenten realisieren, ist Ihre Entscheidung.

- **Versionsverwaltung**
  (siehe Abschnitt „SCM / Versionskontrolle“)

- **Continuous Integration**
  ist ein relativ neues aber sehr mächtiges Konzept, wird teilweise mit CI abgekürzt, was aber leider auch für andere Begriffe stehen kann. Im Grunde ist es eine massive Erweiterung der Versionsverwaltung. Denn hier werden auch Tests und anderes in die Entwicklung eingeführt, das Sie später in der Veranstaltung Software Engineering kennen lernen.

  Beispiel: **Jenkins**

- **Continuous Deployment**
  ist dann gewissermaßen die aktuelle Luxusklasse fürs Software Engineering. Denn hier wird eine Software auch nach der Auslieferung an Kunden kontinuierlich weiter entwickelt. An bestimmten Punkten werden dann die Neuerungen an den Kunden ausgeliefert, um beispielsweise Fehler zu bereinigen (**Patch**en) oder neue Funktionalitäten in die Software einzuführen.

- **Responsive Design**
  hat streng genommen nichts mit Design zu tun. Hier geht es darum, Webanwendungen zu entwickeln, die auf den unterschiedlichsten Displays nutzbar sind. Dieses Thema werden wir in in „Programmieren 1”´ (für Medientechnik) bzw. „Einführung in die Programmierung”´ (für Media Systems) kurz behandeln. Es gibt eine Vielzahl an JavaScript Frameworks, die Ihnen hier viel Arbeit abnehmen, die aber leider fast ausschließlich für HTML 4.01 gedacht sind.

- **Strukturiertes Arbeiten**
  Ein Arbeitgeber, der danach fragt ist für Sie die passende Wahl... außer wenn Sie eigentlich im falschen Studium gelandet sind.

- **UX / UI (User Experience, User Interface)**
  sind Begriffe, die in den Bereich Webdesign und **Usability** fallen.

- **Photoshop**
  gehört ins Webdesign.

- **Design Patterns**
  Seitdem die objektorientierte Programmierung in vielen professionellen Unternehmen genutzt wird, haben sich einige de-facto Standards ausgebildet, die die Arbeit im Team deutlich erleichtern. Damit werden Sie sich in der Veranstaltung Software Engineering auseinander setzen. Eines davon werden Sie aber schon in dieser Veranstaltung kennen lernen.

  Beispiel: **MVC**

- **Agile Softwareentwicklung**
  setzt objektorientierte Softwareentwicklung voraus und ist ein aktuelles Buzz Word: Viele Menschen benutzen es (in der Softwareentwicklung) aber leider gibt es hier wie bei der Objektorientierung eine Reihe von Missverständnissen. So gibt es ein Unternehmen, in dem ernsthaft versucht wird, agile Softwareentwicklung zu nutzen ohne dabei zentrale Aspekte der Objektorientierung zu verwenden. Und das ist unmöglich. Auch dieses Konzept lernen Sie in der Veranstaltung Software Engineering kennen.

  Beispiele: **TDD** (Test-Driven-Development), **Iterationen** im Projektmanagement, **SCRUM**

- **Extreme Programming**
  diese Art der Softwareentwicklung geht in eine andere Richtung als die bislang besprochenen Varianten. Hier wird im Regelfall auf eine strukturierte Vorgehensweise verzichtet, wenn dadurch Kundenwünsche so schnell wie möglich integriert werden können.

  Schlagwörter: **YAGNI**

- **Skalierbarkeit**
  wenn dieser Begriff auftaucht, geht es um die Entwicklung von Anwendungen oder Systemen, die möglichst gut damit umgehen sollen, wenn die Anzahl Nutzer oder Daten, die jeweils auf die Anwendung oder das System zugreifen oder von diesem genutzt werden zum Teil innerhalb kurzer Zeit stark ansteigen oder abfallen können bzw. stark schwanken. Dieser Bereich kommt für Sie leider nicht in Frage, weil Sie dafür mindestens die Veranstaltungen „**Algorithmen und Datenstrukturen**" sowie „**Algorithmendesign**" (**Praktische Informatik** 1 und 2) erfolgreich abgeschlossen haben müssen, die in unserem Department nicht auf dem Lernplan stehen.

Hier noch ein paar Begriffe, die jeder Arbeitgeber in diesem Bereich erwartet, weshalb sie eigentlich überflüssig sind:

- Motivation
- Belastbarkeit
- Teamfähigkeit
- Kenntnis von Webstandards

Und nun noch ein paar Begriffe, bei denen Sie von einer Bewerbung absehen sollten, wenn Sie als Softwareentwickler ins Webdevelopment gehen wollen: So wie es wenig Sinn macht, Webdevelopment und Webdesign von einer Person durchführen zu lassen, macht es auch keinen Sinn, Webdevelopment und Serveradministration von einer Person durchführen zu lassen: Entweder ist diese Person sehr fähig (und damit sehr teuer) oder beherrscht nur einen der beiden Bereiche und das wäre sehr problematisch:

- **Linux-/UNIX-Administration**
  Nicht zu verwechseln mit Linux-/UNIX-Kenntnissen; die müssen Sie bis zum Ende des Studiums zumindest grundlegend erworben haben, wenn Sie Media Systems studieren. Medientechnikstudierende, die in irgend einer Form mit Rechnern arbeiten wollen, sollten hier aber zumindest lernen, wie sie Befehle über die sogenannte Konsole eingeben können.
- **Konfiguration von ...-Servern (z.B. Linux, Apache)** wird gefordert.
  Die Administration von Servern hat nichts mit Softwareentwicklung zu tun, wie Sie sie im Rahmen von Veranstaltungen wie Programmieren 1 und 2 kennen lernen. Hier geht es vielmehr um Aspekte der Rechteverwaltung, Abwehr von netzbasierten Angriffen durch konkurrierende Unternehmen oder das organisierte Verbrechen und ähnliches. Wenn ein Unternehmen also jemanden sucht, der sowohl als Webdeveloper als auch als Serveradministrator fähig ist, dann herrschen dort offenkundig sehr große Wissensdefizite vor oder die Bezahlung ist entsprechend der geforderten Kenntnisse. Das würde dann aber bedeuten, dass Sie erst mit mehrjähriger Praxis in diesem Bereich die nötigen Kenntnisse erworben haben werden.
- Formulierungen wie **„Gängige Webtechnologien wie HTML, CSS und ... sind Ihnen nicht fremd"´** weisen eher darauf hin, dass dem zuständigen Mitarbeiter des Unternehmens diese Programmiersprachen und die ihnen zugrunde liegenden Konzepte sehr fremd sind.
- Ebenfalls abzuraten ist von Anzeigen, in denen Ihre Aufgaben wie folgt beschrieben werden: **„Development of the next generation of ..."´** Denn wenn Sie das wirklich könnten, dann sollten Sie sich lieber einen Investor suchen, der Sie bei der Umsetzung Ihrer Ideen

unterstützt. In 99% aller Fälle geht es hier um eine Unternehmen, dessen „Fachkräfte"derart wenig über die Entwicklungen im Bereich der Softwareentwicklung wissen, dass sie glauben, eine abgerundete Ecke sei bereits eine historisch bedeutsame Produktentwicklung. (Nichts gegen abgerundete Ecken... Die sehen schick aus... findet jedenfalls der Autor dieses Buches.)

- Problematisch ist es, wenn bei Programmiersprachen keine Versionsnummern angegeben werden. Vielleicht kennen Sie genau die geforderte Version nicht, aber darüber lässt sich im Bewerbungsgespräch immer reden. Dagegen macht es beispielsweise einen großen Unterschied, ob Sie tatsächlich HTML5 oder eigentlich nur HTML4.01 beherrschen. Und umgekehrt ist es auch für Ihre berufliche Zukunft relevant, ob Sie für Arbeitgeber tätig werden, dessen SoftwareentwicklerInnen diesen Unterschied kennen oder eben nicht.

- Ebenfalls kritisch ist es, wenn einerseits nach einem „Junior ... Developer"´ gesucht wird, andererseits aber eine sehr umfangreiche Palette an Kenntnissen gefordert wird. Denn einerseits wird damit angegeben, dass ein Einsteiger gesucht wird, der also gerade erst seinen Abschluss gemacht hat, andererseits werden derart viele Kenntnisse gefordert, dass im Grunde drei Jahre Berufserfahrung dafür nötig sind.

Kommen wir nun zu einem Bereich, den Sie schlicht deshalb ignorieren sollten, weil das eine Spezialdisziplin für Wirtschaftsinformatiker bzw. Informatiker mit mehrjähriger Praxiserfahrung in Unternehmen einer Branche ist:

- **ERP** (kurz für **Enterprise Resource Planning)**
  umfasst die verschiedensten Programme, die in Unternehmen zum Einsatz kommen. Hier ist neben der Kenntnis der jeweiligen Software ein detailliertes Verständnis für Unternehmensstrukturen und die Feinheiten der Branche nötig, in dem das Unternehmen tätig ist.

  Beispiele: **SAP**, **Microsoft Dynamics**

1.4 Zusammenfassung

Nach diesem Kapitel wissen Sie, dass es eine Vielzahl von Möglichkeiten gibt, den Begriff des Programmierens zu verstehen, und dass InformatikerInnen den Begriff des Paradigmas nutzen, um zwischen diesen Möglichkeiten zu unterscheiden. Einige davon sind Teil dieses Buches. (Deshalb ja

auch der Titel.) Viele andere werden Sie in den nächsten Jahren kennen lernen, als Teil Ihres Studiums oder auch bei anderer Gelegenheit.

Sie wissen, dass es neben reinen Programmiersprachen Dinge wie IDEs, Bibliotheken, Frameworks und Middlewares gibt, die Ihnen einen Teil Ihrer Arbeit abnehmen.

Dann haben Sie etwas über Teamarbeit gehört, über die drei Teilbereiche der Informatik: Praktische, Theoretische und Technische Informatik. Und Sie haben insbesondere etwas darüber gehört, warum keines dieser drei Teilgebiete das gleiche ist wie Programmierung. Anschließend ging es um die Unterschiede zwischen den Inhalten der Informatik an Uni und Fachhochschule. Abgeschlossen wurde das ganze mit einem kurzen Überblick über den Arbeitsmarkt und die Begriffe, die Ihnen dort begegnen werden.

Was Sie jetzt aber noch nicht wissen ist, wie Sie mit all diesen Dinge umgehen sollen, bzw. wie Sie sie nutzen können. Keine Sorge, genau darum geht es ja in dieser Veranstaltung. Also sein Sie nicht frustriert, wenn Ihnen dieses Kapitel zu oberflächlich erscheint; es ist lediglich ein erster Einblick.

An dieser Stelle ein Anmerkung, die in der Wissenschaft (also auch in der Informatik bzw. in Ihrem Studium) immer und überall gilt: Alles wird kontinuierlich weiter entwickelt und wer glaubt, dass er ein fähiger Wissenschaftler ist, ohne sich in seinem Bereich ständig auf dem aktuellen Stand zu halten, der macht etwas falsch. Dementsprechend sollten Sie sich stets vor Augen halten: Was Sie hier lernen, ist Wissen, dass in wenigen Jahren so nicht mehr aktuell sein wird. Und Sie werden kontinuierlich prüfen müssen, welche Aspekte sich geändert haben. Denn sonst werden Sie in spätestens zehn Jahren nicht mehr verstehen, worüber in der Informatik geredet wird. Und das ist auch ein zentrales Element wissenschaftliches Arbeit: Hinterfragen und selbst prüfen, was man gehört hat und kontinuierlich die bestehenden Systeme verbessern.

Das ist übrigens eine der Besonderheiten der (Medien-)informatik: Einerseits ist sie so abstrakt, dass sie auf viele so abschreckend wirkt wie ein Gemälde von Pablo Picasso, andererseits in einer derart schnellen Veränderung, dass Kenntnisse zum Teil innerhalb von Monaten veraltert sind. Während also Ihre Kommilitonen z.B. in Elektrotechnik 1 und 2 Kenntnisse erlangen, die so noch in fünfzig Jahren nahezu gleich sein werden, müssen Sie als angehende (Medien-)informatikerInnen sich ständig mit den Änderungen auseinander setzen, weil Sie sonst nichts mehr von dem verstehen, was in Ihrem wissenschaftlichen Bereich passiert.

Sollten Sie allerdings zu denjenigen gehören, die eigentlich **Mediendesign**

studieren wollten und die keine Lust haben, sich mit Fragen der Informatik und der Programmierung zu beschäftigen, dann möchte ich Sie auf Ihre Prüfungsordnung bzw. das Modulhandbuch des Studiengangs Media Systems hinweisen: 110 der 180 Credit Points Ihres Studiums liegen im Bereich der Informatik. Es wäre zwar schön, wenn ich durch dieses Buch oder die von mir angebotenen Veranstaltungen Ihr Interesse am Informatikteil der Medieninformatik bzw. von Media Systems wecken kann, aber Media Systems, Medieninformatik und Medientechnik sind Bachelors of Science und da sind die gestalterischen Anteile naturgemäß deutlich dünner gesät, als bei einem Bachelor of Arts. Wo Ihre Design-Kommilitonen kreativ übers Papier streichen, um Gefühlen Ausdruck verleihen, konzipieren Sie mit mathematischen und computerisierten Abstrakta, um umfangreiche Problemstellungen zu lösen.

Nach diesen freundlich gemeinten Ermahnungen wünsche ich Ihnen viel Erfolg bei dieser Veranstaltung und in Ihrem Studium. Lassen Sie sich von Rückschlägen nicht entmutigen und tun Sie das, wofür der Begriff Studium steht: Sich intensiv mit etwas beschäftigen.

Kapitel 2

Von der Nachrichtentechnik zur Programmierung

Computer sind eine Kombination aus Bauteilen und Datenübertragungsleitungen. In vielen Programmiersprachen brauchen ProgrammiererInnen sich nicht direkt um die Datenübertragung zu kümmern. Allerdings folgen aus der Datenübertragung einige Fakten, die wir bei der Programmierung in jeder imperativen Programmiersprache beachten müssen. Tun wir das nicht, dann sind Fehler die Folge, die wir ohne ein allgemeines Verständnis der Grundlagen von Datenübertragungen nicht verstehen und damit lösen können.

2.1 Nachrichtentechnik – So kommen Nullen und Einsen in den Rechner.

Sie haben schon öfter gehört, dass ein Computer im Kern mit Nullen und Einsen arbeitet. Was den ein oder anderen vielleicht zu Spekulationen über die Qualität dieser Geräte gebracht haben könnte... Wer arbeitet schon freiwillig einen Großteil seiner Zeit mit Nullen zusammen?

Was Sie aber in aller Regel in einem Informatikstudium nicht hören, ist dass diese Folgen von Einsen und Nullen nur ein Hilfsmittel sind, eine **Repräsentation** dessen, was tatsächlich vorhanden ist. Doch wenn Sie etwas darüber hören, dann wird Ihnen in aller Regel erzählt, dass damit der Unterschied zwischen fließendem oder nicht fließendem Strom dargestellt wird. Das ist so aber in aller Regel falsch: Es gibt die unterschiedlichsten Formen von Datenübertragungen, deren Signale am Ende als Folgen von Nullen und Einsen **interpretiert** werden. Und nur die am einfachsten zu verstehende Form ist die, bei der Signale erzeugt werden, indem zwischen fließendem und nicht-fließendem Strom unterschieden wird. Dieje-

nigen von Ihnen, die Medientechnik studieren werden sich damit wenigstens fünf Semester lang beschäftigen, auch wenn Ihnen das anfangs also in Veranstaltungen wie Elektrotechnik 1 und 2 gar nicht bewusst sein wird.

Es folgen zwei Beispiele dafür, wie Einsen und Nullen auch anders dargestellt werden können:

- Am Karlsruher Institut für Technologie (KIT[1]) wird an der Möglichkeit geforscht, wie man Daten in Molekülen aus Lachs DNA speichern kann. Hier wurden Nullen mit 0,4 V und Einsen mit 0,9 V repräsentiert.
- Im Bereich der Speicherung mit DNA gibt es aber auch andere Ansätze. Wie Sie aus dem Biologieunterricht wissen, besteht DNA aus vier Molekülen, die als sogenannte Basensequenzen endlose Ketten bilden können. Und richtig, auch diese Basensequenzen kann man nutzen, um damit Einsen und Nullen darzustellen. Hier steht also gar kein Stromfluss für Nullen und für Einsen, weil sie chemisch und nicht elektrisch repräsentiert werden. DNA bietet dabei eine derart hohe Dichte pro Bit an, dass Sie den Inhalt von 50 Millionen BlueRay Disks in einer Kaffeetasse unterbringen können[2]. Na gut, vielleicht war es auch ein Übersetzungsfehler und im Originalartikel war die Rede von einem Kaffeebecher, aber spielt das eine Rolle? Denn das bedeutet, dass Sie die gesamte Videothek der Welt in einem Kaffeebecher herumtragen könnten. Und überlegen Sie jetzt, wie viel Raum eine solche Videosammlung in Form von BlueRays oder Festplatten einnehmen würde.

