Diplomarbeit

zur Erlangung des akademischen Grades

”Master of Science in Engineering” / ”Diplomingenieur (FH)”

≪Python und Zope als Unterrichtswerkzeuge≫

ausgeführt von

Dominique Lederer

Strassergasse 8-12/2/3
1190 Wien

1. Begutachter: Dipl.-Ing. Dr. Karl M. Göschka

2. Begutachter: Dipl.-Ing. (FH) Alexander Weiss

Wien, den 30.5.2007

Ausgeführt an der Fachhochschule Technikum Wien

Studiengang: Informations- und Kommunikationssysteme

FACHHOCHSCHULE
TECHNIKUM WIEN

Kurzfassung

Für Anfänger ist das Erlernen einer Programmiersprache schwierig. Das liegt daran, dass im heutigen Unterricht bevorzugt mit Sprachen aus der Industrie gelehrt wird. Studenten wollen in der Industrie Jobs bekommen, und legen deshalb Wert darauf, dass gefragte Technologien Bestandteil ihrer Ausbildung sind. Die Industrie wiederum will ihren Bedarf befriedigen. Dabei wird übersehen, dass Programmiersprachen keine Technologien selbst, sondern Werkzeuge für Technologien sind. Im Unterricht muss eine Programmiersprache ein Werkzeug sein, mit dessen Hilfe es möglich ist, die fundamentalen Ideen eines Unterrichtsgegenstandes zu vermitteln, ohne in einen Unterricht über die Programmiersprache selbst abzudriften.

Die vorliegende Arbeit stellt Python als ein solches Werkzeug vor. Sie zeigt auf, dass Python, im Gegensatz zu heute häufig im Unterricht zum Einsatz kommenden Sprachen (wie C, C++ oder Java), gut für Anfänger geeignet ist. Aufgrund des einfachen Zugangs können mit Python viel früher relevante Konzepte der Informatik und Softwareentwicklung diskutiert werden. Ein weiterer wesentlicher Vorteil ist, dass auch die Arbeit der Unterrichtenden erleichtert wird. Die Sprache ist kompakt und simpel gehalten und versucht sich dem Entwickler nicht in den Weg zu stellen. Gleichzeitig ist sie eine allgemein anerkannte Sprache und findet Verwendung in der Industrie.

Für fortgeschrittene Konzepte der Softwareentwicklung kann auf komplexere Sprachen umgestiegen werden, wobei die Studenten dabei von ihren Erfahrungen mit Python stark profitieren. Der Unterricht kann aber durchaus weiter auf Python aufbauen. So zeigt diese Arbeit, wie das Komponentenframework Zope hierbei Verwendung finden könnte. Dabei werden Themen wie Komponentenorientierung, Reuse, Datenbanken, das Erleben eines Softwareentwicklungsprozesses und Testen und Dokumentieren von Software erläutert. Auch Zope, das auf Python basiert, hat den Vorteil, dass im Vergleich zu anderen Applikationsframeworks ein einfacherer Zugang schnelle Lernerfolge ermöglicht.

Vorliegende Arbeit zeigt, wie mit Python der komplette Bedarf eines auszubildenden Softwareentwicklers abgedeckt wird. Durch die Schnelllebigkeit der Technik wird es immer wichtiger, die grundlegenden Konzepte einer Wissenschaft zu beherrschen, anstatt das Erlernen eines Werkzeugs, das in der Industrie aktuell ist.

Abstract

For beginners the learning of a programming language proves to be quite difficult. This is due to the fact that today's educational system focuses primarily on languages related to industry. Students want to get jobs in the industry. Therefore they demand to be sufficiently trained in these technologies. The industry again wants to satisfy its need. But programming languages are not technologies by themselves, though, they are tools for technologies. However, programming languages for educational purposes must be such tools which help students gain fundamental understanding of a topic without the need for expert knowledge in a programming language itself.

This paper introduces Python as one example for such a tool. It points out that, unlike languages such C, C++ or Java widely used today, Python is highly suitable for beginners. Due to the easy access it is possible to teach relevant concepts of computer science and software development earlier in the learning process. A further substantial advantage is relief of the instructor´s. The language is compactly and simply held without hinderung the developer. Python is a generally a well respected language and finds use in the industry.

For advanced concepts of software development, teaching can be directed to more complex languages, whereby the students strongly profit from their experience with Python. If knowledge of other languages is not required, it is also possible to stay with Python, which performs well in this situation.

Thus this paper shows how the component-framework Zope could be of use in this field. The following topics, such as component orientation, reuse, databases, experiencing a software development process and tests- and documenting of software are discussed. Also Zope, which is based on Python, has the advantage of easy access, thus creating the possibility of fast successes in learning. This paper shows how the complete need of a software developer is covered by teaching with Python.

Verwendete Abkürzungen

AJAX	Asynchronous JavaScript and XML
AOP	Aspektorientierte Programmierung
APE	Adaptable Persistence Engine
API	Application Programming Interface
BSD	Berkeley Software Distribution
CBSE	Component Based Software Engineering
CI	Corporate Identity
CORBA	Common Object Request Broker Architecture
COTS	Commercial off-the-shelf
CPS	Collaborative Portal Server
CSS	Cascading Style Sheets
CWI	Centrum voor Wiskunde en Informatica
DBC	Design By Contract
DTML	Document Template Markup Language
EJB	Enterprise Java Beans
FH	Fachhochschule
GNU	GNU's Not Unix
GUI	Graphical User Interface
HLL	High Level Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDLE	Integrated Development Environment
MUW	Medizinische Universität Wien
NASA	National Aeronautics and Space Administration
NATO	North Atlantic Treaty Organisation
OOP	Objektorientierte Programmierung
ORB	Object Request Broker
PDF	Portable Document Format

PROLOG	PROgramming in LOGic
PSA	Python Software Activity
PSF	Python Software Foundation
RDBMS	Relational Database Management System
RMI	Remote Method Invocation
RPC	Remote Procedure Calls
RSS	Really Simple Syndication
SDL	Simple DirectMedia Layer
SIG	Python Special Interest Group
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
Tcl	Tool command language
TDD	Test Driven Development
Tk	Toolkit
TTW	Through the Web
UML	Unified Modeling Language
URL	Uniform Resource Locator
WWF	World Wide Fund For Nature
XML	Extensible Markup Language
XP	Extreme Programming
ZCML	Zope Configuration Markup Language
ZEO	Zope Enterprise Objects
ZMI	Zope Management Interface
ZODB	Zope Object Database
Zope	Z Object Publishing Environment
ZPL	Zope Public Licence
ZPT	Zope Page Templates

Eidesstattliche Erklärung

“Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.“

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation und Zielsetzung

Geht es um das Aneignen von Programmierkenntnissen, ist die momentane Situation nach Ansicht des Autors nicht optimal. Sie ist verbesserungswürdig. Gerade in der schnelllebigen Informatik sollte im Unterricht nicht auf Trends oder Modeerscheinungen zurückgegriffen werden, sondern die fundamentalen Ideen der Wissenschaft Informatik mit einem guten Konzept an die Studierenden herangetragen werden. Erste Programmierkenntnisse mit z.B. C oder Java zu vermitteln bzw. vermittelt zu bekommen, ist problematisch. Diese Arbeit hat zum Ziel, einen einfacheren Weg aufzeigen und soll begründen, warum erste Programmiererfahrungen mit Python didaktisch sinnvoller sind.

Die Programmiersprache Python soll als Programmiersprache für den Unterrichtseinsatz vorgestellt werden und es wird gezeigt, wie gewisse Grundkonzepte mittels dieses Werkzeugs vermittelt werden können. Weiterführend wird Z Object Publishing Environment (Zope) als Applikationsserver, basierend auf Python, für fortgeschrittenere Themen der Softwareentwicklung, und vor allem für den praktischen Teil dieser Arbeit verwendet.

Die Arbeit hat das Ziel, die Verbreitung von Python an Schulen und Universitäten zu unterstützen, um damit den Schülern[1] und Studenten den Einstieg in die Programmierung und diese selbst zu erleichtern.

Weiters ist Zope für den Einsatz an Universitäten vorzustellen; der einfache Zugang zu einem Open-Source Produkt soll die Vorteile für Lektoren und Studenten bei fortgeschrittenen Themen der Softwareentwicklung aufzeigen.

Die vorliegende Arbeit soll dabei als Entscheidungs- und Argumentationsgrundlage an genannten Ausbildungseinrichtungen dienen können.

1.2 Aufbau der Arbeit

Die Arbeit gliedert sich in Einleitung, zwei Hauptkapitel und abschließende Diskussion. Sie ist nicht klassisch in Theorie- und Praxisteil gegliedert. Eine solche Aufteilung ist bei dieser Thematik schwer zu vollziehen, die vorgestellte Theorie wird immer wieder durch praktische Beispiele unterstützend erläutert. Im zweiten Kapitel ist eine Beispielimplementierung abgehandelt.

In der Einleitung werden die Argumentationsgrundlagen für die Arbeit aufgebaut. Die in der Arbeit verwendeten Technologien Python und Zope werden vorgestellt.

Nach Erklärung allgemeiner, in der Arbeit verwendeter Begrifflichkeiten und Technologien, folgt im ersten Kapitel die Abhandlung der Programmiersprache Python. Darin wird erläutert, warum sich Python als eine Einstiegsprogrammiersprache, und vor allem für Unterrichtszwecke besonders gut eignet. Danach werden Grundkonzepte des Informatikunterrichts, wie Paradigmen und

[1] Wenn in dieser Arbeit die männliche Form verwendet wird, sind Frauen gleichermaßen gemeint, sofern nicht explizit Gegenteiliges behauptet wird.

Algorithmen mit Python als unterstützendes Werkzeug analysiert. Weiters wird behandelt, welche andere Fachgegenstände neben Informatik von Python profitieren können, wie das Thema Open-Source auf den Unterricht angewandt werden kann, und welche Materialien und Werkzeuge als Unterstützung für den Unterricht bzw. die Unterrichtsgestaltung zur Zeit zur Verfügung stehen.

Zope ist das Thema des zweiten Kapitels. Zope ist ein Applikationsframework basierend auf Python und eignet sich für das Lehren fortgeschrittener Softwareentwicklungsmethoden. Darin wird untersucht, ob Zope als Komponentensystem betrachtet werden kann, und wie das Framework mit Component Based Software Engineering (CBSE) in Verbindung gebracht werden kann. Dass das Testen und Dokumentieren von Software im Unterricht vernachlässigt wird, wird ebenso behandelt, wie das Lehren eines Softwareentwicklungsprozesses. Den Abschluß des Hauptteils bildet die Beispielimplementierung aus einem realen Projekt im Arbeitsumfeld des Autors.

1.3 Fundamentale Ideen als Unterrichtsprinzip

Eine Argumentationsgrundlage dieser Arbeit ist die Definition einer *fundamentalen Idee* nach Schwill [Sch93]. In seiner Arbeit untersucht er die philosophische Sicht der *Idee* nach Plato und Kant, und formuliert diese mit den *Strukturen* nach J.S. Bruner [Bru60] zur fundamentalen Idee. J.S. Bruner hat das didaktische Prinzip an den Strukturen der zugrundeliegenden Wissenschaft, an denen sich der Unterricht orientieren soll, formuliert.

> *"Eine **fundamentale Idee** (bezgl. einer Wissenschaft) ist ein Denk-, Handlungs-, Beschreibungs- oder Erklärungsschema, das*
>
> 1. *in verschiedenen Bereichen (der Wissenschaft) vielfältig anwendbar oder erkennbar ist (Horizontalkriterium),*
> 2. *auf jedem intellektuellen Niveau aufgezeigt und vermittelt werden kann (Vertikalkriterium),*
> 3. *in der historischen Entwicklung (der Wissenschaft) deutlich wahrnehmbar ist und längerfristig relevant bleibt (Zeitkriterium),*
> 4. *einen Bezug zu Sprache und Denken des Alltags und der Lebenswelt besitzt (Sinnkriterium)."*

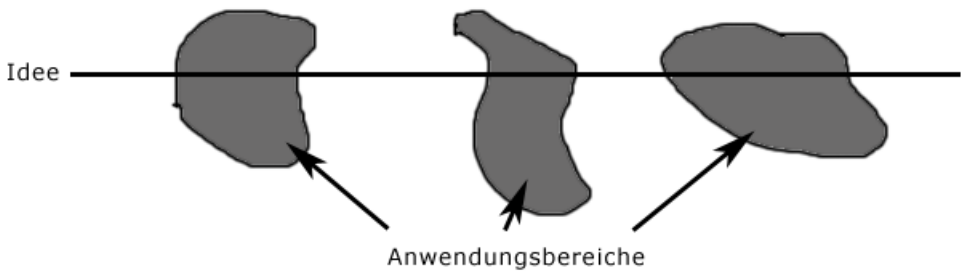

Abbildung 1: Horizontalkriterium nach [Sch93]

Das *Horizontalkriterium* veranschaulicht Schwill wie in Abbildung 1. Eine Idee wird soweit abstrahiert, dass fachspezifische Elemente herausfallen. Damit kann themen- und

fachübergreifend gearbeitet werden. Der Lernende erkennt durch immer wiederkehrende Prinzipien die übergreifende Relevanz der Thematik. Anschließend wird durch Wiederholung das Wichtige gefestigt. Die gemeinsame Idee kann jedoch wiederum nur durch umfassendes Spezialwissen herausgearbeitet werden. Das ist die Aufgabe der Lehrkräfte.

Das *Vertikalkriterium* kann wie ein Faden im Bildungsweg gesehen werden. Diesem Faden wird im Laufe der Ausbildung gefolgt. Dabei steigen das Niveau und die Detaillierung der Materie (Abbildung 2).

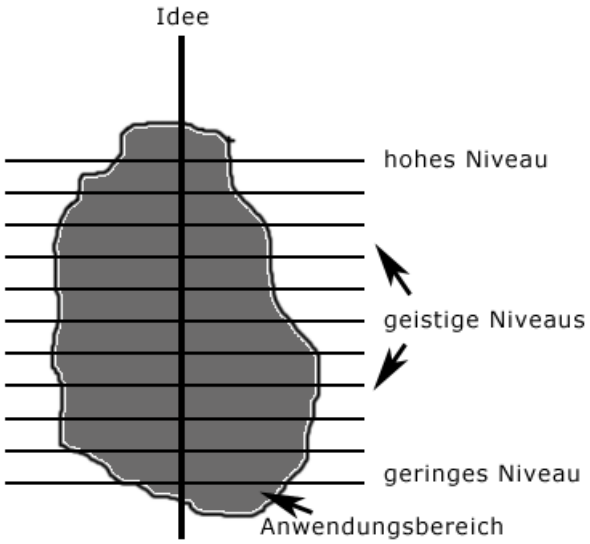

Abbildung 2: Vertikalkriterium nach [Sch93]

Das *Zeitkriterium* ist in der Informatik nicht schwer zu erfüllen. Die Schnelllebigkeit der Technik birgt zwar viele Modeerscheinungen, jedoch sichert das Kriterium die Kontinuität des Unterrichts und den Wert der Kenntnisse und Erfahrungen der Unterrichtenden und verhindert fachliche Moden.[Mod02]

Dabei ist die Trägheit der Lehre eine Art Vorauslese, um das Zeitkriterium einer fundamentalen Idee zu erfüllen.

Der Autor interpretiert das *Sinnkriterium* als das tatsächliche Wertempfinden beim Empfänger der Idee, beim Lernenden. Wie gut kann die Thematik als praxisrelevant oder als sinnvoll erkannt werden? Kann der Bezug zur späteren Verwendung in der Arbeitswelt hergestellt werden?

Der Einsatz fundamentaler Ideen wird von [Mod02] wie folgt beschrieben:

> *"Der Unterricht muss dann so angelegt werden, dass sich diese - wenigen - fundamentalen Ideen bei den Schülerinnen und Schülern bilden können, er muss Kenntnisse und Erfahrungen vermitteln, die anhand dieser Ideen zu ordnen sind, und er muss diese Ideen zu einem geeigneten Zeitpunkt explizit thematisieren, um die spezifischen Möglichkeiten und Beschränkungen der Informatik in Abgrenzung gegen andere Disziplinen erkennbar zu machen."*

Einige Beispiele fundamentaler Ideen und die dazugehörigen Kriterien sind in [Sch94] zu finden.

1.4 Programmiersprachen im Unterricht

Ein vieldiskutiertes Thema ist, welche Sprache sich für den Unterricht am besten eignet. [LGS06, Pal90] meint, dass der Lernerfolg von der Zeit abhängt, die für tatsächliches Programmieren aufgewendet werden kann. Es sollte keine Zeit mit sprachspezifischen Eigenheiten und Syntaxproblemen "verschwendet" werden. Geht es nach [LGS06, Mil93], hat eine Programmiersprache für Unterrichtszwecke einen einfachen Zugang. Sie sollte leicht erlernbar sein, eine klare Struktur haben und vielfältig einsetzbar sein. Die Sprache hat eine einfache Syntax, einfaches I/O Handling, verständliche String-Manipulation, aussagekräftige Schlüsselwörter und verständliches Feedback im Fehlerfall.

Während Pascal und Logo vor einigen Jahren noch oft im Unterricht zum Einsatz kamen, sind beide heute stark aus der Mode gekommen. Gründe dafür sind sicher die mangelnde Einsatzfähigkeit in der Industrie und die, so gut wie nicht mögliche, Verwendbarkeit der Sprache bei steigender Komplexität der Software.

Heute zählen Sprachen wie C, Java und C++ zu den beliebtesten Programmiersprachen. Das ist an der Anzahl der verfügbaren Entwickler, Lehrgänge und Dienstleister weltweit zu erkennen [TIO06]. Studien wie [dWT02], [Md03] und [SW98] belegen, dass Java, C++ und C die Sprachen mit der größten Verbreitung an Universitäten sind.

Trotz dieser Beliebtheit (oder gerade deswegen) wird viel über die Tauglichkeit dieser Sprachen im Unterricht diskutiert, gerade wenn es um geeignete Sprachen für Programmieranfänger geht. Die oben genannten Sprachen werden als überladen betrachtet. Die Studierenden plagen sich eher mit der Notation, als mit dem tatsächlichen Algorithmen. [LGS06] erläutert, dass die meisten Probleme von Programmierneulingen immer dasselbe Muster zeigen:

> *"[...] construct-based problems, which make it difficult to learn the correct semantics of language constructs, and plan composition problems, which make it difficult to put plans together correctly [...] students lack the skills needed to trace the execution of short pieces of code after taken their first course on programming."*

Im Zuge einer Studie an der finnischen Universität in Tampere ist eine Umfrage an europäischen Hochschulen durchgeführt worden. 559 Studenten und 34 Lektoren an 6 verschiedenen Universitäten wurden unter anderem zu deren Schwierigkeiten beim Lernen und Lehren von Programmiersprachen befragt. Die daraus abgeleiteten Folgerungen besagen:

> *"the most difficult concepts to learn are the ones that require understanding larger entities of the program instead of just details [...] abstract concepts like pointers and memory handling are difficult to learn [...] However, the biggest problem of novice programmers does not seem to be the understanding of basic concepts but rather learning to apply them."* [LAMJ05]

Die Studenten haben Probleme, wenn es darum geht, den Gesamtumfang eines Programmes zu erfassen und umzusetzen, dh. wie setze ich eine mir gestellte Aufgabe mit den Konzepten um, die ich gelernt habe. Dabei darf die Syntax einer Sprache nicht im Wege stehen, da sie daran hindert eine Problemlösung zu finden und nur neue, andere Probleme schafft. Zeiger und Speichermanipulation zählen zu den als sehr schwierig eingestuften Konzepten.

Die in diesem Kapitel zitierte Literatur deckt sich mit den Beobachtungen des Autors aus der ei-

genen Ausbildung und der Abhaltung eines C Tutoriums für Programmieranfänger. Die Studenten konnten genau wiedergeben, wie sie das Problem lösen würden, doch konnten sie es nicht "zu Papier" bringen, also als Quelltext wiedergeben. Die Syntax wurde als unnatürlich und teilweise unverständlich empfunden. Manche syntaktische Eigenheiten sind für den Tutor auch schwer zu erklären, da die Studenten die dahinterliegenden Konzepte noch nicht verstehen können. Nahezu alle Fehler resultierten aus einem Fehler in der Syntax. Der Autor konnte dabei ein hohes Maß an Frustration und Demotivation der Studenten im Unterricht feststellen. Die Sinnhaftigkeit des Lehrinhaltes wurde weiters angezweifelt.

Die persönlichen Erkenntnisse des Autors und jene der obig angeführten Literatur lassen Handlungsbedarf erkennen. Diese Arbeit wird im weiteren Verlauf diese Problematik behandeln und zwei Werkzeuge für möglichst reibungsfreien und effektiven Unterricht vorstellen.

1.5 Grundlagen Python

Python ist eine dynamische „High-Level" Programmiersprache, die den interpretierten Skriptsprachen zugeordnet und oft mit Perl, Ruby, Scheme oder Java verglichen wird. Sie wurde mit dem Ziel entwickelt, möglichst einfach und übersichtlich zu sein, gleichzeitig aber nicht an Flexibilität und funktionaler Skalierbarkeit einbüßen zu müssen.

Die Syntax ist sehr übersichtlich gehalten, was unter anderem durch zwingende Einrückung des Quelltextes erreicht wird (siehe Listing 1). Grammatikalisch ist auf wenige Schlüsselwörter optimiert worden[2].

```
1 def faculty(n):
2     if n == 0:
3         return 1
4     else:
5         return n * faculty(n - 1)
```

Listing 1: Python Syntax

Python arbeitet mit dynamischer Typverwaltung. Variablen haben somit keinen festen Typ. Anders als bei statischer Typverwaltung wie in Compilersprachen, wird der Typ zur Laufzeit und abhängig vom Einsatz der Variable bestimmt. Das erspart dem Entwickler vieles an zusätzlicher Arbeit, da der Quelltext, durch die Ersparnis der explizit anzugebenden Typinformation, übersichtlicher und kürzer wird. Weiters können Funktionen, die ja ohne Typinformation geschrieben sind, auf vielen verschiedenen Objekttypen wiederverwendet werden. Man spricht auch von Duck Typing: *„If it looks like a duck and quacks like a duck, it must be a duck"* [DT00]. Die Kehrseite der Medaille sind natürlich Performanceeinbußen.[CG97], [Lej06], [GH]

In Programmiersprachen wie C oder C++ ist der Programmierer für Speicherreservierung und Spreicherfreigabe selbst verantwortlich. Unter C wird das z.B. mit den Funktionen `malloc()` und `free()` durchgeführt. Pythons Strategie hiezu hat den Namen „Referenzzähler" (reference counting). Das Prinzip ist einfach: jedes Objekt enthält einen Zähler, der mit jeder Referenz auf das Objekt inkrementiert wird. Wird eine Referenz wiederum gelöscht, wird der Zähler dekrementiert. Erreicht dieser den Wert Null, kann der Speicher freigegeben werden.

[2]Mit Python 2.5 sind das 31 Schlüsselwörter. Zum Vergleich C/C++ mit 63 und Java mit 53 Schlüsselwörtern.

Der Pythonentwickler ist an kein Programmierparadigma gebunden. Auch wenn Python sich als objektorientierte Sprache vorstellt, ist die Entwicklung mit verschiedenen Programmierstilen möglich. Das erlaubt Flexibilität bei der Erstellung von Programmen. Die Entwicklung ist durch die Sprache nicht an ein Paradigma gebunden, sondern kann das, für die momentane Situation effizienteste Paradigma wählen, eventuell sogar verschiedene Paradigmen anwenden. Das macht natürlich Python für den Unterricht sehr interessant.

Die Python-Distribution wird mit einer Standardbibliothek in Form von Python-Modulen ausgeliefert, die von der Community als „batteries included“ bezeichnet wird. Die Komplexität wurde aus der Sprache entfernt, die Spezialisierung liegt in den Bibliotheken. Das Spektrum reicht von Bibliotheken zur Graphical User Interface (GUI)-Gestaltung über Multimedia und Spielentwicklung bis hin zu verteilten Systemen [Lin02]. Großes Augenmerk wird hierbei auf Webapplikationsentwicklung gelegt, die gängigen Protokolle und Standards sind plattformunabhängig verfügbar[3]. Findet der Entwickler die gewünschte Funktionalität nicht in den Standardbibliotheken, kann er das gewünschte Modul aus einer großen Sammlung von Drittanbietern wählen. Die Module der Standardbibliothek können mit C oder Python erweitert werden, auf gleiche Art können eigene Module entwickelt werden.

```
1 import urllib
2 data = urllib.urlopen("http://www.technikum-wien.at/")
3 text = data.read()
4 data.close()
```

Listing 2: Lesen einer Website durch Verwendung eines Moduls der Standardbibliothek

Python ist portierbar und somit plattformunabhängig. Der Python Interpreter kann auf jeder beliebigen Architektur installiert werden, wenn es dafür einen C Compiler gibt. Die populärsten Betriebssysteme mit verfügbaren Python Interpretern sind Linux, Berkeley Software Distribution (BSD), Mac OS X, Solaris und Microsoft Windows. Für diese gibt es auch sehr gut gewartete Bibliotheken.

Die Python Implementierung steht unter einer Open-Source Lizenz. Der Quelltext ist frei verwendbar und verteilbar, auch für den kommerziellen Gebrauch. Die Python Lizenz[4] wird von der Python Software Foundation (PSF)[5] verwaltet. Hinter Python steht natürlich eine Open-Source Community, die regelmäßig, aktiv und ambitioniert ihr Produkt verbessert. Jeder kann sich an der Entwicklung beteiligen. Internationale Mailinglisten, Foren, Arbeitsgruppen und Konferenzen werden auf den Internet-Seiten der Python Software Activity (PSA)[6] organisiert. Eine wichtige Rolle in der Entwicklung von Python trägt die Special Interest Group (SIG)[7].

Es gibt zahlreiche Projekte, die mit Python am Markt und in der Industrie erfolgreich ihre Arbeit verrichten. Zu den größten Referenzen zählen Firmen wie Google, Yahoo, National Aeronautics and Space Administration (NASA) und viele andere[8].

Peter Norvig, „director of search quality“ bei Google meint:

[3] TCP/IP, FTP, NNTP, HTTP, POP, SMTP, MAPI, SGML, XML, CGI, ...
[4] Python Lizenz: http://www.python.org/psf/license/
[5] Python Software Foundation: http://www.python.org/psf/
[6] Python Software Activity: http://www.python.org/psa/
[7] Special Interest Groups: http://www.python.org/community/sigs/
[8] Erfolgsgeschichten: http://www.python.org/about/success/

> *"Python has been an important part of Google since the beginning, and remains so as the system grows and evolves. Today, dozens of Google engineers use Python, and we're looking for more people with skills in this language."*

Bei Google ist Python ein wichtiges Werkzeug; Google sucht immer wieder nach talentierten Entwicklern, in deren Portfolio Python zu finden ist.

Python ist zu einer bedeutenden Größe in der Softwareindustrie geworden.

1.5.1 Geschichte und Entwicklung

Ihren Namen verdankt die Programmiersprache Python der britischen Comedy-Gruppe Monty Python, nicht der gleichnamigen Schlangenart, wenngleich diese auch zu einem wesentlichen Bestandteil der Python Corporate Identity (CI) geworden ist.

Pythons Erfinder und momentaner Hauptbetreuer ist Guido van Rossum. Python basiert auf ABC, einer Programmiersprache, die in den achtziger Jahren verwendet wird. 1989 beginnt Van Rossum mit der Entwicklung von Python am Centrum voor Wiskunde en Informatica (CWI)[9] in Amsterdam. Das Ziel ist, die Hürden, die ABC an der Weiterverbreitung hindern, zu überwinden. ABC ist wie Pascal: eine gute Lehrsprache für den Unterricht, aber in der Industrie nicht zu gebrauchen. Ein Grund ist Unflexibilität, wenn es um Skalierbarkeit geht. So wird Python von Grund auf für den Einsatz in der Ausbildung an Lehranstalten konzipiert. Hinzu kommen mehr fortgeschrittene Möglichkeiten zur Softwareentwicklung, eine Standardbibliothek voller Erweiterungen und die leichte Anknüpfbarkeit an andere Programmiersprachen (vor allem C), damit Python auch außerhalb des Unterrichts Verwendung finden kann.[Wil02]

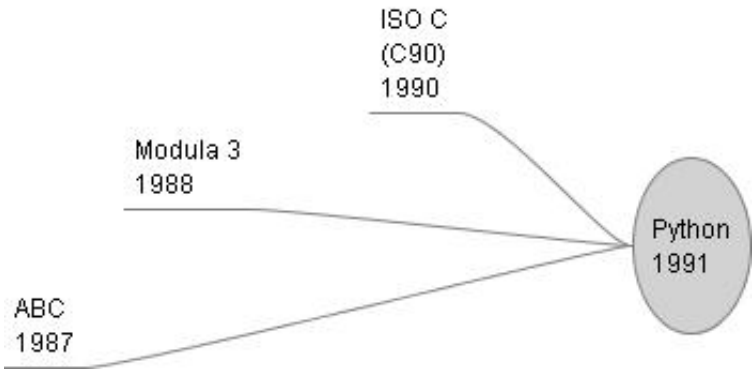

Abbildung 3: Einflüsse anderer Sprachen auf Python

Abbildung 3 zeigt eine kompakte Übersicht zur Entwicklung Pythons. Eine detaillierte, historische Übersicht zur Entstehung von Programmiersprachen im Allgemeinen ist auf [Pix01] zu finden.

Python wird 1991 veröffentlicht; die Entwicklung der ersten Version geht bis Python 1.6.1. Oktober 2000 wird Python 2.0 freigegeben, zum Zeitpunkt dieser Arbeit ist Python in Version 2.5 aktuell, welche seit September 2006 zur Verfügung steht. Momentan wird an Version 3.0 (Py3K, Python 3000) entwickelt, die in einer *Alpha* Version 2007 erwartet wird[10].

[9] http://www.cwi.nl/

[10] Python 3000, Google Tech Talk, Guido van Rossum, 2006

Diese Arbeit greift nun den Versuch auf, die Popularität Pythons anhand einer Google Suche zu messen. Das ist keine sehr wissenschaftliche Messung, spiegelt aber doch eine gewisse Relevanz und vor allem deren Veränderung in den letzten Jahren, in der weltweit größten Suchmaschine wider. Basis der Untersuchung sind die Ergebnisse aus dem Jahr 2002 von [Wil02]:

	Suchergebnisse		Änderung um Faktor
	2002	**2006**	
C	4.930.000	820.000.000	166,33
Java	2.670.000	165.000.000	61,8
C++	1.800.000	58.100.000	32,28
Perl	1.690.000	68.000.000	40,24
Python	838.000	56.600.000	67,54

Tabelle 1: Google-Suche: `Platzhalter-Sprache AND ( language OR code OR program)`. Zugriff auf http://www.google.com/ am 22.10.2006.

