Ergebnisse der Untersuchung zur Eignung einer Programmiersprache für die schnelle Softwareentwicklung – kann der Informatikunterricht davon profitieren?

Ingo Linkweiler, Ludger Humbert

Didaktik der Informatik
Universität Dortmund
D-44 221 Dortmund
i.linkweiler@gmx.de
ludger.humbert@cs.uni-dortmund.de

Zusammenfassung: Die Frage nach geeigneten Programmiersprachen für den Informatikunterricht wird seit Jahrzehnten diskutiert. Dieser Bericht fasst die Ergebnisse einer Untersuchung von Programmiersprachen – und insbesondere der Sprache *Python* – unter dem Aspekt der schnellen Softwareentwicklung zusammen. Es zeigen sich Parallelen bei den Anforderungen an Programmiersprachen zur schnellen Softwareentwicklung zu den Anforderungen aus fachdidaktischer Sicht. Kann der Informatikunterricht von den Erfahrungen im Bereich der schnellen Softwareentwicklung profitieren?

1 Einleitung

Aktuelle Verfahren der schnellen Softwareentwicklung, wie Rapid Prototyping, Extreme Programming und Pair Programming werden in den letzten Jahren im Zusammenhang mit der Softwaretechnik heftig diskutiert. Mittlerweile werden diese Verfahren in der Praxis vielfach eingesetzt und haben sich als brauchbare Methode der Softwareentwicklung erwiesen. Eine geeignete Programmiersprache und Entwicklungsumgebung können dabei den Softwareentwicklungsprozess unterstützen. Im alltäglichen Einsatz haben sich dabei Skriptsprachen wie Tcl, Perl und Python als praxistaugliche Werkzeuge erwiesen. Durch den Einsatz eines Interpreters statt eines Compilers wird interaktive Entwicklung unterstützt.

Im Folgenden werden Teilergebnisse einer Untersuchung der Programmiersprache Python zur Eignung für schnelle Softwareentwicklung vorgestellt. Die Untersuchung deckt Parallelen zwischen den Anforderungen an eine Programmiersprache auf, bezogen sowohl auf professionelle Entwickler als auch auf fachdidaktische Anforderungen.

Hypothesen:

- Die Anforderungen an eine Programmiersprache zur schnellen Softwareentwicklung und zur informatischen Bildung stimmen in einigen Bereichen überein. Es ist möglich, beiden Anforderungen gleichermaßen gerecht zu werden.
- Python und für Python verfügbare Werkzeuge eignen sich sowohl zur schnellen Softwareentwicklung als auch zum Einsatz in der informatischen Bildung.

2 Vorgehensweise

Zur Beurteilung von Programmiersprachen wurde durch Anfragen und Diskussionen mit praktizierenden Informatikern im Bereich der schnellen Softwareentwicklung und Lehrern der Informatik je ein Kriterienkatalog erstellt. Die Zusammenstellung der Anforderungen an eine Programmiersprache in einem gemeinsamen Katalog macht zahlreiche gemeinsame Interessen erkennbar.

In der Untersuchung wird die Programmiersprache Python anhand der aufgestellten Kriterien untersucht und mit anderen Sprachen verglichen. Python wurde als Referenzsprache ausgewählt, denn es bietet unter den derzeit eingesetzten Programmiersprachen zur schnellen Softwareentwicklung eine leicht erlernbare Syntax und scheint auf den ersten Blick viele der gestellten Kriterien zu erfüllen. Die Praxistauglichkeit von Python wurde anhand dreier Fallstudien in Form unterschiedlicher Softwareprojekte überprüft. Des Weiteren wurden Erhebungen zur Lesbarkeit ausgewählter Programmiersprachen durchgeführt.

3 Kriterien und Ergebnisse

In der Softwareentwicklung wurde zu Beginn ein lineares Vorgehensmodell – das Wasserfallmodell – eingeführt. Mit diesem Modell verbundene offensichtliche Probleme führten zur Weiterentwicklung zu evolutionären, partizipativen Softwareentwicklungsmethoden. Daher wird deutlich, dass die Auswahl von konkreten Kriterien immer den aktuellen Forschungsstand berücksichtigen sollte.