Die wissenschaftliche Disziplin, die sich unter anderem mit der Frage beschäftigt, wie man Nullen und Einsen übertragen kann, nennt sich **Nachrichten- und Kommunikationstechnik** und ist ein Teilbereich der **Elektrotechnik**. Während es bei der Datenübertragung innerhalb eines Computers noch recht problemlos möglich ist, mittels Ein- und Ausschalten von Stromflüssen Signale zu erzeugen und zu interpretieren, ist das bei Übertragungen über längere Distanzen eine eher ineffiziente Methode. Deshalb werden hier Schwingungen manipuliert, um die Datenübertragung zu realisieren. Das wichtigste mathematische Werkzeug ist dabei die **Fourier-Transformation** (kurz **FT**). Diese transformiert Nullen und Einsen in eine Schwingung, indem Winkelfunktionen auf eine bestimmte Weise beliebig häufig angewendet werden. (Je häufiger, desto präziser.)

In der Nachrichtentechnik spielen dann Begriffe und Ansätze eine Rolle, die kaum etwas mit dem gemein haben, was die Tätigkeit von InformatikerInnen bestimmt. Während die **Media Systems** Studierenen sich also vor-

[1] `http://www.kit.edu`

[2] `http://www.spektrum.de/news/auf-petabyte-pro-gramm/1182773`

rangig damit beschäftigen Gesamtaufgaben in Teilaufgaben zu unterteilen und mittels abstrakter Konzepte die Teilaufgaben möglichst übersichtlich zu gestalten und sie dabei möglichst effizient zu lösen, nutzen **Medien- und NachrichtentechnikerInnen** Formeln und Funktionen, um Daten so effizient wie möglich über einen bestimmten Leitungstyp zu übertragen.

Kontrolle
Die Nachrichtentechnik entwickelt die Systeme, die es uns überhaupt erst ermöglichen, Informatik zu betreiben. Einsen und Nullen sind zwar die Basis der IT-Systeme, die wir programmieren, was aber tatsächlich bei der Datenübertragung im Computer oder im Internet passiert, um Einsen und Nullen zu übertragen oder besser gesagt um eine jeweils passende Repräsentation für die Übertragung von Einsen und Nullen zu finden, hat damit größtenteils nichts gemein.

2.2 Codierung – Was steht wofür?

Eben haben Sie gelesen, dass Nullen und Einsen nur die Interpretation von z.B. Stromflüssen sind. Und damit sind wir bei einem zentralen Begriff, mit dem die meisten InformatikerInnen sich eher ungern beschäftigen, weil es dabei ausschließlich um die Frage geht, wie etwas interpretiert wird. Der Bereich, von dem hier die Rede ist, wird als **Codierung** bezeichnet.

Vielleicht haben Sie schon etwas vom **ASCI**-Code gehört. Der ASCI-Code ist nichts anderes als eine Tabelle, bei der jedem Buchstaben und verschiedenen anderen Zeichen eine Zahl zugeordnet wird. Wenn Sie nur mit iOS- oder Windows-Rechnern arbeiten, haben Sie mit der Codierung im Regelfall nichts zu tun, aber sobald Sie Daten zwischen Computern austauschen, müssen sie darauf achten. Denn im Hintergrund werden alle Daten, die Sie eingeben in Form von Codes gespeichert. Wenn Sie Daten auf einen anderen Rechner übertragen, dann werden die Codes ausgetauscht. Und nur wenn beide Rechner die gleiche Codierung verwenden empfängt der zweite Computer die Daten, die Sie eingegeben haben.

An dieser Stelle werden wir uns einen Begriff ansehen, der Ihnen immer wieder mit jeweils unterschiedlicher Bedeutung begegnen wird: **Interface**s bzw. **Schnittstelle**n. Ein Interface ist etwas, das die Verbindung zwischen zwei „Dingen"herstellt. Wenn wir über den Datenaustauch zwischen zwei Rechnern reden, die beide unterschiedliche Codierungen verwenden, dann ist eine mögliche Bedeutung des Begriffs Interface der eines Übersetzers. Das Interface kennt dann die beiden Codierungen und wandelt die übertragenen Daten von der einen Codierung in die andere um, damit der Empfänger Daten in einer Codierung erhält, die er versteht.

Hier schon einmal ein Beispiel für eine ganz andere Bedeutung des Begriffs Interface: Bei der Programmierung in objektorientierten Sprachen kann ein Interface eine abstrakte Definition für einen Programmteil sein, der noch programmiert werden muss. Diese Interpretation des Begriffs Interface ist dann sinnvoll, wenn ein Programm im Team entwickelt werden soll. Das Interface enthält dann die Festlegung von Namen für „Befehle", die zwar noch nicht funktionieren, aber bei denen schon festgelegt ist, was sie später tun sollen. So können dann alle Mitglieder des Teams mit Ihren Aufgaben beginnen: Sie können zwar Ihren Programmcode noch nicht ausprobieren, aber da Sie bereits wissen, wie die Befehle lauten, die später eine bestimmte Funktion erfüllen werden, können Sie zumindest schon mit der Entwicklung neuer Programmteile beginnen.

Ähnlich wie für den Begriff des Programmierens gilt auch für den Begriff der Codierung, dass er eine Vielzahl unterschiedlicher Methoden zusammenfasst, die jeweils für einen bestimmten Anwendungsfall sinnvolle Lösungen anbieten. Mehr dazu können Sie in dem großartigen Buch **„Information und Codierung"** von **R.W. Hamming**[3] nachlesen. Codierung ist eines der zentralen Themen der Nachrichtentechnik und wird einführend in Veranstaltungen zur **Technischen Informatik** behandelt.

Im Bereich der **maschinennahen Programmierung** kommt dann noch dazu, dass es zwei unterschiedliche Reihenfolgen gibt, in der Daten im Speicher abgelegt werden können. Die unterscheiden sich jedoch nicht (!) dadurch, dass Sie einfach nur spiegelverkehrt wären. Auch hierüber müssen Sie Bescheid wissen, wenn Sie zwischen zwei Computer Daten austauschen wollen. Denn selbst wenn Sie sichergestellt haben, dass die Codierung des einen Computers richtig in die Codierung des anderen übersetzt wird, kann Ihnen die unterschiedliche Reihenfolge im Speicher noch alles zerschießen. Die meisten von Ihnen werden dieses Problem aber nicht haben, da die Prozessoren heutige Rechner von Apple und Microsoft die gleiche Reihenfolge bei der Datenspeicherung nutzen. Vor einigen Jahren, als Apple noch Prozessoren von Motorola verwendete, war es dagegen eines der zentralen Probleme beim Datenaustausch zwischen Rechnern mit dem Betriebssystem beider Rechner.

Mit Codierungen bekommen es bereits Programmiereinsteiger sehr schnell zu tun, auch wenn wir dann von **Datentyp**en sprechen. Denn letztere basieren im Grunde auf der Codierung: Sobald Sie eine Zahl oder eine andere Zeichenfolge in einem Programm verwenden, kann es dazu kommen, dass Ihr Programm vermeintlich falsche Ergebnisse liefert. Aber wie schon aber

[3] Hamming ist gewissermaßen der Godfather der Nachrichtentechnik.

geschrieben: Das ist nicht der Fehler der Rechners, sondern der Fehler des Entwicklers, also ggf. von Ihnen, da hier die Arbeitsweise des Rechners ignoriert wurde.

Kontrolle Texte und andere Inhalte werden bei der Speicherung codiert. Das bedeutet in der Informatik, dass Sie als Folgen von Binärziffern im Speicher liegen. Je nachdem, wie oder was Sie programmieren, werden Sie detaillierte Informationen über verschiedene Codierungsverfahren benötigen.

2.3 Informatik und Nachrichtentechnik - Die zankenden Geschwister

Wie beschrieben beschäftigen sich **Nachrichtentechnik**erInnen vorrangig mit der Frage, wie Daten mit möglichst geringem Energieaufwand so übertragen werden können, dass sie am Ende der Leitung empfangen und verstanden werden können. **Informatik**erInnen dagegen beschäftigen sich vorrangig mit der Frage, wie Daten möglichst schnell verarbeitet werden können.

Aus diesen unterschiedlichen Perspektiven resultiert eine vollständig andere Herangehensweise an Aufgaben. Und leider führt das häufig zu einer Trennung zwischen NachrichtentechnikerInnen und InformatikerInnen bzw. zwischen Ihnen und Ihren Kommilitonen der Medientechnik bzw. in der Medieninformatik. Dabei ließen sich ungemein spannende Projekte realisieren, wenn Sie es nur schafften, diese Kluft zu überwinden. Somit verfolgt dieses Kapitel auch das Ziel, Ihnen ein wenig Verständnis für dieses ungemein spannende aber auch höchst anspruchsvolle Gebiet zu vermitteln.

Das was wir heute als Computer bezeichnen, ist in aller Regel eine Art Black Box, in der mehrere Hundert Komponenten unabhängig voneinander arbeiten und über verschiedene Arten von Datenübertragungswegen miteinander kommunizieren. Vermutlich sind Sie jetzt skeptisch, da Sie wissen, dass in einem Computer Komponenten wie das Mainboard, Festplatten, Grafikarten und ähnliches vorhanden sind. Der Begriff der Black Box gilt insbesondere bei Smartphones, wo sie im Gegensatz zu den meisten anderen Rechnern nicht einmal mehr eine Grafikkarte oder einzelne Laufwerke erkennen können. Dennoch ist es so, denn wenn Sie beispielsweise ein Mainboard als eine einzige Komponente betrachten, dann ignorieren Sie, dass es sich bereits hier um eine Ansammlung von Dutzenden Komponenten handelt. Und dann gibt es zusätzlich noch Technologien wie die **VLSI**, die very large scale integration, bei der es darum geht, eine sehr

große Anzahl von eigenständigen Einheiten in einem Bauteil zu integrieren.

Wie Sie in den Veranstaltungen der **Technischen Informatik** lernen, besteht ein Prozessor (fachlich korrekt ein **Mikroprozessor**) mindestens aus den drei Komponenten Rechenwerk, Steuerwerk und Speicher sowie den Verbindungen dazwischen, die als Bus bezeichnet werden. Das ist eine Abstraktion, die einem Prozessor aus den 50er Jahren entspricht, die aber genau dem entspricht, was wir bei der **imperativen Programmierung** beachten müssen. Wenn Sie mehr über den Aufbau von Computern mit Mikroprozessoren wissen wollen, werfen sie beispielsweise einen Blick in den Band **„Rechnerarchitektur"** von **Paul Herrmann**.

Wenn wir uns nun die Arbeitsweise eines Computers unvoreingenommen[4] ansehen, dann ist die wichtigste Komponente aber nicht „der"Prozessor, sondern die Vielzahl der Kommunikationswege zwischen all den Komponenten, aus denen er besteht. Die bekannteste Art dieser Kommunikationswege wird als **Bus** bezeichnet. Womit wir bei der Antwort auf die Frage wären, warum anstelle der Bezeichnung Nachrichtentechnik häufig den Begriff der **Kommunikationstechnik** benutzt wird und warum die auch für **Informatik**erInnen so wichtig ist: Wenn die Komponenten unseres Rechners keine Daten austauschen könnten, könnten wir mit Ihnen exakt gar nichts anfangen. Ohne die Arbeit der Kommunikations- und Nachrichtentechnik wäre unsere Arbeit also gar nicht möglich.

Für Einsteiger in die imperative Programmierung scheint diese Aufteilung in Prozessor, Bus und Speicher irrelevant zu sein, doch das ist schlicht falsch: Die meisten Fehler und Missverständnisse von Einsteigern kommen schlicht dadurch zustanden, dass Ihnen häufig nicht erklärt wird, wie eine **Variable** im Speicher abgebildet wird und was der **Datentyp** damit zu tun hat. Damit Sie professionell Software entwickeln können müssen Sie aber die aus der genannten Dreiteilung erwachsenden Probleme und Lösungen kennen: Wenn Sie später scheinbar simple Dinge wie den Zugriff auf eine Datei (Speichern bzw. Laden) selbst programmieren müssen, dann müssen Sie zuerst verstanden haben, dass es eine Datenübertragung zwischen dem Laufwerk und dem Prozessor gibt. Als nächsten müssen Sie verstanden haben, dass Sie niemals auf das Laufwerk, sondern nur auf den Datenstrom zugreifen können, der zwischen Prozessor und Laufwerk existiert. Sie müssen weiterhin beachten, dass die Datenübertragung über den Datenstrom eine gewisse Zeit dauert. Dann müssen Sie verstanden ha-

[4] Mit unvoreingenommen ist hier gemeint, dass Sie sich die Arbeitsweise eines Computers ansehen und nicht die Darstellung auf einem Display bzw. die Eingabemöglichkeiten in Form von Maus, Tastatur usw.

ben, dass bei dieser Datenübertragung einiges schieflaufen kann, was das ist und was Sie ggf. tun müssen, damit die Datenübertragung trotzdem klappt. Weiterhin müssen Sie sich unter Umständen damit auseinander setzen, welche Codierung bei der Speicherung der Daten verwendet werden muss. In dem Fall müssen Sie die Codierung und Decodierung programmieren. Und es gibt noch eine Vielzahl weiterer Probleme, die bei der Nutzung eines Buses auftreten können. Für viele davon hat die **Nachrichtentechnik** Lösungen entwickelt, aber einige müssen Sie selbst im Rahmen der **Programmierung** lösen. Und das können Sie dann und nur dann, wenn Sie intensiv mit all den Problemen beschäftigen, die z.B. bei der Programmierung eines Taschenrechners irrelevant sind. Aber keine Sorge, wir fangen ganz einfach an. So lange Sie die Inhalte dieses Buches konsequent durcharbeiten und in eigenen Programmen umsetzen, werden Sie all das und noch viel mehr im Laufe der Zeit beherrschen.

Umgekehrt machen jedoch auch die Kommunikations- und NachrichtentechnikerInnen häufig den Fehler, die Konzepte und Modellierungen der Informatik als größtenteils untauglich oder überflüssig zu betrachten. Dabei sind diese Ergebnisse der Informatik Lösungen zu genau den Problemen, die bei der Nutzung nachrichtentechnischer Systeme entstanden sind bzw. entstehen. Hier sei wieder einmal auf das große Gebiet der **Praktischen Informatik** verwiesen, dass TechnikerInnen in aller Regel nicht kennen.

Bitte beachten Sie, dass wir hier über Kommunikationstechnik reden; in den **Kommunikationswissenschaften** (Teilbereich der Sozial- und Geisteswissenschaften) geht es um die Kommunikation zwischen Menschen.

Kontrolle Informatik und Nachrichtentechnik sind im Grunde wie zwei Seiten einer Medaille: Die eine kann ohne die andere nicht existieren. Im Regelfall sind beide für die Leistungen der jeweils anderen jedoch blind.

2.4 Von der Nachrichtentechnik zu logischen Gattern

Ein zweiter Bereich, mit dem sich vorrangig die Elektro- bzw. **Nachrichtentechnik** beschäftigt, ist der Aufbau von Komponenten, die logische Operationen ausführen. Diese bestehen seit mehr als fünfzig Jahren aus Kondensatoren. Wenn Sie sich nicht mehr an den Physikunterricht erinnern: Ein Kondensator ist ein Bauteil, über das mittels einer Eingangsleitung gesteuert werden kann, ob Strom weitergeleitet wird oder nicht. Hier sind wir dann auch in einem Bereich, in dem eine 1 gleichbedeutend ist mit fließendem Strom und eine 0 mit nicht fließendem Strom.

Solche logischen Gatter können Sie bei den **FPGA**s, den Field-Programmable Gate Arrays direkt programmieren[5]. Wenn Sie das tun, dann befinden Sie sich in einem der wenigen Gebiete, in dem Elektrotechniker und Informatiker dieselbe Sprache sprechen. Diese Technologie wurde erst Mitte der 80er Jahre entwickelt. Und obwohl wir hier von einer Programmiertechnik reden, werden Sie in Büchern über Programmierparadigmen kein Kapitel finden, dass das entsprechende Paradigma enthält.

