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

-

EMIFROP 0.22

Markus Alpers
B.Sc. und Ausbilder f. Industriekaufleute

3. 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`

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.

Stichwortverzeichnis