Tabelle 1 stellt die Ergebnisse einer Anfrage an die Suchmaschine Google dar. Dabei wird die ursprüngliche Anfrage von [Wil02] aus dem Jahr 2002 mit der gleichen - nur zeitlich aktuelleren - Anfrage verglichen. Wie zu erkennen ist, ist die Anzahl der Treffer immens in die Höhe geschnellt. Das kann in erster Linie durch die technologische und inhaltliche Entwicklung der Suchmaschine begründet werden, wie auch durch die Relevanz der einzelnen Suchanfragen.
Die hohe Trefferanzahl der Sprache C ist mit Vorsicht zu genießen, da angenommen werden kann, dass hier einige Treffer zu C++ mitgerechnet werden. Trotzdem steht C ungeschlagen an der Spitze dieser Auswertung, was auch nicht weiter verwunderlich ist. Nach wie vor wird C in der Systemprogrammierung verwendet, Schnittstellen zu Anwendungsprogrammen sind typischerweise ebenso in C implementiert. Ein C-Compiler exisitiert für eine Vielzahl an Plattformen. Gerade im rasant gewachsenen Embedded Systems Bereich ist C quasi Standard.
Die anderen Programmiersprachen sind interessant zu beobachten, wobei Python im Vergleich zu 2002 das größte Wachstum verbuchen kann und sich im Gesamtvergleich mit Anwendungsentwicklungssprachen wie C++ oder Java durchaus messen kann. Im Vergleich der Skriptsprachen hat Perl vor Python die höhere Trefferanzahl.

	2006
C	168.000.000
Java	26.500.000
C++	11.000.000
Perl	13.000.000
Python	13.200.000

Tabelle 2: Google-Suche: `Platzhalter-Sprache AND ( (language OR code OR programm) AND ( education OR teaching))`. Zugriff auf http://www.google.com/ am 22.10.2006.

Die Daten in Tabelle 1 sind sehr allgemein und umfassen mehr als die gewünschte Thematik dieser Arbeit. Aus diesem Grund wird mit Tabelle 2 die Suchanfrage leicht abgeändert und damit konkretisiert: wie relevant sind die genannten Programmiersprachen im Unterricht? Während Tabelle 1 die Veränderung im Laufe der letzen Jahre darstellt, zeigt Tabelle 2 den Status quo.

Weiterführende Literatur zu Python: [Pyt06], [Swa04], [AD02], [Lut06], [AM05], [Pil04].

1.5.2 High-Level-Sprache und Skriptsprache

High Level Language (HLL) nach [Fed96, Lin02]:

> *"A computer programming language that is primarily designed for, and syntactically oriented to, particular classes of problems and that is essentially independent of the structure of a specific computer or class of computers."*

Nach diesem Zitat ist eine HLL eine Programmiersprache, die unabhängig von der Maschine ist. Programmiersprachen ab der dritten Generation zählen zu den höheren Programmiersprachen oder High-Level-Languages. In der Literatur wird oft auch der Begriff "Systemsprache" verwendet. Eine HLL hat eine für Menschen lesbare Syntax, die von einem Compiler oder Interpreter 1:n in Maschinensprache übersetzt wird. Das bedeutet, dass ein einziger Befehl in einer HLL durch viele Instruktionen in Maschinensprache ausgedrückt wird. Assembler, zählend zur zweiten Generation, übersetzt 1:1 in Maschinensprache, die wiederum zur ersten Generation gezählt wird.

Zwei verschiedene Programme verarbeiten eine HLL in Maschinensprache (Binärcode): Compiler und Interpreter. Ein Interpreter liest ein High-Level Programm und führt es aus. Die Analyse des Quelltextes erfolgt also zur Laufzeit des Programms.

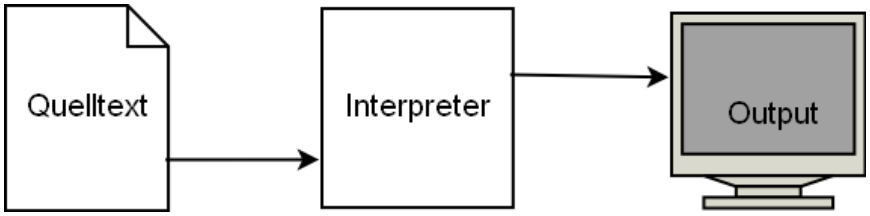

Abbildung 4: Interpreter

Ein Compiler analysiert das Programm und übersetzt es komplett, bevor das Programm ausgeführt wird. Der somit entstandene Code wird Objektcode genannt. Das Programm kann ohne weitere Übersetzung wiederholt ausgeführt werden.

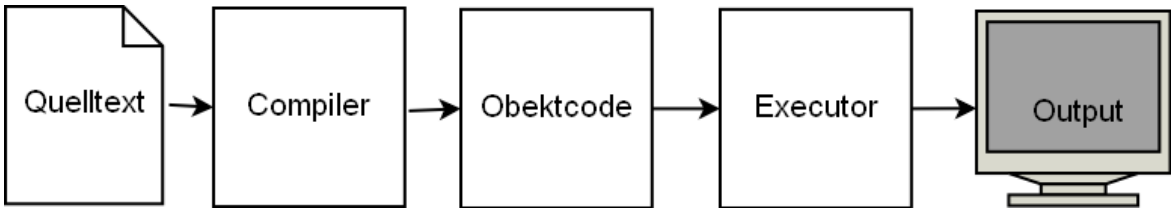

Abbildung 5: Compiler

Python wird zu den interpretierten Programmiersprachen gezählt; Programme werden von einem Interpreter ausgeführt. Es gibt hierbei zwei Varianten: den Kommandozeilenmodus und den Skriptmodus. Im Kommandozeilenmodus werden Befehle Zeile für Zeile eingegeben, der Interpreter gibt das Ergebnis sofort retour[11]:

[11] Im Laufe der Arbeit werden immer wieder Quelltexte eingearbeitet. Dabei gibt es zwei Formen. Die interaktive Session einer Interpreter Shell und den Skriptmodus. Die Interpreter Sessions sind erkennbar an der typischen Python Eingabeaufforderung (engl. "prompt"), die durch drei aufeinanderfolgende spitze Klammern (>>>) symbolisiert wird. Die Eingabe ist dabei immer fett ausgeprägt, die Ausgabe des Interpreters normal.

```
$ python
Python 2.4.4c1 (#1, Oct 13 2006, 12:10:43)
Type "help", "copyright", "credits" or "license" for more information.
>>> print 10 * 2
20
```

Im Skriptmodus wird das Programm in einem File gespeichert. Der Inhalt wird dann vom Interpreter ausgeführt.

Der Skriptmodus führt zum Begriff Skriptsprache. Wie zu Beginn des Kapitels erwähnt, wird Python den interpretierten Skriptsprachen zugeordnet. Eine Skriptsprache ist eine HLL, jedoch mit noch höherer Abstraktion zur Maschine. [Ous98] unterscheidet dabei Systemsprachen von Skriptsprachen wie folgt:

> *"Scripting languages are designed for different tasks than system programming languages, and this leads to fundamental differences in the languages. System programming languages were designed for building data structures and algorithms from scratch, starting from the most primitive computer elements such as words of memory. In contrast, scripting languages are designed for gluing: they assume the existence of a set of powerful components and are intended primarily for connecting components together. [...] Scripting languages are sometimes referred to as glue languages or system integration languages. [...] The growth of the Internet has also popularized scripting languages. The Internet is nothing more than a gluing tool."*

Skriptsprachen haben andere Aufgaben als "Systemsprachen". Sie sind nicht darauf ausgelegt Datenstrukturen oder Algorithmen von Grund auf neu zu entwickeln, sondern als Bindeglied für vorhandene Applikationen zu dienen. Skriptsprachen bedienen sich also anderer Komponenten, um daraus eine eigene Applikation entstehen zu lassen.

Den immer wieder kritisierten Geschwindigkeitsunterschied zu kompilierten Sprachen versuchen Skriptsprachen wie Python zu verbessern. So wird der Quelltext nicht direkt interpretiert, sondern zuerst in den sogenannten "Bytecode" umgewandelt. Dieser wird wesentlich schneller interpretiert. Wirklich Geschwindigkeitskritisches oder Rechenintensives wird in einer kompilierten Sprache geschrieben. Wie schon erwähnt, wird bei Python C verwendet. Weiters relativiert immer schnellere und billigere Hardware die Kritik an der Performance.

Einige weitere Eigenschaften von Skriptsprachen sind in diesem Kapitel schon erwähnt. Zur Übersicht sei eine Darstellung in kompakter Form aufgelistet:

- noch "höhere" Implementierung als eine typische HLL
- interpretiert
- dynamische Typverwaltung
- optimal für schnelle Softwareentwicklung bzw. Prototyping [Lin02]
- hoher Re-Use Faktor
- leichter erlernbar als Systemsprachen

Weitere Beispiele für Skriptsprachen sind Perl[12] oder Tcl[13].

1.6 Grundlagen Zope

Zope[14] ist ein Applikations- und Backend Server Framework, das Entwicklern erlaubt, schnell Protokolle einzubinden, Applikationen (üblicherweise webbasierte) zu entwickeln, und Konnex zwischen anderen internetbasierten Services herzustellen [Ric05a].

Zope wird großteils mit Python entwickelt. Geschwindigkeitskritisches wird in C umgesetzt. Üblicherweise wird die mit Zope mitgelieferte Zope Object Database (ZODB) verwendet, um transaktionssicher[15] (Python) Objekte zu persistieren. Zope ist mit der Zope Public Licence (ZPL)[16], einer freien Softwarelizenz, verfügbar.

Momentan werden zwei stabile Zope Versionen von der Community gepflegt. Mit Oktober 2006 ist Zope 2.10 die letzte stabile Version der Zope 2 Generation. Zope 3.3 ist die letzte stabile Version des neuen Zope 3. Diese Arbeit konzentriert sich auf Zope 3, wobei zuerst einige Unterschiede zu Zope 2 aufzeigt werden sollen.

1.6.1 Zope 2

Bevor Zope ins Leben gerufen wird, entwickelt die Zope Corporation[17] (ursprünglich Digital Creations) 1996 das Produkt Bobo, ein in Python entwickelter Object-Publisher, welcher Entwicklern erlaubt, Python Objekte im Web abzubilden. Bobo ist weiters eine Objektdatenbank und ein Object Request Broker (ORB), der Uniform Resource Locator (URL) in Objektpfade umwandelt. 1998 wird dieses Produkt als Open-Source unter dem Namen Zope veröffentlicht.[Ric05a]

Zope 2 erfreut sich anfangs großer Beliebtheit durch die Through the Web (TTW) Entwicklung und Administration. Komplette Projekte können per Web entwickelt werden, ohne auch nur einmal eine Kommandozeile zu sehen. Die schnelle Projektabwicklung durch die Produktivität mit Python plus die Zope Templatesprachen Document Template Markup Language (DTML) und Zope Page Templates (ZPT) mobilisieren viele Entwickler, ihre Projekte mit Zope umzusetzen.

Zope 2 hat jedoch mit einer Vielzahl von Problemen zu kämpfen. Die anfangs, vor allem bei unerfahrenen Entwicklern, sehr beliebte TTW Entwicklung stößt schnell an Grenzen. So muss für fortgeschrittenere Methoden, wie etwa das Verwenden von regulären Ausdrücken, das Zope 2 Sicherheitssystem umgangen werden. Zope 2 TTW wird vom Framework aus Stabilitätsgründen mit eingeschränktem Zugriff auf Pythonbibliotheken versehen. Mit Zope 3 gibt es hier keine Einschränkung, da Funktionalität mit Hilfe von Pythonpaketen implementiert wird. Innerhalb dieser Pakete kann auf die komplette Funktionalität Pythons zurückgegriffen werden. Das ist unter Zope

[12] Perl: http://www.perl.com/

[13] Tcl: http://www.tcl.tk/

[14] Zope: http://www.zope.org/

[15] siehe Maillinglisteneintrag "ZODB implementiert kein ACID": https://mail.dzug.org/pipermail/zope/2007-June/003937.html

[16] Zope Public Licence: http://www.zope.org/Resources/ZPL/

[17] http://www.zope.com/

2 auch möglich, nur ist die Dokumentation dazu schlecht zugänglich und die Implementierung nur mit einigen Tricks möglich. Listing 3 zeigt ein Beispiel dazu: wird das Initialisieren der Klasse in Zeile 27 vergessen, ist schnell und einfach eine potentielle Sicherheitslücke produziert. Das Zope 2 Framework liefert dem Entwickler hierzu keine Hinweise oder Warnungen.

```
1  from Globals import DTMLFile, InitializeClass
2  from OFS.SimpleItem import SimpleItem
3  from OFS.PropertyManager import PropertyManager
4
5  class Contact(SimpleItem, PropertyManager):
6
7      def __init__(self, first, last, address, postal_code):
8          self.manage_edit(first, last, address, postal_code)
9
10     manage_options = (SimpleItem.manage_options +
11     PropertyManager.manage_options
12         + ({'label': 'Edit', 'action': 'manage_main'},
13            {'label': 'View', 'action': 'index_html' },))
14
15     security.declareProtected('Change Contacts', 'manage_main')
16     manage_main = DTMLFile('dtml/main', globals())
17
18     security.declareProtected('View', 'index_html')
19     index_html = DTMLFile('dtml/index_html', globals())
20
21     security.declareProtected('Change Contacts', 'manage_edit')
22     def manage_edit(self, first, last, adress, postal_code, REQUEST=None):
23     ...
24     if REQUEST is not None:
25         REQUEST.RESPONSE.redirect(...)
26
27     InitializeClass(Contact)
```

Listing 3: Typische Zope 2 Content Klasse

Um existierende Funktionalität zu erweitern, muss unter Zope 2 das sogenannte „Monkey-Patching“ angewandt werden. Hierbei wird die ursprüngliche Funktionalität zur Laufzeit geändert, ohne den Originalquelltext zu verändern. Eine gefährliche Methodik, da der Monkey-Patch nach einer Änderung des Originalquelltextes, mit hoher Wahrscheinlichkeit nicht mehr funktional ist. Bei exzessiver Anwendung dieser Methodik kommt der Entwickler schnell in Teufels Küche. Konkurrierende Monkey-Patches sind schwer nachvollziehbar, die Wartung wird somit teuer erkauft.

Auch die schlechte Trennung von Inhalt, Konfiguration und Präsentation trägt einen Teil dazu bei. Unter Zope 2 muss jede Instanz einer Klasse Attribute und Methoden bereitstellen, die Zope für die Interaktion im Framework benötigt. Das führt zu überladenen Objekten, schränkt die Portierbarkeit ein und erschwert nachträgliche Erweiterungen.[vW05b]

Trotzdem sind in den letzten Jahren hunderte von sogenannten Produkten unter Zope 2 veröffentlich worden. Die Palette reicht von einfachen Erweiterungen, wie z.B. unterschiedliche Datenbankadapter, Wikis, Blogs, bis zu e-Commerce Lösungen. Seine Praxistauglichkeit stellt Zope 2 etwa im Intranet-Einsatz der North Atlantic Treaty Organisation (NATO) oder beim Webauftritt des World Wide Fund For Nature (WWF)[18] unter Beweis. Auch bekannte Content-

[18] WWF: http://www.wwf.at/

managementsysteme wie Plone[19] oder der Collaborative Portal Server (CPS)[20] stützen sich auf Zope.[Möl05]

Für mehr Literatur zu Zope 2 empfiehlt der Autor [MB02], [AL01].

1.6.2 Zope 3

Dieses Kapitel umfasst die allgemeinen Eigenschaften und Merkmale von Zope 3. Tiefergehende Informationen und Praxisbezug sind in Kapitel 3 zu finden.

Das Zope 3 Projekt wird im Februar 2001 ins Leben gerufen. Die Community spricht sich für eine komplette Neuentwicklung des Frameworks aus, um die grundlegenden Schwierigkeiten mit Zope 2 auszumerzen. Seit Oktober 2005 ist Zope 3 auch für den Produktionseinsatz freigegeben.

Die Komponentenarchitektur, als wichtigeste Neuerung, teilt die Zuständigkeit für Inhalt (Content), Dienste (Utilities), Sichten (Views) und Adapter auf. Sie ermöglicht es, Software in kleinen und wiederverwertbaren Bausteinen zu entwickeln. Ziel einer solchen Aufteilung ist vor allem die höhere Wartbarkeit der Software. Ein anderes Datenbank-Backend, oder ein neuer Skin für eine Webapplikation soll keine Konsequenzen auf die tatsächliche Programmlogik haben. Außerdem wird für eine gewisse Übersicht und Ordnung durch Reduzierung der Komplexität gesorgt.

Auch die Wiederverwendung existierender Python-Komponenten aus den Bibliotheken gestaltet sich einfacher, als in der Vorgängerversion. Das dürfte helfen, die bisher vorhandene Spaltung zwischen Python- und Zope-Entwicklern zu überwinden [vW05a].

Damit Komponenten austauschbar bleiben, müssen ihre Schnittstellen definiert sein. Dafür zuständig sind Interfaces, die als Verträge für die Komponenten fungieren. So ist es möglich, dass unterschiedliche Implementierungen für ein Interface entwickelt werden können, da ein Interface nur das „Was“ beschreibt, nicht aber das „Wie“.

Zope 3 Komponenten kurz beschrieben:

Content Komponenten sind Datenobjekte. Die Daten sind anhand eines Schemas definiert. Ihre einzige Aufgabe besteht darin, ihre Daten zu verwalten. Das ermöglicht eine Zope-unabhängige Weiterverwendung unter Python.

Utilities Utilities sind kontextunabhängige Komponenten, die für bestimmte Aufgaben im Framework zuständig sind (z.B. Emailversand, Kodierung, Suche).

Adapter Adapter nutzen eine Komponente mit definiertem Interface, um ein neues Interface bereitzustellen. Das ermöglicht existierende Funktionalitäten zu erweitern, ohne die Originalimplementierung zu verändern.

Views sind Komponenten, die für die Darstellung anderer Komponenten zuständig sind. Views sind normalerweise als Kombination von Python-Code für Logik und ZPT für die Präsentation definiert.

[19] Plone: http://www.plone.org/
[20] CPS: http://www.cps-project.org/

Die Konfiguration des Frameworks und der Applikationen erfolgt bei Zope 2 im Quelltext selbst. Veränderungen der Konfiguration hat eine Änderung des Quelltextes zur Folge. Zope 3 Konfiguration wird in seperaten Extensible Markup Language (XML)-Dateien abgelegt. Der verwendete Dialekt ist Zope Configuration Markup Language (ZCML).

Um Entwicklern und vor allem bestehenden Applikationen die Umstellung auf Version 3 zu ermöglichen, wurde das Projekt Five (Zope 2 + Zope 3 = Five) ins Leben gerufen. Five ist ein Zope 2 Produkt, das einiges an Zope 3 Funktionalität zur Verfügung stellt. Zukünftige Zope 2 Versionen und Five werden langsam zu einer gemeinsamen Zope Version, basierend auf Zope 3, führen.

Der Großteil der momentanen Zope Projekte basiert auf Version 2. Anhand des stark wachsenden Interesses in den Communityforen und Mailinglisten[21], und der stetig steigenden Anzahl an Zope 3 Modulen im Zope Repository[22], kann in Zukunft mit einer größeren Anzahl an Referenzen gerechnet werden. Ende 2006 sind folgende größere Projekte bekannt:

Schooltool [23] ist eine Applikation für die Administration einer Schule.

Launchpad [24] ist ein Framework rund um Opensource Produkte, wie die Linux Distribution Ubuntu[25] oder die Versionierungssoftware Bazaar[26].

Weiterführende Zope 3 Literatur: [vW06], [Ric05a]

[21] Zope Users Mailingliste: http://mail.zope.org/pipermail/zope3-users/
[22] Zope Repository: http://svn.zope.org/
[23] Schooltool: http://www.schooltool.org/
[24] Launchpad: https://launchpad.net/
[25] Ubuntu: http://www.ubuntu.com/
[26] Bazaar: http://bazaar-vcs.org/

2 Python lehren

2.1 Python als erste Programmiersprache?

Warum eignet sich Python als Werkzeug für den Informatikunterricht? Ist Python als Einstiegssprache geeignet? Was zeichnet Python gegenüber anderen Programmiersprachen aus? Welche Anforderungen werden an eine "erste Programmiersprache" gestellt? Was haben die Lehrenden davon? Dieses Kapitel enthält eine Diskussion zu den vorangegangen Fragen.

Gegenwärtig sind hauptsächlich Java und C++ als Programmierprachen in Lehrgängen für Programmieranfänger anzutreffen (vgl. Kapitel 1.4) [Rad06].

> *"While students with good preliminary background, who have already committed themselves to a computing career, usually succeed in such courses, many others remain disappointed or even completely fail."*

[Rad06] schreibt, dass Studenten mit Vorkenntnissen und explizitem Interesse an der Materie es leichter haben, solche Kurse erfolgreich zu absolvieren. Studenten, die bisher keine Erfahrungen in der Programmierung haben, bleiben hier oft auf der Strecke. Die Motivation, die Materie weiter zu verfolgen, ist verständlicherweise gering. Dabei soll Unterricht üblicherweise motivieren und dadurch den Forschergeist wecken. Ob das gelingt, ist eine Frage der Gestaltung des Unterrichts und hängt nur geringfügig von der zu unterichtenden Materie ab. Jedes noch so trockene Thema kann durch interessante und fesselnde Unterrichtsgestaltung schmackhaft gemacht werden. Die Unterrichtsgestaltung hängt auch mit der Wahl der benutzten Werkzeuge, um Wissen zu vermitteln, zusammen.

Das Problem nach [Rad06] ist, dass die heute verwendeten Sprachen alle kommerziell ausgerichtet sind. Kommerzielle Sprachen sind nicht für den Unterricht ausgelegt, sondern für die industrielle Softwareentwicklung. Kommerzielle Applikationen sind komplex. Die Programmiersprachen sind dafür ausgelegt, die Entwicklung dieser Applikationen zu ermöglichen.

Der Autor unterstützt diese Argumentation. Es stellt sich jedoch die Frage, warum nicht gerade aus diesen Gründen Programmiersprachen so einfach und übersichtlich wie möglich sein sollten. Eine Ursache dafür ist sicher eine versuchte Effizienzsteigerung durch diverse Sprachkonstrukte, die den Profi effizienter arbeiten lassen, für Anfänger aber nicht leicht verständlich sind ("make the common case efficient"). [Rad06] meint in seiner Arbeit, dass genau aus diesem Grund Programmierneulinge mit Konzepten konfrontiert werden, die für den Anfang überfordern.

[BM05] bringen einen Vergleich für den sportlichen Leser, indem sie ein Beispiel aus dem Schifahren bringen. Um ein wirklich guter Schifahrer zu werden, erfordert es Übung. Schipisten sind darauf ausgelegt, das Üben zu unterstützen, indem sie je nach Schwierigkeitsgrad farblich markiert sind. Blau für Anfänger, Rot für Fortgeschrittene und Schwarz für Profis. Eine blaue Piste bedeutet aber nicht, das hier auf die wichtigen Elemente des Schifahrens zu verzichten ist. Schwünge, Geschwindigkeitskontrolle, Bremsen und die richtige Übersicht sind auch hier notwendig, sind jedoch unter entschärften Bedingungen ausübbar. Wird ein Anfänger auf die schwarze Piste geschickt, wird er oft stürzen. Die Erfahrung ist für ihn frustrierend, und er wird schnell aufgeben und den Sport verdammen.

Dieser Ausflug in eine anderes Gebiet und die einfache Erkenntnis daraus kann auf jedes Lernen

umgelegt werden. Konfrontiert man den Lernenden zu früh mit zu hoher Komplexität, kann dieser nicht genug Erfolge erleben und wird sich eher frustriert von der Thematik abwenden. Der Stoff muss sich aufbauend an den fundamentalen Ideen des Unterrichtsgegenstandes nähern, und dem Lernenden zwischendurch immer wieder Erfolgserlebnisse ermöglichen.

> *"As students progress through the introductory computer science sequence, we want them to focus on aspects of problem solving, algorithm development, and algorithm understanding. Unfortunately, many modern programming languages require that students jump into more advanced programming concepts at a point that is too soon in their development. This sets them up for possible failure, not because of the computer science but because of the language vehicle being used."*[BM05]

In der Informatik liegt das Augenmerk auf den Aspekten der Problemlösung, der Entwicklung und dem Verstehen von Algorithmen. Die heute im Unterricht angewandten Programmiersprachen führen aber zu früh in komplexere Details der Wissenschaft. Das führt auch dazu, dass nicht Informatik, sondern die Aspekte einer Programmiersprache zum Gegenstand des Unterichts werden.

> *"It is a shame that the languages that our students encounter when they come to learn to program are languages that are designed for commercial use by experienced commercial programmers. Modern languages like Java and C++ contain a host of features that most students will never need, and which we should probably admit are a mystery to many of the students' teachers."*[Jen03]

Sprachen wie Java oder C++ sind in ihrem Umfang so mächtig, das es bestimmt eher selten ist, wirklich routinierte Profis als Lektoren zu bekommen. Glücklich sind diejenigen, die in diesen Genuss kommen.

Industriesprachen

Weßhalb werden Java, C++ und C nun hauptsächlich verwendet? Geht es nach [Jen03], [LGS06], [Zel98], [LM06], [Md03] und [Sta00], ist dieser Zustand industriegetrieben. Studenten sehen Jobangebote, in denen eine bestimmte Programmiersprache als Voraussetzung verlangt wird, und wollen diese lernen, um ihre Qualifikation damit aufzuwerten. Sie haben kein Interesse, eine "Baby-Sprache"[PM06] zu lernen. Lehreinrichtungen bewerten ihre Studienpläne anhand der Nachfrage in der Industrie. Das betrifft ganz besonders Fachhochschulen, wo Lehrpläne in enger Zusammenarbeit mit der Industrie erstellt werden.

Ein Lektor des Autors, der lieber anonym bleiben möchte, ist da ganz anderer Meinung. Die obig gebrachten Überlegungen seien unwahr, vielmehr gestaltet sich der Unterricht oft nach der Verfügbarkeit von Lektoren im Bekanntenkreis der für den Studienplan verantwortlichen Personen. Diese Lektoren bringen eher ihre persönlich bevorzugten Themen und Werkzeuge in den Unterricht ein (wobei letzteres durchaus legitim und wichtig für die Qualität des gebrachten Unterichtsstoffes ist).

Nach Ansicht des Autors ist die Realität wohl eine Mischung aus beiden gerade genannten Punkten. Doch sollte es nicht anders sein? In der Informatik war es früher auch anders. Blickt man auf die Zeit vor der objektorientierten Bewegung zurück, gab es Pascal [Wir71] als allgemein anerkannte Programmiersprache für den Unterricht [Rad06],[Zel98]. Das Problem mit reinen Unter-

richtssprachen wie Pascal ist jedoch, dass sie in der Industrie nicht verwendet werden. Aus diesem Grund wurde Pascal auch weltweit aus den Lehrplänen gestrichen.

Es braucht also eine Sprache, die

- in der Industrie anerkannt ist bzw. in der Industrie Verwendung findet,
- mit der komplexe Softwareprojekte umgesetzt werden können,
- die für den Unterrichtseinsatz geeignet ist.

Betrachten wir Python, sind die ersten beiden Punkte erfüllt (siehe Kapitel 1.5). Wenden wir uns nun dem dritten Punkt zu. Ist Python für den Unterrichtseinsatz geeignet?

Hello World

Hello World ist traditionellerweise das erste Programm jedes Programmieranfängers. Starten wir gleich mit der Python Variante.

```
1 print "Hello World!"
```

Listing 4: Hello World mit Python

Über die Kommandozeile kann nun mit

```
%python helloworld.py
```

das Programm ausgeführt werden. Unter Windows reicht ein Doppelklick auf die Datei.

Sieht sehr einfach aus. [Jen03] meint aber, dass die vermeintliche Trivialität nicht unterschätzt werden sollte. Es muß klar sein, was ein Programm ist und wie das Programm erstellt, ausgeführt und das Ergebnis betrachtet werden kann.

Nun zur C++ Variante des Programms.