Der Grund dafür ist ganz einfach: Diese Art der Programmierung ist eine direkte Umsetzung dessen, was in der Mathematik **boolesche Algebra** genannt wird. Und an dieser Stelle möchte ich eine Lanze für die **Mathematik** brechen: Immer wieder fallen Formulierungen wie „Mathe ist doch nur eine Hilfswissenschaft." Das allerdings ist ein Ausdruck, der eher auf die Ignoranz der Person verweist, die ihn gebraucht. Denn tatsächlich gab es die bei Computern verwendete Logik lange vor dem ersten Computer.

Und Mathematik hat zunächst auch nichts mit Rechnen zu tun: Im alten Griechenland gab es eine Disziplin namens Philosophie. Die Philosophen versuchten die Welt und das was in ihr geschieht zu erklären. Daraus entstanden dann weitere Disziplinen wie die Naturwissenschaften, die sich auf bestimmte Teilgebiete der Welt und des Universums konzentrieren. Wenn Sie also bislang über Philosophie als eine Wissenschaft betrachtet haben, in der es darum geht, wie Menschen sich verhalten sollten, dann haben Sie hier eine zu beschränkte Vorstellung.

Es gab aber auch Forscher, die sich fragten, wie ein Universum aussehen würde, wenn sie den umgekehrten Weg wählen würden. Sie untersuchten deshalb, wie ein Universum aussehen würde, für das Eigenschaften willkürlich festlegt werden. Um das nochmal zu betonen: Im Gegensatz zu allen anderen Wissenschaften setzt sich die Mathematik nicht mit der Beschaffenheit eines einzigen (nämlich unseres) Universums auseinander: Sie geht wesentlich weiter! Sie stellt sich die Frage wie jedes nur denkbare Universum aussehen würde. Und wer so etwas als Hilfswissenschaft abtut, der... Nun ja, wie soll ich es höflich ausdrücken?

Wenn Sie jetzt genau gelesen haben, dann verstehen Sie, warum die Aussage wie „das wurde mathematisch bewiesenïn den Naturwissenschaften häufig einen falschen Eindruck vermittelt: Sie können mathematisch alles beweisen, wenn Sie nur kreativ genug bei der Entwicklung von Annahmen sind. Ein mathematischer Beweis alleine genügt also nicht, um zu bewei-

[5]Diese Art der Programmierung ist Teil der Veranstaltung Informatik 2 für Media Systems. Das bedeutet leider auch, dass Sie am Studienende keine Kenntnisse der Praktischen Informatik haben werden. Dafür werden Sie im Bereich der Technischen Informatik deutlich mehr Kenntnisse besitzen als Ihre Kommilitonen von anderen HAWs bzw. FHs.

sen, dass etwas wahr ist. Sie müssen außerdem beweisen, dass die Voraussetzungen bzw. Annahmen wahr sind, die Sie für Ihre Beweisführung vorausgesetzt haben.

Ein aktuelles Beispiel sind die „Beweise"rund um den Urknall. Die Physik ist zurzeit im Stande, zu beweisen, in welchem Zustand sich das Universum wenige Sekunden nach dem sogenannten Urknall befand. Was also direkt zum „Zeitpunkt"des Urknalls passiert oder was möglicherweise davor „warïst zumindest momentan nicht beweisbar. Es gibt jedoch PhysikerInnen, die darauf beharren, das zu können. Sie legen dafür mathematische Beweise vor, die in sich schlüssig sind. Der Fehler liegt hier aber nicht in der mathematischen Beweisführung, sondern darin, dass sie Annahmen treffen, die zumindest noch nicht bewiesen sind. Wie Sie in „**Mathematik 1**" lernen ist es aber möglich, aus einer falschen Annahme wahre und falsche Schlussfolgerungen zu ziehen. Wenn also PhysikerInnen Annahmen für eine mathematische Beweisführung verwenden, die noch nicht bewiesen sind, dann kommen sie damit zu Aussagen, bei denen unklar ist, ob sie nun wahr oder falsch sind. Also haben die eingangs genannten Forscher nicht etwa den Urknall oder Abläufe rund um den Urknall bewiesen, sondern lediglich gezeigt, welche Abläufe dort stattgefunden haben, wenn bestimmte Annahmen wahr sind.

Jedenfalls sollte Ihnen damit klar sein, dass MathematikerInnen häufig schon Lösungen parat haben, wenn die zugehörigen Probleme noch gar nicht in der Praxis auftreten. Und nein, wir reden hier nicht von Monaten; einige Lösungsansätze, die in den letzten Jahrzehnten zum Einsatz kamen wurden bereits vor mehreren hundert Jahren von Mathematikern konzipiert. Wenn Sie also in einem mathematischen Lehrwerk Formulierungen lesen wie „Es sei ein Universum mit den Eigenschaften...", dann wissen Sie jetzt, dass das wortwörlich gemeint ist: Es geht wirklich darum, ein Universum oder einen Teilbereich eines Universums zu beschreiben, das über bestimmte Eigenschaften definiert wird. Dieses Universum hat aber in aller Regel nicht das geringste mit dem zu tun, was Sie sich als ein Universum vorstellen.

Hier auch noch ein Exkurs: Häufig wird von radikalen (Mono-)theisten behauptet, Naturwissenschaftler würde behaupten, dass es Gott nicht gäbe. Das ist falsch: Naturwissenschaftler untersuchen, wie das Universum aussieht. Sie untersuchen dabei, welche Gesetzmäßigkeiten im Universum nachweisbar sind. Das hat aber nichts mit der Frage zu tun, ob es einen Gott gibt, der all das erschaffen hat.

Aber zurück zu den FPGAs: Wenn Sie die boolesche Algebra nicht beherrschen, können Sie auch kein FPGA programmieren. Unabhängig davon

kommt die boolesche Algebra auch immer dann zum Einsatz, wenn Sie den Rechner etwas prüfen lassen wollen. Für Fortgeschrittene gibt es dann zwar noch Mittel wie die Fuzzy Logic, aber das überlassen wir mal lieber den MathematikernInnen und universitären InformatikerInnen.

Kontrolle
Eine erste Art der Programmierung besteht in der Umsetzung der booleschen Algebra, einer mathematischen Methode, um aus einfachen Ja-/Nein-Fragen komplexe Modelle zu entwickeln. Prozessoren basieren auf nichts anderem.

2.5 Weiter zur Maschinensprache

Wenn wir nun einen Prozessor nicht entwickeln, bzw. seine Arbeitsweise programmieren wollen, sondern ihn so nutzen wollen wie er ist, um Programme zu entwickeln, dann ist die erste Methode die **Maschinensprache**. Danach folgt die **maschinennahe Programmierung**.

Wie Sie wissen reden wir von Prozessoren mit einer gewissen **Bittigkeit**. Heute sind die 32-Bit- und die 64-Bit-Prozessoren am bekanntesten. Diese Bittigkeit bedeutet nichts anderes, als dass der Prozessor bei jedem Rechenschritt eine Zahl mit genau dieser Anzahl an Bits addieren kann. Ein 32-Bit-Prozessor kann also bei jedem Rechenschritt eine Zahl zwischen 0 und $2^{32} - 1$ zu einer anderen Zahl in diesem Bereich addieren[6]. Wie Sie in der **Technischen Informatik** erfahren liegt darin auch der Grund, dass 32-Bit-Prozessoren nur einen Speicher mit rund 4 GB verwalten können: Für mehr Speicherbereiche haben Sie schlicht keine „Hausnummern".

Aufgabe:
Berechnen Sie doch mal eben die maximale Größe für Speicher, die ein 64-Bit-Prozessor nutzen kann. Und suchen Sie nach einem anschaulichen Beispiel, dass diese Zahl verdeutlicht. (Anmerkung, die nötig ist, weil leider einige Studienanfänger das Potenzrechnen nicht beherrschen: 2^{64} ist nicht (!) $2 \cdot 2^{32}$. Die Antwort lautet also nicht „etwas weniger als 8,6 Millionen".)

Die Maschinensprache besteht dementsprechend aus nichts anderem als Zahlen, die in dem Bereich liegen, der von der Bittigkeit des jeweiligen Prozessors abhängt. Ob eine Zahl nun ein Befehl darstellt, ob Sie als Zahl zu verwenden ist oder ob es eine Position im Speicher sein soll, kann nur wissen wer die Maschinensprache eines Prozessors beherrscht; es gibt keine Möglichkeit, einer Zeile eines Programms in Maschinensprache anzuse-

[6] 2^{32-1} entspricht etwas weniger als 4,3 Millionen

hen, wofür sie steht.

Kontrolle
Maschinensprache ist so schlecht lesbar, dass Sie sie am besten gleich ignorieren. Verwechseln Sie aber bitte nicht die Maschinensprache (siehe dieser Abschnitt) mit der maschinennahen Programmierung (Assembler, siehe nächster Abschnitt).

2.6 Maschinennahe Programmierung – Assembler

Nachdem wir jetzt geklärt hätten, wie man einen Prozessor besser nicht programmiert, kommen wir zur ersten Methode, um einen Prozessor sinnvoll zu programmieren: Die maschinennahe Programmierung, die auch als **Assembler** oder Assembler Programmierung bezeichnet wird. Es ist die erste Variante der **imperativen Programmierung** und leider ist sie alles andere als anschaulich. Aber wenn Sie sich hiermit beschäftigen, dann zu **C** und später zu **C++** oder **Java** weitergehen, werden sie verstehen, was die Vorteile dieser Sprachen sind und werden diese zu schätzen wissen.

Im Regelfall lernen Sie zu Beginn einer Veranstaltung, in der Sie die maschinennahe Programmierung kennen lernen, etwas über den Aufbau des Prozessors, den Sie maschinennah programmieren werden. Da tauchen dann Begriffe wie Pipeline, RISC und CISC und andere auf. Das kann Ihnen helfen, bestimmte Abläufe besser zu verstehen. Für den Einstieg genügt es aber, wenn Sie die Dinge kennen lernen, die Sie tatsächlich programmieren können.

Aber bevor wir auf die eingehen, sollten Sie wieder (wie zu Beginn dieses Buches) die Antwort auf die Frage erfahren, was **maschinennahe Programmierung** eigentlich ist. Und die Antwort hierauf ist genau die, die allgemein mit dem Begriff Programmieren verbunden wird: Sie tippen Zeile für Zeile ein, was der Computer (besser gesagt der Prozessor) tun soll und der führt es dann in genau dieser Reihenfolge aus. Einzige Ausnahme: Sie können Sprünge ins Programm einbauen, die dafür sorgen, dass der Prozessor einen anderen Teil Ihres Programms ausführt, aber am zeilenweisen Ablauf ändert sich ansonsten nichts. Diese anderen Teile werden als **Subroutine**n bezeichnet. Es sind Programmteile, die an verschiedenen Stellen im Programm ausgeführt werden sollen. Und sie werden einzig und allein deshalb ausgelagert, weil sie dadurch nur einmal programmiert, aber mehrfach ausgeführt werden können. Das reduziert die Fehleranfälligkeit und erleichtert die spätere Verbesserung des Programms.

In der maschinennahen Programmierung besteht nun jede Zeile aus ei-

nem sogenannten **Mnemon** und zwischen keiner und mehreren Zahlen. Ein Mnemon ist einem Hilfswort, das einem Befehl des Prozessors entspricht. Im Gegensatz zur tatsächlichen Maschinensprache können Sie hier also Buchstabenkombinationen für Befehle benutzen. Aber ob eine Zahl eine Zahl oder eine Speicheradresse ist, das müssen Sie zusammen mit der Definition jedes Mnemons lernen.

Dennoch gibt es Gründe, wegen denen bestimmte Entwickler immer noch in Assembler programmieren und aus denen es tatsächlich Sinn macht, sich auf diese Art der Programmierung zu konzentrieren: Es gibt keine effizientere Möglichkeit, einen Computer zu programmieren und viele Prozessoren lassen sich nicht vollständig in anderen Sprachen programmieren. Der Bedarf an Effizienz muss allerdings sehr hoch sein, denn mit C als höherer Sprache können Sie immer noch recht maschinennah und effizient programmieren.

Das Standardwerk für die maschinennahe Programmierung stammt von **Donald E. Knuth** und hat den Titel „**The Art of Computer Programming**". Es besteht aus sieben Bänden, von denen die ersten drei veröffentlich wurden. Band vier wurde in zwei Teile unterteilt, von denen der erste ebenfalls veröffentlicht wurde. Die übrigen Bände sind noch in Arbeit. Das mag so klingen, als wenn eigentlich ein anderes Buch als Standard gelten sollte, doch im Gegensatz zu allen mir bekannten Autoren untersucht Knuth in seinem Buch jedes grundlegende Problem, das bei imperativer Programmierung vorkommen kann. Insbesondere wird in seinem Band praktisch ab der ersten Seite klar, warum InformatikerInnen welche mathematischen Grundlagen beherrschen müssen.

Dennoch lassen sich die Bücher auch ohne diese mathematischen Grundlagen durcharbeiten: Knuth hat viele Passagen so verfasst, dass klar erkennbar ist, ob sie auch ohne mathematische Grundlagen verständlich sind. Das gleiche gilt für die Vielzahl an Aufgaben, die den Text begleiten. Die Musterlösungen machen beispielsweise rund ein Drittel des ersten Bandes aus. Damit kann jedeR Interessierte sich unabhängig von den persönlichen Kenntnissen weitgehend in das Themengebiet einarbeiten.

Kontrolle
Auch die maschinennahe Programmierung ist eher abstrakt. Die Zeilen eines solchen Programms bestehen aus Buchstabenkürzeln und Zahlen. Was das Programm bei der Ausführung an welcher Stelle tut ist sehr schwer zu erkennen.

2.7 Zusammenfassung

In diesem Kapitel haben Sie erfahren, dass die Nachrichtentechnik die Techniken und Technologien liefert, mit der InformatikerInnen Ihre Ergebnisse in die Praxis umsetzen können. Ohne Nachrichtentechnik gibt es keine IT-Systeme und letztlich würden Sie nicht nur dieses Buch nicht lesen, Sie würden wahrscheinlich etwas ganz anderes studieren, weil Computer und das Internet nicht existieren würden. Unter Umständen hätten Sie nicht einmal die Hochschulreife erreicht.

Sie haben außerdem einen ersten Einblick in das erhalten, was hinter den Nullen und Einsen steht, und dass das etwas ganz anderes ist, als das, was die meisten sich darunter vorstellen. Sie haben ebenfalls erfahren, dass wir auch die Einsen und Nullen in aller Regel nicht wahrnehmen, weil sie sich hinter den Ergebnissen der Codierung verbergen.

Abschließend haben Sie einen ersten Blick darauf erhascht, wie all das zu Programmiersprachen führt. Im nächsten Kapitel geht es dann um Formen der Programmierung, mit denen Sie in den nächsten Jahren voraussichtlich am häufigsten zu tun haben werden.

Kapitel 3

Vorbereitung fürs Programmieren

Im Grunde brauchen Sie zum Programmieren lediglich einen Texteditor sowie ein SDK bzw. einen Compiler oder einen Interpreter für die jeweilige Sprache. Ein Texteditor wird bei jedem beliebigen Betriebssystem mitgeliefert, während Sie sich SDK/Compiler/Interpreter häufig von der Seite des Entwicklers herunterladen können.

Sie werden im Folgenden des Öfteren die Bezeichnung **Werkzeug** (engl. **tool**) lesen. Beim Programmieren ist damit im Regelfall ein Programm gemeint, das Sie in irgendeiner Form bei der Softwareentwicklung unterstützt.

Da Sie bei der Suche nach Werkzeugen fürs Programmieren über eine Vielzahl an Begriffen stolpern werden, folgt hier eine auszugweise Aufstellung häufiger Bezeichnungen, sortiert nach Bedeutung:

- Mit **Quellcode** bezeichnet man den Programmtext, den Sie eingegeben haben.
- Ein **Interpreter** ist eine Software, der Ihren Quelltext Zeile für Zeile übersetzt und kontinuierlich jede übersetzte Zeile sofort ausführt.
- Ein **Compiler** ist eine Software, der das von Ihnen verfasste Programm vollständig übersetzt und dabei eine beschränkte Anzahl an Fehlertypen erkennen kann. Personen, die ein sehr beschränktes Verständnis von Programmierung haben sind deshalb häufig überzeugt, dass kompilierte Sprachen sicher, interpretierte Sprachen dagegen unsicher wären. Tatsächlich handelt es sich schlicht um zwei Paradigmen, die unterschiedliche Vor- und Nachteile haben. Und wenn wir von **IT-Sicherheit** sprechen, dann hat das nichts aber auch rein gar nichts mit kompilierten Sprachen zu tun.