Die ermittelten Anforderungen an eine Programmiersprache wurden in einer Matrix zusammengestellt. Dabei wird die Bedeutung und Wichtigkeit der Kriterien für jede der beiden Interessensgruppen ausgewiesen, um so Konfliktbereiche aber auch

Gemeinsamkeiten zu identifizieren. Ein Ausschnitt aus dem entwickelten Kriterienkatalog (Tab. 1) zeigt die Bandbreite der Fragestellungen. Er sollte nicht dazu verleiten, feste Positionen zu zementieren, sondern zeigen, auf welcher Entscheidungsgrundlage die folgenden Aussagen basieren. Nachfolgend werden ausgewählte Kriterien vorgestellt und die Ergebnisse bezogen auf die untersuchte Sprache Python zusammengefasst.

Portabilität ist im Kontext vernetzter und interaktiver heterogener Informatiksysteme allgemein wünschenswert. Python bietet hier alle nötigen Voraussetzungen: erstellte Programme lassen sich weitestgehend systemunabhängig einsetzen.

Schneller GUI-Entwurf ist eines der wichtigsten Einsatzgebiete des Rapid Prototyping. Gleichzeitig gewinnt er auch im Informatikunterricht an Bedeutung, was an der wachsenden Verbreitung von visuellen Softwareentwicklungssystemen deutlich wird. Für Python sind zahlreiche Bibliotheken und integrierte Entwicklungsumgebungen verfügbar, welche einen hohen Grad an Modularisierung unterstützen.

Werkzeuge zur Modellierung und Entwicklung: Zu modernen Sprachen sollten Werkzeuge existieren, welche eine Anbindung an die Modellierung erlauben. Modellierung ist heute ein wichtiges Teilgebiet des Informatikunterrichts. Die Möglichkeiten der computergestützten Modellierung mit automatischer Quelltexterzeugung können die Implementierung erleichtern und schaffen eine Verknüpfung von der reinen Modellsicht zur Umsetzung als Programm. Moderne Entwicklungsumgebungen bieten dabei Vereinfachungen bei der Quelltexteingabe und helfen, Fehler zu vermeiden. Für Python sind unterschiedliche Werkzeuge verfügbar, die sowohl schulischen wie professionellen Anforderungen gerecht werden.

Einfache, widerspruchsfreie Syntax und Semantik erleichtert die Lesbarkeit und das Erlernen einer Programmiersprache und hilft, Fehler zu vermeiden. Häufig wird ein Pascal-ähnlicher „Pseudocode" zum Erklären von Algorithmen eingesetzt. Python bietet eine Pseudocode-artige Syntax. Die Blockbildung durch Klammern bzw. `begin` und `end` und Anweisungstrennung durch Semikolon ist in Python nicht erforderlich: durch Einrücken von Anweisungen werden Blöcke definiert. Zahlreiche in die Sprache integrierte Datentypen wie Listen, Strings, Hashtabellen, Tupel sowie vollkommen dynamische Variablentypisierung und Polymorphie sind ebenfalls sehr elegante und hervorragend lesbare Merkmale der Syntax. Einige Beispiele:

```
>>> zahl = 3.141
>>> tupel = (7, 42)
>>> text = 'Hallo.'
>>> print text*2
Hallo.Hallo.
>>> hash = {1: 'abc', 2: 'def'}
>>> c=complex(4,-2)
```

```
>>> liste = [0,1,2,3,4,5]
>>> print liste[2:5]
 [2, 3, 4]
>>> print liste[:3]
 [0, 1, 2]
>>> for i in liste:
>>>    tuwas(i)
```

Automatische Speicherverwaltung in modernen Sprachen nimmt dem Entwickler das Reservieren und Freigeben von Speicher ab. So entfernt Python automatisch nicht mehr gebrauchte Objekte aus dem Speicher. Das programminterne Verwalten von Speicher durch den Entwickler sowie der Umgang mit Zeigern ist ein Relikt aus alten Sprachen wie C und Pascal. Zur Darstellung von komplexen Strukturen wie Bäumen werden heute Klassen und Attribute mit Referenzen verwendet.

Dokumentation: Alle Funktionen, Objekte und Module können in Python direkt mit einer Dokumentation versehen werden, die interaktiv im Interpreter aufgerufen werden kann. Aus der Quelltextdokumentation lässt sich mit geeigneten, verfügbaren Werkzeugen eine vollständige Projektdokumentation erzeugen.