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main ()
6 {
7   cout << "Hello World!";
8   return 0;
9 }
```

Listing 5: Hello World mit C++

- Zeile 1: Zeilen, die mit einer Raute(#) beginnen, sind Direktiven für den Präprozessor. Diese Zeilen werden vom Compiler nicht normal ausgewertet, sondern zuvor vom Präprozessor bearbeitet. Dieser reagiert je nach Befehl unterschiedlich. In diesem Beispiel wird dem Präprozessor mitgeteilt, die Datei `iostream` zu inkludieren. Diese enthält Deklarationen der Standard Eingabe-Ausgabe Bibliothek von C++. Sie wird benötigt, da wir einen Befehl aus dieser Bibliothek später im Programm verwenden.

- Zeile 3: Alle Elemente der Standard C++ Bibliothek werden in einem eigenen Namensraum deklariert. Dieser Namensraum hat den Namen `std`.
- Zeile 5: Die `main` Funktion ist der Startpunkt eines jeden C++ Programmes. Eine Funktion wird immer mit runden Klammern definiert. Innerhalb der runden Klammern können Parameter an die Funktion übergeben werden. Die geschwungenen Klammern markieren den Bereich der Funktion. Alles innerhalb dieser Klammer wird beim Aufruf der Funktion ausgeführt. Vor dem Namen der Funktion steht die Definition des Rückgabewertes.
- Zeile 7: Das ist ein C++ Befehl (statement), den wir über eine Library inkludiert haben und einem Namesraum zugewiesen haben. Dieser spezielle Befehl repräsentiert den *Standard Output Stream* in C++. Das Beispiel zeigt, wie eine Sequenz von Zeichen ("Hello World!") in den Stream geschrieben wird. Jeder Befehl muss mit einem Semikolon (;) beendet werden.
- Zeile 8: Der `return` Befehl beendet die Funktion, der Returnwert muss vom gleichen Typ wie die Deklaration der Funktion sein. Der Returncode von 0 bedeutet, das die Funktion fehlerfrei ausgeführt wurde.

Wie zu sehen ist, müssen, bei diesem sehr simplen Beispiel, Themen, die für einen Anfänger doch sehr fortgeschritten sind, erläutert werden. Eine andere, und nach Erfahrung des Autors sehr beliebte Variante um diese Problematik zu umschiffen, ist das gezielte Ignorieren dieser Punkte. Der Vortragende ersucht die Studenten, gewisse Teile des Programmes einfach auszublenden und z.B. nur innerhalb der `main` Funktion zu arbeiten und zu denken.

> *"The C++ version has always forced me to choose between two unsatisfying options: either to explain the #include, void main(), {, and } statements, and risk confusing or intimidating some of the students right at the start, or to tell them 'just don't worry about all of that stuff now, we will talk about it later' and risk the same thing."* [Elk00]

Keine der beiden Varianten ist befriedigend. [Elk00] beschreibt den Aufwand, ein *Hello World* Programm in ein Skriptum zu verfassen, wie folgt.

> *"There are thirteen paragraphs of explanation of "Hello, world" in the C++ version, in the Python version there are only three. More importantly, the missing ten paragraphs do not deal with the "big ideas" in computer programming, but with the minutia of C++ syntax. I found this same thing happening throughout the chapters that I have completed so far. Whole paragraphs simply disappear from the Python version of the text because Python's much simpler syntax renders them unnecessary."*

[Elk00] zeigt damit, dass viel Einführungsarbeit mit C++ auf Spracheigenheiten bezogen ist. Das ist Zeitverschwendung. Studenten könnten mit Python viel früher zu ersten Erfolgserlebnissen kommen, ohne über mysteriöse Gegebenheiten, wie unerklärte und nicht verstandene Konstrukte, in den eigenen Programmen zu stolpern.

Einen Vorteil gegenüber Java hat C++ jedoch: im ersten Programm muss über Klassen nichts bekannt sein. Bei der Java Version des Programmes wird der Student sofort mit einem fortgeschrittenen Konzept konfrontiert, welches aber das Wissen über darunterliegende, objektorientierte Thematik voraussetzt. Abbildung 6 zeigt übersichtlich, welche Konzepte das unter anderem sind. Dabei fehlt bei diesem Beispiel der Konstruktor, ein weiteres neues Konzept.

Der Autor will C++ und Java nicht negativ darstellen. Es soll lediglich aufgezeigt werden, dass ein Start in die Programmierung mit einer der momentan in der Ausbildung verwendeten Sprachen

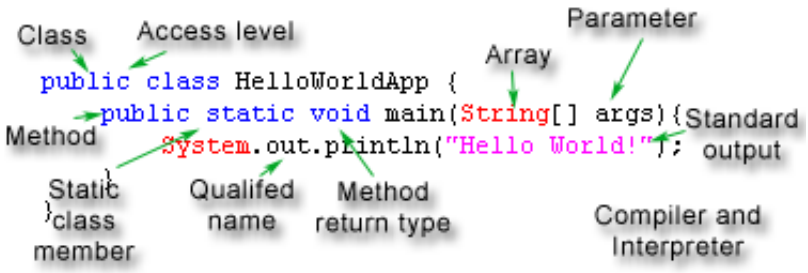

Abbildung 6: Hello World mit Java nach [Rad06]

nicht als optimal angesehen werden kann. Ein großer Vorteil von Python im Vergleich der *Hello World* Programme, ist die unterschiedliche Anzahl der Konzepte, die verstanden werden müssen, um ein erstes Programm zu schreiben. Ein erstes funktionales Programm ist auch ein erstes Erfolgserlebnis und motiviert zu mehr.

Syntax

> *"Compared to languages such as Java or C++, Python has a more intuitive syntax. Python enforces an intended and structured way of writing programs, and the code resembles pseudo code."*[LGS06]

Entwickler werden unter Python gezwungen, mit Einrückung ihres Quelltextes zu arbeiten. Eine Einrückung markiert einen Block im Programm. Der Block endet erst mit der Einrückung. Bei falscher Einrückung wird ein Fehler generiert. Dabei sieht ein Programm in Python fast genauso aus wie das Programm geschrieben in Pseudocode. Das hat den Vorteil, dass Algorithmen einfach nach Pseudocodevorlage implementiert werden können.

> *"[...] with Python, the Pseudocode is in fact also real code, and already executable."*

[Sta00] meint sogar, dass mit Python sei Pseudocode auch echter Quelltext, der schon ausführbar ist.

Die Einrückung zwingt dazu, strukturiert zu schreiben und fördert dadurch ein einheitliches Programmbild, während in anderen Sprachen verschiedene Schreibstile möglich sind, die dazu führen, dass Richtlinien zur Lesbarkeit entwickelt werden müssen.

Programme werden lesbarer, was einige Vorteile mit sich bringt. Lektoren haben es leichter, die Arbeit der Studenten zu evaluieren und finden auch schneller Fehler in Programmen. Da Python keine unterschiedlichen syntaktischen Stile erlaubt, ist es somit für Studenten einfacher, an größeren Projekten gemeinsam zu arbeiten.[Elk00]

> *"Novice programmers find indentation to be quite natural and we have found that it enforces a more readable coding style than students typically adopt when they can use braces."*[BM05]

Listing 6 zeigt ein C++ Beispiel. Frei nach dem Auge würde der zweite Befehl zum if-Block gehören. Durch das Fehlen der geschwungen Klammern umfasst der Block jedoch nur den ersten Befehl, der zweite wird unabhängig von der Verzweigung ausgeführt. Dieser Fehler passiert recht

häufig. Mit Python wäre dieses Beispiel klar, alles Eingerückte nach der Verzweigung ist dem Block zugehörig. Es herrscht eine Art "natürliche Lesbarkeit".

```
1 if(t == 1)
2     cout << "t is 1";
3     t = 2;
```

Listing 6: Beispiel zur Lesbarkeit von eingerücktem Quelltext

> *"We feel that writing structured programs, which are easy to check, follow and maintain, is one of the main lessons in any programming course. Using Python, this lesson is taught automatically."* [LGS06]

Lesbaren Quelltext zu schreiben ist ein wichtiges didaktisches Ziel. Durch Python erreicht der Student das implizit.

Semantik

Durch die dynamische Typisierung erspart man sich das Konzept der Deklaration. Der Typ einer Variable wird zur Laufzeit bestimmt. Eine Variable zeigt immer auf ein Objekt. Dieses Konzept ist leichter zu erläutern und zu verstehen, als die maschinennahe statische Typisierung, wo eine Variable ein Platz im Speicher ist, der zuvor deklariert werden muss. Typisierung ist ebenfalls eine häufige Fehlerquelle, auf die anfangs getrost verzichtet werden kann. Auch Zeiger und Speicherverwaltung sind solche Konzepte.

Die Literatur ist sich einig, dass die genannten Konzepte für eine erste Programmiersprache nicht notwendig sind, aber trotzdem zu einem späteren Zeitpunkt gelehrt werden sollten. Erfahrungen zeigen, dass die Konzepte auch viel einfacher angenommen werden, nachdem Studenten ihre ersten Erfolge mit Python erlebt haben und in der Lage sind, eigene komplexe Programme zu schreiben. Der Umstieg auf eine Hochsprache ist unproblematisch.[Jen03, Sha03, Rad06, Zel98, Elk00, PM06]

> *"We believe that it is advantageous for beginning students to spend time on the green runs[blaue Piste] learning the rudimentary ideas relating to algorithms and data structures. If you are among those of our colleagues who are frustrated with Java or C++, we encourage you to consider Python and join us on the green runs, it works."*

[Elk00] nimmt wieder das Beispiel des Schifahrens als Illustration und vergleicht Python als erste Programmiersprache mit dem Fahren auf einer blauen Piste, wo Studenten, unter entschärften Bedingungen, das Arbeiten mit Algorithmen und Datenstrukturen lernen.

Entwicklungsumgebung

Python wird mit dem Integrated Development Environment (IDLE) (Abbildung 9), einer auf das Nötigste reduzierten Entwicklungsumgebung ausgeliefert. Features sind Syntaxhighlighting, automatisches Einrücken des Quelltextes, ein einfacher Debugger und die integrierte Python Dokumentation.

Mit IDLE bekommen Programmieranfänger ein zurechtgeschnittenes Werkzeug, das nicht mit Funktionalitäten und Features überladen ist. Python ist mit IDLE für diverse Plattformen verfügbar. Der Vorteil dabei ist, dass Studenten ihre Programme unabhängig vom Betriebssystem erstellen und benutzen können [Jen03].

> *"The development environments that they [C++, Java] use are often designed for commercial programming and contain a baffling array of features that are beyond their comprehension. The problem is that novice mistakes can often generate the most arcane error messages because novices make errors that commercial programmers never would. This produces errors that are comprehensible only to the truly initiated or enlightened."* [Jen03]

Der Autor hat als Tutor eines C Kurses die Problematik mit überladenen Frameworks kennen gelernt. Hierbei wurde Microsoft Visual C++ verwendet. Die Studenten mussten sich, neben den schon erläuterten Schwierigkeiten mit der Sprache selbst, noch mit Funktionalitäten des Frameworks plagen. Es scheiterte teilweise schon an der Installation. Die Konfiguration der Compilerpfade war auch eine Wissenschaft für sich. Der Compiler meldete oft Warnings, deren Erläuterung mit dem Wissenstand der Studenten zu dieser Zeit eher zur weiteren Verwirrung beigetragen hat.

Python aus der Sicht der Unterrichtenden

Aus der Sicht eines Lektors ist Python eine Arbeitserleichterung. Die Syntax ist wie erwähnt sehr einfach und übersichtlich gehalten. Dadurch machen die Studenten auch weniger Fehler, fehlende Klammern oder Strichpunkte als Hauptfehlerursache sind nicht möglich. Fehler sind generell einfacher zu finden, da durch die erzwungene Einrückung, ein einheitlicher Stil für Lesbarkeit sorgt.[BM05, Jen03, LGS06, Elk00]

> *"The interpreter enables fast and interactive demonstration of programming concepts, and gives immediate and understandable feedback on potential errors."* [LGS06]

Die Python Interpreter Shell gibt die Möglichkeit, den Unterricht interaktiver zu gestalten. Lektoren haben die Möglichkeit, vor den Studenten mit sofortigem Feedback des Interpreters Themen zu erläuten oder Probleme zu besprechen. Studenten haben durch die Interpreter Shell die Möglichkeit, "schnell etwas auszuprobieren" und können somit beliebig experimentieren.

Unterrichtsmaterial, sowohl für Lektoren als auch für Studenten gibt es mittlerweise ausreichend. Kapitel 2.6 gibt darüber einen Überblick.

Kritik

Berichte vom erfolgreichen Einsatz im Unterricht gibt es viele [BM05, Jen03, LGS06, Sha03, Rad06, Old05, Elk00, PM06]. Die Literatur belegt, dass Python eine optimale Alternative zu Sprachen wie C++ oder Java im Unterrichtseinsatz ist. Doch auch die Kritikpunkte sollen nicht unerwähnt bleiben.

Performance Das Thema Performance von Skriptsprachen wurde in der Einleitung behandelt.

Im Unterrichtseinsatz ist diese zu vernachlässigen, da sie ja nur zeitkritische Applikationen betrifft, die im einführenden Unterricht wohl kaum anzufinden sind. Der Perfomanceunterschied ist in der Standard-Softwareentwicklung nicht zu erkennen, im Gegenzug gibt es aber eine merkbar einfachere Handhabung der Sprache und schnellere Entwicklungszeiten (rapid prototyping [Lin02]).

Geheimnisprinzip Die nicht vorhandene sprachliche Regelung des Geheimnisprinzips in der Objektorientierung wird oft kritisiert. Es ist also syntaktisch nicht möglich, private Methoden oder Attribute auf Objekten zu definieren. Interessanterweise wird dieser Schwachpunkt nicht als tragisch beschrieben.

> *"Pedagogically, this does not seem to be a major weakness, since it is still possible to teach the principles of data-hiding; it is just not enforced by the language."* [Zel98]

[Old05] lehrt den Studenten *getter* und *setter* für Attribute eines Objektes zu schreiben. Auch [LGS06] haben, trotz Befürchtungen, keinen negativen Einfluß festgestellt.

Dynamische Typisierung Der Autor betrachtet die Kritik an dynamischer Typisierung als "Glaubensfrage" und möchte dazu keine Stellung nehmen. Fakt ist, dass mit der nächsten Python Version (Python 3000) eine optionale Typisierung möglich gemacht wird.

Versionen [Old05] meint, dass der schnelle Entwicklungszyklus von Python, der sich natürlich nicht an akademische Semesterpläne hält, zu Problemen führt, wenn sich Spracheigenschaften zwischen den Releases ändern. Der Autor kann das nicht bestätigen, da die Python Community hier sehr wohl eine sichere Releasepolitik betreibt. Einzig neue Sprachfeatures können, falls passend, in die Curricula eingearbeitet werden. Alter Quelltext wurde durch neue Releases noch nie defunktional.

Einrückung des Quelltextes Überraschenderweise hat [Elk00] festgestellt, dass Studenten leichter die Blocknotation mit geschwungenen Klammern verstehen, als die mit Einrückung. Es ist nur eine Sache der Gewohnheit. Die Vorteile der Einrückung (wie in diesem Kapitel erwähnt) überwiegen aus der Sicht des Autors.

2.2 Programmierparadigmen

Das Wort **Paradigma** kommt aus dem Griechischen und bedeutet Vorbild, Beispiel, Muster oder Abgrenzung. Es hat viele unterschiedliche Bedeutungen, die sich in der Wissenschaft, der Linguistik, Organisationstheorie etc. wiederfinden. Für diese Arbeit und dieses Kapitel ist das Programmierparadigma interessant:

Ein **Programmierparadigma** ist das einer Programmiersprache oder Programmiertechnik zugrundeliegende Prinzip. Es ist eine Sichtweise, die zur Lösung eines Problems mittels einer Programmiersprache eingenommen wird.[Gra03]

Python ist primär eine objektortientierte Sprache, unterstützt also das objektorientierte Softwareparadigma:

> *"Python is a dynamic object-oriented programming language that can be used for many kinds of software development."* [Pyt06]

Jedoch ist der Entwickler bei der Verwendung von Python an kein spezielles Paradigma gebunden. Python kann somit als **Multiparadigmen-Sprache** bezeichnet werden.

In diesem Kapitel wird auf folgende Paradigmen kurz eingegangen und deren Realisierung mit Python erläutert:

- Imperative Programmierung
  - Prozedurale Programmierung
  - Modulare Programmierung
  - Objektorientiere Programmierung
- Deklarative Programmierung
  - Funktionale Programmierung
  - Logische Programmierung
- Weitere Paradigmen
  - Design by Contract
  - Aspektorientierte Programmierung

2.2.1 Programmierparadigmen im Unterricht

Nach [Pla93] und [Wes99] sind Multiparadigmensprachen für den Unterricht gut geeignet. Es ist wichtig den Studenten in kein Korsett hineinzuzwingen, sondern ein breites Schema an Möglichkeiten zur kreativen Entfaltung anzubieten, denn

> *"verschiedene Menschen nehmen zur Lösung von gleichen Aufgaben unterschiedliche Sichtweisen ein, und ebenso kann es sein, dass ein Mensch für unterschiedliche Aufgaben verschiedene Lösungsansätze verfolgt. Für die wenigsten Probleme gibt es die optimale Lösung, und so empfiehlt es sich, der- oder demjenigen, der das Problem löst, die Wahl zu überlassen, wie er oder sie am besten mit einer Aufgabe umgeht. Daher ist es für den praktischen Einsatz von Programmiersprachen notwendig, diese unterschiedlichen Sichtweisen, die zu verschiedenen Lösungsstrategien führen, in geeigneter Form ausdrücken zu können."* [Gra03]

Das bietet den Studenten den Freiraum, zu Beginn ihrer Programmierkarriere mittels **eines** Werkzeugs, nämlich der Programmiersprache, unterschiedliche Denkweisen und Programmieransätze zu erproben. Die Studenten können sich voll und ganz auf das Verstehen der, den Paradigmen zugrundeliegenden Prinzipien konzentrieren und begreifen, dass ein Paradigma nicht von der verwendeten Programmiersprache abhängt, sondern von der Sichtweise auf eine Problemstellung und deren Lösung.

> *"Python ermöglicht sowohl den Logo-Zugang mit Rekursion und Listen, als auch den sequentiellen Zugang der imperativen Sprachen (und wer will - es geht auch funktional). Sprachen wie Delphi oder C++ zwingen den Schüler erst einmal, 'nur nichts*

falsch' zu machen. Das mag Sprach-Juristen erfreuen, lässt aber den Forschungsdrang der Schüler verkümmern."[Urb06]

2.2.2 Imperative Programmierung

Imperative Programmierung ist das älteste Programmierparadigma und ist an die Architektur des Von-Neumann-Rechners angelehnt. Ein Programm ist eine Folge von Anweisungen, die nacheinander abgearbeitet werden; dieses befindet sich zu jeder Zeit in einem Zustand, der vom Arbeitsspeicher und der Programmumgebung (Dateisystem, Peripherie) definiert ist. Seiteneffekte (Ein-/Ausgabe, Speichermodifikation, Variablen) bewirken eine Änderung des Zustandes über der Zeit.

Prozedurale Programmierung

Die prozedurale Programmierung, die als Synonym für imperative Programmierung gesehen werden kann, ermöglicht das Zusammenfassen von Anweisungen zu Unterprogrammen (Prozeduren). Unterprogramme werden dann von verschiedenen Stellen des Programms aufgerufen und ausgeführt.

Das ist der erste Ansatz, Aufgaben in Teilaufgaben zu zerlegen und so eine einfachere, übersichtlichere und vor allem wiederverwertbare Programmierung zu ermöglichen.

Es bietet sich an, dieses Paradigma als erstes im Unterricht vorzustellen, da Von-Neumann üblicherweise sehr früh im Informatikunterricht ein Thema ist, und so ein einfacher Konnex hergestellt werden kann.

Um den zeitlichen Ablauf eines Programmes zu veranschaulichen, kann der Python Interpreter als unterstützendes Visualisierungswerkzeug im Unterricht verwendet werden:

```
>>> fahr = 0
>>> while (fahr < 120):
...     celsius = 5./9 * (fahr - 32)
...     print 'Fahrenheit %6.1f = Celsius %6.3f' % (fahr, celsius)
...     fahr += 20
...
Fahrenheit    0.0 = Celsius -17.778
Fahrenheit   20.0 = Celsius -6.667
Fahrenheit   40.0 = Celsius  4.444
Fahrenheit   60.0 = Celsius 15.556
Fahrenheit   80.0 = Celsius 26.667
Fahrenheit  100.0 = Celsius 37.778
```

Die prozedurale Variante des obigen, sehr simplen, Beispiels ergäbe dann mit ausgelagerter Berechnung:

```
1 def fahr2cels(fahr):
2     celsius = 5./9 * (fahr - 32)
3     print 'Fahrenheit %6.1f = Celsius %6.3f' % (fahr, celsius)
4     return celsius
```

```
5
6 fahr = 0
7 while (fahr < 120):
8     fahr2cels(fahr)
9     fahr += 20
```

Listing 7: Prozedurales Programmierbeispiel

Modulare Programmierung

Modulare Programmierung geht einen Schritt weiter und zerlegt die Prozeduren zusammen mit Daten in logische Einheiten: die Module. Der nächste Schritt ist, Aufgaben in größeren Softwareprojekten zu abstrahieren. Die jeweiligen Module werden einzeln geplant, entwickelt und getestet.

Mit Python ist jede Datei ein eigenes Modul. Ein Verzeichnis mit Modulen wird Package[27] genannt. Die Python Standardbibliothek und alle Erweiterungen werden als Packages entwickelt und eingebunden.

Wenn die Prozedur aus Listing 7 in eine eigene Datei names `mymodule.py` ausgelagert wird, muss diese als Modul eingebunden werden:

```
>>> from mymodule import fahr2cels
>>> fahr2cels(0)
Fahrenheit    0.0 = Celsius -17.778
```

Um das Modul nun für externe Pythonmodule zugänglich zu machen, muss die Datei `__init__.py` erstellt werden, die das Verzeichnis als Pythonpackage markiert. Importiert wird das Modul auf unterschiedliche Arten. Dazu ein Beispiel:

```
>>> from mydirectory.mymodule import fahr2cels
```

Das Kennenlernen der Modularisierung fördert das Verständnis der Austauschbarkeit und Wiederverwendbarkeit von Software in einem sehr frühen Stadium des Unterrichts. Ohne viele Vorkenntnisse kann der Sinn und Zweck einer solchen Aufteilung erläutert werden. Zu diesem Zweck könnten sich jeweils zwei Gruppen von Studenten zusammenfinden, die unterschiedliche Aufgaben zu lösen haben. Die Aufgabe ist, ein Programm mittels eines weiteren Moduls zu entwerfen. Dieses Modul wird trotz unterschiedlicher Aufgabenstellung von beiden Studentengruppen benötigt. Interessant ist, die Studenten im ersten Anlauf unwissend in den Einzelgruppen arbeiten zu lassen. Danach werden die ”Partnergruppen“ einander vermittelt und die jeweiligen, sicher unterschiedlich implementierten, Lösungen des Moduls von den Studenten zu einem gemeinsamen Modul angeglichen. Natürlich wird das nicht ganz friktionsfrei von statten gehen. Die Änderungen werden mit großer Wahrscheinlichkeit auch die beiden Programme der Gruppen betreffen.

Im zweiten Anlauf setzt man, mit neuer Aufgabenstellung, die Studenten schon vorher über ihre jeweiligen Partner in Kenntnis. Nun soll ein gemeinsames Modul konzipiert werden, bevor man sich über die eigene Implementierung im Detail Gedanken macht. Die Schlußfolgerungen daraus werden anschließend diskutiert, die Studenten haben die ersten Untiefen der kollektiven Softwareentwicklung erlebt.

[27] http://docs.python.org/tut/node8.html

Objektorientierte Programmierung

Die Objektorientierte Programmierung (OOP) erweitert das prozedurale Paradigma um Objekte.

> *"Ein Objekt ist ein Gegenstand des Interesses, insbesondere einer Beobachtung, Untersuchung oder Messung. Objekte können Dinge, Personen, oder Begriffe sein [...] ein Objekt besitzt einen bestimmten Zustand und reagiert mit einem definierten Verhalten auf seine Umgebung [...] jedes Objekt besitzt eine Identität, die es von anderen Objekten unterscheidet [...] ein Objekt kann ein oder mehrere andere Objekte kennen."*[Bal04]

Heide Balzert schreibt weiters, dass der Zustand eines Objektes dessen Attribute und die jeweiligen Verbindungen zu anderen Objekten umfasst. Das Verhalten eines Objektes wird durch die Menge seiner Operationen (Methoden, Funktionen) beschrieben. Eine Änderung oder eine Abfrage des Zustandes ist nur mittels Operationen möglich (Geheimnisprinzip). Objekte werden durch Klassen definiert und können ihre Eigenschaften und Operationen vererben, was eine Schachtelung des Quelltextes ermöglicht.

Folgendes Listing zeigt eine typische Python Klasse:

```
1  class Bankkonto:
2
3     def __init__(self,startbetrag):
4        """Konstruktor"""
5        self.kontostand = startbetrag
6
7     def einzahlung(self, betrag):
8        self.kontostand = self.kontostand + betrag
9
10    def auszahlung(self, betrag):
11       self.kontostand = self.kontostand - betrag
12
13    def anzeigen(self):
14       print self.kontostand
```

Listing 8: einfache Python Klasse

Wie zu Beginn dieses Kapitels erwähnt, ist Python eine objektorientierte Programmiersprache. In Python ist alles ein Objekt, wie folgendes Beispiel zeigt.

```
>>> konto = Bankkonto(100)
>>> Bankkonto
<class __main__.Bankkonto at 0x401dfbcc>
>>> konto
<__main__.Bankkonto instance at 0x401e972c>
>>> konto.anzeigen
<bound method Bankkonto.anzeigen of <__main__.Bankkonto instance at 0x401e972c>>
```

Python lässt sich somit hervorragend für den aufbauenden Unterricht einsetzen. Während mit Java ab dem ersten Programm der objektorientierte Ansatz verstanden werden muss, kann bei Python der Unterricht mit dem verständlicheren, prozeduralen Paradigma begonnen werden. Danach arbeitet sich der Unterricht über den modularen Ansatz zur Objektorientierung.

Als Abweichung zur OOP Definition ist das Geheimnisprinzip in Python zu sehen. Es gibt keine private/public Schlüsselwörter. Somit kann jedes Attribut einer Pythonklasse, egal aus welchem Kontext, geändert werden. Es existiert jedoch eine Namensvereinbarung, die Methoden mit doppeltem Unterstrich (z.B. `__auszahlung()`) vor ihrem Namen, Zugriffe außerhalb der Klasse untersagt. Das mag für traditionelle OOP Entwickler eigenartig klingen, für den Unterricht scheint es aber keine Einschränkung zu sein (siehe Kapitel 2.1).

2.2.3 Deklarative Programmierung

Wie der Name ausdrückt, ist ein Programm *deklarativ*, wenn es beschreibt, *was* passiert. Die imperative Programmierung ist das genaue Gegenstück, sie beschreibt, *wie* etwas umgesetzt wird.

In der deklarativen Programmierung wird eine problemerläuternde Spezifikation geschrieben, die von der Implementierung der jeweiligen Sprache analysiert und abgearbeitet wird. Structured Query Language (SQL) ist eine bekannte deklarative Sprache; eine SQL Abfrage beschreibt gewünschte Daten bzw. den Retourwert, der tatsächliche Algorithmus wird vom Relational Database Management System (RDBMS) dahinter ausgeführt. Ein weiteres Beispiel ist Hypertext Markup Language (HTML); hier beschreibt der Templateentwickler das Aussehen der Webseite mittels Markup, jedoch nicht den tatsächlichen Renderingprozess des Browsers.

Imperative Programme definieren also explizit einen Algorithmus, um das gewünschte Ergebnis zu erzielen. Deklarative Programme hingegen spezifizieren das Ergebnis und überlassen die Umsetzung der Implementierung dahinter.

Funktionale Programmierung

In der funktionalen Programmierung wird mit Funktionen gearbeitet. Hierbei ist zu beachten, dass nicht die Funktionen (Unterprogramme, Prozeduren) aus der Informatik, sondern Abbildungen aus der Mathematik gemeint sind.

Abbildung nach [Har03b]:

> "Seien M, N Mengen und jedem $x \in M$ sei genau ein $y \in N$ zugeordnet. Durch M, N und diese Zuordnung wird eine Abbildung von M nach N definiert [...] Abbildungen sind Funktionen, wenn M und N Teilmengen der reellen Zahlen sind."

In der Literatur wird, ungeachtet dessen, nur von Funktionen gesprochen. Aus Gründen der Einfachheit und des eindeutigen Paradigmennamen, wird auch in dieser Arbeit weiterhin von Funktionen gesprochen.

Ein Programm nach dem funktionalen Paradigma besteht allein aus Funktionsdefinition,

```
>>> def square(x): return x*x
...
>>> def twice(f,x):return f(f(x))
...
```

Funktionsanwendung

```
>>> square(4)
16
```

und Funktionskomposition. Bei der Kompostion werden mehrere Funktionen zusammengesetzt.

```
>>> twice(square, 4)
256
```

Funktionen können als Argumente und Rückgabewerte verwendet werden, man spricht von *Funktionen höherer Ordnung*.