- Ein Debugger ist eine Software, die Ihren Quelltext auf Fehler untersucht.
- Ein **SDK** (kurz für Software Development Kit bzw. Softwareentwicklungskit) ist eine Softwaresammlung, die neben einem Interpreter und/oder Compiler eine Reihe weiterer Programme beinhaltet, die Sie bei der Entwicklung von Programmen unterstützt. SDKs werden jeweils für eine bestimmte Sprache entwickelt. Gelegentlich wird auch der Begriff **Toolchain** verwendet. Im hier verwendeten Sinne entspricht eine Toolchain einem SDK.
- Eine **IDE** (kurz für Integrated Development Environment bzw. Integrierte Entwicklungsumgebung) ist ein Sammlung von Programmen, die meist unabhängig von einer bestimmten Sprache zum Programmieren nutzbar sind. Im Gegensatz zu einer SDK beinhalten IDEs in aller Regel einen eigenen Editor, der u.a. Syntaxerkennung bietet. Mehr noch: Viele IDEs sind so entwickelt, dass es möglich ist, eine Vielzahl von SDKs in ihnen zu nutzen und damit eine Vielzahl von Programmiersprachen in Ihnen zu nutzen.

Die nachfolgenden Werkzeuge gehören in den fortgeschrittenen Bereich, mit dem Sie im Rahmen dieses Buches nichts zu tun haben werden. Sie sollten Sie allerdings kennen, damit Sie wissen, wonach Sie später suchen müssen, um Probleme zu vermeiden, die sehr viel Zeit kosten können. Ihre Beherrschung unterscheidet professionelle Software Entwickler von Quereinsteigern.

- **Deployment Support/Tools** sind Werkzeuge, die Sie dabei unterstützen, ein Programm zu verteilen. Stellen Sie sich dazu vor, dass Sie ein Programm entwickeln, das von allen 15.000 Mitarbeitern Ihres Unternehmens genutzt wird. Ohne Deployment Support/Tool müssten Sie nun an jeden Arbeitsplatz gehen, die alte Version deinstallieren, u.U. den Rechner neustarten und dann die neue Version einspielen. Und wenn Sie fertig sind, gehen Sie in Rente. Mit Deployment Support/Tools dagegen lassen Sie lediglich die Änderungen (auch als **Delta**s bezeichnet) automatisiert von einem zentralen Server aus an die Rechner verteilen und dort installieren.
- **Refactoring** bezeichnet eine Überarbeitung einer Software. Hier geht es aber nicht darum, einzelnen Fehler zu beheben, sondern darum, strukturelle Änderungen in eine Software einzuarbeiten. (Hier wären wir bei einem weiteren Aspekt des Software Engineering angelangt.)

Die folgenden Abschnitte enthalten jeweils die Informationen, die sie zur Vorbereitung der Programmierung auf einem Windows-Rechner benötigen,

da auf den Rechnern unseres Departments Windows zum Einsatz kommt. Für die Linux- und MacOS-User unter Ihnen gibt es hier leider keine bzw. nur wenige Hinweise. Nutzen Sie bitte über die üblichen Quellen im Netz.

3.1 Assembler und C – Vorbereitung für die maschinennahe und imperative Programmierung

In den Veranstaltungen zur maschinennahen und imperativen Programmierung unseres Departments kommen zurzeit **ARM Cortex-M0 Prozessoren** zum Einsatz. Bei diesen Prozessoren können Sie von der Webpage von IAR ein kostenloses SDK herunterladen. Im Rahmen der Veranstaltung Informatik 3 (für Media Systems) erhalten Sie außerdem Informationen darüber, wie Sie Ihr System einrichten können und wie Sie die dort verwendete IDE beziehen können. Diese stammt von der Fa. Keil, heißt µVision und wird auch als **MDK** bezeichnet. Weiterhin benötigen Sie für die Entwicklung von Software für einen ARM Prozessor ein Entwicklerboard, auf dem der Prozessor bereits montiert ist, sowie eine debugging Unit. In beiden Fällen handelt es sich um Hardware, die Sie im Labor für Ihre Versuche erhalten.

Wenn Sie so lange nicht warten wollen und wesentlich weniger Geld ausgeben wollen, um sich in die Programmierung von ARM-Prozessoren einzuarbeiten, dann möchte ich Ihnen einen Computer mit ARM-Prozessor empfehlen, der von der Universität in Cambridge extra für Studienanfänger entwickelt wurde. Es handelt sich um den **Raspberry Pi** (kurz RasPi oder Pi), der in der Version 2 für rund 45,- € zu haben ist. (Maus und Tastatur sowie eine MicroSD-Card als Laufwerk und ein passendes Ladekabel fehlen hier noch, aber für knapp 70,- € sollten Sie ein komplettes System bekommen.) Das praktische am RasPi ist, dass Sie ihn in die Tasche stecken können.

Wenn Sie dagegen die maschinennahe Programmierung auf einem Intel- oder AMD-Prozessor mit IAx86-Architektur[1] durchführen wollen, können Sie sich die **GCC** (kurz für **GNU Compiler Collection**) kostenlos herunterladen und installieren. Hierbei handelt es sich um eine Sammlung von Compilern für die verschiedensten Sprachen. Im Gegensatz zu vielen anderen Compilern erhalten Sie die GCC unter der sogenannten **GNU GPL** (Lizenz). Kurz gesagt bedeutet das, dass Sie hierauf kostenfreien Zugang haben, dass die Entwickler Ihnen darüber hinaus aber noch wesentlich mehr Rechte einräumen. Die GCC sind in verschiedenen Paketen enthalten, die

[1] Das sind die Prozessoren, die zurzeit üblicherweise in Desktop-Rechnern verbaut werden.

eine etwas komfortablere Installation erlauben.

Windows-Nutzern sei hier das **MinGW** empfohlen, das zusätzlich zum GCC auch ein minimalistisches GNU installiert. Mit letzterem können Sie unter Windows auf der Konsole wie in einer Linux-Umgebung arbeiten. Alternativ können Sie auch auf Microsofts **Visual Studio** Professional oder Ultimate zurückgreifen, das Sie als Studierende kostenlos über das Dreamspark Programm erhalten. Davon gibt es noch die Community Edition, die grundsätzlich kostenlos von Microsoft angeboten wird. Beachten sie aber bitte, dass Sie in diesem Fall unter Umständen Software entwickeln, die ausschließlich auf Windows Rechnern lauffähig ist. Für den Aufruf des GCC-Assembler Compilers müssen Sie abschließend noch die PATH-Variable von Windows um das /bin-Verzeichnis Ihrer MinGW-Installation erweitern.

MacOS-Nutzer brauchen eine derart umfangreiche Software nicht, da Ihr Betriebssystem bereits die entsprechenden Werkzeuge an Bord hat. Allerdings müssen Sie immer noch einen Assembler Compiler installieren, wozu Sie am besten ebenfalls auf die GCC zurückgreifen. Einsteiger werden zuvor noch **XCode** installieren müssen. Dabei handelt es sich um eine IDE, die Sie über den App Store herunterladen können. Suchen Sie anschließend im Netz nach einer Anleitung, wie Sie die GCC in XCode einbinden können. Unter MAC OS X wählten Sie 2010 im XCode
`Menü -> Preferences -> Downloads` den Eintrag „Command Line Toolsünd hatten nach dem Abschluss des Downloads die GCC vollständig installiert. Die Installation sollte also auch Einsteigern problemlos möglich sein. Genau wie bei Microsofts Visual Studio sollten Sie allerdings daran denken, dass Sie bei der Nutzung von XCode unter Umständen Software entwickeln, die ausschließlich auf Apple-Rechnern lauffähig ist.

Warum es hier keine Informationen für **Linux**-User gibt? Weil Linux-User im Regelfall selbständig genug sind, um sich diese Informationen selbst zu beschaffen. Sollten Sie Linux-Neuling sein, empfehle ich Ihnen die Linux Einführung auf der Plattform edX. Dieser Kurs kann wie die meisten Kurse auf edX auch kostenlos belegt werden und wird im Gegensatz zu den meisten Kursen kontinuierlich angeboten; Sie sind hier also nicht an einen bestimmten Zeitraum gebunden. Er wurde von der FSF entwickelt und ist nach anfänglichem Marketing für die FSF eine fundamentale und sehr nützliche Einführung in das offene und freie Betriebssystem. Leider kommt GNU ein wenig zu kurz, aber Sie erhalten zumindest die entsprechenden Links.

Wenn Sie einen RasPi erworben haben, dann gibt es eine Einführung in Linux im sogenannten **Raspberry Pi Educational Manual**

http://vx2-downloads.raspberrypi.org/Raspberry_Pi_Education_Manual.pdf, das zwar für die erste Version des RasPi entwickelt wurde, aber auch bei der aktuellen Version genutzt werden kann, da diese weitgehend abwärtskompatibel ist. Darin ist neben der Einführung in die Grundlagen verschiedener Arten der Programmierung auch eine Einführung in die Administration von Linux enthalten. Die sollten Sie als RasPi-Nutzer auf jeden Fall durcharbeiten.

Außerdem sollten Sie noch ein gutes Lehr- und Nachschlagewerk für die Assemblerprogrammierung erwerben. Hier gibt es allerdings auch einige Bände, die Sie legal kostenlos herunterladen dürfen. Grundsätzlich möchte ich Ihnen für diese Fälle die folgende Seite ans Herz legen:
http://hackershelf.com/topics/
Suchen Sie hier bitte unter dem Suchbegriff **assembly**, nicht assembler, da im Englischen die Bezeichnung nicht Assembler Language sondern Assembly Language lautet.

Persönlicher Tipp: Greifen Sie dort zu „**Programming from the Ground up**"von Jonathan Bartlet.

Sie wollen wissen, was mit GNU gemeint ist? Gute Frage. Aber die Antwort lautet: Lesen Sie es selbst nach: https://www.gnu.org/ Es geht hier letztlich um eine der wichtigsten Institutionen, die Sie als angehende InformatikerInnen kennen sollten: Die **FSF** (kurz für Free Software Foundation). Ob nun die **GI** (kurz für Gesellschaft für Informatik), **IEEE**, die FSF oder **ITU-T** die absolute Spitzenposition bezüglich der Bedeutung für InformatikerInnen einnimmt, kann ich Ihnen nicht sagen, aber sie sind allesamt außerordentlich wichtig. Beschäftigen Sie sich deshalb damit.

Ergänzungen zu C und C++

Leider befand sich zum Zeitpunkt, als diese Zeilen geschrieben wurden noch kein für Einsteiger empfehlenswertes Buch auf hackershelf.com. Allerdings gibt es für die Programmierung in C ein Standardwerk, das u.a. vom Entwickler der Sprache verfasst wurde: „**The C Programming Language**"von **Ritchie** und **Kernighan**.

3.2 Java - Vorbereitung für die Programmierung in Java

Wichtig für Wiederholer: Stellen Sie sicher, dass Sie auf jedem Rechner die gleiche Version von Java installiert haben. Zwar ist Java weitgehend abwärtskompatibel, aber die Entwickler der Sprache erklären immer wie-

der einzelne Bestandteile für veraltet. Wenn Sie dann noch eine alte JDK verwenden, bedeutet das, dass Sie ggf. aktuellen Code nicht programmieren können und veraltete Vorgehensweisen vom Compiler akzeptiert werden. Gerade bei Hausaufgaben führt das dann schnell zu Problemen.

Für Java gibt es zunächst zwei Anbieter von SDKs. Zum einen können Sie eines der offiziellen SDKs von Oracle herunterladen, zum anderen gibt es das OpenJava, das unter einer anderen Lizenz entwickelt wird.

Auf der Webpage des Entwicklers Oracle gibt es Java2SE, Java2EE, Java for Mobile und noch ein gutes Dutzend weiterer Java Downloads. Das was Sie für den Einstieg brauchen ist **Java2SE**. Zum Grund dafür kommen wir am Ende dieses Abschnitts, für die Installation brauchen Sie es nicht zu wissen.

Nachdem Sie sich mit den Nutzungsbedingungen einverstanden erklärt haben, können Sie auf eine Reihe an Downloads zugreifen, die sich dadurch unterscheiden, ob Sie nun einen 32- oder 64-Bit-Prozessor haben, welches Betriebssystem Sie nutzen und ob Sie die Installationsdatei vollständig (offline) oder nur zum geringen Teil (für eine online-Installation) herunterladen wollen. Wenn Sie sich die Zeit nehmen, genau hinzusehen, dann sollten Sie hier die für Ihr System optimale Version wählen können. Im Grunde ist es bei Windows, Linux oder MacOS aber egal, welche Version (32/64 Bit bzw. online/offline) Sie wählen, so lange das Betriebssystem und der Prozessor stimmen.

Einen Buchtipp für die Programmierung in Java kann an dieser Stelle nicht gegeben werden, was auch daran liegt, dass mit jeder neuen Version wieder essentielle Änderungen in die Sprache eingeführt werden. Dazu kommt, dass Java durch seinen Middleware-Charakter ohnehin derart umfangreich ist, dass kein Buch existieren kann, das auch nur größtenteils vollständig ist. Dazu ist das Framework schlicht zu umfangreich. Sie glauben mir nicht? Dann werfen Sie einen Blick in die Java API; das ist die vollständige Java Dokumentation: `http://docs.oracle.com/javase/8/docs/api/` Und dort sind lediglich all die Klassen enthalten, die von Oracle selbst angeboten werden. Am besten setzten Sie in Ihrem Browser auch gleich ein Lesezeichen auf diese Seite, denn beim Lernen von und Arbeiten mit Java werden Sie wohl nichts so oft brauchen wie diese Seite. Sollte inzwischen die nächste Version von Java veröffentlicht worden sein, dann ändern Sie bitte die entsprechende Ziffer im Link ab.

Bei dieser Gelegenheit lernen Sie auch schon die nächste wichtige Abkürzung kennen: **API** steht kurz für **Application Programming Interface**, was als **Programmierschnittstelle** übersetzt wird. Eine API galt als ein typisches Kennzeichen objektorientierter Programmiersprachen, doch das ist falsch:

Eine Vielzahl an Lösungen wurden bereits von anderen Entwicklern programmiert und intensiv getestet, sodass Sie sich darauf konzentrieren können, die Spezialfälle zu programmieren, die bei Ihrer Software auftreten. Tatsächlich finden Sie so etwas jedoch selbst für Assembler. Der Begriff **Klassenbibliothek** wird des Öfteren als Synonym für eine API verwendet. An sich steht dieser Begriff aber allgemeiner für Sammlungen von Klassen, die Lösungen für häufig auftretende Probleme beinhalten.

Da der Begriff der **Klasse** untrennbar mit Java verbunden ist, hier eine kurze Erklärung, die zwar viel zu simpel ist, aber für den Anfang genügen soll: In Java bestehen Programme wie in alle objektorientierten Programmiersprachen aus sogenannten Modulen, die in Java als **Package**s und Klassen bezeichnet werden. Jedes dieser Module beinhaltet einen Programmteil, der für sich genommen bereits ein einen sinnvollen Teil des Gesamtprogramms erfüllen kann. Um bei besonders großen Projekten mehr Übersicht zu erhalten werden dort mehrere Klassen in einem Package gesammelt. Packages können Sie sich in Java wie Verzeichnisse bei einem Betriebssystem vorstellen. Genau wie dort kann ein Package mehrere Packages enthalten, die sowohl Klassen als auch Packages enthalten können.

Anschließend müssen Sie u.U. unter Windows (ähnlich wie bei der Vorbereitung für die maschinennahe Programmierung) noch die PATH-Variable erweitern. In diesem Fall müssen Sie dort das Verzeichnis angeben, in dem u.a. die Datei javac liegt. Prüfen Sie anschließend über den Befehl javac –v in der Eingabeaufforderung, ob Sie die PATH-Variable richtig gesetzt haben. Anmerkung für einen späteren Umstieg auf Linux: Dort heißt die „Eingabeaufforderung“ **Terminal**, **Konsole** oder **shell**, wird aber auch als bash, sh, z-sh und ähnliches bezeichnet, wobei das teilweise konkrete Programmnamen sind, die ein Terminal bzw. eine Konsole starten.