Modularisierung, Wiederverwendbarkeit, Softwaretest: Das „Zusammenstecken" von Software aus fertigen Komponenten (Baukastenprinzip) ist kostengünstig und spart Entwicklungszeit für Eigenentwicklungen. Möglichst vielseitige, polymorphe und sichere Bibliotheken bestimmen häufig die Geschwindigkeit der Softwareentwicklung. Kostengünstige Verfügbarkeit und gute Recherchemöglichkeiten zum Finden von Bibliotheken zu einem gegebenen Problem sind wichtige Faktoren. Insbesondere bei der Arbeit mit Prototypen sind wie auch im Ingenieurwesen häufige Tests einzelner Komponenten wie auch des gesamten Prototyps nötig. Das Modulkonzept von Python mit integrierten sowie interaktiven Testmöglichkeiten im Interpreter machen Python hier zu einem geeigneten Werkzeug.

Paradigmen der Softwareentwicklung: Obwohl heute das Gros der Softwareentwicklung nach dem objektorientierten Paradigma erfolgt, sollten die anderen Paradigmen nicht vernächlässigt werden. Python bietet die Möglichkeit, mehrere Paradigmen (prozedural, objektorientiert, funktional) der Softwareentwicklung unter dem Dach einer einzigen Programmiersprache zu nutzen. Damit wird Python der Forderung nach einer breit angelegten Sprache weitgehend gerecht. Nur die prädikativen Entwicklungsmöglichkeiten sind derzeit, verglichen mit rein prädikativen Sprachen wie Prolog, eingeschränkt und nicht sauber realisierbar. Im schulischen Einsatz besteht jedoch die Gefahr, dass die unterschiedlichen Paradigmen gemischt werden können. Professionelle Entwickler mögen das durchaus als Vorteil werten, im schulischen Einsatz kann es dazu führen, dass die Paradigmen nicht als eigenständige Konzepte vermittelt werden.

Widerspruchsfrei und so einfach wie möglich zum Ziel: Eine Programmiersprache sollte möglichst auf unnötige Anweisungen und Deklarationen verzichten können. Gerade im Anfangsunterricht ist es nicht einfach, die Bedeutung der Schlüsselworte `void`, `public`, `static` zu fundamentieren. Das *Hello-World*-Beispiel in Java zeigt diese Problematik, in vielen anderen Sprachen (wie C++) sieht es ähnlich

aus. Python gestattet Programmierung ohne redundante Anteile, die Anweisungen `print` und `input` sind direkt ohne weitere Zusätze nutzbar. Die Deklaration von Unterprogrammen sowie Funktionen ähnelt der mathematischen Definition einer Funktion und kommt ohne Zusätze von Typinformation und Geltungsbereichen aus.

```
Java:
class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
Basic, Python:
print "Hello World"
```

Lernumgebungen: Um die informatische Modellierung zielgerichtet unterrichtlich umsetzen zu können, ist es notwendig, geeignete Lernhilfen nutzen zu können. Dabei kommt der Unterstützung möglichst verschiedener Zugangsmöglichkeiten zum Problembereich eine herausragende Bedeutung zu. Zum Einsatz von Python steht dem Lehrer bereits ein gutes Angebot an Hilfsmitteln für anschaulichen Unterricht zur Verfügung, wie Umsetzungen von *Karel* bzw. *Niki der Roboter* oder der grafischen objektorientierten Lernsoftware *Von Stiften und Mäusen* und *PyCard* (angelehnt an *Hypercard*).

Übersicht Der Kriterienkatalog in Tabelle 1 zeigt einen Ausschnitt der überprüften Kritierien. Der vollständige Katalog wird bei Fertigstellung der gesamten Untersuchung Ende November auf den Internetseiten *[Li02]* veröffentlicht.

4 Fallstudien und Erhebungen

Die Auswertung existierender empirischer Untersuchungen [Pr02, Wa02, WP02], bei denen Entwickler gegebene Probleme in ausgewählten Programmiersprachen lösen sollten, hat gezeigt, dass moderne objektorientierte Skriptsprachen bezüglich der Entwicklungszeit und Quelltextgröße den restlichen Sprachen meist deutlich überlegen sind. Innerhalb der Skriptsprachen gehört Python dabei zu den jeweils besten Sprachen. Laufzeitverluste durch den Interpreter können in der Praxis oft vernachlässigt werden.