```
>>> square
<function square at 0x401dee64>
```

Python liefert einen Standardsatz an Werkzeugen zur funktionalen Programmierung. Die Befehle `map()`, `filter()`, `reduce()` und `lambda`, ebenso wie die Packages `functools`[28] und `functional`[29].

Die oben genannten Befehle sind jedoch seit längerem *deprecated* und werden mit Python 3000 durch andere, schon vorhandene Konstrukte, wie *list-comprehensions*, abgelöst[30]. Eine list-comprehension sei hier als Veranschaulichung in mathematischer und implementierter Form aufgezeigt: $S = \{x | x \in \mathbb{N}, x^2 < 2000\}$

Die Implementierung dazu lautet:

```
>>> from itertools import count
>>> S = [x for x in count() if pow(x,2) < 2000]
```

Die funktionale Programmierung ist sicher nicht für Programmieranfänger zu empfehlen. Für einen fortgeschrittenen Kurs oder als unterstützendes Werkzeug für Mathematik oder Physikunterricht ist sie durchaus geeignet. Es kann mit derselben Sprache eine komplett neue Sichtweise erlernt werden.

Weiterführender Literaturhinweis: [Bar90]

Logische Programmierung

Bei der logischen Programmierung ist die Logik eines Programmes selbst als Programm zu betrachten. Logik kann als problemnahe und effiziente Programmiersprache verwendet werden. Ein Programm besteht aus einer Menge von Fakten, Regeln und Anweisungen. Folgende schematische Darstellung ist der Grundgedanke der logischen Programmierung:

$$algorithm = logic + control$$

Wobei *logic* dem rein logischen (deklarativen) Aspekt, also dem Wissen über die Zielsetzung des Algorithmus (auch Spezifikation genannt), und *control* dem prozeduralen Aspekt, also der Strategie der Anwendung von Ableitungsregeln, entspricht.

[Sch99] schreibt über die Vorteile des logischen Paradigmas wie folgt:

[28] functools: http://docs.python.org/lib/module-functools.html
[29] functional: http://oakwinter.com/code/functional/
[30] Python 3000 and reduce(): http://www.artima.com/weblogs/viewpost.jsp?thread=98196

> *"Dem Anfänger wird ein erheblicher Teil des Programmieraufwandes abgenommen. Er beschreibt sein zu lösendes Problem mit Fakten und Regeln, ohne sich um die Lösungssuche selbst zu kümmern. Das hat den Vorteil, dass er mit einfachen Anfragen an sein Programm, komplizierte Lösungsmechanismen auslösen kann, die er schrittweise erkunden und für sich nutzen lernt [..] Es werden rekursive Denk- und Arbeitsweisen, die zu den fundamentalen Ideen der Informatik gehören, fast spielerisch erlernt."*

Betrachten wir folgendes Beispiel:

Alle verheirateten Männer hassen ihre Schwiegermütter
Hannes ist der Mann von Christa
Tanja ist die Mutter von Christa

Hannes hasst Tanja

Die deklarative Interpretation erlaubt, über die logische Korrektheit einer Klausel zu diskutieren.

$hasst(Z, X)$ gilt, wenn $mutter(X)$ und $kind(Y, X)$ und $verheiratet(Z, Y)$ gelten. Oder auch

$$\forall X(\forall Z(mutter(X) \wedge kind(Y, X) \wedge verheiratet(Z, Y) \rightarrow hasst(Z, X))$$

Die prozedurale Interpretation liest die Klausel als Definition einer Prozedur durch eine Reihe von Prozeduraufrufen:

Um $hasst(Z, X)$ zu beweisen, beweise $mutter(X)$, dann $kind(Y, X)$ und schliesslich $verheiratet(Z, Y)$.

Somit sind Programmiersprachen nach dem logischen (deklarativen) Paradigma auch prozedural anzusiedeln und können daher auch als Multiparadigmen-Sprachen bezeichnet werden. Der bekannteste Vertreter der logischen Programmiersprachen ist PROgramming in LOGic (PROLOG).

Logische Programmierung wird unter Python nicht unterstützt, wenngleich einige Versuche[31] und Vorschläge[32] in diese Richtung existieren. Die entsprechende SIG enthält wenig Information, was mangelndes Interesse signalisiert. Auf dem Gebiet der logischen Programmierung scheint PROLOG der Platzhirsch zu sein. Anscheinend gibt es so gut wie keinen Bedarf, hier zusätzliches Werkzeug zu schaffen.

2.2.4 Weitere Paradigmen

Mit Python können noch weitere Paradigmen angewandt werden. Diese zählen, nach Ansicht des Autors, jedoch nicht zu den fundamentalen Paradigmen, sondern sind Softwaredesignelemente bzw. Konzepte in der Softwareentwicklung.

[31] http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/360698 und http://bedevere.sourceforge.net/
[32] http://codespeak.net/pypy/dist/pypy/doc/constraints-and-logic.html

Design by Contract

[Plö97] erklärt Design by Contract (DBC) mit folgenden Worten:

> *"A systematic approach to building correct software systems [..] an effective framework for quality assurance [...] A method for documenting software."*

DBC ist somit ein Werkzeug, um Fehler in der Softwareentwicklung zu vermeiden, und dient gleichzeitig der Quelltextdokumentation. Dabei werden mit Vorbedingungen, Nachbedingungen und Invarianten *Verträge* definiert, die zur Laufzeit des Programms überprüft werden.

In der Programmiersprache Eiffel ist DBC Bestandteil der Programmiersprache selbst. Unter Python ist DBC eine Erweiterung des objektorientierten Paradigmas, die mit Metaklassen[33] umgesetzt wird. Es existieren fertige Packages[34], um DBC zu implementieren.

Aspektorientierte Programmierung

Aspektorientierte Programmierung (AOP) ist ebenfalls eine Weiterentwicklung von OOP. Bei AOP wird der Quelltext um eine weitere Ebene, die *Aspekte*, abstrahiert. Aspekte sind Funktionalitäten, die wenig bis nichts mit den jeweiligen Objekten zu tun haben, aber dennoch oft benötigt werden, und deswegen als Aspekte ausgelagert werden. Diese Aspekte können nun jederzeit zur Laufzeit zugeschalten werden und somit die Funktionalität der Objekte erweitern oder verändern. Somit soll die Komplexität tief verschachtelter Objekte gelockert werden. Die entwickelte Software wird wartbarer und wiederverwendbarer.[CW05]

Dieses Paradigma ist im Vergleich zu den bisherig vorgestellten Paradigmen ein sehr junges und wird seinen Einzug in die Lehre erst in einigen Jahren erfahren. Das lässt sich mit der sogenannten *Trägheit der Lehre* erklären.

> *"It takes several years before ideas developed in research settings can be tested, evaluated, and disseminated to practicing professionals so that they can be used."* [FW78]

Relevante Technologien aus der Industrie benötigen bis zu zehn Jahre, um tatsächlich flächendeckend unterrichtet zu werden. Als Beispiel sei die Unified Modeling Language (UML) erwähnt, die 1997 als Standard akzeptiert wird, deren Einzug in die Lehre aber selbst heute noch nicht abgeschlossen ist. Der Vorteil dieses Prozesses ist die natürliche Auslese von Modeerscheinungen, was wiederum das Zeitkriterium einer fundamentalen Idee bestätigt.

Für Python existieren einfach zu integrierende AOP Module wie Pythius[35]. Teilweise sind diese Module sehr simpel und abstrakt gehalten, sind aber für die Untermalung der Theorie im Unterricht vollkommen ausreichend.

[33] Python Metaklasse: http://www.python.org/2.2/descrintro.html#metaclasses
[34] PyDBC: http://www.nongnu.org/pydbc/
[35] Pythius: http://pythius.sourceforge.net/

2.3 Algorithmen

Ein Algorithmus ist eine endliche Menge von Regeln, die eine endliche Folge von Operationen zur Lösung eines Problems beschreibt [Knu98].

Er ist ein schrittweises Verfahren zur Berechnung von gesuchten Ausgabegrößen aus gegebenen Eingabegrößen mit nachfolgenden Eigenschaften nach [Ade05].

deterministisch: Ein Algorithmus ist deterministisch, wenn die Abfolge der auszuführenden Schritte eindeutig vorgegeben ist. Es existiert also ein festgelegter Ablauf. Zu jedem Zeitpunkt ist der nächste Schritt eindeutig definiert.

determiniert: Bei gleichen Eingangsgrößen liefert der Algorithmus immer das gleiche, eindeutige Ergebnis.

finit: Er ist mit endlichen Mitteln beschreibbar (natürliche Sprache, Quelltext, etc.).

terminierend: Der Algorithmus bricht nach endlich vielen Schritten ab.

Der Algorithmus ist ein Rezept, um ein Problem zu lösen. Für die Lösung eines Problems gibt es oft unterschiedliche Algorithmen, die sich in Ausführungszeit und Quelltextlänge unterscheiden (Abbildung 7).

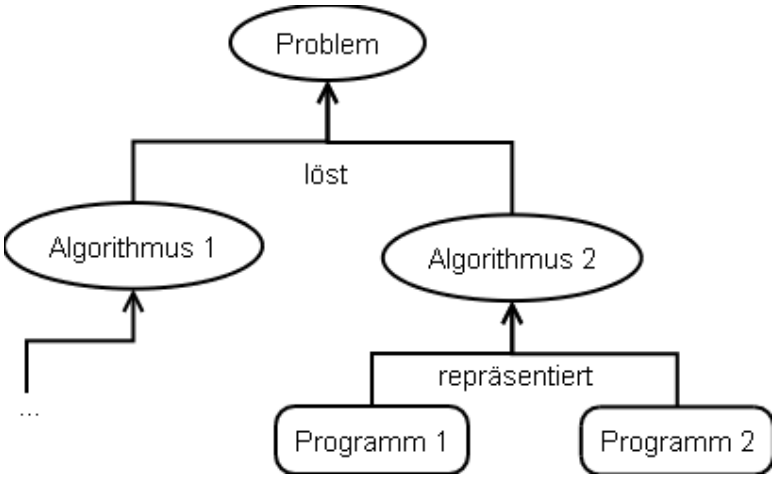

Abbildung 7: Problem - Algorithmus - Programm

Als Beispiel zur Erläuterung eines Algorithmus in natürlicher Sprache, wird der Euklidische Algorithmus aus [Euk03] zitiert.

> *"Nimmt man bei Vorliegen zweier ungleicher Zahlen abwechselnd immer die kleinere von der größeren weg, so müssen, wenn niemals ein Rest die vorangehende Zahl genau mißt, bis die Einheit übrig bleibt, die ursprünglichen Zahlen gegeneinander prim sein."*

Der moderne Euklidische Algorithmus ersetzt die wiederholten Subtraktionen durch eine Division mit Rest. Das kann mit Pseudocode wie folgt (Listing 9) beschrieben werden.

```
1 euklid(a,b)
2   solange b ungleich 0
```

```
3       temp wird a mod b
4       a wird b
5       b wird temp
6   return a
```

Listing 9: Euklidscher Algorithmus - Pseudocode

Eine tatsächliche Implementierung des Algorithmus zeigt Listing 10.

```
1 def euklid(a,b):
2     while b > 0:
3         a, b = b, a % b
4     return a
```

Listing 10: Euklidscher Algorithmus - Python

Das Erlernen von grundlegenden Algorithmen ist eines der wichtigsten fachdidaktischen Ziele. Es ist nicht nur ein fachübergreifendes Konzept, sondern auch ein essentielles Werkzeug, um komplexe Problemstellungen lösen zu können. Dabei ist es wichtig, den Studierenden einen Algorithmus auch ausprobieren zu lassen. Das einfache Erläutern mit natürlicher Sprache und Pseudocode, ist zwar als Unterstützung und erste Beschreibung notwendig, hat aber nicht den entsprechenden Lerneffekt. Es muss eine Programmiersprache als Hilfsmittel angewandt werden. Dazu eignet sich natürlich Python, da man sich durch die einfache und Pseudocode-ähnliche Syntax mehr auf die Implementierung und das Verstehen des Algorithmus konzentrieren kann, als auf das Verstehen der Programmiersprache. Diese Aussage wird durch nachfolgendes Zitat aus [Cho02] gestützt.

> *"Design and analysis of algorithms are a fundamental topic in computer science and engineering education. Many algorithms courses include programming assignments to help students better understand the algorithms. Unfortunately, the use of traditional programming languages forces students to deal with details of data structures and supporting routines, rather than algorithm design. Python represents an algorithm-oriented language that has been sorely needed in education. The advantages of Python include its textbook-like syntax and interactivity that encourages experimentation."*

[Cho02] spricht sogar von einer *algorithmus-ortientierten* Sprache, wenngleich dieser Ausdruck etwas übertrieben erscheint. Jeder Algorithmus lässt sich in jeder beliebigen Programmierprache implementieren und zeigt die jeweiligen verschiedenen Lösungswege eines Algorithmus auf. [Cho02] scheint damit einfach die Eignung Pythons für die praktische Anwendung von Algorithmen im Unterricht unterstreichen zu wollen.

Im Folgenden werden nun zwei Algorithmen erläutert und in verschiedenen Programmiersprachen implementiert. Der Autor hat dazu die Sortieralgorithmen *bubblesort* und *quicksort* gewählt.

Bubblesort

Dieser Algorithmus verhält sich bekannterweise nicht besonders effizient. Es werden, der Reihe nach, zwei benachbarte Elemente einer Liste verglichen und vertauscht, falls diese in der falschen Reihenfolge vorliegen. Bis zur fertig sortierten Liste sind im Regelfall mehrere Durchläufe notwendig. Wird aufsteigend sortiert, steigen die größeren Zahlen wie Blasen im Wasser auf. Aus diesem Grund wurde der Algorithmus *bubblesort* getauft.

Trotzdem gibt es einige Gründe, gerade Bubblesort für den Unterricht auszuwählen. Diese sind modifiziert nach [Urb06]:

- Es ist lehrreich zu erkennen, warum dieser Algorithmus schlecht ist.
- Auf Basis des schlechten Beispiels lassen sich andere Sortieralgorithmen leichter erläutern.
- Die quadratische Laufzeit $O(n^2)$ ist sofort erkennbar.
- Es kann versucht werden, noch schlechtere Sortieralgorithmen zu finden. Das weckt den Ehrgeiz der Studierenden und fördert das Interesse für die Thematik.
- Im Gegenzug kann an Verbesserungen des Algorithmus gearbeitet werden. Beispielsweise könnten Elemente, die einmal am oberen Ende platziert wurden, nicht mehr überprüft werden (siehe Listing 11 und Listing 12).

```
1  def bubblesort(list):
2      swapped = True
3      while (swapped):
4          swapped = False
5          j = 0
6          while j < len(list) -1:
7              if list[j+1] < list[j]:
8                  swapped= True
9                  list[j], list[j+1]= list[j+1], list[j]
10             j = j + 1
11     return list
```

Listing 11: Bubblesort mit Python

```
1  def BubbleSort(list):
2      for i in range(0, len(list)):
3          for j in range(0, len(list) - 1 - i):
4              if list[j] > list[j + 1]:
5                  list[j], list[j + 1] = list[j + 1], list[j]
6      return list
```

Listing 12: Verbesserter Bubblesort mit Python

Listing 11 zeigt den normalen *bubblesort* Algorithmus mit Python, Listing 12 den verbesserten Algorithmus. Als Vergleich dazu zeigt Listing 13 die Implementierung in Java. Dabei ist schon Overhead in der Java-Implementierung zu erkennen, obwohl die Implementierung sehr einfach ist. Um die Javavariante zu schreiben, muss der Studierende die Objektnotation und das dahinterliegende, objektorientierte Paradigma beherrschen.

```
1  public class BubbleSort {
2      public static void bubblesort (int[] list){
3          for(int i=1; i<list.length; i++) {
4              for(int j=0; j<list.length-i; j++) {
5                  if (list[j]>list[j+1]) swap(list, j, j+1);
6              }
7          }
8      }
9
10     private static void swap (int[] list, int i, int j) {
11         int temp = list[i];
12         list[i] = list[j];
```

```
13          list[j] = temp;
14      }
15  }
```

Listing 13: Verbesserter Bubblesort mit Java

Quicksort

Der *quicksort* Algorithmus arbeitet nach dem Prinzip "teile und herrsche" (*divide and conquer*). Dabei wird versucht, statt einer großen Liste, viele kleine Listen zu sortieren und diese danach wieder zusammenzufügen.

1. Rekursionsabbruch bei null bis einem Element in der Liste.
2. Auswählen eines Elementes als *Pivot* Element.
3. Aufteilen der Liste in zwei Teile:
   - Liste mit Elementen größer als das Pivot
   - Liste mit Elementen kleiner als das Pivot
4. rekursive Anwendung auf beide Teillisten

Die Wahl des Pivot Elementes ist entscheidend für die Effizienz des Algorithmus. Im *worst-case* erhält man ein Zeitverhalten von $O(n^2)$, das Mittel ist $O(nlogn)$. Das geschieht, wenn der Pivot immer so gewählt wird, dass es zu keiner gleichwertigen Aufteilung der Liste kommt. Um das zu vermeiden, wird üblicherweise die *median of three* Methode zur Auswahl des Pivots gewählt. Dabei wird aus dem ersten, letzen und mittleren Element der Liste der Median gewählt, welcher als Pivot für den Sortieralgorithmus fungiert. Zwecks Einfachheit und aus Platzgründen wird in den folgenden Beispiellistings darauf verzichtet, und der Pivot als

- zufälliges Element (Listing 14),
- Element aus der Mitte der Liste (Listing 16),
- erstes Element (Listing 17) oder
- letztes Element (Listing 15)

definiert.

Mit dem nachfolgendem Listing wird der Algorithmus mit C und Python implementiert.

```
1  #include <stdlib.h>
2
3  #define LEN 10
4
5  void quicksort(double list[], int left, int right);
6
7  int main(void) {
8    double list[LEN] = {7, 3, 1, 9, -2, 0, 16, 3, 11, 4};
9
10   quicksort(list,0,LEN-1);
11
```

```
12   return 0;
13 }
14
15 void swap(double list[], int i, int j) {
16   double tmp = list[i];
17   list[i] = list[j];
18   list[j] = tmp;
19 }
20
21 int Random(int i, int j) {
22   return i + rand() % (j-i+1);
23 }
24
25 void quicksort(double list[], int left, int right) {
26   int last = left, i;
27
28   if (left >= right) return;
29
30   swap(list,left,Random(left,right));
31   for (i = left + 1; i <= right; i++)
32     if (list[i] < list[left])
33       swap(list,++last,i);
34   swap(list,left,last);
35   quicksort(list,left,last-1);
36   quicksort(list,last+1,right);
37 }
```

Listing 14: Quicksort mit C

```
1  def quicksort(list,anfang,ende):
2      if anfang<ende:
3          p = partition(list,anfang,ende)
4          quicksort(list,anfang,p-1)
5          quicksort(list,p+1,ende)
6
7  def partition(list,von,bis):
8      pivot = list[bis]
9      i = von-1
10     for j in range(von,bis):
11         if list[j]<=pivot:
12             i = i+1
13             list[i],list[j] = list[j],list[i] #swap
14     list[i+1],list[bis] = list[bis],list[i+1] #swap
15     return i+1
16
17 list = [7, 3, 1, 9, -2, 0, 16, 3, 11, 4]
18 quicksort(list,0,len(list)-1)
```

Listing 15: Quicksort mit Python

Listing 15 lässt sich etwas verkürzen. Listing 16 und 17 zeigen, dass ein stark verkürzter Quelltext unter Python noch lesbar bleibt.

```
1  def quicksort(list):
2      if len(list)<=1: return list
3      pivot = list[len(list)/2]
4      return quicksort(filter(lambda i,p=pivot: i<p, list)) + \
```

```
5             filter(lambda i,p=pivot: i==p, list) + \
6             quicksort(filter(lambda i,p=pivot: i>p, list))
7
8 list = [7, 3, 1, 9, -2, 0, 16, 3, 11, 4]
9 quicksort(list)
```

Listing 16: Quicksort mit Python (funktionales Paradigma)

```
1 def quicksort(list):
2     if not list: return []
3     return quicksort([x for x in list if x < list[0]]) + [list[0]]+ \
4             quicksort([x for x in list[1:] if x >= list[0]])
5
6 list = [7, 3, 1, 9, -2, 0, 16, 3, 11, 4]
7 quicksort(list)
```

Listing 17: Quicksort mit Python (list-comprehensions)

Zugegebenerweise sieht die C Implementierung (Listing 14) für das geübte Auge nicht viel schwieriger aus als die Python Implementierungen. Bei genauerem Hinsehen sind die Unterschiede jedoch merkbar. Konzepte wie Mainfunktion, Typen von Funktionen, Typisierung bei Variablen und Moduleinbindungen können doch zum Stolperstein werden, wenn einem Studenten die Sprache C nicht besonders geläufig ist. Python hingegen kommt mit viel weniger Grundkonzepten aus (siehe Kapitel 2.1). Überraschenderweise ist die Implemtierung mittels list-comprehensions (Listing 17), trotz aller Kürze, noch erstaunlich lesbar und verständlich.

2.4 Python in Fachgegenständen

Wie aus den vorangegangenen Kapiteln hervorgeht, eignet sich Python hervorragend für den Einsatz im Informatikunterricht. Einerseits ist Python eine optimale Sprache, um das Programmieren zu lehren, andererseits können die grundlegenden Konzepte wie Programmierparadigmen und Algorithmen gut erarbeitet werden.

In welchen Fachgegenständen kann Python noch verwendet werden? Und warum ist es gut, einen Unterrichtsgegestand mit Python zu unterstützen?

Diese Fragen sollen an einem Beispiel aus der **Mathematik** erläutert werden.