Jetzt aber zur Antwort auf die Frage, was denn die verschiedenen Java-Versionen unterscheidet. Hier werden wir nicht zu sehr ins Detail gehen, daher nur so viel:

- Warum steht da eine 2 in Java2SE?
  Obwohl mit jeder neuen Version (Java ist im Moment, in dem diese Zeilen geschrieben werden bei Version 8 angelangt) essentielle Änderungen an der Sprache durchgeführt werden, entschieden sich die Entwickler nach Version 2 die Zahl 2 im Namen zu behalten. Das ist alles.

- Wir brauchen ja das SDK. Aber was ist die JRE?
  Sie wissen bereits, dass Sie ein Softwareentwicklungskit zum Entwickeln von Software nutzen können. Dieses beinhaltet eine Laufzeit-

umgebung (in diesem Fall die JRE, kurz für Java Runtime Environment), die dazu dient, Java Programme auf einem Computer laufen zu lassen, auf dem kein JDK installiert ist.
Laufzeitumgebungen gibt es für eine Vielzahl von Programmiersprachen. Es handelt sich dabei um so etwas wie eine Übersetzungssoftware, die es dem Betriebssystem ermöglicht, Programme in der jeweiligen Sprache auszuführen.
Und richtig verstanden: Kein Programm wird direkt von einem Betriebssystem ausgeführt, sondern das Betriebssystem startet gegebenenfalls die Laufzeitumgebung für eine Sprache, in der das zu startende Programm dann ausgeführt wird. Ein häufiger Irrtum von Nutzern besteht dagegen darin, anzunehmen, dass beispielsweise unter Windows Dateien mit der Endung .exe von Windows selbst ausgeführt werden; vielmehr gilt auf für Dateien im .exe-Format das gleiche, was für alle Programm gilt, die ausgeführt werden sollen: Es muss auf dem Rechner eine Laufzeitumgebung (oder ein Interpreter) für die entsprechende Sprache installiert sein, sonst kann das Programm nicht ausgeführt werden. Einzige Ausnahme: Ein Programm liegt in Maschinensprache vor. Dann und nur dann kann es direkt vom Prozessor ausgeführt werden.

- Was bedeutet dieses SE in Java2SE? Was ist dieses Java2EE? Und was sollen all die anderen Versionen?
  - **Java SE** (kurz für Standard Edition) ist gewissermaßen der Kern der Sprache/der Middleware. Es enthält alles, was Sie benötigen, um Java Programme zu entwickeln bzw. um sie auf einem Rechner laufen zu lassen.
  - **Java EE** (kurz für Enterprise Edition) ist eine erweiterte Fassung von Java SE, die in Unternehmen eingesetzt werden kann, bei denen es wenigstens einen zentralen Server gibt, über den die Kommunikation von Java Anwendungen koordiniert wird.
  - **Java ME** (kurz für Micro Edition) ist dagegen eine höchst effiziente Version, die auf Geräten mit geringen Kapazitäten zum Einsatz kommen kann. (Alte Rechner, Smartphones, usw.)
  - **Java FX** (kurz für Effects) beinhaltet Klassen, mittels derer Internetapplikationen entwickelt werden können.

Nachdem Sie das JDK heruntergeladen und installiert haben, brauchen Sie noch eine Möglichkeit, um Java Programme einzugeben oder zu ändern, denn das JDK enthält keinen eigenen Editor. Hier genügt für die ersten Schritte ein **einfacher Texteditor**. Verwechseln Sie das bitte nicht mit einer

Textverarbeitung: Ein Texteditor zeigt Ihnen den eingegebenen Text standardmäßig ohne irgendwelche Hervorhebungen wie Fettdruck und ähnliches. Im Gegensatz dazu zeigt Ihnen eine Textverarbeitung Texte mit Hervorhebungen, bei denen dann der Teil des Quelltexts ausgeblendet wird, über den diese Hervorhebungen erzeugt werden.

Standardmäßig hat jedes Betriebssystem mindestens einen Texteditor an Bord. Bei Windows finden Sie diesen unter Zubehör bzw. indem Sie bei Windows 8 in der „Kachelansichtßchlicht das Wort Editor eintippen. Bei älteren Windows-Versionen können Sie das Suchfenster im Windowsmenü nutzen.

Als ersten Schritt zu mehr Komfort gibt es Texteditoren mit **Syntaxerkennung**. Beispielsweise den Notepad++. Solche Texteditoren heben automatisch syntaktische Fehler hervor, indem sie z.B. eine rote Linie unterhalb fehlerhaften Stellen anzeigen. Sie ändern dabei aber im Gegensatz zu Textverarbeitungen nichts am Quellcode.

Fortgeschrittene werden Ihnen jetzt empfehlen, doch sofort zu einer IDE wie **Eclipse** zu greifen, da die doch so viel besser zum Programmieren seien. Doch bitte lassen Sie das vorerst: Wie schon zuvor geschrieben haben selbst kostenlose IDEs inzwischen einen derartigen Umfang an Komfortfunktionen, dass sie für Einsteiger eher verwirrend als hilfreich sind. Es ist aber richtig: Nach spätestens sechs Monaten sollten Sie auf jeden Fall damit beginnen, eine IDE zu nutzen.

3.3 HTML, PHP und MySQL – Vorbereitung für die Entwicklung verteilter Anwendungen

Wenn Sie eine Webpage entwickeln wollen, dann benötigen Sie dazu folgende Dinge:

1. Einen einfachen Texteditor wie **Notepad++**.
2. Ein Softwarepaket, das Ihnen die nötige Infrastruktur für eine webbasierte Anwendung bietet. Beispiele: **XAMPP** oder **EasyPHP**
3. Eine Software, die es Ihnen ermöglicht, Dateien auf einen Webserver zu übertragen. Ein einfaches kostenloses Programm ist **FileZilla**.

Im Gegensatz zur imperativen Programmierung müssen Sie sich hier nur um wenige Dinge kümmern, obwohl hier tatsächlich wesentlich mehr Installationsschritte zu erledigen sind, als das bei C, C++ oder Java der Fall ist. Denn all die nötigen Schritte übernimmt hier die Installationsroutine

von XAMPP bzw. EasyPHP.

Da Sie alle drei Programme im Netz legal kostenlos beziehen können und die Installation keine fortgeschrittenen Kenntnisse erfordert, sei dazu an dieser Stelle nichts weiter gesagt.

Sie sollten allerdings noch wissen, welche Sprachen und Strukturen hier zum Einsatz kommen. Wie Sie zu Beginn des Kapitels über verteilte Anwendungen erfahren werden, werden Sie diese Punkte später in Veranstaltungen wie „Einführung in relationale Datenbankenäusführlich kennen lernen.

1. Zunächst benötigen Sie für eine Webanwendung einen Server, der die Daten der Seite vorhält und über das Internet erreichbar ist.
   An dieser Stelle empfehle ich Ihnen die Installation von EasyPHP. EasyPHP wird (genau wie XAMPP) Ihren Rechner so vorbereiten, dass der Apache Server auf Ihrem Rechner läuft und als Webserver genutzt werden kann. Aus Sicherheitsgründen sollten Sie diesen Server aber nur so lange laufen lassen, wie Sie ihn zum Testen benötigen. Denn wenn Sie keine fortgeschrittenene Kenntnisse in IT-Sicherheit und Serveradministration haben, werden Sie sonst mit höchster Sicherheit Sicherheitslücken auf Ihrem Rechner einrichten.

2. Dann benötigen Sie eine Sprache, mittels derer Sie die Elemente Ihrer Webanwendung definieren können. Über diese Sprache regeln Sie also, wie die einzelnen Bestandteile der Webpage zu einzelnen Ansichten verbunden werden. Hier greifen wir zu **HTML**, für das Sie im Gegensatz zu C, C++ oder Java keinen Compiler oder Interpreter installieren müssen: Den HTML-Interpreter finden Sie bereits in Form eines Webbrowsers auf praktisch jedem Rechner vor.
   HTML hat allerdings einen entscheidenden Nachteil für unsere Zwecke: Die Sprache ist durchgehend statisch. Wenn Sie also Änderungen auf der Seite einführen wollen, die beispielsweise von den Eingaben des Nutzers abhängen, dann können Sie das mit HTML alleine nicht erreichen.

3. Deshalb folgt als nächstes eine imperative Sprache, wobei wir hier **PHP** wählen, das im Gegensatz zu C und ähnlichen dynamisch typisiert ist. Sie werden sich erinnern: Das bedeutet, dass der Interpreter der Sprache vieles für Sie übernimmt, dass Sie also weniger Details selbst programmieren müssen. Dafür müssen Sie aber umso genauer bzw. umso mehr darüber Bescheid wissen, wie der Interpreter genau arbeitet.

PHP wird wird von vielen Webservern interpretiert. In unserem Fall übernimmt das der Apache Server, der Teil der Easy-PHP- bzw. der XAMPP-Installation ist.
Mittlerweile wird allerdings JavaScript mehr und mehr zu einem ernstzunehmenden Konkurrenten von PHP, weil es wesentlich mehr Möglichkeiten bietet und eine Vielzahl Beschränkungen dort nicht existieren. In HTML5 ist es als Standard für die Programmierung der Funktionalität einer Webanwendung eingestellt, sodass Sie hier zum Testen nicht unbedingt einen Webserver benötigen.

4. Außerdem brauchen wir noch eine Möglichkeit, um Daten langfristig zu speichern. Denn der Apache Server stellt zwar unsere Webpage bereit, HTML realisiert mit CSS die Darstellung und PHP realisiert die dynamische Nutzbarkeit der Anwendung, aber wenn wir nur diese Sprachen nutzen, können wir Nutzereingaben nicht dauerhaft speichern und wieder verwenden.
   Gerade wenn wir eine Webanwendung entwickeln, über die wir Verträge abschließen wollen (egal, ob es dabei um ein browserbasiertes Spiel, einen Webshop oder was auch immer geht), genügt das nicht. Und das führt uns direkt zu einer Datenbank. Datenbanken werden über eigene Sprachen angesprochen, die häufig ein SQL vorkommt. SQL steht kurz für **Structured Query Language**, übersetzt sind das also standardisierte Abfragesprachen. Und hier ist **MySQL** die Sprache, die wir im Kurs verwenden.
   Aber auch hier brauchen Sie wieder keinen Interpreter oder Compiler herunterladen; diese Aufgabe übernimmt bereits XAMPP bzw. EasyPHP für Sie.

Trotz der komfortablen Installation z.B. durch EasyPHP müssen Sie sich all diese Komponenten merken, weil Sie bei einer professionellen Entwicklung all diese Komponenten selbst installieren und konfigurieren müssen. Und schließlich geht es im Studium darum, dass Sie sich auf eine professionelle Tätigkeit vorbereiten.

3.4 Alle - Vorbereitung für die Teamarbeit

Wie im ersten Kapitel aufgeführt ist die Arbeit an einer Software heute eine Aufgabe, die in Teams durchgeführt wird. Und hier müssen Sie Werkzeuge nutzen, die über einen Server koordiniert werden, um auch nur ansatzweise effizient zu sein. Wenn sie dagegen die Vorstellung haben, Ihren Teil der Software zu entwickeln, dann eine Kopie davon an Ihre Teamkollegen zu verschicken und sich anschließend die Kopien der Arbeitsergebnisse der

Kollegen zu besorgen, dann sollten Sie sich schleunigst von diesem Dilletantentum verabschieden! So arbeiten ausschließlich Personen, die nichts aber auch gar nichts von Softwareprojekten verstehen.

Die aktuell effizienteste Möglichkeit, um ein solches Softwareprojekt über eine SCM zu verwalten heißt **Git**. Und diese Möglichkeit ist nicht nur außerordentlich effizient und praktisch, sondern obendrein noch kostenlos. Deshalb werden wir Sie hier von Anfang an einsetzen, damit Sie am Ende Ihres Studiums nicht einmal ansatzweise auf die Idee verfallen, Änderungen an Softwareprojekten per Kopie zu verteilen.

Im Arbeitsleben werden Sie allerdings häufig auf eine ältere Form des SCM treffen. Die Bezeichnung dafür lautet **SVN** bzw. **Subversion**. Subversion hat deutliche Nachteile gegenüber Git, ist aber immer noch weit verbreitet, weil viele Nutzer schlicht nicht willens sind, sich in die Änderungen einzuarbeiten, die mit einem solchen Umstieg verbunden sind. Media Systems Studierende sollten sich deshalb in jedem Fall später (z.B. in der Veranstaltung Software Engineering) auch in Subversion einarbeiten.

Wie schon im ersten Kapitel beschrieben besteht der große Vorteil von Git darin, dass Sie auf das Repository keinen Zugriff haben müssen, um an der Software zu arbeiten; Subversion geht im Kern davon aus, dass Sie sich in einem Unternehmensnetzwerk befinden und deshalb ständigen Zugriff auf das Repository haben.

Neben Git gibt es noch eine Webplattform mit dem Namen **GitHub**. Git ist an Software das einzige, das Sie installieren müssen, um ein SCM zu beginnen. Für die Speicherung des Repository können Sie dann einen beliebigen Rechner nutzen, so lange dieser über das Internet oder ein anderes Netzwerk regelmäßig erreichbar ist. GitHub selbst ist ein Service, der Ihnen anbietet, dass Sie dort Ihre Repositories lagern. Ähnlich wie andere Cloud-Dienste gilt auch hier, dass Sie bei einem kostenlosen Account keine privaten Repositories erhalten. U.U. gibt es hier aber für Studierende Sonderangebote.

Wichtig: Sollten Sie GitHub (oder einen ähnlichen Dienst) nutzen und Ihre Repositories öffentlich sein, dann bedeutet das auch, dass Ihre Projekte für jedermann/-frau frei zugänglich sind. Nutzen Sie dagegen Git mit einem Server, für den Sie Zugangsbeschränkungen erlassen können, dann gilt das natürlich nicht (automatisch).

Das praktische bei Git ist, dass Sie gar keinen Server für die Repositories brauchen, um anzufangen: Sobald Sie die Software heruntergeladen (`https://git-scm.com/`) und installiert haben, können Sie jedes be-

liebige Verzeichnis unter die Versionskontrolle stellen und können damit arbeiten, als gäbe es bereits ein Repository. Sobald Sie den nötigen Web- bzw. Cloudspace haben, können Sie Ihr/e Projekt/e dort in ein Repository einbinden, bzw. aus ihrem/n Projekt/en heraus ein Repository anlegen.

Die Einführung in die Arbeit mit Git auf der Webpage git-scm.com ist sehr gut, weshalb an dieser Stelle keine weiteren Erläuterungen erfolgen. Nur so viel: Arbeiten Sie sich dort ein, damit Sie von Beginn an damit arbeiten können.

3.5 Alle – Nutzung des Netzlaufwerks zur Speicherung eigener Daten

Leider wissen viele Studierende nicht, dass ihnen in der Hochschule ein Netzlaufwerk zur Verfügung steht, auf dem Sie alle Daten speichern können und speichern deshalb Ihre Ergebnisse „auf dem Rechnerän dem Sie in der Hochschule sitzen. Dabei werden die Daten aber gerade nicht auf dem Rechner gespeichert, an dem sie sitzen, sondern in ihrem Rechnerprofil. Und jedes Mal, wenn Sie sich in der Hochschule an einem Rechner anmelden wird dieses Profil über das Netzwerk auf den Rechner übertragen, an dem Sie sich anmelden. Wir hatten schon Studierende, die deshalb eine halbe Stunde und länger an ihrem Rechner saßen, bis sie endlich den Desktop sahen. Aber das liegt nicht etwa daran, dass die Rechner oder das Netzwerk langsam wären, sondern einzig und alleine daran, dass diese Studierenden zum Teil 30 und mehr Gigabyte an Daten in Ihrem Rechnerprofil gespeichert hatten. Nutzen Sie deshalb bitte für alle Arbeiten am Rechner Ihr Netzlaufwerk. Denn die Daten, die dort gespeichert werden werden erst dann auf Ihren aktuellen Rechner übertragen, wenn Sie sie benötigen. Und keine Sorge: Die Daten werden nicht über das WLAN übertragen, sind also im Regelfall so schnell auf Ihrem Rechner, als hätten Sie sie auf einem USB-Stick gespeichert.