Zur Überprüfung der Praxisrelevanz wurden vom Autor Fallstudien in Form zweier Softwareprojekte durchgeführt.

Kriterien	für schnelle Entwicklung	für informatische Bildung	Eignung von Python
Portabilität und Dateizugriff	✓	✓	++
Schneller GUI-Entwurf	✓	○	++
Geeignete Hilfsmittel und Werkzeuge	✓	✓	++
Einfache Syntax und Semantik	✓	✓	++
Modularisierung und Wiederverwendbarkeit	✓	✓	++
Dokumentation	✓	✓	++
Softwaretest	✓	✓	++
Fehlersuche und Vermeidung	✓	✓	++
Nebenläufigkeit (Threading)	✓	✓	+
Erweiterungsfähigkeit und Einbettung	✓	×	++
Automatische Speicherverwaltung	✓	✓	++
Objektorientierte Entwicklung	✓	✓	++
Funktionale Entwicklung	○	✓	+
Prädikative Entwicklung	○	✓	-
Einfache Lesbarkeit und Erlernbarkeit	○	✓	++
Austausch von Algorithmen	○	✓	++
Lernumgebung	○	✓	+
Motivation	✓	✓	++

✓ ist erforderlich, ○ ist nützlich, × ist unwichtig

++ sehr gut geeignet, + gut geeignet, ○ ist geeignet, - eingeschränkt geeignet, – nicht geeignet

Tabelle 1: Auswertung der Kriterien

Projekt 1: Von Stiften und Mäusen

Ziel des ersten Software-Projektes war die schnelle Implementierung der Ausbildungssoftware *Von Stiften und Mäusen* [Cz99]. Im Vordergrund stand somit die direkte Implementierung eines gegebenen Konzeptes. Die Implementierung ist inklusive Test ohne Probleme innerhalb von drei Arbeitstagen durchgeführt worden, so dass diese Fallstudie als erfolgreiches Beispiel einer schnellen Implementierung angesehen werden kann.

Projekt 2: PyNassi

Ziel des zweiten Projekts war der Entwurf eines Prototypen und die anschließende vollständige Implementierung eines Editors für Struktogramme. Hierbei kamen typische Verfahrensweisen des Prototyp-Entwurfs zum Einsatz, etwa separater Dialogentwurf, experimentelle Implementierungen mit interaktiven Tests im Interpreter und Kommunikation mit einen „Testanwender". Innerhalb weniger Tage war eine vorzeigbare Benutzungsoberfläche und Grundfunktionalität des Programms realisiert. Neben syntaktischen und semantischen Merkmalen haben sich in diesem Projekt die unabhängige und interaktive Testmöglichkeit einzelner Module als vorteilhaft erwiesen.