```
>>> def dreieckszahl(n):  return n*(n+1)/2
...
>>> [dreieckszahl(n) for n in range(1,10)]
[1, 3, 6, 10, 15, 21, 28, 36, 45]
```

[Urn04] bedient sich des Beispiels der Dreieckszahl (Abbildung 8) und meint, dass der Mathematikunterricht der perfekte Einstieg in das Programmieren ist.

> *"What I like [...] is the connection between rule-driven (program driven) number sequences, such as 1, 3, 6, 10.. and the corresponding, visually appealing shapes. This forges connections between algebra and visualizations, but in a way more basic than by means of Cartesian coordinates (or any coordinates). Moreover such sequences provide an ideal beginning for learning to program..."*

Abbildung 8: Dreieckszahl

Weiters geht daraus hervor, dass die Verknüfung von Theorie, praktischer Umsetzung und dem visuellem Feedback durch den Interpreter, oder eine grafische Implementierung[36] den Lernerfolg erhöhen kann.

Diese Aussage wird von [Bro97] gestützt, wo es heißt, dass der höchste Lernerfolg durch *aktive Lehrmethoden* und *direkt gesteuerte Erfahrung* erzielt wird. Eine aktive Lehrmethode ist z.B. die Erarbeitung und Veranschaulichung der gelehrten Inhalte mittels Python.

Auch andere Themengebiete der Mathematik lassen sich gut mittels Python unterstützen. Induktion, Rekursion, Algebra, Vektoren, Funktionen, Wahrscheinlichkeit und Kryptographie[37] sind nur Ausschnitte möglicher Teilgebiete.

Folgend nun ein Auszug von Fachgegenständen, welche mit Python als unterstützendes Werkzeug bereichert werden können.

- **Betriebssysteme** Prozesse, Threads, Semaphore, Queues, Stacks, ...
- **Internettechnologien** Client-Server Entwicklung, Protokolle, ...
- **Genetik**[38]
- **3D Engineering**[39]
- ...

Es gibt viele Anwendungsgebiete, und diesen sind auch keine Grenzen gesetzt. Durch die aktive Community und die Offenlegung des Quelltextes ist es möglich, Python mit fehlender Funktionalität zu erweitern. So kann der Unterricht auf die speziellen Bedürfnisse des Lehrenden angepasst werden.

2.5 Open-Source und Community

Python ist eine Open-Source Software. Die Community kann als Unterrichtsmaterial angesehen werden sowohl vom Unterrichtenden als auch von den Studenten. Wie kann die Community genutzt werden, was kann sie den Studenten vermitteln?

[36] z.B. mit VPython: http://vpython.org/
[37] Python Cryptography Toolkit: http://www.amk.ca/python/code/crypto
[38] pgapy: http://pgapy.sourceforge.net/ oder Biopython: http://biopython.org/
[39] Blender: http://www.blender.org/

> *"Die Nutzung von freier Software im Schulunterricht, die u.a. der Begründer der Free Software Foundation Richard Stallman fordert, wird seit Jahren durch verschiedene Projekte und Organisationen [...] unterstützt. Als Argumente für den Einsatz von Open-Source Software dienen dabei vor allem die kostenlose und lizenzrechtlich unproblematische Anschaffung von Betriebssystemen und Programmen sowohl für die Schule als auch für die einzelnen Schülerinnen und Schüler, aber auch ethische Aspekte."*

[Sch06] beruft sich auf [Bin00], [Sta03], [Gra04]. Dabei ist von freier Software die Rede, die in dieser Arbeit der Open-Source Software gleichgestellt wird. Trotzdem gibt es Unterschiede, die aber nicht Gegenstand dieser Arbeit sind. Weitere Informationen dazu sind bei [Neu02] und [Har03a] zu finden.

Python kann somit lizenztechnisch völlig bedenkenlos an die Studenten verteilt werden; die Installation ist einfach und vor allem kostenlos. Wie schon erwähnt, gibt es Python für diverse Plattformen und Betriebssysteme. Das ist auch ein großer Vorteil für die Systemadministration des Lehrinstituts, die Python einfach auf allen Rechnern zur Verfügung stellen kann.

Als Beispiel möchte der Autor seine Tätigkeit als Tutor in der Fachhochschule (FH) Technikum Wien erläutern, wo die Programmiersprache C mit Microsoft Visual C++.net 2005 als Werkzeug gelehrt wird. Die Lektoren dieses Unterrichtsgegenstandes sind hervorragend, jedoch ist die Wahl des Tools mehr als fragwürdig. Die Probleme während des Tutorium sind, zu einem nicht unbeträchtlichen Anteil, Probleme mit dem Werkzeug, das einfach viel zu umfangreich und undurchsichtig für die simple Erstellung eines C Programms ist. Die Studenten werden schon allein bei der Installation mit Themen konfrontiert, die zu diesem Zeitpunkt für sie unverständlich und deshalb frustrierend sind. Der Compiler meldet C++ Fehlermeldungen, die ohne Hintergrundwissen schwer zu erklären sind. Weiters ist die Installation so aufwändig, dass die Software nur in bestimmten Räumen zur Verfügung steht. Manche Studenten konnten die Installation auf ihren privaten Laptops nicht durchführen. Eine Verbesserung wäre das Entwickeln auf einer Linux Shell, wobei natürlich ein einführender Unterricht davor notwendig wäre. Alternativ könnte z.B. lcc-win32[40], ein Windows C-Compiler mit IDE verwendet werden.

Bei diesem Beispiel schneidet Microsoft Visual C++ nicht gut ab, obwohl das Framework frei zu erwerben ist. Die Software ist jedoch nicht frei, der Quelltext ist nicht zugänglich. Wäre das der Fall, könnte theoretisch ein motivierter Student die Unannehmlichkeiten der Installation ausräumen und hier eine Vereinfachung vornehmen.

Wichtig ist auch, zu erwähnen, dass freie Software das Erkennen von Konzepten, anstatt das Erlernen von Produkten fördert. Es ist möglich, zu sehen "wie es andere tun", und so die eigene Sicht und Arbeitsweise zu verbessern.

Auf die Community kann jederzeit zurückgegriffen werden, wenn es um Fragestellungen, Diskussion oder Vorschläge geht. Auch kommerzielle Produkte haben Communities, der Autor kann jedoch bestätigen, dass Communities freier Software mit einer komplett anderen Motivation und Hilfsbereitschaft versehen sind. Am Beispiel der Python EDU-SIG[41] sieht man, wie weltweit gemeinsam Unterrichtsmaterialen erarbeitet werden. Die Erfahrungen damit werden dann in der eigenen Newsgroup[42] diskutiert.

[40] lcc-win32: http://www.cs.virginia.edu/%7Elcc-win32/

[41] Python EDU-SIG: http://www.python.org/community/sigs/current/edu-sig/

[42] EDU-SIG newsgroup archiv: http://mail.python.org/pipermail/edu-sig/

Diese Eigendynamik sollte nach Ansicht des Autors genutzt werden. Beispielsweise wäre die Einrichtung einer eigenen deutschsprachigen Newsgroup für alle Studenten, die Informatik mit Python lernen, ein durchaus interessantes Experiment. Natürlich müsste diese Idee an die jeweiligen Institute und zuständigen Lektoren kommuniziert werden. Diese Idee könnte generell auf andere Lehrgegenstände übertragen werden, da die Institute alle ihre eigenen Lösungen verwenden, und sich nur einige wenige Studenten in den hauseigenen Kommunikationsplattformen finden. Eine Kanalisierung auf einer Plattform klingt vielversprechend. Man stelle sich vor, wie ein Mathematikforum für österreichische Studenten funktionieren könnte, wenn wirklich alle wissen, dass dort Information, Austausch und Hilfe von Gleichgesinnten zu erhalten ist. Ein Versuch wäre es wert. Erste Ansätze in dieser Richtung sind schon zu beobachten, die Ausführung scheint aber mangelhaft zu sein[43].

Der Open-Source Gedanke und die im letzten Zitat erwähnten ethischen und sozialwissenschaftlichen Zusammenhänge sind ein sehr interessantes Diskussionsthema abseits von technischen Fächern, die viel mit dem Umfeld des wissenschaftlichen Arbeitens, der Politik und Wirtschaft zu tun haben. Eine umfassende, auch philosophische Betrachtung zu dieser Thematik liefert [CFG05].

Auch auf kritische Betrachtung zu Open-Source sei mit [Ric05b] verwiesen.

2.6 Material, Werkzeuge und Frameworks für den Schuleinsatz

Um Unterrichten mit Python weiter zu verbreiten, ist es unerlässlich, auf qualitativ hochwertiges Lehrmaterial und Werkzeuge zurückgreifen zu können. Schließlich sollten diese Materialen als Grundlage eines jeden Unterichtenden, von Entscheidungsträgern der Lehre abgesegnet werden. Weiters fällt es mit gutem Material leichter, das eigene Curriculum zu gestalten. Werkzeuge sind wiederum wichtig für die Durchführung des Unterrichts, wobei hier das Augenmerk auf die einfache Installation und Handhabung zu legen ist.

- Zuerst ist **Python** selbst zu erwähnen. Die Installation unter Windows ist unkompliziert. Einzig das Zielverzeichnis der Installation muß eingestellt werden. Die Dateiendung `.py` wird mit Python registriert. Die Installation umfasst zusätzlich das Python IDLE, welches für das erste Programmierbeispiel sofort verwendet werden kann. Abbildung 9 zeigt dazu einen Screenshot.

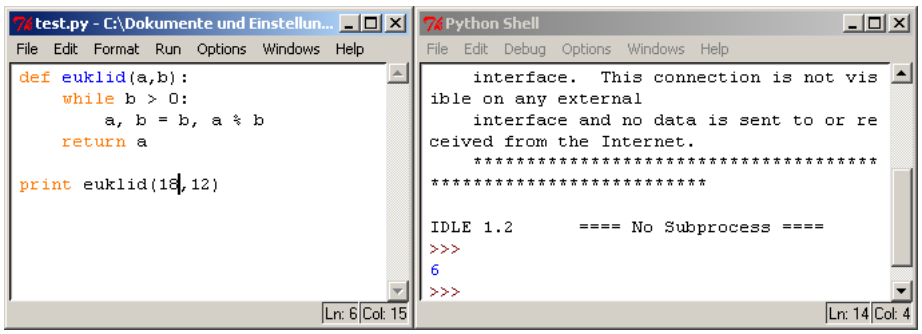

Abbildung 9: Python IDLE unter Windows

[43] z.B. http://www.schule.at/

Die Installation unter Linux ist ebenfalls einfach, denn entweder wird ein fertiges Packet der gewählten Linux-Distribution verwendet oder händisch von den Sourcen kompiliert (`configure, make, make install`). Unterschiedliche Python Versionen können in getrennten Verzeichnissen installiert und nebenläufig betrieben werden.

Für Materialien, Dokumentation und Hilfestellungen ist die Python Website [Pyt06] der erste Anlaufpunkt.

- Ein speziell für Programmierneulinge zugeschnittenes Programm hat [Lin06] erarbeitet. Dessen zweite Auflage ist mit Ende 2006 gerade neu erschienen. Der Autor unterrichtet selbst an einem österreichischen Gymnasium und hat dieses Buch als Begleitbuch für den Einstieg in das Programmieren geschrieben. Anders als der Titel "**Python für Kids**" vermuten lässt, ist die Lektüre auch für ältere Semester geeignet, sofern diese noch Programmierneulinge sind. Natürlich ist es auch als unterstützendes Lehrmaterial zu betrachten.

  [Lin06] hat für sein Buch ein eigenes Grafikmodul geschrieben. **xturtle** ist eine Erweiterung bzw. Verbesserung des `turtle` Moduls. Die Idee des `turtle` Moduls stammt aus der Programmiersprache Logo aus dem Jahre 1966, und ist ein Werkzeug für den Einführungsunterricht in die Programmierung. Dabei wird eine programmierbare "Schildkröte", die nach entsprechenden Befehlsvorgaben hin- und herlaufen kann, verwendet. Falls der Zeichenstift abgesenkt ist, zeichnet diese ihren zurückgelegten Weg auf (Abbildung 10).

  > *"It's intended to be used in those educational situations, where an important requirement is easy access to graphics. "*[Lin06]

  Das `turtle` Modul verwendet Tkinter[44] für die Darstellung von Grafiken und ist momentaner Bestandteil der Python Distribution. `turtle` wird voraussichtlich mit Python 2.6 von `xturtle` abgelöst[45].

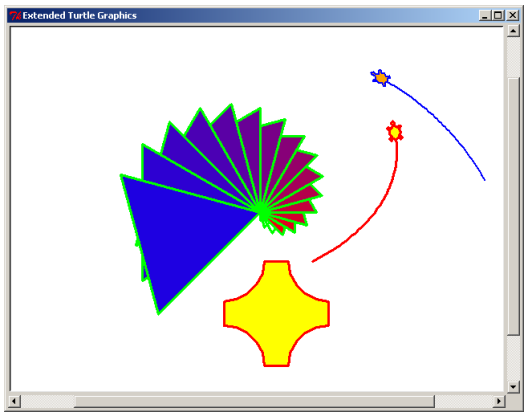

Abbildung 10: xturtle

Auf der Website des Autors[46] finden sich zudem andere Unterrichtsmaterialien, wie z.B. Folien für den Einführungsunterricht.

[44] Tkinter: http://wiki.python.org/moin/TkInter
[45] Diskussion: http://mail.python.org/pipermail/python-dev/2006-June/066677.html
[46] weitere Materialien: http://ada.rg16.asn-wien.ac.at/ python/

- Wolfgang Urban arbeitet ebenfalls an einem österreichischen Gymnasium und stellt auf seiner Website[47] umfassendes Material für Mathematik, Physik und Informatikunterricht zur Verfügung. Darunter implementiert er auch beispielhaft die Fibonacci-Folge in unterschiedlichen Variationen und mit verschiedenen Programmiersprachen. Mit diversen Sortieralgorithmen unter Python setzt er sich genauer auseinander.
- "**How to Think Like a Computer Scientist**"[AD02] ist ein Lehrbuch, das unter der GNU's Not Unix (GNU) Free Documentation License verfügbar ist. Die Autoren haben dieses Buch während des Aufbaus ihres Curriculums verfasst und in weiterer Folge fortlaufend verbessert. Aufgrund des positiven Feedbacks aus aller Welt wurde das Buch für Java, Logo und C++ adaptiert. Eine zweite Version von [AD02] ist derzeit in Arbeit und kann auf der Website des Projektes[48] eingesehen werden. Geplanter Fertigstellungstermin ist Anfang Juni 2007.
- Materialien für fortgeschrittene Entwickler bietet Greg Wilson mit "**Software Carpentry**"[49]. Der Kurs richtet sich an Softwareentwickler, die schon praktische Erfahrung in Umgang mit Projekten gesammelt haben. Basis des Kurses ist Python. Das Thema ist die Wissenschaftlichkeit der Softwareentwicklung und die Vermittlung von Techniken, die das Arbeitsleben eines Entwicklers erleichtern sollen. Python wurde für diesen Kurs gewählt, um "den zu vermittelnden Inhalten am wenigsten im Weg zu stehen". Das Material ist unter einer Open-Source Lizenz verfügbar.
- Die "**Python Bibliotheca**"[50] ist eine Lehrplattform, die Tutorials, Workshops und Lehrmaterialien für den Unterricht mit Python bietet. Unter anderem ist ein Video[51] mit Interviews bekannter Python-Größen, wie Guido van Rossum, zu finden.
- **VPython**[52] ist eine Sammlung von Python Klassen, die einen einfachen Zugang zu Echtzeit 3D Animationen erlaubt. Anwendungsbereich ist beispielsweise der Physikunterricht, wo das Arbeiten mit Vektoren oder physikalischen Abläufen wie Gravitation, visualisiert werden können (Abbildung 11).

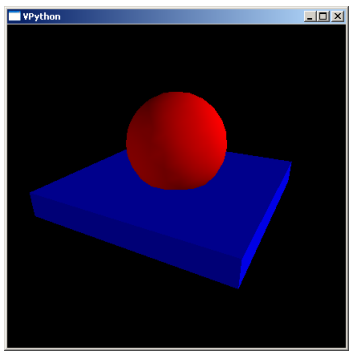

Abbildung 11: Gravitationssimulation mit VPython

Auch das Visualisieren von Graphen ist mit VPython möglich (Abbildung 12).

[47] Wolfgang Urban: http://www.hib-wien.at/leute/wurban/
[48] Open Book Project: http://ibiblio.org/obp/thinkCS/python.php
[49] Software Carpentry: http://www.swc.scipy.org/
[50] Python Bibliotheca: http://www.ibiblio.org/obp/pyBiblio/
[51] Video: "A Python Love Story": http://www.ibiblio.org/obp/pyBiblio/pythonvideo.php
[52] VPython: http://vpython.org/

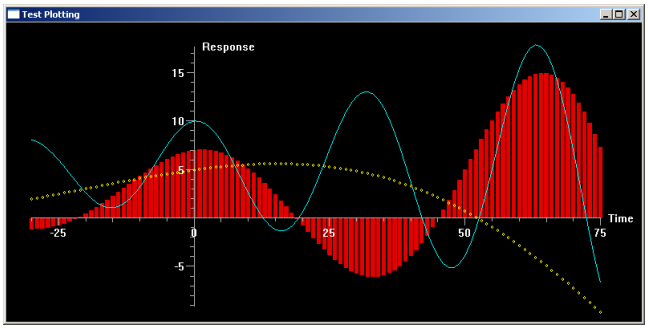

Abbildung 12: Darstellung eines Graphen mit VPython

- Mit **Tkinter**[53] und **wxPython**[54] wird die GUI Programmierung unter Python ermöglicht. Tkinter bedeutet Toolkit (Tk) Interface, ist also eine Schnittstelle zu Tk, und Tk ist eine GUI-Bibliothek geschrieben in der Skriptsprache Tool command language (Tcl) (man spricht dabei auch von Tcl/Tk[55]). Tkinter ist Bestandteil der Python-Standardbibliothek. wxPython ist ein Wrapper um die C++-Bibliothek wxWidgets[56]. Beide Module sind auf verschiedenen Betriebssystemen einsetzbar. Tkinter eignet sich vor wxWidgets durch die geringe Komplexität für den Einstieg in die GUI-Programmierung. wxPython skaliert besser und ist daher für die Entwicklung von komplexeren GUIs geeignet.
- **Gato**[57] ermöglicht die Visualisierung von Algorithmen auf Baumstrukturen (wie z.B. Suchbaumalgorithmen, kürzester Weg, Kruskal oder Dijkstra). Das Tool wird auf der Universität Köln entwickelt und bei Kursen über Algorithmen und diskrete Mathematik verwendet (Abbildung 13).

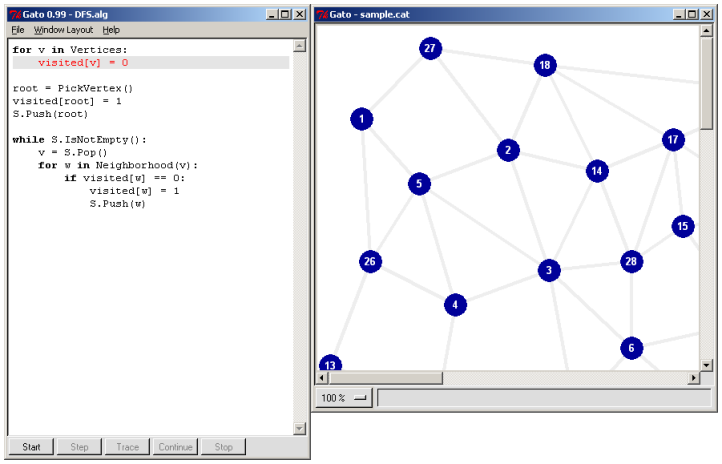

Abbildung 13: Gato

- Für die Entwicklung von Spielen gibt es eine Sammlung von Modulen, die unter **Pygame**[58] zusammengefasst sind. Pygame basiert auf der freien Simple DirectMedia Layer (SDL) -

[53] Tkinter: http://wiki.python.org/moin/TkInter
[54] wxPython: http://www.wxpython.org/
[55] Tcl/Tk: http://tcl.sourceforge.net/
[56] wxWidgets: http://wxwidgets.org/
[57] Gato: http://gato.sourceforge.net/
[58] PyGame: http://www.pygame.org/

Bibliothek, die ein Application Programming Interface (API) für Eingabe, Sound und Grafik zur Verfügung stellt. Damit wird ein einfacher, high-level Zugang zur Spieleprogrammierung ermöglicht.

3 Softwareentwicklung im Unterricht mit Zope

3.1 Komponentenarchitektur

Zope 3[59] besteht aus einer Komponentenarchitektur. Um diese zu erläutern, soll zuerst der Begriff einer Softwarekomponente und der des CBSE erläutert werden.

In der heutigen Softwareentwicklung gibt es ein hohes Maß an Anforderungen (modifiziert nach [Goe05] und [Nin96]):

- Hohe Qualität wird erwartet.
- Die Software muss billig sein.
- Usability: die Software muß einfach und möglichst ohne Lernaufwand zu bedienen sein.
- Neue Versionen und Updates werden in immer kürzeren Zeitabständen verlangt. Auf Änderungen muss schnell reagiert werden.
- Stabilität, Zuverlässigkeit und Robustheit.
- Neue Software muss bestehende, alte Software integrieren und erweitern.
- Software wird immer komplexer.

Auf der anderen Seite erwartet man im Softwareentwicklungsprozess folgende Merkmale:

- Die Erstellung der Software muss billig sein.
- Projektlaufzeiten müssen eingehalten werden.
- Prozess- und qualitätsorientierte Entwicklung.

[Goe05] schreibt weiters:

> *"Der Widerspruch aus den Kundenwünschen und den Anforderungen an die Entwicklung führt fast zwangsläufig dazu, dass wir es heute als selbstverständlich akzeptieren, dass Software Fehler enthält. Wir wundern uns nicht mehr, wenn wir mehrmals am Tag unsere Systeme neu booten müssen, weil sie komplett abgestürzt sind (system crash). Den damit verbundenen Verlust an wertvoller Arbeitszeit oder vielleicht sogar Datenverlust nehmen wir als unerfreulich, aber alltäglich einfach so hin. Würden wir uns bei einem Gebäude oder einem Auto auch so leicht zufrieden geben? Wohl kaum!"*

CBSE soll dabei Abhilfe schaffen. Das Ziel komponentenbasierter Softwareentwicklung ist, die Produktivität und Qualität in der Softwareentwicklung zu steigern. Anstatt jedesmal "das Rad neu zu erfinden", soll Software aus Standardkomponenten "zusammengebaut" werden. Software soll nicht mehr entwickelt, sondern gekauft werden. Dabei spricht die Literatur von Commercial Off-The-Shelf (COTS) Komponenten.

> *"It [CBSE] focuses on reusing and adapting existing components, as opposed to just coding in a particular style. CBSE encourages the composition of software systems, as*

[59] Im Folgenden wird immer von Zope ab der Version 3 gesprochen. Ist die Rede von einer früheren Version, wird explizit die Versionsnummer angeführt.

> *opposed to programming them. CBSE is a process that aims to design and construct software systems using reusable software components.“*[Kva04]

Ziel ist es, eine Industrialisierung der Softwareentwicklung zu erreichen. Damit würde die Softwareentwicklung zu einer klassischen Ingenieurswissenschaft mit einheitlichen Standards werden. Es gäbe einen eigenen Markt für den Handel mit Komponenten (Abbildung 14).

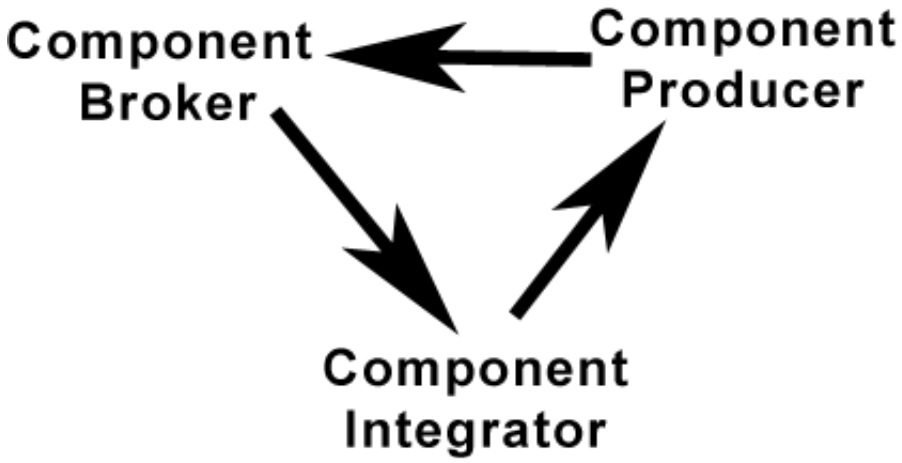

Abbildung 14: Beispiel eines Komponentenmarktes nach [Nin96]

Ein hochgestecktes Ziel, wenn man die rasante technologische Entwicklung betrachtet. Kaum werden Vorschläge ausgearbeitet, sind sie auch schon wieder veraltet. Die Herausforderung besteht darin, die Thematik frei nach dem Horizontalkriterium der fundamentalen Idee nach Schwill zu bearbeiten. Wie können Standards geschaffen werden, die möglichst frei von Technologie, aber trotzdem mit jeder Technologie eingehalten werden können? Und wie kann es in der kommerziellen Welt der Konkurrenz zu *einer* einheitlichen Lösung, also zu einem Standard, kommen?

Dazu kommen noch die Akzeptanzschwierigkeiten bei vielen Entwicklern, die ihre Arbeit als kreativen Prozess sehen und sich durch die Überstrukturierung von Prozessen bedrängt fühlen. [Str99] sieht die Softwareentwicklung sogar als eine Kunst und bekräftigt unter anderem Folgendes:

> *“Bedenkt man, daß der Computer eine universelle Maschine ist und Software praktisch beliebige Vorgänge beschreiben kann, wird klar, dass der Ansatz eines universellen Vorgehensmodells zum Scheitern verurteilt ist.“*

[Str99] unterteilt Softwareentwicklung, unter Berufung auf andere Quellen, in zwei unterschiedliche Kulturen, die griechische und die römische Kultur. Wobei zur kurzen Erläuterung hier die Schubladen *Entwickler* und *Manager* zu verstehen sind. Wenngleich sehr überspitzt formuliert, erläutert diese Aufteilung eine weitere Problematik, die das Finden einer gemeinsamen und akzeptierten Ingenieurswissenschaft erschwert.

Spinnt man den Faden weiter, kann eine Verbindung zur unterschiedlichen Weltanschauung von Open-Source Software und kommerzieller Closed-Source Software gesehen werden. CBSE geht laut Literatur davon aus, dass Komponenten als *Blackbox* ausgeliefert werden. Das heißt, der Kunde kennt die Implementierung nicht, nur die Schnittstellendefinition. Dieser Ansatz widerspricht zutiefst der Open-Source Philosophie. CBSE kann jedoch nur funktionieren, wenn der Kunde die

Implementierung der Komponente nicht verändern kann, andererseits wäre die Idee fehlerfreier Software wieder hinfällig. Der Hersteller kann nur für *sein* Produkt garantieren, das er mit seinem Spezialwissen qualitativ hochwertig hält. Ändert der Kunde etwas an der Komponente, erlischt die Garantie und Fehlern sind wieder Tür und Tor geöffnet. Eine mögliche Abhilfe wäre die Auslieferung von Komponenten als *glass-box* [Stü02], wobei der Quelltext der Komponenten mit ausgeliefert wird, dieser jedoch nicht verändert werden darf.

Diese Arbeit will und kann hier keine Lösung erarbeiten. Die bisherige Abhandlung zeigt, dass CBSE uns die nächsten Jahre begleiten wird, und das Thema Modularisierung, Reuse und Testen von Software einen wichtigen Platz in der Ausbildung angehender Softwareentwickler einnehmen muß.

Was ist nun eigentlich eine Komponente? [Stü02] hat die unterschiedlichen Definitionen aus der Literatur zusammengefasst und folgende Merkmale abgeleitet:

- Komponenten sind klar abgrenzbar: sie können unabhängig entwickelt und eingesetzt werden.
- Sie stellen Schnittstellen nach außen zur Verfügung und kapseln ihre Daten.
- Sie haben definierte Abhängigkeiten von ihrem Kontext.

Eine nach [Ost04] allgemein anerkannte Definition einer Komponente ist:

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

Wenden wir uns jetzt der Komponentenarchitektur von Zope zu. Die Zope Komponentenarchitektur kann nicht im globalen Umfeld einer CBSE Komponente gesehen werden. Eine Komponente unter Zope ist eine Stufe davor anzusiedeln, nämlich in der Entwicklung selbst. Zope selbst ist ein Applikationsserver, der auf einer Komponentenarchitektur (`zope.component`, `zope.interface`) und weiteren Komponenten basiert. Diese *core* Komponenten können durch die Komponentenarchitektur flexibel angepasst und erweitert werden, ohne die originale Funktionalität zu verändern.

Passend dazu definiert [Stü02] drei Typen von Komponenten unterschiedlicher Granularität: Elementarbausteine, Baugruppen und Systeme.

> *"Elementarbausteine sind der kleinste Komponententyp. Sie haben beliebige Abhängigkeiten von anderen Komponenten und sind nicht für sich ausführbar [...] Baugruppen sind Komponenten mittlerer Größe. Sie bestehen aus Elementarbausteinen und/oder anderen Baugruppen. Sie haben klar begrenzte Abhängigkeiten von anderen Komponenten und sind nicht für sich ausführbar. [...] Systeme sind die größten Komponenten. Sie sind aus Baugruppen zusammengesetzt, haben minimale Abhängigkeiten von anderen Komponenten und sind für sich ausführbar."*

Eine Zope Komponente ist als Baugruppe einzustufen, während die Komponente, von der CBSE spricht, eher dem System zuzuordnen ist. Die Zope Komponentenarchitektur ist im Prinzip ein Entwicklerwerkzeug zum Software Reuse, das den gleichen Merkmalen, wie bei [Stü02] dargelegt, unterliegt. Einziger Unterschied ist, dass die Zope Komponente nicht ausgeliefert und *deployed*,

sondern für die Entwicklung einer Applikation[60] verwendet wird. Diese Applikation kann theoretisch als Komponente nach CBSE verwendet werden.

Zope selbst hat keine komplexe Verteilungstechnologie wie z.B. Enterprise Java Beans (EJB) mit Remote Method Invocation (RMI) hat. Zope kann per Standard mit XML-Remote Procedure Call (RPC) umgehen[61]. Dafür ist es möglich, auf verteilte Konzepte wie Common Object Request Broker Architecture (CORBA) oder Webservices zu setzen. Für Python exisitieren mehrere ORB[62], und für den Zope Server gibt es eine experimentelle Komponente für das Simple Object Access Protocol (SOAP)[63].

Auch wenn die Zope Archtitekur aus den genannten Gründen nicht der Idee des CBSE entspricht, verfolgt sie doch den Komponentengedanken und ist, nach Ansicht des Autors, für die Vermittlung der Thematik im praktischen Bezug gut im Unterricht anwendbar. Der Einsatz von Python wurde im ersten Abschnitt dieser Arbeit behandelt. Ist den Studenten die Sprache geläufig, geht es im Unterricht nur noch um das Erlernen neuer Konzepte. Die Anwendung erfolgt mit gewohntem Werkzeug.

Das modulare Komponentendenken ist eine fundamentale Idee der Softwareentwicklung und gehört vermittelt. Der Autor kennt neben Zope kein anderes Framework, das ohne in die Komplexität von verteilter Software einzutauchen, diese Thematik leicht zugänglich macht.

3.2 ZODB

Wie in Kapitel 1.6 schon erwähnt, wird Zope mit der ZODB ausgeliefert. Die ZODB ist für die Persistierung der Pythonobjekte zuständig. Damit die in der Applikation verwendeten Objekte bei einem Neustart des Zope Servers nicht verloren gehen, werden diese Objekte persistiert.

ZODB arbeitet mit dem `pickle`[64] Modul, um Objekte schreiben und lesen zu können. Daten werden dabei von komplexen Objekten in einen Bytestream umgewandelt. Dieser kann nun z.B. in einem File gespeichert werden.

```
>>> pickle.dump(object, file)
```

Um aus einem Bytestream wieder ein Objekt zu erhalten, ist ebenfalls nur eine Zeile Quelltext notwendig.

```
>>> object = pickle.load(file)
```

Man spricht dabei von *pickling* (Serialisierung) und *unpickling* (Deserialisierung). Die ZODB kümmert sich um die gesamte Maschinerie: Caching, Schreiben und Lesen. Das Ganze ist natürlich transaktionssicher[65]. Um ein Objekt zu persistieren reicht eine simple Vererbung.

```
>>> from persistent import Persistent
```

[60] Unter Zope typischerweise eine Webapplikation.

[61] Die Zope Community hat bisher keinen Bedarf in der Integration aufwendigerer Technologien gesehen.

[62] z.B. omniORB: http://omniorb.sourceforge.net/

[63] http://svn.zope.org/soap/

[64] pickle: http://docs.python.org/lib/module-pickle.html

[65] siehe Maillinglisteneintrag "ZODB implementiert kein ACID": https://mail.dzug.org/pipermail/zope/2007-June/003937.html

```
>>> class MyPersistentClass(Persistent):
...     def __init__(self):
...         self.id = ''
...         self.name = ''
...         self.description = ''
...
```

Als Standard schreibt die ZODB in ein File im Dateisystem. Diese Implementierung wird "FileStorage" genannt. Es ist möglich, auf andere Implementierungen zuzugreifen. Diese sind "DirectoryStorage"[66], "ClientStorage" und "Adaptable Persistence Engine (APE)"[67].

Aus diesen Alternativen ist die "ClientStorage" erwähnenswert, mit der Daten mittels Zope Enterprise Objects (ZEO) auf mehrere Rechner verteilt werden. Damit ist Clustering von Webapplikationen allein auf Basis kostenloser Open-Source-Software möglich [Möl05].

Wie kann die ZODB nun im Unterricht verwendet werden? Es ist sinnvoll, im Zuge der Verwendung von Zope in einigen wenigen Unterrichtseinheiten auf das Thema einzugehen. Dabei sollte jedoch nicht zu sehr ins Detail gegangen werden. Es bietet sich an, die ZODB zur Veranschaulichung der unterschiedlichen Handhabung von objektorientierten und relationalen Datenbanken zu zeigen.

> *"An advantage of relational database systems is their programming-language neutrality. Data are stored in tables, which are language independent. An application must read data from tables into program variables before use and must write modified data back to tables when necessary. This puts a significant burden on the application developer. A significant amount of application logic is devoted to translation of data to and from the relational model."* [Ful00]

Bei relationalen Datenbanken wird, unabhängig von der Implementierung mittels SQL, auf die Daten zugegriffen und diese mit der Applikation verwoben. Dabei ensteht ein nicht unerheblicher Aufwand für den Entwickler, der sich um die Handhabung der Daten selbst kümmern muss.

> *"An alternative is to retain the tabular structure in the program. For example, rather than populating objects from tables, simply create and use table objects within the application. In this case, high-level tools can be used to load tables from the relational database. With sufficient knowledge of database keys, tools could automate saving data when tables are changed. A disadvantage of this approach is that it forces the application to be written to the relational model, rather than in an object-oriented fashion. The benefits of object orientation, such as encapsulation and association of logic with data are lost [...] Object databases provide a tighter integration between an applications object model and data storage. Data are not stored in tables, but in ways that reflect the organization of the information in the problem domain. Application developers are freed from writing logic for moving data to and from storage."*

[Ful00] schreibt hier von zwei Varianten. Die eine ist eine Hybridlösung, nämlich das Objektmapping. Hierbei werden die Daten in der Applikation verwendet, die notwendigen Abfragen an die relationale Datenbank geschehen im Hintergrund. Hier ist der Entwickler jedoch zu stark an das relationale Denken gebunden.

[66] DirectoryStorage: http://dirstorage.sourceforge.net/
[67] APE: http://hathawaymix.org/Software/Ape

Die zweite Variante ist das Verwenden einer objektorientierten Datenbank, wie die ZODB. Hierbei rücken Daten und Integration näher zueinander, ohne jedoch problematisch zu verschmelzen. Die Entwicklung muss sich nicht um das Speichern und Lesen von Daten kümmern.

In Verbindung mit der ersten Variante soll SQLAlchemy[68] erwähnt werden, ein Python SQL Toolkit und objektrelationaler Mapper. SQLAlchemy kann als Hibernate[69] der Pythonwelt betrachtet werden. Die folgende Interpreter Session zeigt stark verkürzt die Anwendung eines Objektmappers, wo gegen Ende zu sehen ist, wie der Mapper eine SQL-Abfrage im Hintergrund absetzt.