An dieser Stelle auch ein Hinweis auf die Nutzung des WLANs: Leider verstehen viele Studierende nicht, dass jeder Aufruf einer Webpage eine gewisse Datenmenge über das Netzwerk überträgt, über das sie mit dem Internet verbunden sind. Und wenn nun fünfzig Studierende gleichzeitig ein Dutzend YouTube-Videos starten oder den neuesten Client für World of Warcraft herunterladen, dann bedeutet das eben, dass das Netz bis an die Grenze ausgelastet ist. Das verstehen Sie nicht? Nun, es ist genau wie im Straßenverkehr: Wenn jede/r mit dem Auto zur Arbeit fährt, dann kommt eben keine/r mehr richtig voran. Und dieser Vergleich entspricht genau dem, was beim Surfen im Netz das Problem ist: Die meisten Nutzer verhalten sich, als wenn sie alleine auf der Welt wären und als wenn Ressour-

cen unbegrenzt vorhanden wären. Wenn es dann einen Engpass gibt, dann heißt es immer: Das Netz ist schlecht, die Stadt soll mehr Straßen bauen, usw. usf. Doch die Ressourcen sind begrenzt. Ja, das gilt auch in den LTE-Netzen, egal was der Verkäufer Ihres Handy-Providers Ihnen versprochen hat.

Kapitel 4

Höhere Programmierung

Der Begriff der höheren Programmiersprachen wird heute eigentlich nur noch am Rande verwendet, weil die Abgrenzung zur maschinennahen Programmierung (und genau dafür steht der Begriff) zum Normalfall geworden ist. Sollten Sie also darüber stolpern und sich fragen, was denn eine höhere Programmiersprache ist, dann merken Sie sich einfach: Dazu zählen alle imperativen Programmiersprachen außer Maschinensprache oder Assembler. Das mag im Detail nicht ganz zutreffend sein, genügt aber für den Moment vollkommen.

4.1 Nach B kam C

Die mangelnde Verständlichkeit von maschinennahen Programmen führte dazu, dass Programmiersprachen entwickelt wurden, die Befehle und Zeilenstrukturen beinhalteten, die leichter lesbar waren.

Die Zeile

```
if (a < b) then print "a ist kleiner als b"
```

dürfte auch von Menschen lesbar sein, die lediglich über grundlegende Englischkenntnisse, aber kaum über Computerkenntnisse verfügen.

Im Gegensatz dazu dürfte die Zeile

```
CMP R6 MSP
```

selbst bei denjenigen unter Ihnen für Stirnrunzeln sorgen, die bereits Projekte in Java oder C++ entwickelt haben. (Hier handelt es sich um eine maschinennahe Programmzeile, die bei einem ARM-Prozessor einen Vergleich

zwischen zwei Zahlen durchführt.)

Eine dieser höheren Sprachen wurde von ihrem Entwickler **Dennis Ritchie** schlicht **C** genannt. Es gab vorher unter anderem eine Sprache namens B, die als Vorlage für C diente. Die Informatiker der Anfangszeit waren weniger an Marketing interessiert, weshalb Bezeichnungen wie Java, Ruby, Python usw. erst ab den 90er Jahren üblich wurden. Vorher wurden häufig einzelne Buchstaben oder Abkürzungen wie im Falle der Sprache **PROLOG** genutzt, was schlicht für programmable logic steht.

C ist bis heute eine sehr wichtige Sprache, weil sie dafür entwickelt wurde, um **Betriebssystem**e zu entwickeln und dabei möglichst wenig maschinennah programmieren zu müssen. Zusätzlich können Sie mit ihr grundsätzlich jede Form imperativer Programme entwickeln. Für Sprachen, die wie C für alle möglichen Zwecke eingesetzt werden können, wird die Bezeichnung **general purpose programming** (kurz GPP) verwendet.

Das Buch „**The C programming language**"von **Kernighan** und **Ritchie** ist eines der ersten Bücher zur Einführung in die Programmierung mit C. Es ist eher schwer zu nutzen, aber wenn Sie sich durch diesen Band durchgearbeitet haben, dann beherrschen Sie die Grundlagen der imperativen Softwareentwicklung, die InformatikerInnen beherrschen müssen.

Wenn Sie in einer Statistik nachsehen, wie viele Programmierer C nutzen, dann werden Sie feststellen, dass diese einen immer geringeren Anteil aller Programmierer ausmachen. Das hat damit zu tun, das C für die Entwicklung verteilter Anwendungen relativ wenig Unterstützung anbietet. Aber denken Sie deshalb nicht, C sei belanglos geworden; es gibt schlicht wesentlich mehr Bereiche, in denen heute programmiert wird, als in den 70er Jahren, in denen C entwickelt wurde.

Eine **verteilte Anwendung** ist nichts anderes als ein Programm, das auf mehreren miteinander vernetzten Rechnern aktiv ist. Die Probleme, die dabei durch die Kommunikation zwischen den Rechnern entstehen sind eines der anspruchsvollsten Themen, mit denen Sie sich auseinander setzen können.

Kontrolle
Höhere Programmiersprachen sind Programmiersprachen, die für Menschen leichter lesbar sind, als das bei Assembler der Fall ist. C ist hier einer der wichtigsten Vertreter, auch wenn es insbesondere bei der Entwicklung von verteilten Systemen eher nicht eingesetzt wird.

4.2 C++ : C mit Objektorientierung

Auch wenn höhere Programmiersprachen übersichtlicher und verständlicher als maschinennahe Programmiersprachen sind, ändert das nichts daran, dass irgendwann der Punkt erreicht ist, an dem auch sie nicht genug Übersichtlichkeit bieten. Vielen Informatikern war das bereits in der Frühzeit der Programmierung klar. Seit Mitte der 50er Jahre wurden deshalb immer neue Konzepte erarbeitet, die dann die Basis für verschiedene Sprachen bildeten, die mehr Strukturierungsmöglichkeiten beinhalten, als das bei C der Fall ist.

Für die Zwecke dieser Einführung soll es genügen, wenn Sie wissen, dass C++ der Sprache C entspricht, aber zusätzlich Möglichkeiten zur objektorientierten Programmierung bietet. Wenn nun von **objektorientierter Programmierung** die Rede ist, dann gibt es das Problem, dass es hierfür zwei Interpretationen gibt, die zu gänzlich unterschiedlichen Programmierstilen führen:

4.2.1 Objektorientierung nach Alan Kay

Alan Kay war ein Forscher am **MIT**, der den Begriff der Objektorientierung mitprägte aber diese Bezeichnung später als einen großen Fehler bezeichnete. Denn was er meinte war eine Programmierung, bei der der Fokus auf dem **Nachrichtenaustausch zwischen virtuellen Objekten** liegt. Wohlgemerkt, der Fokus liegt auf dem Nachrichtenaustausch, nicht auf den Objekten selbst.

Wenn Sie sich jetzt daran erinnern, was das wichtigste bei einem Computer ist (die Datenübertragung zwischen den Komponenten des Rechners), dann verstehen Sie auch, warum dieses Konzept der logische Schluss ist. Wenn Sie dann noch an den Aufbau des Internet denken, dann können Sie sich vorstellen, wie grundsätzlich und vorausschauend dieses Konzept ist. Und nochmal: In diesem Bereich kommen wir nur dann zu sinnvollen und effizienten Programmen, wenn **Informatik**erInnen und **Nachrichtentechnik**erInnen zusammen arbeiten.

Aufgabe:
Können Sie jetzt nachvollziehen, warum Kay die Bezeichnung Objektorientierung als großen Fehler bezeichnet hat?

Dieser Begriff suggeriert, dass die virtuellen Objekte das wichtige sind und lassen naive ProgrammiererInnen die Bedeutung des Nachrichtenaustauschs vergessen. Da wäre der Begriff des **Message Sending** wesentlich passender gewesen. Aber so ist das eben, wenn ein neues Konzept entwickelt wird; da wird eine einprägsame Bezeichnung genutzt und dann gerät alleine da-

durch das eigentliche Konzept in Vergessenheit.

Aus diesem Grund sind auch praktisch alle Programmiersprachen, die im Internet zum Einsatz kommen für dieses Einsatzgebiet praktisch nicht geeignet: Für die Probleme beim Nachrichtenaustausch, namentlich zeitliche Verzögerungen und Verluste bieten sie im Regelfall nur beschränkte Lösungsmöglichkeiten, was dementsprechend eher zu mittelmäßigen Programmen führt. Und hier gilt wieder, dass InformatikerInnen und NachrichtentechnikerInnen leider kaum zusammen arbeiten. Täten Sie das, dann würden just die Probleme wesentlich besser gehandhabt werden, die beim programmierten Nachrichtenaustausch auftreten: Die NachrichtentechnikerInnen würden die Probleme bei der Datenübertragung sinnvoll lösen und die InformatikerInnen würden die Probleme bei der Softwareentwicklung sinnvoll lösen.

Eine Sprache, die Message Sending bzw. Objektorientierung nach Kay umsetzt, heißt **Erlang**. Es handelt sich hier um eine Sprache, die mehrere Paradigmen unterstützt. Richtig gelesen: Es gibt Sprachen, die mehrere **Paradigmen** umsetzen. Und tatsächlich ist das bei den meisten Sprachen der Fall. **Java** war beispielsweise bis zur Version 7 eine rein imperative und klassenbasiert objektorientierte Sprache. Seit Version 8 beinhaltet Sie mit der funktionalen Programmierung aber auch ein Konzept der deklarativen Programmierung. Die Version 9 wird kein neues Paradigma einführen, sondern es wird eine massive Restrukturierung geben, die den Speicherbedarf von Java-Programmen deutlich reduzieren wird, die aber auch bei der Programmierung in Java Folgen haben wird.

4.2.2 Objektorientierung nach Lieschen Müller

Damit kommen wir jetzt zu dem, was heute üblicherweise unter Objektorientierung verstanden wird:

Anstelle eines Programms entwickeln wir im Kern lauter kleine Programme, die jeweils einen gewissen Funktionsumfang anbieten und grundsätzlich als **Klasse**n bezeichnet werden. Soll eine bestimmte Funktionalität genutzt werden, erhält die Klasse, die sie enthält einen entsprechenden Befehl, was dann als Methodenaufruf bezeichnet wird. Bitte beachten Sie: Ein Methodenaufruf ist mehr als nur ein einfacher Befehl, aber zu den Unterschieden kommen wir bei der Einführung in die imperative Programmierung, wenn wir uns die sogenannten Funktionen ansehen.

Und auch wenn Klassen, Methoden und Methodenaufrufe zentrale Themen der objektorientierten Programmierung sind, hat dieses Verständnis ungefähr so viel mit Objektorientierung zu tun, wie das Verleimen zweier

Holzleisten mit dem Tischlerhandwerk: Es gibt noch wesentlich mehr, was Sie verstanden haben müssen, um wirklich zu verstehen, was Objektorientierung ist.

Das ist auch der Grund, warum man mit der Einführung in die Objektorientierung bereits eine vollwertige Vorlesung für ein oder zwei Semester füllen kann.

Kontrolle
Wenn C Programme zu umfangreich werden, kann man auf C++ zurückgreifen, da es den gleichen Umfang an Befehlen und Strukturen bietet, aber mit der Objektorientierung weitere Strukturierungsmöglichkeiten anbietet. Java ist ebenfalls in diesem Sinne eine objektorientierte Sprache. Beachten Sie bitte, dass bei allen dreien Objektorientierung nicht im Sinne von Alan Kay umgesetzt und angewendet wird, auch wenn das durchaus möglich wäre.

Darüber, was das im Detail bedeutet und wozu es gut ist, haben Sie jetzt noch nichts erfahren. Zerbrechen Sie sich da also bitte nicht den Kopf. Es braucht im Regelfall mehrere Jahre, um diese Punkte verinnerlicht und weitgehend verstanden zu haben.

4.3 Java – C++ ohne maschinennähe

C und damit C++ bieten wie beschrieben die Möglichkeit recht nah an der Maschine zu programmieren, auf der ein Programm laufen soll. Das bringt einen großen Nachteil mit sich: Angreifer können durch geschickt entwickelte Programme in laufende Prozesse eingreifen. Außerdem muss ein C bzw. C++ Programm individuell auf jeden Prozessor zugeschnitten werden, auf dem es laufen soll. Bei der Vielzahl an Prozessoren, die heute in mobilen Endgeräten zum Einsatz kommt ist aber genau dieser letzte Punkt ein ernstes Problem: Nicht nur müssen die Entwickler die Software für jeden Prozessor anpassen, sie müssen insbesondere die Details jedes dieser Prozessoren kennen, sonst entwickeln Sie im besten Falle ineffiziente, im schlimmsten Fall leicht angreifbare Programme. Und wer möchte schon, dass die neueste App ein Einfallstor für Viren und Trojaner wird?!

Deshalb wurde u.a. **Java** entwickelt: Zu einem Zeitpunkt, zu dem nicht nur für Frau Merkel das Internet Neuland war (Anfang der 90er Jahre) entwickelte ein Team bei **Sun Microsystems** diese neue Sprache zusammen mit einem passenden mobilen Endgerät. Um Entwicklern die Umgewöhnung zu erleichtern, wurden viele Konventionen und Regeln in Java so umgesetzt, wie das bereits in C und C++ der Fall war.

Wenn Sie also bislang dachten, **Apple** sei das Unternehmen, das (mit dem iPhone) das erste Smartphone entwickelt hat, dann liegen Sie schlicht falsch. Hier wie in mehreren anderen Fällen hat Apple (genau wie **Microsoft**) ein Konzept, das andere bereits zuvor ausgearbeitet hatten schlicht zum richtigen Zeitpunkt in einem Produkt umgesetzt und es zum Verkauf angeboten, als es ausreichend Menschen gab, die bereit waren, dafür Geld auszugeben. Das gleiche gilt für grafische Nutzeroberflächen und die Bedienung eines Computers mit der Mouse. Die wurden ebenfalls nicht von Apple entwickelt, sondern von einem Unternehmen namens Xerox Parc. Allerdings kam dort (im Gegensatz zu Steve Jobs, der das Gelände besuchte) niemand auf die Idee, dass mit so etwas Geld verdient werden könnte.

Übrigens ist auch die Möglichkeit, ein Javaprogramm unverändert auf unterschiedlichen Systemen zu nutzen ein Kriterium der Objektorientierung. Dabei spricht man von **Portabilität**.

Insbesondere bei Entwicklern, die vorrangig in C oder C++ aber auch in anderen imperativen Sprachen entwickeln, herrscht bis heute das Vorurteil vor, Java sei eine viel zu langsame Sprache und deshalb überflüssig, ja generell sei **Objektorientierung** unsinnig.

Hier sollten Sie sich merken, dass es im Regelfall keine unsinnigen Sprachen gibt; **Sprachen werden entwickelt, um einen bestimmten Zweck zu erfüllen.** Ist dieser Zweck tatsächlich nützlich und ist die Sprache sinnvoll und für den Zweck effizient konzipiert, dann wird sie im Regelfall einige Jahrzehnte verwendet. Wer dann einer solchen Sprache die Sinnhaftigkeit abspricht zeigt damit lediglich, dass er den Zweck nicht versteht. Und natürlich kann Java nicht die Geschwindigkeit einer Sprache wie C++ erreichen: Java übernimmt die Arbeit, jedes Programm auf einer möglichst großen Anzahl von Rechnern und Smartphones laufen zu lassen. Das bedeutet einen teilweise höheren Aufwand und damit laufen diese Programme in Java langsamer als in C++, wenn es fähige C++-ProgrammiererInnen in C++ umsetzen. Andererseits müssen diese eben auch sehr fähig sein und umfangreiche Kenntnisse über die Unterschiede zwischen rund dreißig Betriebssystemen und Prozessoren kennen und sich kontinuierlich in neue Systeme einarbeiten. Da das kaum jemand leisten kann, der als Entwickler bezahlbar ist, werden die meisten Spiele nur für ein System entwickelt oder sie sind nicht gut auf die einzelnen Systeme angepasst.

Einige von Ihnen werden jetzt einwenden, dass es doch von **Steam** eine Plattform gibt, auf der Spiele unabhängig vom System laufen. Hier gilt das gleiche, was schon bei Java gilt: So lange diese Spiele nicht individuell für jede Plattform entwickelt werden, kann auch nicht die volle Palette an Möglichkeiten genutzt werden, die das System bietet. Also werden einige

Spiele auf dieser Plattform langsamer laufen als wenn Sie speziell an das System angepasst wären.

Die meisten Spieleentwickler nutzen heute allerdings keine Programmiersprache mehr, sondern sogenannten **Game Engines**. Das sind Softwarepakete, die bereits eine Vielzahl an **Bibliotheken** beinhalten, sodass die Mitglieder von Entwicklerstudios sich nur noch auf den Ablauf des Spiels konzentrieren müssen und ein Team von Designern für die Grafik und den Sound benötigen. Um mit einer Game Engine zu arbeiten brauchen Sie deshalb kein Informatikstudium mehr abschließen. Im Gegenteil: Da diese Softwarepakete genau das übernehmen, was fähige InformatikerInnen tun, gibt es in der Spielebranche nur wenige Stellen für vollwertige InformatikerInnen. Im Gegenteil: Als Absolvent z.B. von Media Systems ist die Nutzung einer Game Engine eigentlich ein Rückschritt: Das System übernimmt nicht nur vieles, was Sie sonst umsetzen müssten, es verhindert auch vieles, das Sie kennen und schätzen gelernt haben.