Erhebung 1: Lesbarkeit von Programmiersprachen

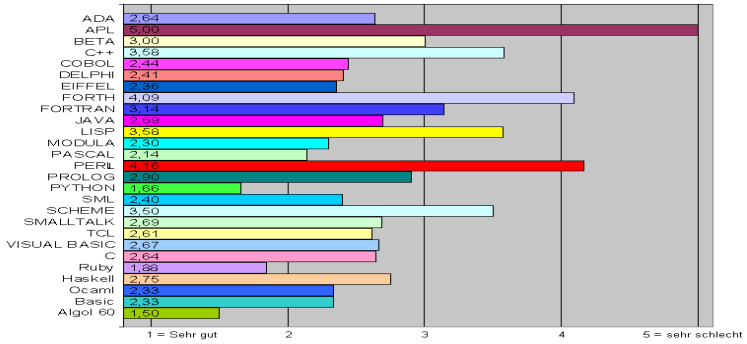

Abbildung 1: Ergebnisse der explorativen Befragung zur Lesbarkeit

Lesbarkeit ist ein Begriff, der sich nur schwierig definieren lässt. Jeder Mensch hat dabei eigene Vorstellungen, des Weiteren ist die Lesbarkeit vorhandenen Codes stark vom Programmierer anhängig. Trotzdem gibt es offenbar gemeinsame Faktoren, welche die allgemeine Lesbarkeit günstig beeinflussen. Um einen allgemeinen Trend erkennen zu können, wurden Programmierer und Lehrer in Universitäten, Newsgroups und Unternehmen zur Lesbarkeit von Programmiersprachen befragt. Die Bewertung erfolgte auf einer Skala von 1 (für sehr gut lesbar) bis 5. Ziel der Befragung ist die Exploration von Einstellungen bezüglich des unscharfen Begriffs der Lesbarkeit im Vergleich verschiedener Programmiersprachen. Demnach gelten Algol, Pascal, Python, Ruby und SML als besonders gut lesbar, während APL und Perl als schwierig zu lesen beurteilt wurden.

Erhebung 2: Erfahrungen mit der Programmiersprache Python

Die zweite Erhebung erfolgte unter Anwendern, welche Python überwiegend zur informatischen Bildung einsetzen. Es wurde nach Erfahrungen und Problemen beim Einsatz von Python gefragt. Festgestellte Probleme und Kritiken bezogen sich dabei kaum auf die Syntax und Semantik der Sprache. Vielmehr wurde festgestellt, dass ein Gros der Kritiken durch Festhalten an alten Vorstellungen begründet ist. Einige Beispiele:

1. Zitat: *If it's not popular, it can't be any good"; (read between the lines: "if it didn't come from Microsoft, it's no good"); or the even deadlier slag "it won't help our grads get a job". This leads to pushback from other teachers and some students who feel I'm teaching something totally irrelevant and "out in left field". Visual Basic, C, C++, Java and Turing are the alternatives proposed.*
2. Dynamische Typisierung erleichtert zwar den Einstieg in die Programmierung, führt aber bei Wechsel auf streng typisierte Sprachen wie Java oder C zu Problemen.
3. Schüler sind nicht auf die Problematik der Zeilentrennung und Blockbildung in anderen Sprachen vorbereitet.

Sicherlich sind diese Kritiken nachvollziehbar. Allerdings sollte im modernen Informatikunterricht das Lehren von Konzepten der Informatik in den Vordergrund gerückt werden. Die oft genannten Alternativen C++, Java, Visual Basic und Delphi haben derzeit als Programmiersprachen möglicherweise mehr Praxisrelevanz, behindern aber den Lernvorgang durch unnötig komplexe Syntax und Sprachelemente, die Schülern oft nur schwer vermittelt werden können (vgl. Java-Beispiel in Abschnitt 3).

5 Fazit

Es wurden Belege zur Stützung der ausgewiesenen Hypothesen zusammengetragen. Dabei wurde deutlich, dass es eine nichtleere Schnittmenge zwischen den Anforderungen der modernen Softwareentwicklung und Anforderungen, die für Programmiersprachen für den Informatikunterricht gelten, gibt.

Die speziell für den Informatikunterricht entwickelten Programmiersprachen (z. B. PASCAL-E, Logo) fanden zwar Eingang in die Schule, halten aber modernen Anforderungen an die Softwareentwicklung nicht mehr stand. Ihnen fehlen Anbindungen an Modulbibliotheken, die auch aktuellen Anforderungen entsprechen. Diese werdene notwendig, wenn gemäß dem in [Hu02] geforderten Primat der Arbeit in vernetzten Strukturen und der Einbindung in die konkrete Modellierung Rechnung getragen werden soll.

Eine zweite Argumentationslinie sollte ebenfalls Berücksichtigung finden. Immer wieder werden umfangreiche Softwareprodukte mit der Möglichkeit verbunden, wiederkehrende Arbeitsabläufe durch „Makros" zusammenzufassen und damit „auf Knopfdruck" ablaufen zu lassen. Dazu wurden eigene Makrosprachen entwickelt (z.B. Filter in Photoshop/Gimp, Macromedia: LINGO). Inzwischen zeichnet sich ab, dass durch mächtige „echte" objektorientierte Skriptsprachen hier ein grundlegendes Manko überwunden werden kann: die Kenntnis einer Skriptsprache reicht aus, um in verschiedenen Anwendungen Arbeitsabläufe programmgesteuert automatisieren zu können.