```
>>> class User(object):
...     def __init__(self, username, password):
...         self.userid = None
...         self.username = username
...         self.password = password
...     def get_name(self):
...         return self.username
...     def __repr__(self):
...         return "User id %s name %s password %s" % (repr(self.userid), \
...                             repr(self.username), repr(self.password))
...
>>> mapper(User, users_table)
>>> sess = create_session()
>>> user = User('hak', 'tom')
>>> sess.save(user)
>>> sess.flush()
INSERT INTO users (username, password) VALUES (:username, :password)
{'password': 'hak', 'username': 'tom'}
>>> user
User id 1 name 'hak' password 'tom'
```

3.3 Softwareentwicklungsprozess erleben

Anhand einer Implementierung kann man Studenten in Form eines Projekts den professionellen Softwareentwicklungsprozess erleben lassen. Was ist dabei zu beachten? Wie kann der Unterricht gestaltet werden? Was sollten die Studenten daraus mitnehmen? Die letzte Frage erläutert [FW78] schon im Jahre 1978. Ein Student sollte nach Abschluss der Lehrveranstaltung folgende Kenntnisse besitzen:

- *"Do Software Development by accurately using at least one state-of-the-art method in each of the four major areas of analysis, design, implementation, and testing;*
- *Test and Measure software by devising and carrying out well-designed experiments that will validate a software system's quality (e.g., reliability, efficiency) and that will identify any parts of the system needing work;*
- *Manage effectively a moderate sized project (3-7 people for 1-2 years)*
- *Communicate effectively with users, managers, and other technical people;*

[68] SQLAlchemy: http://www.sqlalchemy.org/
[69] Hibernate: http://www.hibernate.org/

- *Learn new software engineering methods rapidly, and be able to keep up with relevant advances in computer science.“*

Wenngleich die zitierte Literatur schon veraltet erscheint, kann sie doch auf erstaunlich präzise Art vermitteln, was bei der Gestaltung einer solchen Lehrveranstaltung wichtig ist:

- Methodik und Werkzeuge (z.B. Testen)
- Kommunikation, Projektmanagement
- Motivation bzw. Wecken des Forschergeistes

Das Schlüsselwort, mit dem alle diese Ziele erreicht werden können, heißt *Praxisbezug*. [uMG05] hat untersucht, unter welchen Kriterien Studenten eine Lehrveranstaltung als praxisnahe empfinden (vgl. Sinnkriterium der fundamentalen Idee nach [Sch93]):

Lernhandeln Die Veranstaltung bietet viel Freiraum für den eigenen Beitrag.

Überraschungseffekt Die Lehrveranstaltung bietet ein neues oder im Ausbildungskontext überraschendes Erlebnis für die Teilnehmer.

Authentizität und Nutzen Die jeweiligen Lehrinhalte sind für die Studenten nachvollziehbar praxisrelevant. Authentizität entsteht insbesonders, wenn der Lektor über eigene Praxiserfahrung verfügt und über diese berichten kann. Letzthin kommt man um die Erkenntnis nicht herum, dass nur solche Personen Software Engineering authentisch und praxisnahe unterrichten können, die selbst in der Praxis Software Engineering betrieben haben oder betreiben.

[uMG05] zählt Möglichkeiten auf, Praxisnähe im Unterricht zu vermitteln.

Echte Kunden, echte Anwendung Für Praxisnähe spricht natürlich ein Beispiel aus der Praxis. Studenten profitieren von echten Kunden mit echten Anwendungen. Wobei es kaum möglich ist, im Hochschulbetrieb mit kommerziellen Kunden im Unterricht umzugehen. Es gibt jedoch Erfolge mit Non-Profit-Organisationen. Die Schwierigkeit besteht darin, Jahr für Jahr Kunden vom ihrem gewonnenen Mehrwert zu überzeugen. Ein Argument dafür wäre Networking mit den Studenten. Begabte Studenten können quasi “gescoutet“ werden. Die Hauptaufgabe des Lektors sollte die Gestaltung des Curriculums sein. Dieser wäre aber mit der jährlichen Kooperation mit realen Kunden überfordert. Abhilfe könnte hier eine Unterstützung von Seiten des Unterrichtsministeriums sein, z.B. mit Förderungen für kooperationswillige Firmen bzw. Schaffen von Strukturen für die Abwicklung solcher Lehrveranstaltungen. Momentan ist es eher üblich, den Kunden selbst in einer Art Rollenspiel zu vertreten. Dabei treten aber Glaubwürdigkeits- und Akzeptanzprobleme auf.

Kundengespräche Eng mit dem ersten Punkt dieser Aufzählung verknüpft, sind die Kundengespräche. Dabei können parallel dazu gleich professionelle Gesprächsstrategien gelehrt werden. Die Studenten lernen aus den Informationen des Kunden das Essentielle herauszufiltern. Das Projektteam muss geschlossen, aber mit für den Kunden übersichtlicher Struktur auftreten. Hierbei können auch Gespräche mit unterschiedlichen Kundengruppen trainiert werden, beispielsweise ein Meeting mit der technischen Abteilung oder mit dem Geschäftsführer selbst. Letzteres wird bei tatsächlichen Kunden wohl eher unrealistisch sein.

Professionalität in der Form Das korrekte Erbringen z.B. eines Angebotes, ist nicht nur in inhaltlicher Form, sondern auch als Erscheinungsbild in der Praxis wichtig. Ergebnisse müssen

übersichtlich, und für den Kunden ersichtlich, die relevante Botschaft übermitteln.

Kundenorientiertes Denken Was dem Team wichtig ist, ist meist klar. Doch was ist dem Kunden wichtig? Aus den Gesprächen soll hier eine Abstrahierung stattfinden. Das Projekteam, das wohl eher technisch denken wird, muss sich hier in die Denkweise des Kunden hineinversetzen und diese verstehen. Danach müssen die Studenten diese Erkenntnis mit ihren Zielen oder Vorgaben abstimmen und daraus einen Kompromiss schließen. Einerseit muss der Kunde befriedigt werden, aber der Spaß an der Arbeit darf nicht verloren gehen!

Komplexere Projekte und Reverse Engineering Beim Reverse Engineering wird eine bestehende Software analysiert und rekonstruiert bzw. erweitert oder verbessert.

> *"Programming from scratch: Most courses teach students to code from scratch, rather than by modifying existing programs or by working from model solutions. Moreover, students rarely read good programs. It's as if we asked students to write good prose without first reading good prose [..] Modify and combine programs as well as creating them. Teach students to work with program structures devised by others, to reuse components, to adhere to standards, and to value good documentation."*[Sha00]

Nicht nur das Erstellen einer Software von Beginn an, sondern auch das Weiterverwenden und Verbessern existierender Software sollte einem Studenten vermittelt werden. Gute Software kann analysiert und Beispielen schlechter Software gegenübergestellt werden. Natürlich fehlt in einem Semester die Zeit für ein großes Softwareprojekt. Eine Möglichkeit besteht jedoch darin, das Projekt über eine längere Laufzeit zu planen und das Projektteam (die Studenten) immer wechseln zu lassen. Hier bleibt die Frage offen, wie überschaubar und kontrollierbar dieser Ablauf für den Lektor wäre. [KB01] beschreibt die Erfahrungen bei der Durchführung eines Reverse-Engineering Projekts im Hochschulbetrieb:

> *"[...] bieten gerade Projekte, die auf existierender Software aufbauen eher praxisorientierte Bedingungen. Sie sind ökonomischer, da sie den Aufwand, der in existierender Software steckt (Implementations- und Testaufwand), nachnutzen. Damit ist die Bewältigung komplexer Software auch unter Hochschulbedingungen möglich."*

Und hier kommt Zope wieder ins Spiel. Eine Neuentwicklung ist ebenso möglich, wie Reverse Engineering einer bestehenden Software. Weiters kann Zope selbst analysiert werden, um die internen Funktionalitäten und Abläufe besser zu verstehen. Oder besser gesagt, um zu lernen, wie spezifische Problemstellungen von anderen Softwareentwicklern umgesetzt werden.

Harte Termine Die gestellte Aufgabe muss zu einem festgelegten Termin erfüllt werden. Der Termin sollte so gewählt werden, dass sich mit durchschnittlichem Aufwand der Gesamt- oder Teilprojektumfang nicht realisieren lassen würde. So müssen die Studenten Abstriche (Mehrarbeit, Qualitäts- oder Funktionalitätseinbußen,...) machen, um die Deadline halten zu können. Die daraus entstanden Konsequenzen werden diskutiert und müssen auch vor dem Kunden glaubhaft gerechtfertigt werden können.

Professioneller Werkzeugeeinsatz Arbeitsweise, Werkzeuge und Frameworks, die in der Industrie Verwendung finden, fördern das Verständnis der Praxisrelevanz bei den Studenten. Wobei hier wiederum das Hauptaugenmerk mehr auf die fundamentalen Konzepte zu legen ist,

als auf Modeerscheinungen. Das zu erkennen, erfordert ein hohes Maß an Wissen und Erfahrung in der entsprechenden Materie. Somit sollte keine Spezialisierung auf ein Framework oder Tool stattfinden, sondern dieses als Werkzeug zum Curriculum betrachtet werden, dessen Inhalte es zu lehren gilt. Zope bietet, neben einfacher Installation und Wartung im Universitätsbetrieb, den Zugang zu komplexerer softwaredidaktischer Thematik, ohne sich zu stark auf das Framework zu spezialisieren. Die Infrastruktur kann also einfach eingerichtet und jedem Studenten, auch für den privaten Gebrauch zu Hause, verfügbar gemacht werden.

"Dirty Tricks" [uMG05] schreibt, dass künstlich herbeigeführte Serverabstürze, korrupte Code Repositories usw. helfen, realitätsnahen Praxisbezug herzustellen. Diese Methode erscheint fast gehässig, zumal die Studenten sicher genug mit eigenen Problemen innerhalb des Projekts zu tun haben. Trotzdem ist eine Reflexion der erlebten Komplikationen sowohl technischer als auch sozialer Natur sicherlich lehrreich.

Da der Fokus eines solchen Projekts nicht primär auf der Programmierung liegt, sollten die notwendigen Kenntnisse schon vermittelt worden sein. Wird der Lektor trotzdem mit unterschiedlichen Wissensständen der Studenten konfrontiert, muss diesen der klare Mehraufwand kommuniziert werden. Eventuell müssen sich die Betroffenen eine Einführung in die Programmierung selbst erarbeiten, wenn das notwendig erscheint. Bei einer Sprache wie Python wird das natürlich um ein Vielfaches erleichtert, wie ab Kapitel 2.1 ausführlich diskutiert wurde. Trotzdem sollte beachtet werden, dass Studenten mit Erfahrung in der Programmierung bestimmt mehr aus dem Projekt mitnehmen, da sie sich durch ihr vorhandes Wissen, vollständig auf das Erlernen von Methodik und Werkzeugen konzentrieren können.

Die Aufgaben in der Projektarbeit können natürlich je nach Fähigkeiten der Studenten unterschiedlich verteilt werden, jedoch muß darauf geachtet werden, dass keine "Trittbrettfahrer" mitgeschleppt werden. Ein weiteres Hilfsmittel zur Wissensverteilung bei unterschiedlichem Know-how in Projektgruppen ist das Pair-Programming aus dem Extreme Programming (XP) [Bec00], einem Vorgehensmodell in der Softwarentwicklung. Dabei sitzen immer zwei Studenten gemeinsam vor einem Bildschirm, um eine Aufgabe zu lösen. Einer von beiden kontrolliert die Eingabegeräte, der andere sieht über die Schulter, schlägt Fragen nach oder dokumentiert die momentanen Arbeitsschritte.

> *"Our experiments and experiences with pair-programming in the computer science classroom have been favorable. Ultimately, students are able to complete programming assignments faster and with higher quality. Students communicate with each other more, appear to learn faster, and are happier and less frustrated. The temptation to cheat is greatly reduced. Additionally, the workload of the educators is reduced."* [WU01]

Zahlreiche Studien, wie [MWBF02], [LW00] belegen den positiven Effekt von Pair-Programming im Unterrichtseinsatz.

Der Autor kann die Erkenntnisse aus der Literatur bestätigen, da er selbst an der FH Technikum Wien, im Zuge der Spezialisierungsveranstaltung Informationsmanagement, mit sechs weiteren Studenten in ein Projekt involviert war. Vision des Projekts war ein einfach bedienbares, webbasiertes Projektmanagement-Tool, realisiert mit dem Fokus auf offene Standards.

Das Projekt gliederte sich in zwei Phasen: das Vorprojekt und das Hauptprojekt. Die Projektdauer betrug zwei Semester. Im Zuge des Vorprojekts wurden vor allem eine umfangreiche Marktstudie

durchgeführt, die anwendbare Technik evaluiert, die erwünschten Anforderungen spezifiziert und ein Pflichtenheft erstellt. Dieses stellte die Grundlage für die Aufgaben im Hauptprojekt dar. Auch wurde das Vorprojekt selbst projektmanagementorientiert durchgeführt und dahingehend genutzt, einiges über Abläufe und Probleme in einem konkreten Projekt zu erfahren.

Im Vorprojekt, der Analysephase des Gesamtprojekts, wurde mit einem Unternehmen kooperiert, welches Interesse am eventuell entstehenden Produkt äußerte. Bei einem Meeting in den Räumlichkeiten der Firma wurde die Produktidee präsentiert und Feedback eingeholt. Die Praxisrelevanz war für das Team kein Thema, sie war offensichtlich vorhanden. Die Motivation war dementsprechend in hohem Maße gegeben. In dieser Phase wurde, bis auf einen Mini-Prototypen, nichts entwickelt. Die Arbeit wurde hauptsächlich auf die Vorgehensweise in der Analysephase, die Kommunikation und Abstimmung im Team und das Projektmanagement fokusiert. Die Probleme, die dabei auftraten, waren sehr lehrreich, obwohl alle involvierten Studenten nebenher berufstätig waren und somit dementsprechende Erfahrungen schon gesammelt hatten.

Aus den Erfahrungen des Vorprojekts wurde für das Hauptprojekt die Erstellung eines Prototypen für ein Projektmanagementtool mit zwei integrierten lokalen Diensten und einem integrierten externen Dienst, sowie der Entwurf und die Realisierung einer Schnittstelle nach außen als Projektziel definiert. Die ursprüngliche Idee, eine fertige Software zu entwickeln, wurde aus Komplexitätsgründen, die sich erst im Verlauf der Analysephase ergaben, eingestellt. Selbst die Entwicklung eines Webservice-orientierten Prototypen geriet unter Zeitdruck, und so wurde die Funktionalität der GUI stark minimiert.

Im Hauptprojekt wurde primär entwickelt. Das Team bestand aber nur aus zwei berufstätigen Entwicklern, weshalb die Wissensverteilung nicht homogen war. Durch den Einsatz von Pair-Programming konnte der Wissensstand insofern angeglichen werden, als eine sinnvolle Teameinteilung und Entwicklung möglich wurde.

Mit Ende der zwei Semester und mit Abschluss der Lehrveranstaltung wurde die Entwicklung des Prototypen fertiggestellt. Es wäre durchaus interessant, diesen Prototypen in den nachfolgenden Semestern zu behandeln. Kann der Prototyp weiterentwickelt oder verbessert werden? Welche Fehler wurden bei der Entwicklung gemacht? Eventuell kann die gleiche Funktionalität auf Basis einer anderen Technologie implementiert werden. Möglichkeiten gäbe es viele. Die Frage ist jedoch, welche Studentengruppe motiviert wäre, sich genau mit dieser Thematik auseinanderzusetzen, denn wie schon weiter oben beschrieben, ist die Motivation für den Lerneffekt wichtig.

3.4 Testen und Dokumentieren von Software

Testen und Dokumentieren von Software gewinnt erst bei großen Softwareprojekten an Relevanz. Dementsprechend wird das Testen von Software im Unterricht vernachlässigt, da nur in seltenen Fällen große Projekte realisiert werden, bei denen das Testen unerlässlich wird. Bei kleinen Projekten ist es auch schwer, den doch enormen Aufwand vor den Studenten zu rechtfertigen. Die Fehler scheinen bei normaler Benutzung der Software viel schneller gefunden zu werden.

Selbst in der Industrie scheint es, als ob das Testen von Software nicht durchgängig als wichtiges Element im Softwareentwicklungsprozess gesehen wird. Eine Studie aus dem Jahr 2003 [TM03] besagt, dass 52% der befragten Softwareentwickler ihren Code testen, 34% manchmal und 14% überhaupt nicht. Testen wird als lästiges und langweiliges Anhängsel gesehen. Hier ist ein deut-

licher Nachholbedarf ersichtlich. Nach Erfahrung des Autors ist das nur theoretisch in die Lehre vorgedrungen, die praktische Anwendung kommt jedoch zu kurz.

> *"Developers reuse code but do not test it to the extent we expected [...] simple to use tools are lacking when it comes to test creation and test analysis [...] knowledge on the importance of testing in general and* unit *testing in particular seem low"* [TM03]

Entwickler testen so gut wie keine Reuse-Implementierung. Gerade bei der Zunahme an Reuse und der Komponentenorientierung von Software verlagert sich die Relevanz von der Entwicklung zum Testen. Die Komponenten selbst müssen mit Tests ausgeliefert werden, und die Benutzung dieser erfordert weitere Integrationstests. Weiters werden Tools zur Erstellung und Validierung von Tests als unbefriedigend empfunden. Die Wichtigkeit der Thematik wird generell unterschätzt.

Im Folgenden werden die unterschiedlichen Testmöglichkeiten mit Zope besprochen. Eine tatsächliche Umsetzung davon wird in der Beispielimplementierung, Kapitel 3.5 präsentiert. Weiters wird auf das *doctesting*, einer Testmethode mit gleichzeitiger Dokumentation eingegangen.

Unit Tests

Ein *Unit test* ist ein White-Box Test, wobei nur eine spezielle Komponente für sich allein getestet wird. Der Test wird von der Umwelt (wie andere Komponenten) nicht beeinflusst. Er testet die kleinste überhaupt testbare Einheit (Unit) einer Software. Ziel ist die Funktionalität jeder einzelnen Komponente zu garantieren. Nach dem Test Driven Development (TDD) [GW03] werden *Unit tests* vor der Programmierung einer Komponente entwickelt, die Funktionalität der Komponente wird dann für den Test entwickelt (siehe Abbildung 15). Dieses Vorgehen zwingt den Entwickler, sich im Vorfeld intensiver mit der gewünschten Funktionalität einer Komponente auseinanderzusetzen. Dabei wird im Vorfeld das Application Programming Interface (API) durch die Testfälle definiert und danach implementiert.

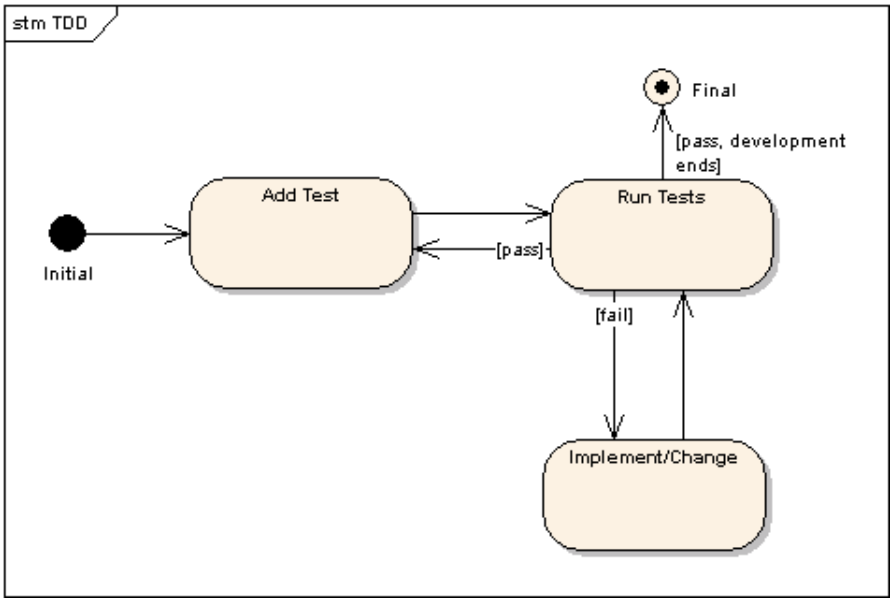

Abbildung 15: Test Driven Development (TDD)

Python wird mit dem `unittest` Modul[70] ausgeliefert und ist die Pythonvariante von *JUnit*[71], bekannt aus der Java-Welt. Das Modul kann somit unter Zope verwendet werden.

Nachfolgend ist ein einfacher Testcase mit dem `unittest` Modul unter Python dargestellt:

```
>>> import unittest
>>> class MyTestCase(unittest.TestCase):
...     def test_TwoMinusThree(self):
...         self.assertEqual(2 - 3, -1)
...         self.failIfEqual(2 - 3, 1)
...
>>> unittest.main()
```

Integration Tests und Functional Tests

Integration tests testen die Interaktion zwischen Komponenten, für die schon erfolgreich laufende *Unit tests* geschrieben wurden. Dabei wird zum ersten Mal das Zusammenspiel der Komponenten in einem gemeinsamen Kontext getestet. Die nächste Steigerung ist der Functional Test, bei dem das System als Black-Box betrachtet wird. Die Tests laufen aus der Sicht eines Benutzers.

> *"[...] in the Zope world, functional tests are really integration tests. Functional tests would treat Zope like a black box. "Functional tests" in Zope, however, do not simulate a browser client program and connect through the Hypertext Transfer Protocol (HTTP) port. They only simulate the necessary objects, such as the browser request. That means, they are more like integration tests."*[vW06]

Unter Zope gibt es keine Trennung der beiden Testbegriffe. Es wird unkorrekterweise einheitlich von *Functional tests* gesprochen.

Doctests

Doctesting ist eine eigene Art Pythons, Dokumentation mit Tests zu verschmelzen. Üblicherweise sind Tests selbst schon eine gute Dokumentation, wenn man die Funktionalität einer Komponente verstehen will. Meist sind sie aber nicht gerade komfortabel zu lesen.

> *"Doctest is a system for writing tests within Python documentation strings. The emphasis is on documentation. Tests are provided as example code, set off with Python prompts. Doctest tests lend themselves toward a literate form of test code."*[FP04]

Ein *Doctest* wird entweder in den *Docstring*[72] im Quelltext geschrieben, oder als eigenes Textfile abgelegt. Letzteres empfiehlt sich bei umfangreicher Testlänge. Aus den *Docstrings* können seperate Dokumentationen, z.B. als HTML oder Portable Document Format (PDF) extrahiert werden. *Doctests* werden im Stil einer Interpreter Session geschrieben. Erklärende Worte dazu sind im *reStructuredText* Format[73] zu wählen. Listing 18 zeigt ein kurzes Beispiel zur Veranschaulichung.

[70] PyUnit: http://pyunit.sourceforge.net/
[71] JUnit: http://junit.org/
[72] Docstring: http://en.wikipedia.org/wiki/Docstring
[73] reStructuredText: http://docutils.sourceforge.net/rst.html

```
...
class SessionCredentials(object):
    """Credentials class for use with sessions.

    A session credential is created with a login and a password:

      >>> cred = SessionCredentials('scott', 'tiger')

    Logins are read using getLogin:

      >>> cred.getLogin()
      'scott'

    and passwords with getPassword:

      >>> cred.getPassword()
      'tiger'

    """
    def __init__(self, login, password):
        self.login = login
        self.password = password

    def getLogin(self): return self.login
    def getPassword(self): return self.password
...
```

Listing 18: Doctest from the Zope 3 SessionCredentials Plugin

Doctests können *Unit tests* und *Integration tests* in einem besser les- und schreibbaren Format abbilden. Der Entwickler kann anhand einer "Erzählung" seinen Quelltext dokumentieren und zugleich testen. Die Vorteile liegen auf der Hand. Die Handhabung schlägt zwei Fliegen mit einer Klappe und ist leichter zu erlernen als z.B. das *Unit test* API. Diese Art von Dokumentationstest erscheint dem Autor ein gutes Werkzeug, um die Thematik Testen und Dokumentieren im Unterricht darzulegen.

3.5 Beispielimplementierung

Die Beispielimplementierung behandelt einen Auszug aus einem konkreten Projekt des Autors. Das Projekt Stadtgespräche[74] ist ein digitales Museum für fotographische Fundstücke von Schriften, Logos, Zeichen, Pictogrammen, Buchstaben, Wappen und anderen typografischen Werken. Der Museumsdirektor Markus Hanzer beschreibt Stadtgespräche (Abbildung 16) wie folgt:

> *"Stadtgespräche versteht sich als ein offenes Forum der Diskussion und Auseinandersetzung über visuelle Alltagskommunikation und versucht durch Gegenüberstellungen und analytische Beschreibungen einen Blick auf mögliche Hintergründe und Motive für konkrete Gestaltungsformen zu werfen."*

Ursprünglich wurde das Projekt im Jahre 2000 auf Basis von Zope 2 unter dem Namen Typemuseum entwickelt. Aus rechtlichen Gründen musste das Projekt neu entwickelt werden. Zu diesem Anlass gibt es einen Technologierelaunch auf Basis von Zope 3.

Die folgenden Seiten zeigen nur Ausschnitte aus den Quelltexten, in denen teilweise aufbauend erläutert wird. Daher ist es notwendig, den tatsächlichen Endstatus der Quelltexte im Anhang dieser Arbeit abzulegen.

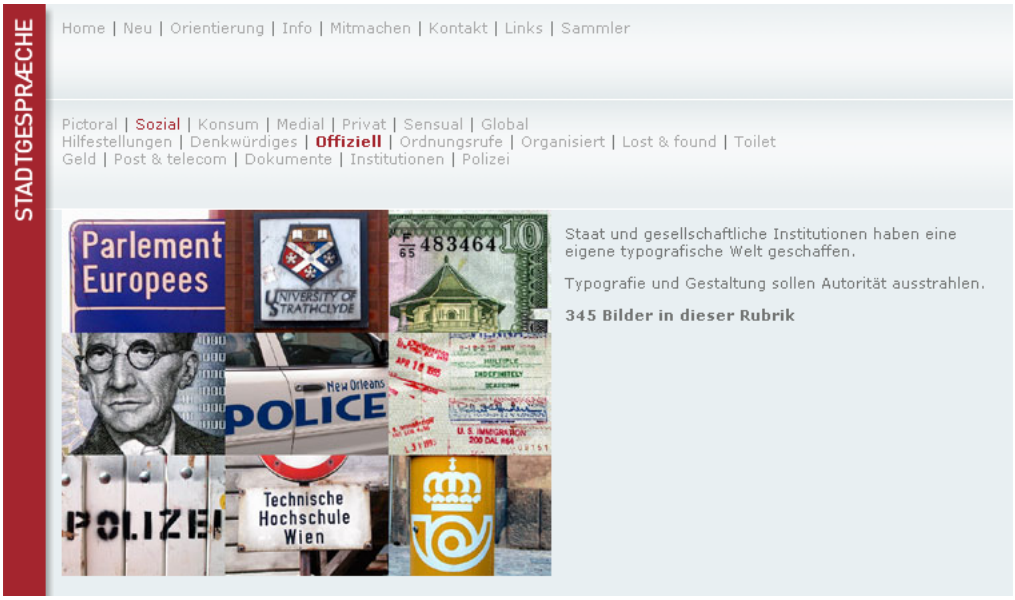

Abbildung 16: Frontend Stadtgespäche

Das Prinzip ist einfach. Als Geschäftsobjekt kann ein Eintrag im Museum definiert werden. Ein Eintrag ist eine Grafik mit zugehörigen Metadaten, die in einer hierarchischen Struktur abgebildet wird. Aus dieser Struktur wird die Navigation für das Frontend generiert.

[74] Stadtgespräche: http://www.stadtgespraeche.com

Interface

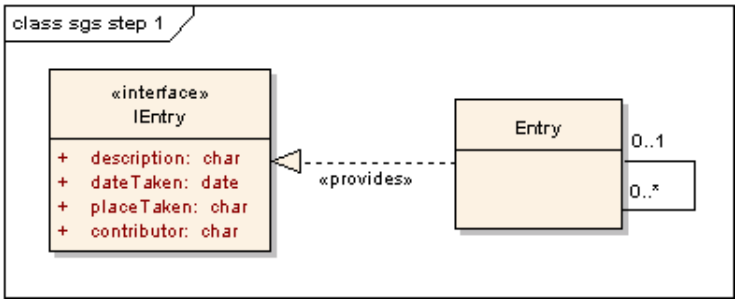

Abbildung 17: Interface

Im ersten Schritt wird das Interface definiert. Interfaces sind kein Standardkonzept von Python und wurden von Zope zusätzlich implementiert. Aus diesem Grund sind Interfaces Klassen. Das hat wiederum den Vorteil, dass Interfaces vererbt werden können.

Interfaces unter Zope 3 dienen nicht nur der Dokumentation, sondern können auch adaptiert werden. Jedes Objekt kann jedes Interface zur Verfügung stellen oder adaptieren.

Nach Erstellen eines Python Moduls mittels leerer `__init__.py` im Python Pfad (normalerweise in der Zope Instanz unter `lib/python`), wird das Interface erzeugt. Nach Konvention sind die Interfaces eines Moduls immer in der Datei `interfaces.py` zu finden.

```
1  from zope.app.file.interfaces import IImage
2  from zope.app.container.interfaces import IContainer
3  from zope.app.container.constraints import contains
4  from zope.schema import Text, TextLine, Date
5
6  class IEntry(IImage):
7      """ Interface for Entry  """
8
9      description = Text(title=u"Image Description", required=False)
10
11     dateTaken = Date(title=u"Date when picture was taken", required=True)
12     placeTaken = TextLine(title=u"Place where picture was taken",
13                           required=True)
14
15     contributor = TextLine(title=u"email of contributor", required=True)
16
17
18 class IEntryContainer(IContainer):
19     """ Container Interface for an Entry """
20     contains(IEntry)
```

Listing 19: IEntry interface (interfaces.py)

- Zeile 6: `IEntry` erbt das `IImage` Interface aus `zope.app.file.interfaces`. Damit ist es möglich, die Funktionalität des Zope Basis Objektes `Image` zur Verfügung zu stellen. Dieses speichert unter anderem die Bilddaten als Attribut auf dem Objekt.
- Zeile 7: Python Docstring für die Dokumentation des Interfaces.

- Zeile 9-15: Das Schema für die Attribute wird festgelegt. Ein Schema bringt eine statische Typisierung in die dynamisch typisierte Sprache Python. Hauptanwendung ist dabei das Generieren und Validieren von Userinput. Zuständig dafür ist beispielsweise `zope.formlib`. Später wird der Zope Server aus diesen Informationen passende Formulare generieren.
- Zeile 18-20: Hier ist eine andere Verwendung für Interfaces zu sehen. Objekte, die IEntryContainer zur Verfügung stellen, erlauben, andere Objekte zu beinhalten, die wiederum das Interface IEntry bereitstellen. Man spricht auch von einem *Marker-Interface*.

Content Objekt

Im nächsten Schritt legen wir ein Content Objekt an, das obige Interfaces implementieren wird.

```
1  from zope.interface import implements
2  from zope.app.container.btree import BTreeContainer
3  from zope.app.file.image import Image
4
5  from interfaces import *
6
7  class Entry(BTreeContainer, Image):
8      implements(IEntry, IEntryContainer)
9
10     __name__ = __parent__ = None
11
12     description = u""
13
14     dateTaken = None
15     placeTaken = u""
16
17     contributor = u""
```

Listing 20: persistentes Content Objekt (entry.py)

- Zeile 7: Unsere Klasse erbt die Funktionalität von `Image` und `BTreeContainer` und damit implizit auch eine automatische Persistierung in der ZODB. `BTreeContainer` ermöglicht, weitere Objekte *in* einer `Entry` Instanz zu speichern. Die Klasse fungiert somit als Container. Die enthaltenen Objekte werden aus Performancegründen als Binary Tree abgelegt.
- Zeile 8: Das Objekt implementiert hiermit die beiden von uns erstellten Interfaces. Somit ist es möglich, das Content Objekt ineinander zu verschachteln.
- Zeile 10: Diese beiden Attribute werden benötigt, um das Objekt in der Hierarchie auffindbar zu machen (`zope.location`).
- Zeile 12-17: Um die Anforderungen der Interfaces zu erfüllen, werden die im Interface definierten Attribute leer initialisiert.

Konfiguration

Per ZCML wird dem Zope Server nun mitgeteilt, dass die neue Komponente bereit ist, in Betrieb genommen zu werden.

```
1  <configure
2          xmlns="http://namespaces.zope.org/zope"
3          i18n_domain="zope"
4          >
5
6  <class class=".entry.Entry" >
7      <require
8          permission="zope.View"
9          interface=".interfaces.IEntry" />
10     <require
11         permission="zope.View"
12         interface=".interfaces.IEntryContainer" />
13     <require
14         permission="zope.ManageContent"
15         set_schema=".interfaces.IEntry" />
16 </class>
17
18 </configure>
```

Listing 21: ZCML Konfiguration (configure.zcml)

- Zeile 6-16: Der `class` Tag registriert und konfiguriert das Objekt innerhalb von Zope. Standardmäßig ist der Zugriff auf alle Elemente einer Klasse von der Zope Security Maschinerie unterbunden. In diesen Zeilen wird der Lesezugriff auf alle durch die Interfaces `IEntry` und `IEntryContainer` definierten Attribute für die Zope Permission `zope.View` freigegeben. Der Schreibzugriff wird für das Schema unter `IEntry` zugelassen.

Um das Objekt per Zope Management Interface (ZMI) verwalten zu können, erstellen wir eine eigene Konfiguration im Untermodul `browser`. Das signalisiert die Zugehörigkeit zu einem View der Hypertext Transfer Protocol (HTTP) Klasse. Dafür wird das Unterverzeichnis erstellt und als Python Modul gekennzeichnet. In die obige Datei `configure.zcml` wird das neu zu berücksichtigende Modul wie folgt eingefügt.

```
<include package=".browser" />
```

Das `browser` Modul wird in Zukunft alle Views betreffend Webbrowser enthalten.

```
1  <configure xmlns="http://namespaces.zope.org/browser">
2
3  <addMenuItem
4        class="..entry.Entry"
5        title="SGS Entry"
6        permission="zope.ManageContent"
7        view="AddSGSEntry.html"
8        />
9
10 <addform
11       label="Add SGS Entry"
12       name="AddSGSEntry.html"
13       schema="..interfaces.IEntry"
```

```
14        content_factory="..entry.Entry"
15        class="zope.app.file.browser.image.ImageAdd"
16        permission="zope.ManageContent"
17        />
18
19  <containerViews
20        for="..interfaces.IEntryContainer"
21        contents="zope.ManageContent"
22        add="zope.ManageContent"
23        />
24
25  </configure>
```

Listing 22: ZCML Browser Konfiguration (browser/configure.zcml)

Diese Konfiguration ermöglicht dem Administrator nun via ZMI neue Instanzen des `entry` Content Objektes zu erstellen (siehe Abbildung 18). Die dafür benötigten Webformulare, inklusive Validierung, wurden durch die vorherigen Interfacespezifikationen erstellt.

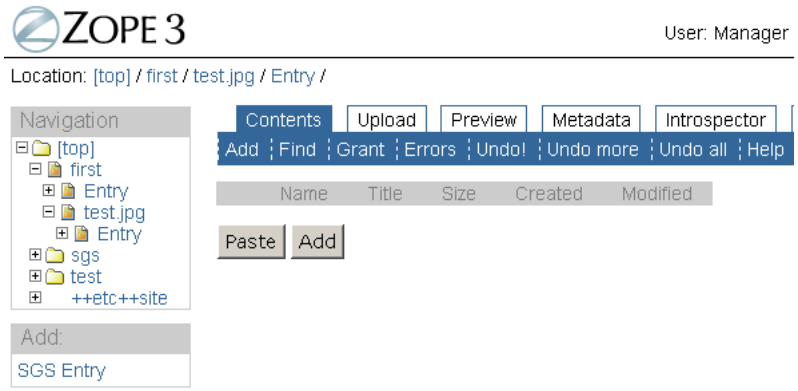

Abbildung 18: Zope Management Interface (ZMI)

Adapter

Die Persistierung und Konfiguration eines Eintrages ist erledigt. Nun wird dieser um die Funktionalität einer Navigation erweitert. Dabei wird darauf geachtet, dass browser-spezifische Funktionalität abgekapselt, und weiters die bisherige Implementierung nicht geändert wird. Dabei kommt ein Werkzeug des Zope Komponenten Frameworks zum Einsatz: der Adapter.

Mittels Adapter kann die bestehende Funktionaliät erweitert werden, ohne die Implementierung des zu adaptierenden Objektes zu ändern.

Die Datei `interfaces.py` wird um das Interface des Adapters erweitert.