Dagegen müssen Sie anspruchsvolle Aufgaben als (Medien-)InformatikerIn erfüllen können, um eine Game Engine zu entwickeln oder Ihren Funktionsumfang zu erweitern. In Ihrem Studium können Sie das später ausprobieren: Sie werden eine Game Engine namens **Blender** kennen lernen. Diese wird von den meisten Studierenden abgelehnt, da die Nutzeroberfläche nicht wie die von vielen Game Engines oder Programmpaketen für Computergrafik aussieht. Tatsächlich ist Blender die wahrscheinlich beste Game Engine für (Medien-)InformatikerInnen: Da Sie hier alles und ohne Beschränkung erweitern oder verändern können und da Blender vollständig kostenlos und frei verfügbar ist, können Sie genau das tun, was Sie später als professionelle EntwicklerIn tun müssten. (Zum Vergleich: Wenn Sie keine akademische Lizenz erhalten, dann zahlen Sie für Engines wie Maya 3D mehrere tausend Euro. Doch selbst wenn Sie die Software erhalten, dürfen Sie daran nahezu nichts ändern.)

Kontrolle
Java ist eine imperative und klassenbasierte objektorientierte Programmiersprache, deren Programme leicht auf andere Systeme portiert werden können. Es unterstützt zusätzlich seit Version 8 die funktionale Programmierung und damit ein deklaratives Paradigma.

4.4 Verteilte Anwendungen

Die drei Sprachen, mit denen wir uns bislang beschäftigt haben setzen nicht voraus, dass unser Rechner sich in einem Netzwerk befindet, und dass es möglich ist, Daten mit den anderen Rechnern dieses Netzwerks auszutau-

schen. Nun wissen Sie aber, dass heute annähernd jedes computerbasierte System (also auch Smartphones) zumindest zeitweilig vernetzt ist. Wie Sie durch die einleitenden Kapitel wissen, wurden Rechner im Regelfall schon immer vernetzt und nur im Heimbereich hatten Nutzer einen Computer ohne Netzzugang. (Hieraus resultiert auch die veraltete Unterteilung in Heimcomputer und PCs.)

Wenn wir nun ein Programm entwickeln wollen, das vernetzte Rechner nutzen soll oder sogar nur bei vernetzten Rechnern einsetzbar sein soll, dann müssen wir die Strukturen, die sich daraus ergeben auch in unseren Programmen integrieren. Ein solches Programm wird übrigens als **verteilte Anwendung** bezeichnet, vor allem wenn es genau genommen aus mehreren individuell agierenden Programmen besteht. Wir müssen dann (siehe Alan Kay und die Objektorientierung) beachten, dass Daten zwischen Rechnern transportiert werden müssen. Und das bedeutet, dass wir eine Absicherung für die Fälle schaffen müssen, in denen Daten nicht das Ziel (also einen anderen Rechner) erreichen oder in denen das Ziel aus irgendwelchen Gründen nicht versteht, was es mit diesen Daten tun soll. Wir müssen insbesondere bei Verbindungen über das Internet auch beachten, dass die Datenübertragung einen Zeitversatz hat, und dass wir keine genaue Zeitabstimmung zwischen den Rechnern realisieren können. Die genauen Ursachen und möglichen Auswirkungen verstehen Sie, wenn Sie Veranstaltungen zum Thema **Netzwerk**e und **Nachrichtentechnik** belegen. Es folgen in Kürze einige Beispiele, um Ihnen einen ersten Eindruck zu vermitteln.

Aus der dafür nötigen Denkweise resultieren auch zwei Begriffe: Server und Client. Die naive Vorstellung lautet hier, dass ein Server ein Rechner im Netz ist, der eine bestimmte Funktionalität anbietet, und dass ein Client ein anderer Rechner im Netz ist, der vom Server eine solche Leistung anfordert. Das ist allerdings nicht richtig; ein **Server** ist lediglich ein Programm, das eine bestimmte Funktion anbietet, und ein **Client** ist ein Programm, das eine Funktion abruft. Sie können also auf einem Rechner verschiedene Server und Clients betreiben, wobei bei Betriebssystemen in aller Regel eine Vielzahl an Servern und Clients aktiv ist. (Hier gibt es noch andere Programmarten über die wir aber erst im Rahmen von Veranstaltungen wie „Betriebssysteme"reden werden.) Einsteiger, die aus der Apple- oder Microsoftwelt kommen sind häufig bei der Installation von Linux überrascht, dass Sie Mailserver und andere Server installieren können, aber Sie wissen jetzt, warum das so ist.

Dennoch werden häufig einzelne Rechner als Server oder Client bezeichnet. Das ist insbesondere dann kein Problem, wenn ein solcher Rechner ausschließlich eine entsprechende Funktion im Netz übernimmt. Aber Sie

wissen jetzt, dass Sie kein Netz benötigen, wenn Sie ein netzbasiertes Programm entwickeln wollen, weil Sie ja auf einem Rechner sowohl den Server als auch den Client betreiben können. Und ja: Sie können dann einen Datenaustausch zwischen Client und Server auf Ihrem Rechner praktisch genauso durchführen, als wenn beide auf unterschiedlichen Rechnern installiert und über ein Netz verbunden wären. Deshalb können Sie z.B. eine Webanwendung, die später im Internet nutzbar sein soll auf Ihrem Rechner entwickeln und sie dort auch testen, selbst wenn keine Internetverbindung vorhanden ist.

Machen Sie sich in solchen Fällen aber bewusst, dass Sie dann die zentrale Fehlerquelle bei verteilten Anwendungen ausblenden: Da Sie keine Daten über das Netz austauschen, wissen Sie nicht, ob die Anwendung am Ende auch tatsächlich so funktioniert, wie Sie sich das vorstellen: Die Datenübertragung kostet Zeit und diese Zeit ist deutlich höher, wenn die Daten über ein Netzwerk übertragen werden, als wenn Sie innerhalb eines Rechners übertragen werden.

4.4.1 Entwicklung von Webanwendungen – MySQL und PHP versus Ruby on Rails

Bevor wir an dieser Stelle weiter machen, hier ein wichtiger Hinweis: Bis vor wenigen Jahren entwickelten die meisten Softwareentwickler Anwendungen, die auf einem System liefen und die eine Netzwerk nur nutzten, um Nachrichten darüber auszutauschen. **Webanwendung**en haben wie alle **verteilten Anwendungen**, mindestens einen Server- und einen Clientteil, die tatsächlich auf getrennten Rechnern aktiv sind. Die einfachste Form von Webanwendungen kennen Sie wahrscheinlich unter dem Namen Internetseiten. Doch das sind nicht einfach nur Dokumente mit Bildern und Videos, die Sie sich auf Ihren Rechner bzw. Ihr Smartphone herunterladen können, sondern es sind immer öfter komplette Anwendungen. Allerdings werden diese Anwendungen zum Teil auf dem Server und zum Teil auf dem Client ausgeführt. Wenn Sie sich intensiver mit diesem Bereich beschäftigen, werden Sie Sprachen wie **PHP** kennen lernen, die ausschließlich als Server bzw. auf einem Webserver genutzt werden können. Allerdings ist dieser Ansatz veraltet: Aktuelle Sprachen wie **JavaScript** können sowohl als Server als auch als Client eingesetzt werden. Das ist allerdings für Einsteiger bzw. Erstsemester meist nur schwer umsetzbar, da Sie hier bewusst und gut begründet entscheiden müssen, auf welche Programmteile Nutzer Zugriff haben dürfen.

Wenn Sie also denken, die Programmierung in **HTML** (der am häufigsten eingesetzten Sprache für Webanwendungen) sei langweilig und man könnte damit nur einfache Internetseiten programmieren, dann liegen Sie falsch;

das war in der Version 4.01 so, die Ende 1999 veröffentlicht wurde. Mit der Version 5, die im Herbst 2014 veröffentlicht wurde, ist dieses Thema endgültig passé: Basierend auf HTML 5 ergänzt um Sprachen wie PHP oder JavaScript können Sie Anwendungen entwickeln, die genau das gleiche leisten wie eine beliebige Anwendung, die Sie auf einem einzelnen Rechner nutzen können. Der Begriff Webanwendung ist im Grunde ein Synonym für den Begriff der verteilten Anwendung.

Zu Beginn dieses Kapitels haben Sie erfahren, dass im Grunde keine Sprachen existieren, die die zentralen Probleme angehen, die bei der Datenübertragung im Netz aufkommen. Der Grund besteht darin, dass für die meisten InformatikerInnen eben das System im Mittelpunkt steht, auf dem eine Software ausgeführt wird. Die Kommunikationswege dazwischen und der Zeitfaktor bei der Übertragung werden im Grunde immer nur als lästiges Übel angesehen oder gleich gänzlich ignoriert. Das gleiche gilt für die Nutzung von Webanwendungen durch Menschen.

Ein anschauliches Beispiel konnten Sie bei ebay in der Anfangszeit erleben: Damals hatten die Entwickler ignoriert, dass kurz vor Abschluss einer Auktion besonders viele Aufrufe und Gebote für ein Angebot erfolgten. Also konnten die Server gar nicht alle Angebote „sofort"verarbeiten. Dementsprechend wurde es zu einer Art Glücksspiel, ein Gebot kurz vor Versteigerungsschluss abzugeben: Unter Umständen wurde Ihr Gebot scheinbar ignoriert, weil es erst nach Auktionsende von den ebay-Servern verarbeitet werden konnte. Dieser Fehler im System wurde inzwischen soweit als möglich bereinigt. Dies ist außerdem ein Beispiel für die Bedeutung des Begriffs **Skalierbarkeit**.

Damit wieder zurück zu den eingangs genannten Sprachen: **Ruby on Rails** ist nicht die Sprache selbst, sondern ein Framework namens Rails, das Ruby so erweitert, dass Sie damit **Webanwendungen** entwickeln können.

Eine zweite Möglichkeit (und deutlich älter), um Webanwendungen zu entwickeln besteht in der Kombination aus drei Sprachen: **MySQL** ist eine Sprache, mit der Sie Datenbanken nutzen können und **PHP** ist eine imperative Sprache, mit der Sie die Funktionalität von Elementen einer Webanwendung programmieren können. Dazu kommt noch **HTML**, was eine **Markup Language** ist. Markup Languages sind Programmiersprachen mittels derer sich die Struktur von Anwendungen unabhängig von der Darstellung und der Funktion programmieren lassen. HTML ist eine Markup Language mit der sich die Struktur einer Webanwendung und seit Version 5 die Bedeutung der Inhalte programmieren lässt.

Viele Softwareentwickler reden in Bezug auf Markup Languages vom so-

genannten **Script**en. Für diesen Begriff gibt es keine präzise Definition. Wenn ProgrammiererInnen ihn benutzen, dann geht es in aller Regel um etwas, das zwar programmiert werden muss, damit eine bestimmte Aufgabe erfüllt wird, das aber vom jeweiligen Entwickler keinerlei logisches Denkvermögen erfordert. Sie müssen im Falle vom Scripten also nur verschiedene Befehle aneinander reihen oder ineinander verschachteln, ohne sich weiter Gedanken darüber zu machen, wie die miteinander interagieren: Sie tun es schlicht nicht. Teilweise wird auch bei Konfigurationsdateien vom scripten gesprochen, obwohl hier (im Gegensatz zu HTML4.01 oder LaTeX) sehr viel Grundlagenwissen nötig ist. Wenn Sie beispielsweise nicht genau wissen, was der Unterschied zwischen SSH und SSL ist, dann sollten Sie von der Konfiguration eines Servers die Finger lassen.

Haben Sie im ersten Semester eine Veranstaltung zum Webpage Development besucht, dann können Sie eine Webpage entwickeln, denn das ist gar nicht so schwer. Aber viele Konzepte und Abläufe werden Ihnen kaum klar werden. Wenn Sie allerdings die nötige Zeit investieren, dann werden Sie sich basierend auf dieser Veranstaltung die Grundlagen erarbeiten können, um eine vollwertige Webanwendung zu entwickeln.

Ein Tipp für den Fall, dass Ihnen jemand zu **Ruby on Rails** rät: Ja, es ist richtig, dass Rails ein großartiges Framework ist, mit dem Sie selbst komplexe Anwendungen für multinationale Konzerne entwickeln können. Aber Sie merken es schon an der Formulierung: Es ist für Einsteiger schlicht zu komplex und die Vielzahl an Optionen, mit denen Sie von Beginn an konfrontiert werden, lenkt Sie von den Punkten ab, die Sie als Einsteiger verinnerlicht haben müssen. Hier würde ich eher empfehlen, dass Sie sich in **HTML5** und **JavaScript** einarbeiten. Leider sind aber die meisten Anleitungen im Netz (selbst wenn dort die Rede von HTML 5 ist) immer noch Einführungen in HTML 4, bei denen praktisch alles ignoriert wird, was an Version 5 so großartig ist. Teilweise werden hier sogar Techniken vermittelt, die bereits in der Version 4 als schlampig galten. Der Grund ist recht simpel: Wie so oft erklären dort Menschen die Programmierung, die zwar die alte Version beherrschen, aber die schlicht zu faul oder dumm sind, um zu erkennen, dass Version 5 keine kleine Erweiterung um ein paar nette Effekte ist, sondern eine vollständige Überarbeitung, bei der Aspekte berücksichtigt wurden, die nirgends in Version 4 auftauchen und auch nichts mit dem zu tun haben, was in Version 4 vorhanden ist.

Sie fragen, was eine Datenbank ist? Wie so oft, wenn **Informatik**erInnen es mit gleichartigen Daten oder Abläufen zu tun haben, entwickeln Sie entsprechende Strukturen, die letztlich dazu dienen, Fehler zu reduzieren und Abläufe effizienter zu gestalten. **Datenbank**en sind eine weitere Lösung, die so entstanden ist: Wann immer es um große Mengen gleich-

artiger oder gleichartig strukturierter Daten geht, die nach verschiedenen Kriterien untersucht oder geändert werden müssen, wird eine Datenbank verwendet. **Relationale Datenbanken** bestehen dabei aus Tabellen, bei denen jede Spalte einem Kriterium entspricht und jede Zeile einem sogenannten **Datensatz**. Beispielsweise würde bei einer relationalen Kundendatenbank jede Zeile einem Kunden entsprechen und Einträge in den Spalten wären nach Aspekten wie Name, Vorname, Anschrift, usw. unterteilt.

Bei Webanwendungen dienen Datenbanken (wie bei allen Anwendungen) verschiedenen Zwecken. Zum einen wäre da die klassische Kundendatenbank. Dann gibt es Datenbanken, in denen Einträge auf den Webanwendungen verwaltet werden. Auch die Speicherung jedes Klicks und jeder Taste, die NutzerInnen gedrückt haben wird mit einer Datenbank realisiert. Aber es gibt noch wesentlich mehr Einsatzmöglichkeiten. Wie oben genannt geht es schlicht darum, große Mengen gleichartiger Daten in einer strukturierten Form aufzubewahren, um möglichst schnell darauf zuzugreifen.

Kontrolle
Beginnen Sie beim Webapplication Development mit einer Einführung in MySQL und PHP sowie HTML5. Wenn Sie die objektorientierte und funktionale Programmierung beherrschen, dann können Sie auch JavaScript anstelle von PHP verwenden. Allerdings ist hier ein häufiger Fehler, dass dann HTML nur noch dazu genutzt wird, die JavaScript-Anwendung zu starten, sodass sie in einem beliebigen Browser genutzt werden kann. Ruby on Rails ist ein sehr mächtiges Werkzeug aber für Einsteiger nur beschränkt empfehlenswert. Dazu kommt, dass Rails häufig in Kombination mit weiteren Frameworks verwendet wird, was den Einstieg zusätzlich erschwert. Außerdem ist es leider noch immer nicht so effizient wie JavaScript.

4.5 Konzepte bei der Programmierung

Wie schon mehrfach angeführt gab und gibt es zu jeder Zeit eine Vielzahl von Programmiersprachen, die jeweils für bestimmte Zwecke ausgelegt sind. Zum Teil sind die Unterschiede nur in wenigen Details begründet. Um Ihnen einen kleinen Überblick darüber zu verschaffen, was es noch für Sprachen gibt und wofür diese nützlich sind, folgt eine kleine und dementsprechend unvollständige Aufstellung. Zum Teil werden hier weitere Begriffe eingeführt, die für die Programmierung insgesamt wichtig sind.