```
22  from zope.interface import Interface
23
24  class IEntryNav(Interface):
25      """used as Adapter for menu creation
26      """
27
```

```
28      def setEntryMenuName(menuname):
29          """ set the navigation menu name  """
30
31      def getEntryMenuName():
32          """ get the navigation menu name """
33
34      def getEntryMenuOrder():
35          """ get the navigation menu order """
36
37      def setEntryMenuOrder(order):
38          """ set the navigation menu order """
39
40      def getChildren():
41          """ get all IEntry children from this location """
42
43      def getSiblings():
44          """ get all IEntry Siblings from this location
45              (self is included)
46          """
```

Listing 23: Adapter Interface für Navigationsfunktionalität (interfaces.py)

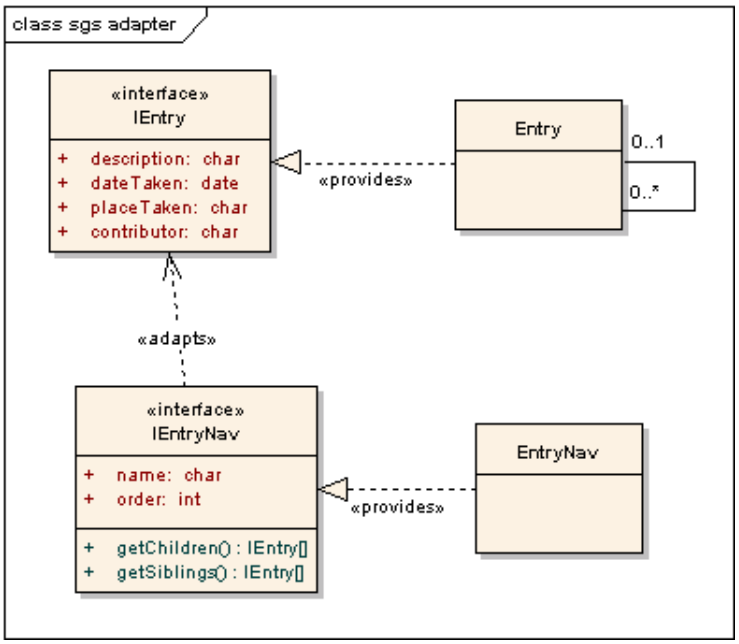

Abbildung 19: Adapter

`Entry` soll um ein Attribut erweitert werden, das den Namen des Navigationspunktes repräsentiert. Weiters soll jeder Menüpunkt eine Gewichtung zwecks Sortierung der Navigation erhalten. Der Name des Navigationspunktes wird als *Annotation* auf dem ursprünglichen Objekt gespeichert. Dazu muss `Entry` ein Markerinterface zur Verfügung stellen, welches signalisiert, dass auf diesem Objekt Metadaten (Annotations) gespeichert werden dürfen. Das kann entweder via ZCML oder wie folgt (Listing 24) geschehen.

```
8 implements(IEntry, IEntryContainer, IAttributeAnnotatable)
```

Listing 24: Hinzufügen des Marker Interfaces für Annotations (entry.py)

```
19 from zope.component import adapts
20 from zope.annotation.interfaces import IAttributeAnnotatable
21 from zope.annotation import IAnnotations
22
23 EntryNavAnnotations_KEY = "sgs.entrynav"
24
25 class EntryNav(object):
26     implements(IEntryNav)
27     adapts(IEntry)
28
29     def __init__(self, context):
30         self.context = self.__parent__  = context
31         annotations = IAnnotations(context)
32         mapping = annotations.get(EntryNavAnnotations_KEY)
33         if mapping is None:
34              mapping = annotations[EntryNavAnnotations_KEY] = {'name': '',
35                                                                'order': 0}
36         self.mapping = mapping
37
38     def setEntryMenuName(self, menuname):
39         self.mapping['name'] = menuname
40
41     def getEntryMenuName(self):
42         return self.mapping['name']
43
44     def setEntryMenuOrder(self, order):
45         self.mapping['order'] = order
46
47     def getEntryMenuOrder(self):
48         return self.mapping['order']
49
50     def getChildren(self):
51         return self.context.values()
52
53     def getSiblings(self):
54         siblings = []
55         for sibling in self.context.__parent__.values():
56             if IEntry.providedBy(sibling):
57                 siblings.append(sibling)
58         return siblings
```

Listing 25: Adapter für Navigationsfunktionalität (entry.py)

- Zeile 26: Diese Klasse stellt `IEntryNav` als Interface zur Verfügung,
- Zeile 27: und fungiert als Adapter für IEntry (siehe Abbildung 19).
- Zeile 30-36: Um die beiden neuen Attribute zu persistieren, wird der IAnnotations Adapter auf dem `Entry` Objekt angewendet.
- Zeile 38-48: getter und setter der neuen Attribute.
- Zeile 50-58: Implementierung der Methoden, die die spätere Navigationsgenerierung unterstützen.

Um den Adapter zu registrieren, muss der Konfigurationsteil im `entry` Modul erweitert werden (Listing 26).

```
...
<adapter factory=".entry.EntryNav"
         trusted="true"  />

<class class=".entry.EntryNav" >
    <require
        permission="zope.View"
        interface=".interfaces.IEntryNav" />
</class>
...
```

Listing 26: ZCML Konfiguration für den Adapter der Navigationsfunktionalität (configure.zcml)

Ein Test für die Adapterimplementierung ist überfällig. Listing 30 aus dem Anhang erläutert diesen.

Als Nächstes wird ein Browser-View geschrieben, der die Navigation nach Abbildung 16 generiert. Der View kann in Folge von einem ZPT verwendet werden, um diesen für den Benutzer graphisch aufzubereiten.

```
1  from operator import itemgetter
2
3  from zope.traversing.api import getParents
4  from zope.traversing.interfaces import IPhysicallyLocatable
5  from zope.publisher.browser import BrowserView
6  from zope.traversing.browser import absoluteURL
7
8  from sgs.entry.interfaces import IEntryNav
9
10 class EntryNavigation(BrowserView):
11
12     def __call__(self):
13         root = IPhysicallyLocatable(self.context).getNearestSite()
14         parents = getParents(self.context)
15
16         # remove all parents below the first found site
17         while parents[-1] != root:
18             parents.pop()
19
20         parents.pop() #remove Site from parents
21         parents.reverse()
22
23         parentSiblings = [ sorted( [
24             {'name':IEntryNav(sibling).getEntryMenuName(),
25              'order':IEntryNav(sibling).getEntryMenuOrder(),
26              'url':absoluteURL(sibling, self.request),
27              'active':((sibling in parents) and 'parent') or False}
28             for sibling in IEntryNav(item).getSiblings() ] ,
29                                 key=itemgetter('order')) for item in parents ]
30
31         mySiblings = sorted( [
32             {'name':IEntryNav(sibling).getEntryMenuName(),
33              'order':IEntryNav(sibling).getEntryMenuOrder(),
34              'url':absoluteURL(sibling, self.request),
35              'active':((sibling == self.context) and 'self') or False} \
36             for sibling in IEntryNav(self.context).getSiblings() ] ,
37                                 key=itemgetter('order'))
```

```
38
39
40          children = sorted( [
41              {'name':IEntryNav(child).getEntryMenuName(),
42               'order':IEntryNav(child).getEntryMenuOrder(),
43               'url':absoluteURL(child, self.request) }
44              for child in IEntryNav(self.context).getChildren() ],
45                             key=itemgetter('order'))
46
47          parentSiblings.append(mySiblings)
48
49          if len(children):
50              parentSiblings.append(children)
51
52          return parentSiblings
```

Listing 27: Generierung der Navigation (browser/__init__.py)

- Zeile 13-21: Das soeben aktive Objekt sucht seine Elternobjekte in der Hierarchie und ordnet diese in eine Liste ein.
- Zeile 23-29: Für jeden Eintrag in der soeben generierten Liste steht ein aktiver Menüpunkt in der Navigation. Für jeden dieser Menüpunkt werden die Geschwistereinträge auf gleicher Ebene gesucht. Die Information wird als Liste von Dictionaries generiert. Gleichzeitig wird die Navigation korrekt sortiert.
- Zeile 31-45: Der gleiche Vorgang wie zuvor findet für die Einträge auf gleicher Ebene und die Kinder des aktiven Objektes statt.

Im Anhang ist der komplette Quelltext der Beispielimplementierung zu finden, sowie weitere Tests, die die Generierung der Navigation und eine Katatalogindizierung testen.

Weitere Schritte in der Implementierung betreffen sowohl die Präsentationslogik, als auch das Layout mit HTML und Cascading Style Sheets (CSS) sowie ein Redakteursinterface. Das wird in dieser Arbeit nicht behandelt. Mögliche weitere denkbare Funktionalitäten wären eine Tagging Engine, mit der es möglich ist, Einträge zu verschlagworten und somit eine flache Kategorisierung zu erreichen (im Gegensatz zur Hierarchie der momentanen Implementierung). Ein fertige Komponente hierfür scheint `lovely.tag`[75] zu sein.

3.6 Die Zukunft von Zope

Zope ist und war ein innovativer Vorreiter bei Webtechnologien. In der Implementierung von Webapplikationen liegt die große Stärke des Frameworks. Gleichzeitig ist durch die einfache Handhabung ein leichter Zugang in die Materie möglich. Die Barrieren bei Lösungen aus der Java-Welt sind durchaus höher. Somit kann eine Verwendung in der Lehre erwogen werden.[Faa07]

Ein aktuelles Projekt der Zope Community ist *Grok*[76]. Grok ist ein Ansatz, den Zugang zur Zope Komponentenarchitektur weiter zu vereinfachen. Momentan ist das Projekt in Entwicklung und sollte weiter beobachtet werden.

[75] lovely.tag: http://svn.zope.org/lovely.tag/
[76] Grok: http://grok.zope.org/

> *"I hope with Grok we will have drastically increased the approachability of Zope 3. This will hopefully make it the web framework of choice for more web developers (Python), even given the intense competition. I also hope that Zope 3 will continue to adopt technologies from outside the Zope realm, sharing code and concepts with other Python-based web frameworks. At the same time, Zope 3's components will become more separate from a monolithic application server and will start to be reused more within other web frameworks [...] Zope 2 applications will continue to evolve using Five towards using Zope 3 technology. The platforms Zope 2 and Zope 3 will continue to merge."*[Faa07]

Zope, vor allem Zope 3, ist jedoch nur einem kleinen Teil der Entscheidungsträger bekannt. Das sollte ein Hauptaugenmerk der Zope Community in naher Zukunft sein. Beispielsweise ist die Zope Website schwer vernachlässigt, was bestimmt ein Grund ist, warum so mancher Interessierte scheitert. Das Framework und dessen Möglichkeiten, vor allem auch Erfolgsgeschichten sollten besser an die Öffentlichkeit kommuniziert werden. Einen Beitrag dazu soll diese Arbeit leisten.

3.7 Alternative Python Frameworks

Es existieren diverse alternative Python Frameworks, die oft in einem Zug mit Zope 3 erwähnt werden. Dabei muss beachtet werden, dass diese Frameworks eher auf spezielle Bedürfnisse im Web zugeschnitten sind und mehr Frontend- als Backendentwicklung unterstützen. Es scheint, dass die meisten dieser Frameworks auf den Web2.0-Hype aufspringen und sich dabei auf die Erstellung von Wiki, Weblog, Forum, *Really Simple Syndication (RSS)* und ein *Asynchronous JavaScript and XML (AJAX)* gesteuertes Userinterface spezialisieren. Die folgenden Frameworks basieren alle auf Python und werden mit Objektmapper-Funktionalität (siehe Kapitel 3.2) zur Verfügung gestellt.

- Turbogears[77]
- Django[78]
- Pylons[79]

Allerding ist Zope 3 nach Ansicht des Autors das geeignete Tool, um anhand interessanter und einfacher Applikationsentwicklung, notwendige Thematik verständlich in den Unterricht zu integrieren. Diese Themen sind Testen von Software, kollaboratives Arbeiten, wissenschaftliches Arbeiten (Nachvollziehbarkeit ist auch in der Softwareentwicklung notwendig!), Komponentenverständnis und Wiederverwendbarkeit, die ihren Einzug in die Lehre vor allem in praktischer Hinsicht noch nicht gefunden haben! Obig angeführte Frameworks scheinen jedoch für einfache Webapplikationsprojekte einen einfacheren Zugang zu bieten.

[77] Turbogears: http://www.turbogears.org/
[78] Django: http://www.djangoproject.com/
[79] Pylons: http://pylonshq.com/

4 Diskussion

4.1 Zusammenfassung und Bewertung

In der vorliegenden Arbeit wurde die Eignung Pythons als Werkzeug für den Einsatz im Unterricht analysiert. Beweggründe für das Schreiben der Arbeit waren die positiven Erfahrungen des Autors mit Python in seinem Arbeitsumfeld und Beobachtungen während seines Studiums an der FH Technikum Wien, wo mit C und Java unterrichtet wird. Der Autor wollte untersuchen, ob und wie sich Python für den Einstieg in die Programmierung als auch für fortgeschrittene Softwareentwicklung, eignet. Beim Thema Softwareentwicklung lag das Augenmerk auf dem Applikationsserver Zope[80].

In der Einleitung wurde mit der Vorstellung der fundamentalen Idee einer Wissenschaft, die, nach Ansicht des Autors, gerade in der informatischen Bildung, im Zusammenhang mit Programmiersprachen, nicht berücksichtigt wird, begonnen. Durch die Komplexität der Programmiersprachen verliert sich der Unterricht in Details einer Sprache oder eines Tools. Die zu vermittelnden fundamentalen Ideen kommen dabei zu kurz. Danach wurde ein kurzer Überblick über die Probleme beim Erlernen einer Programmiersprache gegeben. Python und Zope wurden im Anschluss vorgestellt. Dabei ist der Autor auf die Geschichte und die Unterschiede zwischen den Versionen (vor allem zwischen Zope 2 und Zope 3) eingegangen.

Das erste Kapitel hat sich mit Python beschäftigt. Vor allem der Einsatz von Python als erste Programmiersprache wurde anhand eines Vergleichs mit C++ und Java illustriert. Dabei wurde anhand des banalen *Hello World* Beispiels[81] erläutert, wo die didaktischen Schwachstellen liegen, wenn eine Systemsprache als erste Programmiersprache gewählt wird. Hierbei wurde davon ausgegangen, dass Studenten wenig bis keine Programmiererfahrung haben. Im weiteren Verlauf des Kapitels setzte sich die Arbeit mit Programmierparadigmen und Algorithmen auseinander, um herauszustreichen, welche Vorteile Python hier für sich verbuchen kann. Auch der Aspekt von Open-Source und deren Community wurde in Hinblick auf deren möglichen positiven Einfluss auf den Unterricht beleuchtet. Das Kapitel wurde mit der Vorstellung diverser Materialien und Werkzeuge, die für die Gestaltung des Unterrichts mit Python herangezogen werden können, beendet.

Dabei wird offenbar, dass der Einsatz von Systemsprachen für das Lehren von Programmieraspekten nicht besonders geeignet ist. Um ein einfaches Programm zu schreiben, sind zu komplexe und fortgeschrittene Zusammenhänge zu durchschauen, die den Studenten schwer im Vorfeld beizubringen sind. Das Ergebnis sind frustrierte und desinteressierte, potentielle Softwareentwickler. Ziel des Informatikunterrichts sollte es sein, zu begeistern. Dafür ist es notwendig, den Studenten schnelle Erfolge zu ermöglichen. Das Kapitel zeigt, dass dieses Ziel mit Python, aufgrund des Aufbaus der Sprache an sich, einfacher zu erreichen ist. Auch die Seite des Unterrichtenden wurde analysiert. Dabei ist festgestellt worden, dass für die Gestaltung eines Curriculums mit Python genügend bestehendes Material vorhanden ist, an dem sich orientiert werden kann. Die Wahl einer Skriptsprache wie Python erleichtert Studenten wie Lektoren die Arbeit und ermöglicht, den Unterricht auf die fundamentalen Ideen der Informatik zu konzentrieren.

[80] Im Folgenden wird immer von Zope ab der Version 3 gesprochen. Ist die Rede von einer früheren Version, wird explizit die Versionsnummer angeführt.

[81] Das traditionelle erste Programm eines jeden angehenden Softwareentwicklers.

Es wird festgestellt, dass die Entscheidung, Sprachen wie C, C++ oder Java im Unterricht zu verwenden, stark industriegetrieben ist, was durch das Fehlen einer allgemein anerkannten Lehrsprache, wie es Pascal früher einmal war, noch verstärkt wird. Studenten neigen auch dazu, sich eher für Programmiersprachen zu interessieren, die sie in ihrem späteren Berufsleben verwenden können. Diese beiden Motivationen ergeben die momentane Situation in der Lehre.

Für die Zukunft ist es wünschenswert, dass erkannt wird, dass nicht das Wissen um eine spezielle Programmiersprache wichtig ist, sondern das Erlernen der grundlegenden Konzepte. Sind diese erst einmal verstanden, können sie mit jeder Programmiersprache umgesetzt werden. Dafür ist Python aus der Sicht des Autors bestens geeignet. Als Unterstreichung dieser Aussage soll das folgende Zitat von Eric Steven Raymond[82] dienen.

> *"Python is, of all the languages I have learned, the one that does the least to get between me and the problem. It´s the most effective to translate pure thoughts into action."*

Doch ist Python auch außerhalb der Lehre sinnvoll anzuwenden? Der aufmerksame Leser hat festgestellt, dass verstandene Konzepte aus der Lehre auf alle Anforderungen außerhalb der Lehre angewendet werden können. Diese sind unabhängig von Programmiersprachen. Aber Menschen in eingefahrenen Systemen gehören wachgerüttelt. Diese Arbeit verweist auf den erfolgreichen Einsatz von Python in der Softwareindustrie.

Im zweiten Hauptkapitel wurde das Lehren von fortgeschrittenen Themen der Softwareentwicklung mit Hilfe des Applikationsservers Zope thematisiert. Zope wurde in der Einleitung als Komponentenframework vorgestellt. Damit hat sich dieses Kapitel zu Beginn auseinandergesetzt. Danach wurde versucht, Zope mit dem CBSE in Verbindung zu bringen und einzuordnen. Anschließend wurde die ZODB und deren möglicher Einsatz in der Lehre vorgestellt. Das Entwickeln einer Software ist ein komplexer Prozess und schwierig im Unterricht zu vermittelt. Damit hat sich ein Teil dieses Kapitels beschäftigt, ebenso wie mit dem stark vernachlässigten Testen und Dokumentieren von Software.

Die Arbeit stellt fest, dass auch fortgeschrittene Themen in der Softwareentwicklung mit Python behandelt werden können. Zope wurde mit Python entwickelt und steht selbst als Beispiel für den erfolgreichen Einsatz Python´s in der Industrie. Zope hat seinen Fokus auf Webapplikationen, ist dabei aber nicht auf diesen Einsatz beschränkt. Der Autor meint, dass der Einsatz von Zope einen einfachen Zugang für die Entwicklung von komplexeren Applikationen im universitären Einsatz ermöglicht. Dabei kann der Fokus auf Reuse, komponentenorientieres Denken und Testen bzw. Dokumentieren von Software gelegt werden.

Berichte vom Einsatz von Zope *in* der Lehre gibt es keine. Zope wird eher *für* die Lehre verwendet. Als Beispiel sei die Medizinische Universität Wien (MUW) genannt, wo momentan die zentrale Prüfungsverwaltung mit Zope und Python entwickelt wird. Aufgrund der fehlenden Erfahrung im Umgang mit Zope innerhalb eines Curriculums, sollten Versuche gestartet werden, um festzustellen, ob sich Zope dabei bewähren kann. Die Zeit ist reif für einen Versuch, wenngleich der momentane komplexe Einstieg in die Komponentenarchitektur von Zope eventuell in einer Schulung zum Framwork selbst enden könnte. Die Hürde, das Komponentenframework zu verstehen und anzuwenden, ist momentan noch etwas zu hoch. Daran wird aber gearbeitet. Die Zope Community bemüht sich zur Zeit um ein Projekt namens *Grok* (siehe Kapitel 3.6), das die hohe

[82] http://en.wikipedia.org/wiki/Eric_Steven_Raymond

Anfangsbarriere senken möchte.

4.2 Kritik

Nachfolgende Aufstellung stellt eine zusammenfassende Liste der Argumentation dar, wobei die jeweiligen Alleinstellungsmerkmale fett hervorgehoben sind. Diese werden den negativen Kritikpunkten gegenübergestellt. Dabei ist zu beachten, dass diese Aufstellung nicht alle Vor- und Nachteile von Python und Zope, sondern bewusst die der vorgestellten Lösung, Python und Zope als Unterrichtswerkzeuge, zeigt.

Python:

+ **Multiparadigmensprache**
+ **erzwungene Einrückung des Quelltextes**
+ einfacher Zugang zur Programmierung möglich
+ einfache Installation
+ einfache Entwicklungsumgebung
+ Python kann als Werkzeug im Unterricht verwendet werden und wird nicht Gegenstand des Unterrichts selbst
+ der Untericht kann sich auf die fundamentalen Ideen des zu vermittelnden Wissens konzentrieren
+ schnelle Lehr- und Lernerfolge
+ Erleichterung bei der Gestaltung von Curricula
+ auch für junge Zielgruppe geeignet (Unterstufe)
+ gute Vorbereitung auf komplexere Sprachen
+ große Standardbibliothek
+ dynamische Typisierung
+ Open-Source Produkt
+ interpretierte Sprache
− langsamer als Systemsprachen
− kein Compiler
− keine Pointer, Typisierung und Speicherverwaltung
− keine Systemsprache
− Geheimnisprinzip bei der OOP nicht explizit in der Sprache integriert

Zope:

+ **objektorientierte Peristenzlösung - ZODB**
+ Komponentenframework auf Open-Source Basis
+ im Vergleich zu ähnlichen Frameworks einfacher Zugang zu komplexen Themen der Softwareentwicklung
+ mögliches Werkzeug für das Vermitteln eines Softwareentwicklungsprozesses im Unterricht
− Einstieg in die Komponentenarchitektur noch zu schwierig (vgl. Grok in 3.6)
− im Untericht noch nicht erprobt

4.3 Nächste Schritte

Gregor Lingl bietet in Österreich die Lehrerfortbildung für Python an. Um den Schülern Kosten für sein Buch [Lin06] zu ersparen, müsste es approbiert werden. Weiters könnten mehr Angebote in der Lehrerfortbildung organisiert werden. Kurt Winterstein, ein Lehrer an einer österreichischen Schule meint:

> *"Langfristig müssten die LehrerInnen, die sich fortbilden möchten, mehr entlastet werden. Das gilt ganz allgemein für die LehrerInnenfortbildung, weil der Schulalltag meinem Empfinden nach viel stressiger geworden ist und die Muße für eine Fortbildung fehlt."*[Win07]

An Universitäten und Fachhochschulen muss geprüft werden, ob Lehrpläne angepasst werden können. Als erster Schritt sollte ein Studiengang gewählt werden, wo das einfacher möglich ist. Beispielsweise ein Studiengang, wo das Programmieren ein allgemeinbildendes Fach ist, von dem wenig andere Lehrveranstaltungen abhängen. Gleichzeitig muss Überzeugungsarbeit geleistet werden, um Python als gute Einstiegssprache und weiterführende Sprache für fortgeschrittene Themen in der Ausbildung vorzustellen. Vorliegende Arbeit kann als Anhaltspunkt dazu verwendet werden.

Bei Zope gestaltet sich ein Versuch einfacher. Hier kann eine Projektarbeit ins Leben gerufen werden. Theoretische Teile können leicht in Überblicksveranstaltungen einfließen. Erkenntnisse daraus sollten analysiert und öffentlich gemacht werden, da es noch keine wissenschaftliche Arbeit zum Thema "Zope im Unterricht" gibt.

Die Entwicklung von *Grok* (siehe Kapitel 3.6) sollte beobachtet werden, um zu erfahren, ob *Grok* den Zugang, zum momentan doch komplexen Einstieg in die Komponentenarchitetur von Zope, erleichtert.

Langfristig könnte eine softwaretechnische Ausbildung mit Python und Zope alle wichtigen Konzepte der Softwareentwicklung abdecken.

4.4 Ähnliche Arbeiten

[Lin02] stellt die Frage: Eignet sich die Skriptsprache Python für schnelle Entwicklungen im Softwareentwicklungsprozess? Dabei analysiert er Python in Syntax und Semantik und zeigt Parallelen zwischen den Anforderungen an Programmiersprachen zur schnellen Softwareentwicklung, und den Anforderungen aus fachdidaktischer Sicht. Seine Arbeit kann die eingängliche Frage positiv beantworten. Durch die Parallelen kommt er zu dem Schluss, dass Python für den Einsatz in der informatischen Bildung gut geeignet ist.

[Mil04] erläutert in seiner Arbeit, dass informatische Bildung, vor allem das Programmieren an sich, in Zukunft den gleichen didaktischen Wert wie "herkömmliches" Lesen und Schreiben haben wird. Er beschreibt, wie so ein Unterricht aussehen kann und streicht dabei Python als präferierte Sprache für diesen Einsatz hervor. Inspiriert wurde die Arbeit dabei von [vR99].

[PG06] widmet sich dem Unterrichtsfach Informatik und zeigt, wie schwer es ist, dieses Fach aufgrund der Schnelllebigkeit der Materie zu lehren. Die Arbeit untersucht den Informatikunterricht an Österreichs Schulen im Jahr 2006. Gleichzeitig wird herausgearbeitet, wie sich Informatik von anderen Unterrichtsgegenständen grundsätzlich unterscheidet.

[Mod02] beschäftigt sich in seiner Arbeit mit Didaktikansätzen in der informatischen Lehre. Er diskutiert dabei ausführlich die in der vorliegenden Arbeit verwendete "fundamentale Idee" nach [Sch93].

[Ada04] zeigt, wie der Applikationsserver Zope im Unterricht eingesetzt werden kann. Dabei wird allerdings Version 2 des Frameworks eingesetzt. Der Bericht zeigt den erfolgreichen Einsatz von Zope innerhalb eines Webapplikationsentwicklungs-Curriculums.

Literatur

[AD02] ALLEN DOWNEY, JEFFREY ELKNER, CHRIS MEYERS: *How to Think Like a Computer Scientist- Learning with Python*. Green Tea Press, 2002.

[Ada04] ADAMS, D. ROBERT: *Teaching Web Application Development Using Zope*. Technischer Bericht, Grand Valley State University, 2004.

[Ade05] ADERMANN, NILS: *Die Definition des Algorithmusbegriffs und Darstellung der wesentlichen Merkmale eines Algorithmus an ausgesuchten Beispielen*. Technischer Bericht, Burggymnasium Essen, 2005.

[AL01] AMOS LATTEIER, MICHEL PELLETIER: *The Zope Book*. Sams, 2001.

[AM05] ALEX MARTELLI, DAVID ASCHER, ANNA RAVENSCROFT: *Python Cookbook*. O'Reilly Media, 2005.

[Bal04] BALZERT, HEIDE: *Lehrbuch der Objektmodellierung.* Spektrum-Akademischer Vlg, 2004.

[Bar90] BARENDREGT, HENDRIK PIETER: *Functional Programming and Lambda Calculus.* In: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, Seiten 321–363. 1990.

[Bec00] BECK, KENT: *Extreme programming explained: embrace change*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[Bin00] BINGEL, PETER: *Von Freier Software und Bildung*. Linux Magazin, 03, 2000.

[BM05] BN MILLER, DL RANUM: *Teaching an Introductory Computer Science Sequence with Python*. Technischer Bericht, Luther College, Iowa, 2005.

[Bro97] BROCH, ROLAND: *Computergestützte Lernprogramme für die berufliche Weiterbildung in der Kommunikationsindustrie - Computer Based Training im Kontext traditioneller Lehrmethoden*. Diplomarbeit, Bergische Universität, 1997.

[Bru60] BRUNER, JEROME: *The Process of Education*. Harvard University Press; New Edition (November 1, 2006), 1960.

[CFG05] CHRISTIAN F. GÖRLICH, LUDGER HUMBERT: *Open Source - die Rückkehr der Utopie? Philosophische Miszellen*. In: *Open Source Jahrbuch 2005. Zwischen Softwareentwicklung und Gesellschaftsmodell*. Bernd Lutterbeck, Robert A Gehring , Matthias Bärwolff, 2005.

[CG97] CARLO GHEZZI, MEHDI JAZAYERI: *Programming Language Concepts*. Wiley, 1997.

[Cho02] CHOU, P. H.: *Algorithm Education in Python*. In: *Proceedings of Python 10*, 2002.

[CW05] CRAIG WALLS, RYAN BREIDENBACH: *Spring in Action*. Manning, 2005.

[DT00] D. THOMAS, A. HUNT: *Programming Ruby: A Pragmatic Programmer's Guide*. Addison-Wesley, 2000.

[dWT02] DERAADT, M., R. WATSON und M. TOLEMAN: *Language Trends in Introductionary Programming Courses*. Informing Science and IT Education Conference, 2002.

[Elk00] ELKNER, JEFFREY: *Using Python in a High School Computer Science Program*. Technischer Bericht, Yorktown High School, Arlington, Virginia, 2000.

[Euk03] EUKLID, CLEMENS THAER: *Die Elemente. Buch I - XIII*. Deutsch (Harri), 2003. http://aleph0.clarku.edu/

[Faa07] FAASSEN, MARTIJN: *Email, Re: interview for my diploma thesis about Zope 3 in education*. `http://mail.zope.org/pipermail/zope3-dev/2007-February/021738.html`, 2007. [Online; Stand 18. Februar 2006].

[Fed96] *Federal Standard 1037C*. General Service Administration, 1996.

[FP04] FULTON, JIM und TIM PETERS: *Literate Testing: Automated Testing with doctest*. In: *Pycon 2004*, 2004.

[Ful00] FULTON, JIM: *Introduction to the Zope Object Database*. In: *Proceedings of the 8th International Python Conference*, 2000.

[FW78] FREEMAN, PETER und ANTHONY I. WASSERMAN: *A proposed curriculum for software engineering education*. In: *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, Seiten 56–62, Piscataway, NJ, USA, 1978. IEEE Press.

[GH] GENE HSIAO, STEPHEN JENKS: *High Performance Computing With Scripting Languages*. Technischer Bericht, University of California.

[Goe05] GOESCHKA, KARL MICHAEL: *Component Based Software Engineering, Teil 1*. In: *FH Technikum Wien, Studiengang ICSS, Lehrveranstaltung Komponentensysteme*, 2005.

[Gra03] GRABMÜLLER, MARTIN: *Multiparadigmen-Programmiersprachen*. Technischer Bericht, Technische Universität Berlin, 2003.

[Gra04] GRASSMUCK, VOLKER: *Freie Software: Zwischen Privat- und Gemeineigentum*. `http://freie-software.bpb.de/Grassmuck.pdf`, 2004. Bundeszentrale für politische Bildung, [Online; Stand 7. Jänner 2007].

[GW03] GEORGE, BOBY und LAURIE WILLIAMS: *An initial investigation of test driven development in industry*. In: *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, Seiten 1135–1139, New York, NY, USA, 2003. ACM Press.

[Har03a] HART, TIMOTHY D.: *Open Source in Education*. Technischer Bericht, University of Maine, 2003.

[Har03b] HARTMANN, PETER: *Mathematik für Informatiker*. vieweg, 2003.

[Jen03] JENKINS, TONY: *The First Language - A Case for Python?* In: *4th Annual LTSN-ICS Conference (Galway)*, Seite 7, 2003.

[KB01] KLAUS BOTHE, ULRICH SACKLOWSKI: *Praxisnähe durch Reverse Engineering-Projekte: Erfahrungen und Verallgemeinerungen*. Technischer Bericht, Humboldt-Universität zu Berlin, 2001.

[Knu98] KNUTH, DONALD E.: *The art of computer programming, volume 1: fundamental algorithms*. Addison-Wesley Longman, Amsterdam, 1998.

[Kva04] KVALE, AXEL ANDERS: *Empirical Study of Component Based Software Engineering with Aspect Oriented Programming*. Diplomarbeit, Norwegian University of Science and Technology, 2004.

[LAMJ05] LAHTINEN, ESSI, KIRSTI ALA-MUTKA und HANNU-MATTI JÄRVINEN: *A study of the difficulties of novice programmers*. In: *ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, Seiten 14–18, New York, NY, USA, 2005. ACM Press.

[Lej06] LEJDFORS, CALLE: *Techniques for implementing embedded domain specific languages in dynamic languages*. Diplomarbeit, Lund University, 2006.

[LGS06] LINDA GRANDELL, MIA PELTOMAKI, RALPH-JOHAN BACK und TAPIO SALAKOSKI: *Why complicate things?: introducing programming in high school using Python*. In: *ACE '06: Proceedings of the 8th Australian conference on Computing education*, Seiten 71–80, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[Lin02] LINKWEILER, INGO: *Eignet sich die Skriptsprache Python für schnelle Entwicklungen im Softwareentwicklungsprozess?* Diplomarbeit, Universität Dortmund, 2002.

[Lin06] LINGL, GREGOR: *Python für Kids*. Vmi Buch, 2006.

[LM06] LINDA MANNILA, MICHAEL DERAADT: *An Objective Comparison of Languages for Teaching Introductory Programming*. Technischer Bericht, Åbo Akademi University, University of Southern Queensland, 2006.

[Lut06] LUTZ, MARK: *Programming Python*. O'Reilly Media, 2006.

[LW00] L.A. WILLIAMS, R.R. KESSLER: *The effects of "pair-pressure" and "pair-learning" on software engineering education*. In: *Software Engineering Education & Training, 2000*, Seiten 59–65, 2000.

[MB02] MICHAEL BERNSTEIN, SCOTT ROBERTSON: *Zope Bible*. Wiley, 2002.

[Md03] M. DERAADT, R. WATSON: *Language Tug-Of-War: Industry Demand and Academic Choice*. Technischer Bericht, University of Southern Queensland, 2003.

[Mil93] MILLBRANDT, G.: *Using problem solving to teach a programming language in computer studies*. Journal Of Computer Science Education, 8(2), 1993.

[Mil04] MILLER, JOHN ALEXANDER: *Promoting Computer Literacy through programming Python*. Doktorarbeit, University of Michigan, 2004.

[Möl05] MÖLLER, ERIK: *Applikations-Baukasten*. c't, 1, 2005.

[Mod02] MODROW, ECKART: *Pragmatischer Konstruktivismus und fundamentale Ideen als Leitlinien der Curriculumentwicklung am Beispiel der theoretischen und technischen Informatik*. Doktorarbeit, Martin-Luther-Universität Halle-Wittenberg, 2002.

[MWBF02] MCDOWELL, CHARLIE, LINDA WERNER, HEATHER BULLOCK und JULIAN FERNALD: *The effects of pair-programming on performance in an introductory programming course*. SIGCSE Bull., 34(1):38–42, 2002.

[Neu02] NEUNERT, KIM: *Verwendung freier Software in der schulischen Bildung*. Diplomarbeit, Universität Augsburg, 2002.

[Nin96] NING, JIM Q.: *A Component-Based Software Development Model*. compsac, 00:0389, 1996.

[Old05] OLDHAM, JOSEPH D.: *What happens after Python in CS1?* J. Comput. Small Coll., 20(6):7–13, 2005.

[Ost04] OSTERRIEDER, CHRISTIAN: *Komponentenmodelle für Web-Anwendungen*. Diplomarbeit, Universität Salzburg, 2004.

[Ous98] OUSTERHOUT, JOHN K.: *Scripting: Higher Level Programming for the 21st Century*. IEEE Computer magazine, März, 1998.

[Pal90] PALUMBO, DAVID B.: *Programming Language/Problem-Solving Research: A Review of Relevant Issues*. Review of Educational Research, 60(1), 1990.

[PG06] POSCH-GRUBER, STEFAN: *Das Unterrichtsfach Informatik im Kontext informatischer Bildung*. Doktorarbeit, Institut für Softwaretechnologie - Technische Universität Graz, 2006.

[Pil04] PILGRIM, MARK: *Dive Into Python*. Apress, 2004.

[Pix01] *Programming Language Study*. `http://merd.sourceforge.net/pixel/language-study/`, 2001. [Online; Stand 22. Oktober 2006].

[Plö97] PLÖSCH, R.: *Design by Contract for Python*. Technischer Bericht, Johannes Kepler Universität Linz, 1997.

[Pla93] PLACER, JOHN: *The promise of multiparadigm languages as pedagogical tools*. In: *CSC '93: Proceedings of the 1993 ACM conference on Computer science*, Seiten 81–86, 1993.

[PM06] PATTERSON-MCNEILL, HOLLY: *Experience: from C++ to Python in 3 easy steps*. J. Comput. Small Coll., 22(2):92–96, 2006.

[Pyt06] *Python Webseite*. `http://www.python.org`, 2006. [Online; Stand 22. Oktober 2006].

[Rad06] RADENSKI, ATANAS: *"Python first": a lab-based digital introduction to computer science*. In: *ITICSE '06: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, Seiten 197–201, New York, NY, USA, 2006. ACM Press.

[Ric05a] RICHTER, STEPHAN: *Zope 3 Developer´s Handbook*. Sams Publishing, 2005.

[Ric05b] RICHTER, SUSANNE: *Critique for the Open Source Development Model*. Technischer Bericht, Freie Universität Berlin, 2005.

[Sch93] SCHWILL, ANDREAS: *Fundamentale Ideen der Informatik*. Technischer Bericht, Universität Oldenburg, 1993.

[Sch94] SCHWILL, ANDREAS: *Fundamentale Ideen: Eine Studie zur Methodologie der Informatik*. Technischer Bericht, Universität Paderborn, 1994.

[Sch99] SCHUBERT, SIGRID: *Begleitmaterial zur Vorlesung Einführung in die Didaktik der Informatik*. Technischer Bericht, Universität Dortmund, 1999.

[Sch06] SCHMEISSER, JOHANNES: *Open-Source-Software-Entwicklung im Informatikunterricht*. Technischer Bericht, Freie Universität Berlin, 2006.

[Sha00] SHAW, MARY: *Software engineering education: a roadmap*. In: *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, Seiten 371–380, New York, NY, USA, 2000. ACM Press.

[Sha03] SHANNON, CHRISTINE: *Another breadth-first approach to CS I using python*. SIGCSE Bull., 35(1):248–251, 2003.

[Stü02] STÜTZLE, RUPERT: *Wiederverwendung ohne Mythos: Empirisch fundierte Leitlinien für die Entwicklung wiederverwendbarer Software*. Doktorarbeit, Institut für Informatik der Technischen Universität München, 2002.

[Sta00] STAJANO, FRANK: *Python in Education: Raising a Generation of Native Speakers*. Technischer Bericht, University of Cambridge, 2000.

[Sta03] STALLMAN, RICHARD: *Why schools should use exclusively free software*. `http://www.gnu.org/philosophy/schools.html`, 2003. [Online; Stand 7. Jänner 2007].

[Str99] STRITZINGER, ALOIS: *Komponentenbasierte Softwareentwicklung. Konzepte und Techniken für das Programmieren und Modellieren in Smalltalk*, 1999.

[SW98] STEPHENSON, C. und T. WEST: *Language choice and key concepts in cs 1*. Journal of Research on Computing Education, 31(1), 1998.

[Swa04] SWAROOP, CH: *A Byte of Python*. `http://www.swaroopch.info/`, 2004. [Online; Stand 22. Oktober 2006].

[TIO06] *TIOBE Programming Community Index*. `http://www.tiobe.com/tpci.htm`, 2006. [Online; Stand 16. Dezember 2006].

[TM03] TORKAR, RICHARD und STEFAN MANKEFORS: *A Survey on Testing and Reuse*. In: *SwSTE '03: Proceedings of the IEEE International Conference on Software-Science, Technology & Engineering*, 2003.

[uMG05] MARTIN GLINZ, ROBERT STOYAN UND: *Methoden und Techniken zum Erreichen didaktischer Ziele in Software-Engineering-Praktika*. Technischer Bericht, Institut für Informatik der Universität Zürich, 2005.

[Urb06] URBAN, WOLFGANG: *Materialien für Mathematik, Physik, Informatik*. `http://www.hib-wien.at/leute/wurban/index.html`, 2006. [Online; Stand 31. Dezember 2006].

[Urn04] URNER, KIRBY: *Python in the Mathematics Curriculum*. `http://www.python.org/pycon/dc2004/papers/15/`, 2004. [Online; Stand 6. Jänner 2007].

[vR99] ROSSUM, GUIDO VAN: *Computer Programming for Everybody: A Scouting Expedition for the Programmers of Tomorrow*. `http://www.python.org/doc/essays/cp4e.html`, 1999. CNRI Proposal 90120-1a, Corporation for National Research Initiatives, [Online; Stand 16. Februar 2007].

[vW05a] WEITERSHAUSEN, PHILIPP VON: *Agil serviert, Zope X3.0 mit Komponentenarchitektur*. iX, 8, 2005.

[vW05b] WEITERSHAUSEN, PHILIPP VON: *Formula X, Developing Web Appplications with Zope X3*. Linux Magazine, 54, 2005.

[vW06] WEITERSHAUSEN, PHILIPP VON: *Web Component Development with Zope 3*. Springer, 2006.

[Wes99] WESTBROOK, D.S.: *A multiparadigm language approach to teaching principles of-programming languages*. In: *Frontiers in Education Conference, 1999. FIE '99. 29th Annual*, Seiten 11B3/14–11B3/18, 1999.

[Wil02] WILLIAMS, MICHAEL: *Teaching scientific programming using Python*, 2002.

[Win07] WINTERSTEIN, KURT: *Email, Re: Anfrage für ein Interview zu meiner Diplomarbeit*, 2007.

[Wir71] WIRTH, NIKLAUS: *The programming language pascal*. Acta Informatica, 1(1):35–63, 1971.

[WU01] WILLIAMS, LAURIE und RICHARD L. UPCHURCH: *In support of student pair-programming*. In: *SIGCSE '01: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, Seiten 327–331, New York, NY, USA, 2001. ACM Press.

[Zel98] ZELLE, JOHN M.: *Python as a First Language*. Technischer Bericht, Wartburg College, 1998.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Anhang

A Quelltexte

Alle Quelltexte zu dieser Arbeit und der Status des Projektes Stadtgespräche, sind unter `http://www.stadtgespraeche.com/technik` bzw. `http://www.mira4.com/stadtgespraeche` zu finden.

interfaces.py

```
1  from zope.interface import Interface
2  from zope.app.file.interfaces import IImage
3  from zope.app.container.interfaces import IContainer
4  from zope.app.container.constraints import contains
5  from zope.schema import Text, TextLine, Date
6
7
8  class IEntry(IImage):
9      """ Interface for Entry  """
10
11     description = Text(title=u"Image Description", required=False)
12
13     dateTaken = Date(title=u"Date when picture was taken", required=True)
14     placeTaken = TextLine(title=u"Place where picture was taken",
15                           required=True)
16
17     contributor = TextLine(title=u"email of contributor", required=True)
18
19
20 class IEntryContainer(IContainer):
21     """ Container Interface for an Entry """
22     contains(IEntry)
23
24 class IEntryNav(Interface):
25     """used as Adapter for menu creation
26     """
27
28     def setEntryMenuName(menuname):
29         """ set the navigation menu name  """
30
31     def getEntryMenuName():
32         """ get the navigation menu name """
33
34     def getEntryMenuNameFirstLetter():
35         """ get the navigation menu name first letter """
36
37     def getEntryMenuOrder():
38         """ get the navigation menu order """
39
40     def setEntryMenuOrder(order):
41         """ set the navigation menu order """
```

```
42
43      def getChildren():
44          """ get all IEntry children from this location """
45
46      def getSiblings():
47          """ get all IEntry Siblings from this location
48              (self is included)
49          """
```

Listing 28: interfaces.py

entry.py

```
1  from zope.interface import implements
2  from zope.component import adapts
3  from zope.app.container.btree import BTreeContainer
4  from zope.app.file.image import Image
5  from zope.size.interfaces import ISized
6  from zope.size import byteDisplay
7  from zope.annotation.interfaces import IAttributeAnnotatable
8  from zope.annotation import IAnnotations
9  from zope.app.i18n import ZopeMessageFactory as _
10
11 from interfaces import *
12
13 class Entry(BTreeContainer, Image):
14     implements(IEntry, IEntryContainer, IAttributeAnnotatable)
15
16     __name__ = __parent__ = None
17
18     description = u""
19
20     dateTaken = None
21     placeTaken = u""
22
23     contributor = u""
24
25
26 EntryNavAnnotations_KEY = "sgs.entrynav"
27
28 class EntryNav(object):
29     implements(IEntryNav)
30     adapts(IEntry)
31
32     def __init__(self, context):
33         self.context = self.__parent__  = context # see PvWh book site 269
34         annotations = IAnnotations(context)
35         mapping = annotations.get(EntryNavAnnotations_KEY)
36         if mapping is None:
37             mapping = annotations[EntryNavAnnotations_KEY] = {'name': '',
38                                                               'order': 0}
39         self.mapping = mapping
40
41     def setEntryMenuName(self, menuname):
42         self.mapping['name'] = menuname
43
```

```
44      def getEntryMenuName(self):
45          return self.mapping['name']
46
47      def getEntryMenuNameFirstLetter(self):
48          if len(self.mapping['name']):
49              return self.mapping['name'][0]
50
51      def setEntryMenuOrder(self, order):
52          self.mapping['order'] = order
53
54      def getEntryMenuOrder(self):
55          return self.mapping['order']
56
57      def getChildren(self):
58          return self.context.values()
59
60      def getSiblings(self):
61          siblings = []
62          for sibling in self.context.__parent__.values():
63              if IEntry.providedBy(sibling):
64                  siblings.append(sibling)
65          return siblings
66
67
68
69  class EntrySized(object):
70      implements(ISized)
71      adapts(IEntry)
72
73      def __init__(self, container):
74          self._container = container
75
76      def sizeForSorting(self):
77          """See `ISized`"""
78          return ('item', len(self._container))
79          #return ('byte', self._container.getSize())
80
81      def sizeForDisplay(self):
82          """See `ISized`"""
83          bytes = self._container.getSize()
84          byte_size = byteDisplay(bytes)
85
86          num_items = len(self._container)
87
88          mapping = byte_size.mapping
89          if mapping is None:
90              mapping = {}
91          mapping.update({'items': str(num_items)})
92
93          if num_items == 1:
94              return _("${items} item / " + byte_size , mapping=mapping)
95          else:
96              return _("${items} items / " + byte_size , mapping=mapping)
```

Listing 29: entry.py

entry.txt

```
1  ====================
2  Test EntryNav Adapter
3  ====================
4
5    >>> from sgs.entry.entry import Entry
6    >>> from sgs.entry.interfaces import IEntryNav
7
8    >>> root = testEntryStructure()
9
10 Let's pick out entry1_1_1 and test getChildren and getSiblings
11
12   >>> entry =  root[u'entry1'][u'entry1_1'][u'entry1_1_1']
13   >>> entrynav = IEntryNav(entry)
14   >>> len(entrynav.getChildren())
15   0
16
17   >>> len(entrynav.getSiblings())
18   3
19
20 Now test it with entry1_1
21
22   >>> entry =  root[u'entry1'][u'entry1_1']
23   >>> entrynav = IEntryNav(entry)
24   >>> len(entrynav.getChildren())
25   3
26
27   >>> len(entrynav.getSiblings())
28   2
29
30 Check if the MenuName Annotation is set and remembered
31
32   >>> entrynav.getEntryMenuName()
33   u'Menschen'
34
35   >>> entrynav.setEntryMenuName(u'Sozial')
36   >>> entrynav.getEntryMenuName()
37   u'Sozial'
38
39 Create another adapter and check if the menuname is still the same
40
41   >>> otheradapter = IEntryNav(entry)
42   >>> otheradapter.getEntryMenuName()
43   u'Sozial'
44
45
46 Try the same with the EntryNav "order" key
47
48   >>> entrynav.getEntryMenuOrder()
49   12
50   >>> entrynav.setEntryMenuOrder(1)
51   >>> entrynav.getEntryMenuOrder()
52   1
53   >>> otheradapter = IEntryNav(entry)
54   >>> otheradapter.getEntryMenuOrder()
```

```
1
```

Listing 30: entry.txt

configure.zcml

```
<configure
        xmlns="http://namespaces.zope.org/zope"
        i18n_domain="zope"
        >

<include package=".browser" />

<class class=".entry.Entry" >
    <require
        permission="zope.View"
        interface=".interfaces.IEntry" />
    <require
        permission="zope.View"
        interface=".interfaces.IEntryContainer" />
    <require
        permission="zope.ManageContent"
        set_schema=".interfaces.IEntry" />
</class>

<adapter factory=".entry.EntryNav"
     trusted="true"  />

<adapter factory=".entry.EntrySized"  />

<class class=".entry.EntryNav" >
    <require
        permission="zope.View"
        interface=".interfaces.IEntryNav" />
</class>

</configure>
```

Listing 31: configure.zcml

tests/test_entryNav.py

```
import unittest
from doctest import DocFileSuite, ELLIPSIS

#import zope.component.testing
import zope.app.testing.setup
#from zope.annotation.attribute import AttributeAnnotations
from zope.app.folder import rootFolder

from sgs.entry.interfaces import IEntryNav
from sgs.entry.entry import EntryNav
from sgs.entry.entry import Entry

```

```
13 def setupEntryStructure():
14     """ Set up a reasonably complex folder structure
15
16      _____________ rootFolder ____________________________
17     /                                                    \
18  folder1 ____________________________________        folder2
19    |                         \              |
20  folder1_1 ___________________        folder1_2    folder2_1
21    |           \           \              |            |
22  folder1_1_1 folder1_1_2 folder1_1_3  folder1_2_1  folder2_1_1
23     """
24     root = rootFolder()
25     entry = root[u'entry1'] = Entry()
26     IEntryNav(entry).setEntryMenuName(u'Pictorial')
27     IEntryNav(entry).setEntryMenuOrder(2)
28     entry = root[u'entry1'][u'entry1_1'] = Entry()
29     IEntryNav(entry).setEntryMenuName(u'Menschen')
30     IEntryNav(entry).setEntryMenuOrder(12)
31     entry = root[u'entry1'][u'entry1_1'][u'entry1_1_1'] = Entry()
32     IEntryNav(entry).setEntryMenuName(u'Frauen')
33     IEntryNav(entry).setEntryMenuOrder(113)
34     entry = root[u'entry1'][u'entry1_1'][u'entry1_1_2'] = Entry()
35     IEntryNav(entry).setEntryMenuName(u'Gruppen')
36     IEntryNav(entry).setEntryMenuOrder(111)
37     entry = root[u'entry1'][u'entry1_1'][u'entry1_1_3'] = Entry()
38     IEntryNav(entry).setEntryMenuName(u'Ausrutschen')
39     IEntryNav(entry).setEntryMenuOrder(112)
40     entry = root[u'entry1'][u'entry1_2'] = Entry()
41     IEntryNav(entry).setEntryMenuName(u'Verkehr')
42     IEntryNav(entry).setEntryMenuOrder(11)
43     entry = root[u'entry1'][u'entry1_2'][u'entry1_2_1'] = Entry()
44     IEntryNav(entry).setEntryMenuName(u'Pferd')
45     entry = root[u'entry2'] = Entry()
46     IEntryNav(entry).setEntryMenuName(u'Sozial')
47     IEntryNav(entry).setEntryMenuOrder(1)
48     entry = root[u'entry2'][u'entry2_1'] = Entry()
49     IEntryNav(entry).setEntryMenuName(u'Hilfestellungen')
50     IEntryNav(entry).setEntryMenuOrder(21)
51     entry = root[u'entry2'][u'entry2_1'][u'entry2_1_1'] = Entry()
52     IEntryNav(entry).setEntryMenuName(u'Offiziell')
53     IEntryNav(entry).setEntryMenuOrder(22)
54     return root
55
56
57 def setUp(test):
58     zope.app.testing.setup.placefulSetUp(site=True)
59     zope.component.provideAdapter(EntryNav)
60
61 def tearDown(self):
62     zope.app.testing.setup.placefulTearDown()
63
64 def test_suite():
65     return unittest.TestSuite((
66         DocFileSuite('entry.txt',
67                      package='sgs.entry',
68                      setUp=setUp,
```

```
                        tearDown=tearDown,
                        optionflags=ELLIPSIS,
                        globs={'testEntryStructure': setupEntryStructure}),
        ))

if __name__ == '__main__':
    unittest.main(defaultTest='test_suite')
```

Listing 32: tests/test_entryNav.py

browser/__init__.py

```
from operator import itemgetter

from zope.traversing.api import getParents
from zope.traversing.interfaces import IPhysicallyLocatable
from zope.publisher.browser import BrowserView
from zope.traversing.browser import absoluteURL

from sgs.entry.interfaces import IEntryNav

class EntryNavigation(BrowserView):

    def __call__(self):
        root = IPhysicallyLocatable(self.context).getNearestSite()
        parents = getParents(self.context)

        # remove all parents below the first found site
        while parents[-1] != root:
            parents.pop()

        parents.pop() #remove Site from parents
        parents.reverse()

        parentSiblings = [ sorted( [
            {'name':IEntryNav(sibling).getEntryMenuName(),
             'order':IEntryNav(sibling).getEntryMenuOrder(),
             'url':absoluteURL(sibling, self.request),
             'active':((sibling in parents) and 'parent') or False}
            for sibling in IEntryNav(item).getSiblings() ] ,
                            key=itemgetter('order')) for item in parents ]

        mySiblings = sorted( [
            {'name':IEntryNav(sibling).getEntryMenuName(),
             'order':IEntryNav(sibling).getEntryMenuOrder(),
             'url':absoluteURL(sibling, self.request),
             'active':((sibling == self.context) and 'self') or False} \
            for sibling in IEntryNav(self.context).getSiblings() ] ,
                        key=itemgetter('order'))


        children = sorted( [
            {'name':IEntryNav(child).getEntryMenuName(),
             'order':IEntryNav(child).getEntryMenuOrder(),
             'url':absoluteURL(child, self.request) }
            for child in IEntryNav(self.context).getChildren() ],
```

```
45                              key=itemgetter('order'))
46
47          parentSiblings.append(mySiblings)
48
49          if len(children):
50              parentSiblings.append(children)
51
52          return parentSiblings
```

Listing 33: browser/__init__.py

browser/configure.zcml

```
1  <configure xmlns="http://namespaces.zope.org/browser">
2
3  <addMenuItem
4        class="..entry.Entry"
5        title="SGS Entry"
6        permission="zope.ManageContent"
7        view="AddSGSEntry.html"
8        />
9
10 <addform
11       label="Add SGS Entry"
12       name="AddSGSEntry.html"
13       schema="..interfaces.IEntry"
14       content_factory="..entry.Entry"
15       class="zope.app.file.browser.image.ImageAdd"
16       permission="zope.ManageContent"
17       />
18
19 <containerViews
20       for="..interfaces.IEntryContainer"
21       contents="zope.ManageContent"
22       add="zope.ManageContent"
23       />
24
25 <view
26       for="sgs.entry.interfaces.IEntry"
27       name="getEntryNavigation"
28       class=".EntryNavigation"
29       permission="zope.View"
30       />
31
32 <page
33     for="*"
34     name="test.html"
35     template="test.pt"
36     permission="zope.View"
37     />
38
39 </configure>
```

Listing 34: browser/configure.zcml

browser/tests.py

```
import unittest
from zope.testing import doctest
from zope.publisher.browser import TestRequest
import zope.app.testing.setup

from sgs.entry.interfaces import IEntryNav
from sgs.entry.entry import EntryNav
from sgs.entry.browser import EntryNavigation
from sgs.entry.tests.test_entryNav import setupEntryStructure

def setUp(test):
    zope.app.testing.setup.placefulSetUp(site=True)
    zope.component.provideAdapter(EntryNav)

def tearDown(test):
    zope.app.testing.setup.placefulTearDown()


class EntryNavigationTestCase(unittest.TestCase):

    def setUp(self):
        setUp(self)

    def tearDown(self):
        tearDown(self)

    def test_navigation_firstlevel(self):
        request = TestRequest()
        root = setupEntryStructure()

        view = EntryNavigation(root[u'entry1'], request)

        self.assertEqual(view(), [[{'url': 'http://127.0.0.1/entry2', \
'active': False, 'name': u'Sozial', 'order': 1}, {'url': \
'http://127.0.0.1/entry1', 'active': 'self', 'name': u'Pictorial', \
'order': 2}], [{'url': 'http://127.0.0.1/entry1/entry1_2', 'name': \
u'Verkehr', 'order': 11}, {'url': 'http://127.0.0.1/entry1/entry1_1', \
'name': u'Menschen', 'order': 12}]])

    def test_navigation_midlevel(self):
        request = TestRequest()
        root = setupEntryStructure()

        view = EntryNavigation(root[u'entry1'][u'entry1_1'], request)

        self.assertEqual(view(), [[{'url': 'http://127.0.0.1/entry2', \
'active': False, 'name': u'Sozial', 'order': 1}, {'url': \
'http://127.0.0.1/entry1', 'active': 'parent', 'name': u'Pictorial', \
'order': 2}], [{'url': 'http://127.0.0.1/entry1/entry1_2', 'active': False, \
'name': u'Verkehr', 'order': 11}, {'url': 'http://127.0.0.1/entry1/entry1_1'\
,'active': 'self', 'name': u'Menschen', 'order': 12}], [{'url': \
'http://127.0.0.1/entry1/entry1_1/entry1_1_2', 'name': u'Gruppen', \
'order': 111}, {'url': 'http://127.0.0.1/entry1/entry1_1/entry1_1_3', \
'name': u'Ausrutschen', 'order': 112}, {'url': \
```

```
'http://127.0.0.1/entry1/entry1_1/entry1_1_1', 'name': u'Frauen', \
'order': 113}]])

    def test_navigation_lastlevel(self):
        request = TestRequest()
        root = setupEntryStructure()

        view = EntryNavigation(root[u'entry1'][u'entry1_1'][u'entry1_1_1'],
                               request)

        self.assertEqual(view(), [[{'url': 'http://127.0.0.1/entry2', \
'active': False, 'name': u'Sozial', 'order': 1}, {'url': \
'http://127.0.0.1/entry1', 'active': 'parent', 'name': u'Pictorial', \
'order': 2}], [{'url': 'http://127.0.0.1/entry1/entry1_2', \
'active': False, 'name': u'Verkehr', 'order': 11}, {'url': \
'http://127.0.0.1/entry1/entry1_1', 'active': 'parent', 'name': \
u'Menschen', 'order': 12}], [{'url': \
'http://127.0.0.1/entry1/entry1_1/entry1_1_2', 'active': False, \
'name': u'Gruppen', 'order': 111}, {'url': \
'http://127.0.0.1/entry1/entry1_1/entry1_1_3', \
'active': False, 'name': u'Ausrutschen', 'order': 112}, {'url': \
'http://127.0.0.1/entry1/entry1_1/entry1_1_1', 'active': 'self', \
'name': u'Frauen', 'order': 113}]])


def test_suite():
    suite = unittest.TestSuite()
    suite.addTest(unittest.makeSuite(EntryNavigationTestCase))
    suite.addTest(doctest.DocFileSuite(
        'catalog.txt',
        setUp=setUp,
        tearDown=tearDown,
        globs={'testEntryStructure': setupEntryStructure}))
    return suite

if __name__ == '__main__':
    unittest.main(defaultTest='test_suite')
```

Listing 35: browser/tests.py

browser/catalog.txt

```
SGS Catalog Test
================

This test covers a catalog search for Entries.

  >>> from zope.component import provideUtility
  >>> from sgs.entry.interfaces import IEntryNav

We use a fake int id utility here so we can test independent of
the full-blown zope environment::

  >>> from zope import interface
  >>> import zope.app.intid.interfaces
  >>> class DummyIntId(object):
```

```
  ...     interface.implements(zope.app.intid.interfaces.IIntIds)
  ...     MARKER = '__dummy_int_id__'
  ...     def __init__(self):
  ...         self.counter = 0
  ...         self.data = {}
  ...     def register(self, obj):
  ...         intid = getattr(obj, self.MARKER, None)
  ...         if intid is None:
  ...             setattr(obj, self.MARKER, self.counter)
  ...             self.data[self.counter] = obj
  ...             intid = self.counter
  ...             self.counter += 1
  ...         return intid
  ...     def getObject(self, intid):
  ...         return self.data[intid]
  ...     def __iter__(self):
  ...         return iter(self.data)
  >>> intid = DummyIntId()
  >>> provideUtility(intid, zope.app.intid.interfaces.IIntIds)

Now let's register a catalog::

  >>> from zope.app.catalog.interfaces import ICatalog
  >>> from zope.app.catalog.catalog import Catalog
  >>> catalog = Catalog()
  >>> provideUtility(catalog, ICatalog)

And set it up the field index::

  >>> from zope.app.catalog.field import FieldIndex
  >>> catalog['menuNames'] = FieldIndex(
  ...              interface=IEntryNav,
  ...              field_name='getEntryMenuNameFirstLetter',
  ...              field_callable=True
  ... )

Now let's create some objects so that they'll be cataloged::

  >>> root = testEntryStructure()

Now iterate through them and index them

  >>> def index_all(node):
  ...     for item in node.values():
  ...         catalog.index_doc(intid.register(item), item)
  ...         index_all(item)
  ...
  >>> index_all(root)

There should be 10 intid items in the catalog

  >>> len(list(catalog.apply({'menuNames':(None, None)})))
  10
  >>> len(list(catalog.searchResults(menuNames=(None, None))))
  10

```

```
71  >>> for item in list(catalog.searchResults(menuNames=(None, None))):
72  ...     print item.__name__
73  entry1
74  entry1_1
75  entry1_1_1
76  entry1_1_2
77  entry1_1_3
78  entry1_2
79  entry1_2_1
80  entry2
81  entry2_1
82  entry2_1_1
83
84  >>> from sgs.entry.interfaces import IEntryNav
85  >>> for item in list(catalog.searchResults(menuNames=('S', 'S'))):
86  ...     IEntryNav(item).getEntryMenuName()
87  u'Sozial'
```

Listing 36: browser/catalog.txt