4.5.1 Dynamisch versus statisch – Ruby und Python versus C und Java

Ruby und **Python** sind Programmiersprachen, die in Konkurrenz zu Java stehen. Der auffälligste Unterschied besteht darin, dass Ruby **schwach typisiert** ist. (Alternativ spricht man auch von **dynamischer Typisierung**.) Im Gegensatz dazu sind **C** und **Java stark bzw. statisch typisiert**. Beide Varianten haben Vor- und Nachteile. Und leider neigen die meisten Entwickler dazu, die Variante als schlecht zu bezeichnen, die sie als zweites kennen lernen. Wer das tut hat aber leider nichts mit professionellen **Informatiker**Innen zu tun, selbst wenn er/sie in einer Sprache bzw. einem **Paradigma** wirklich gut ist.

4.5.2 Typisierung von Daten

Bislang haben wir lediglich über Codierung gesprochen aber nicht über Typisierung. Wie Sie bereits wissen werden alle möglichen Daten, die sie vom Computer verarbeiten lassen in einer anderen Form gespeichert als die, in der sie angezeigt werden. Wenn wir nun von statischer oder starker Typisierung sprechen, dann bedeutet das, dass Sie bei einem Wert, den Sie programmieren so etwas Ähnliches wie eine Codierung festlegen. Der Typ eines Wertes, den Sie so vergeben wird entsprechend als **Datentyp** bezeichnet.

Einer der ersten Fälle, in denen Sie mit Typisierung zu tun bekommen ist das Rechnen mit ganzzahligen und ganzrationalen Zahlen. Diese werden nämlich vom Rechner unterschiedlich gespeichert: Fließkommazahlen werden nicht in der Form gespeichert, die Sie aus dem Mathematikunterricht kennen, aber eine Einführung in diese Materie überlasse ich den Kollegen der **Technischen Informatik**. Jetzt aber ein Beispiel für die möglichen Varianten, wie eine Programmiersprache mit ganzen Zahlen umgehen kann:

Wenn Sie die Zahl 5 programmieren, als Typ der Zahl ganzzahlig (**Integer**) festlegen und anschließend durch 2 teilen (oder jede andere Zahl ungleich +/- 5 oder +/- 1), dann ergibt sich bekanntlich eine ganzrationale Zahl. Je nach Programmiersprache gibt es nun unterschiedliche Möglichkeiten, was dabei passiert:

1. Bei statisch typisierten Sprachen erfolgt in aller Regel eine Fehlermeldung, denn Sie haben definiert, dass Ihre Zahl ganzzahlig ist. Also muss das Ergebnis ebenfalls ganzzahlig sein. Es gibt dennoch Möglichkeiten, um eine solche Aufgabe in einer solchen Sprache lösen zu lassen. Dazu sind jedoch zusätzliche Programmzeilen nötig.

2. Wenn keine Fehlermeldung erfolgt, ist das das Ergebnis häufig nicht das, was Sie erwarten. In den Fällen, wo die Sprache vorsieht, dass das Ergebnis als ganzzahliger Wert gespeichert wird, wird nun entweder auf- oder abgerundet.

   ABER! Ob auf- oder abgerundet wird, das hat nichts mit den Rundungsregeln zu tun, die Sie aus der Schule kennen: Es gibt also vier Möglichkeiten, wie eine Programmiersprache damit umgeht, wenn Sie eine Division von zwei ganzzahligen Werten einprogrammieren und bei der keine Ganzzahl berechnet wird.

   (a) In der Sprache wird stets abgerundet:
   5 : 2 = **2**.
   -5 : 2 = **-3**,
   denn -2,2 abgerundet ergibt -3 und nicht -2!

   (b) In der Sprache wird stets aufgerundet:
   5 : 2 = **3**
   -5 : 2 = **-2**,
   denn -2,2 aufgerundet ergibt -2 und nicht -3!

   (c) In der Sprache sind die Rundungsregeln enthalten, die Sie aus dem Mathematikunterricht der Schule kennen. Das ist der seltenste Fall.

   (d) Die Sprache gibt in solchen Fällen eine Fehlermeldung oder eine **Exception** aus. Als EntwicklerIn müssen Sie dann das Programm entsprechend korrigieren.
   Wichtig:
   Eine Exception ist KEINE Fehlermeldung. Es ist vielmehr ein Komfortfaktor einzelner Programmiersprachen, der sie darauf hinweist, dass Ihr Programm in bestimmten Fehlern nicht so ablaufen wird, wie Sie das wahrscheinlich erwarten. Je nach Komfort der jeweiligen Sprache gibt es auch Exceptions, die auf Situationen hinweisen, die durchaus wie gewünscht verlaufen können, wo Ihnen die Programmiersprache also quasi den Tipp gibt, zu prüfen, ob Sie dieses Verhalten so haben wollen oder ob das nicht doch ein logischer Fehler ist. In Java haben Sie sogar die Möglichkeit, eigene Exceptions zu programmieren, um bestimmte Ausnahmen ganz bewusst anders verarbeiten zu lassen als das sonst der Falle wäre.

3. Bei **dynamisch typisierten Sprachen** wird der Datentyp je nach Bedarf automatisch von der Programmiersprache angepasst. Hier programmieren wir also in aller Regel nicht den Datentyp. Generell steht der Begriff des **Typecasting** für eine solche Anpassung, die in einigen

statisch typisierten Sprachen möglich ist.

Typecasting gibt es noch für andere Datentypen, es ist also nicht nur auf die Umwandelung des Datentyps bei einer Zahl beschränkt, sondern bei allen denkbaren virtuellen Objekten.

Aber auch beim Typecasting ist das Ergebnis nicht automatisch das, was Sie denken. Das hat wiederum mit der Speicherung von Daten zu tun. Wie Sie wissen basiert die Speicherung von Daten in einem Computer auf Zahlen der Basis 2. Und damit basiert die Speicherung von Nachkommastellen auf Potenzen von $\frac{1}{2}$. In Informatikveranstaltungen werden Sie dazu einige Beispiele rechnen, hier seien nur zwei genannt: $\frac{5}{2} = 2 + 1/2$. Binär lässt sich das in der Form $[10, 1]_2$ darstellen: $(1 \cdot 2) + (0 \cdot 1) + (1 \cdot \frac{1}{2})$. Versuchen wir einmal, die Zahl 0,3 als Binärzahl darzustellen:

$$
\begin{aligned}
&(0 \cdot 1) + (0 \cdot \tfrac{1}{2}) + 0,3 \\
&= (0 \cdot 1) + (0 \cdot \tfrac{1}{2}) + (1 \cdot \tfrac{1}{4}) + 0,05 \\
&= (0 \cdot 1) + (0 \cdot \tfrac{1}{2}) + (1 \cdot \tfrac{1}{4}) + (0 \cdot \tfrac{1}{8}) + 0,05 \\
&= (0 \cdot 1) + (0 \cdot \tfrac{1}{2}) + (1 \cdot \tfrac{1}{4}) + (0 \cdot \tfrac{1}{8}) + (0 \cdot \tfrac{1}{16}) + 0,05 \\
&= (0 \cdot 1) + (0 \cdot \tfrac{1}{2}) + (1 \cdot \tfrac{1}{4}) + (0 \cdot \tfrac{1}{8}) + (0 \cdot \tfrac{1}{16}) + (1 \cdot \tfrac{1}{32}) + 0,01875 \\
&= ...
\end{aligned}
$$

Aber es gibt kein endgültiges Ergebnis. Dementsprechend kann weder der Computer noch die Programmiersprache eine Zahl wie 0,3 richtig darstellen. Er könnte sie als $3 * 10^{-1}$ darstellen, aber da das nur eine begrenzte Anzahl an Spezialfällen aller möglichen ganzrationalen Zahlen löst, ist das keine Lösung, die wir immer nutzen können.) Also wird eine Zahl wie 0,3 nur annäherungsweise gespeichert. Und wenn sie dann ausgegeben wird, kann so etwas wie 0,3000010002 oder 0,298991 herauskommen. Auch hierfür gibt es Lösungen, die aber auch wieder bedeuten, dass Sie zusätzliche Zeilen programmieren müssen.

Wenn wir mit der Programmierung in **C** beginnen, werden Sie häufig mit solchen Fällen zu tun haben. Sie werden dann (wie auch sonst grundsätzlich bei der Programmierung) Lösungen entwickeln müssen, damit der Rechner die Zahlen verwendet und speichert, die für Ihre Aufgabenstellung eine richtige Lösung darstellen.

Dies ist allerdings kein Beispiel dafür, was **Informatik**erInnen von ProgrammiererInnen unterscheidet: Gute ProgrammiererInnen wissen, dass

es solche Probleme gibt, sie kennen die Ursachen und sie entwickeln Programme so, dass diese Probleme in allen Varianten gelöst werden. (Sonst sind es Dilletanten, was leider auf viele Quereinsteiger zutrifft.) Allerdings lernen InformatikerInnen in Ihrem Studium verschiedene systematische Methoden, um solche Probleme effizient anzugehen. Der entsprechende Bereich heißt **Praktische Informatik**, wobei die entsprechenden Veranstaltungen in aller Regel unter Titeln wie **Algorithmen und Datenstrukturen**, **Algorithmendesign** und **Algorithmik** angeboten werden.

Aber zurück zu Ruby: Wie gesagt handelt es sich hier um eine dynamisch typisierte Programmiersprache, während C, C++ und Java statisch typisiert sind. Es spielt keine Rolle, welche der beiden Varianten Ihnen lieber ist, das einzige, was eine Rolle spielt ist, dass Sie langfristig lernen, beide Formen der Typisierung zu beherrschen.

Eine weitere Programmiersprache, die neben Ruby in den letzten Jahren immer bekannter wurde und dynamische Typisierung bietet, ist **Python**.

Kontrolle
Es gibt dynamisch und statisch typisierte Sprachen. Der Unterschied besteht darin, dass bei den dynamisch typisierten Sprachen die Programmiersprache Methoden hat, um den Datentyp eines Wertes automatisch anzupassen. Das ist komfortabel, aber es ist nicht intuitiv. Denn während Sie bei einer statisch typisierten Sprache selbst programmieren müssen, wie ein Typcasting ausgeführt wird, müssen Sie bei jeder dynamisch typisierten Sprache genau wissen, wie diese einzelne Sprache das „automatische"Typecasting durchführt. Wenn Sie das eine oder das andere nicht beherrschen, dann programmieren Sie den Computer nicht, um das zu tun, was er tun soll, sondern Sie programmieren Ergebnisse, die schlichtweg falsch sind. (Und glauben Sie mir, das wollen Sie nicht bei der Steueranlage eines Flugzeugs. . .)

4.5.3 Funktionen – Dynamisch versus statisch

Dieser Abschnitt dürfte auch für die meisten fortgeschrittenen unter Ihnen eine Überraschung beinhalten: Nicht nur Datentypen, sondern auch Funktionen können bei einzelnen Sprachen dynamisch während der Laufzeit eines Programms geändert werden.

Wichtig:
Bitte denken Sie jedoch nicht, dass das Programmieren einer Funktion eine Form der **funktionalen Programmierung** ist: Genau wie bei der **imperativen Programmierung** nutzen Sie dort etwas, das als Funktion bezeichnet wird, um Abläufe aus dem Programm auslagern. Diese Art der Funk-

tionsdefinition sorgt dafür, dass Sie den Inhalt der Funktion an beliebigen Stellen Ihres Programms verwenden können. Bei der funktionalen Programmierung ist eine Funktion dagegen eine Umsetzung des sogenannten **Lambda-Kalkül**s, das wir uns erst bei der Einführung in die **deklarative Programmierung** ansehen werden.

Doch für die Einsteiger zunächst die Erklärung, was eine Funktion ist: Eine **Funktion** fasst in der Programmierung mehrere Programmzeilen zusammen und lässt sich über einen Bezeichner von beliebigen Stellen eines Programms aus aufgerufen werden.

Die meisten Programmierer kennen Funktionen dagegen nur als ein Mittel der Programmierung, das sich nicht ändern kann, während das Programm läuft. Das ist aber ein Irrtum, der darauf basiert, dass das bei Programmiersprachen wie C, C++ oder Java so ist.

Tatsächlich werden Funktionen während des Programmablaufs wie alle anderen Daten eines Programms im Speicher des Rechners abgelegt und bei Bedarf von dort geladen. Und weil der Speicher eines Rechners zu beliebigen Zeiten geändert werden kann, ist es natürlich auch grundsätzlich möglich beliebige Teile eines Programms abzuändern, das dort abgelegt wurde.

Kontrolle
Auch wenn die meisten bekannten Sprachen das nicht können, ist es grundsätzlich möglich, dass auch Funktionen eines Programms dynamisch programmiert werden.

4.6 First-class Objects

Die meisten Programmierneulinge lernen das Programmieren mit Variablen kennen und entwickeln dabei die Vorstellung, dass eine Variable nur einen Wert oder eine Menge an Werten (z.B. die sogenannten Arrays) sein kann. Tatsächlich kann aber auch eine Funktion der Wert einer Variablen sein. Ist das bei einer Programmiersprache der Fall, dann können Sie nicht nur eine Funktion mit einem Wert aufrufen, sondern Sie können eine Funktion quasi wie einen beliebigen Wert an eine andere Funktion übergeben. Das bedeutet, dass Sie dann die Möglichkeit haben, den Ablauf einer Funktion während eines Programmablaufs dynamisch anpassen können.

Diejenigen von Ihnen, die bereits imperativ programmiert haben werden jetzt behaupten, dass das doch klar sei, weil so „schon immer"der Wert einer Funktion an eine Variable übergeben wurde. Damit zeigen Sie, dass Sie

den Absatz missverstanden haben: Dort steht, dass auch eine Funktion als ganzes und eben nicht nur der Wert, den sie berechnet in einer Variablen gespeichert werden kann. Warum das so ist werden wir uns ansehen, wenn wir klären, was genau eine Variable eigentlich ist.

Als Sammelbegriff für alles, was einer Funktion übergeben werden kann wird der Begriff des **first-class Object** verwendet. Wenn also die Rede davon ist, dass in einer Programmiersprache Funktionen first-class Objects sind, dann bedeutet das nichts anderes, als dass Sie in dieser Sprache eine Funktion genauso als Objekt an eine andere Funktion übergeben können, wie Sie das mit einer Variablen gewohnt sind.

Kontrolle
Im Gegensatz zur meist anzutreffenden Überzeugung von ProgrammiererInnen spricht eigentlich nichts dagegen, auch Funktionen als Argumente an Funktionen zu übergeben. Und wenn eine Programmiersprache das unterstützt, dann reden wir davon, dass in dieser Sprache Funktionen first-class objects sind.

4.7 Zusammenfassung

In diesem Kapitel haben Sie einen Überblick erhalten, wie die drei Sprachen C, C++ und Java zusammen hängen und wo die Unterschiede liegen. Sie haben eine Vielzahl an Begriffen kennen gelernt, die bei der Programmierung von Hochsprachen von Belang sind.

Sie haben verstanden, dass es keine beste Sprache oder sinnlose Sprachen gibt, sondern dass Sprachen jeweils für eine bestimmte Problemstellung entwickelt wurden. Deshalb gibt es dann auch ganz unterschiedliche Arten (Paradigmen) des Programmierens und hier haben Sie konkrete Fälle kennen gelernt, um den Begriff des Paradigmas mit Leben zu füllen.

Sie wissen jetzt, dass Sie es gelegentlich mit einem bestimmten Programmierparadigma zu tun haben und teilweise lediglich mit einem Spezialfall, der so nur in einer einzelnen oder bei einigen wenigen Sprachen umgesetzt wird. Diese Kenntnisse sind ein weiterer Unterschied zwischen einem dilletantischen Quereinsteiger und einem ernstzunehmenden Softwareentwickler.

Danach haben Sie etwas über Konzepte erfahren, die C-, C++- und Java-ProgrammiererInnen nicht verstehen und die beispielsweise in Ruby, Python und JavaScript zum Einsatz kommen.

Teil II

Fortsetzung für Studierende der Informatik, Elektrotechnik und verwandter Studiengänge

Der Abschnitt zu den Grundlagen der imperativen Programmierung muss noch an die Latex-Syntax angepasst werden.

Teil III

Einführung in die objektorientierte Programmierung und Softwareentwicklung

Die Abschnitte dieses Teils müssen noch an die LeTeX-Syntax angepasst werden.

Stichwortverzeichnis