Werden Sprachen wie Flash oder HyperCard im Unterrichtskontext eingesetzt, so wird ein „halbes" Konzept vermittelt, da ein wichtiges Element (nämlich die Möglichkeit, eigene Klassen erstellen zu können) nicht Bestandteil dieser „Sprachen" ist. Python bietet als Skriptsprache hervorragende Möglichkeiten schneller prozeduraler und objektorientierter Programmierung, gleichzeitig lassen sich funktionale Ideen einsetzen. Es wird dabei sowohl professionellen Entwicklern wie auch fachdidaktischen Anforderungen gerecht. Diese Eigenschaften wurden durch die im Rahmen der Untersuchung durchgeführten Fallstudien und Erhebungen bestätigt.

Es fällt meist nicht schwer, Ausbilder und Schüler von den Möglichkeiten der Programmiersprache Python zu überzeugen. Immer wieder ist jedoch die Frage nach dem praktischen Nutzen zu hören: *Der Markt sucht derzeit Entwickler für XYZ, und XYZ ist weit verbreitet. Warum soll ich dann Python lehren (bzw. lernen)? Ich möchte auch auf den Markt vorbereiten.*

Wie diese Arbeit zeigt, eignet sich Python hervorragend zum Erlernen von Konzepten. Sind Konzepte einmal verstanden, fällt die Adaption auf andere Sprachen leicht. Programmiersprachen kommen und gehen, die grundlegenden Konzepte hingegen bleiben aber gleich. Daher sollte eine Programmiersprache gewählt werden, welche die Konzepte unterstützt, und nicht primär auf den aktuellen Markt geachtet werden. In wenigen Jahren können die Marktanforderungen sich komplett wandeln. Bereits heute werden aufgrund der gewachsenen Bedeutung der schnellen Softwareentwicklung Entwickler in diesen Bereichen gesucht, daher sollte man die zukünftige Bedeutung von Skriptsprachen wie Python nicht unterschätzen.

Es bleibt daher zu hoffen, dass Entscheidungsträger die Vorteile von Python gegenüber klassischen Sprachen erkennen und die Ergebnisse dieser Arbeit bei ihrer Auswahl berücksichtigen.

Literaturverzeichnis

[Bu02] Burley, Brent: Python Bibliotheca, Python resources for teachers and students. http://www.ibiblio.org/obp/pyBiblio/interactive.php – geprüft: 06/2002.

[Cz99] Czischke, Dick, Hildebrecht, Humbert, Ueding, Wallos: Von Stiften und Mäusen. Verlag für Schule und Weiterbildung, 1999.

[DEM01] Downey, Allen; Elkner, Jeff; Meyers, Chris*:* How to Think Like a Computer Scientist: Learning with Python. Green Tea Press, 2001.

[Hu02a] Humbert, Ludger: Informatik lehren – Zeitgemäße Ansätze zur Qualifikation von Schülern. http://ddi.cs.uni-dortmund.de:8000/ddi_bib/forschung/pub/Informatik-lehren.pdf – Universität Dortmund – geprüft: 07/2002.

[Hu02b] Humbert, Ludger: Welche Programmiersprache unterstützt meine Konzepte für den Informatikunterricht. http://ddi.cs.uni-dortmund.de/dortmund2002/nrw/humbert.html – Universität Dortmund – geprüft: 07/2002 .

[Li02] Linkweiler, Ingo: Python im Kontext des Softwareentwicklungsprozesses – Ergebnisse und Software zur Untersuchung. http://www.ingo-linkweiler.de/diplom – geprüft: 09/2002.

[LF02] Löwis, Martin von; Fischbeck, Nick: Python 2. 2. Aufl. München. Addison Wesley, 2002.

[Pr02] Prechelt, Lutz: An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl. http://www.ipd.uka.de/~prechelt/Biblio/jccpprtTR.pdf – geprüft: 03/2000.

[Py02] Python Language Website: http://www.python.org/ – geprüft: 07/2002.

[Sw02] Schwill, Andreas: Programmiersprachen im Informatikunterricht. Fachbereich Informatik, Universität Oldenburg. http://www.informatikdidaktik.de/Forschung/Schriften/PSimUnterrMatNatTag.pdf – geprüft: 05/2002.

[Wa02] Waclena, Keith: Programming Languages: What's wrong with them. The University of Chicago - http://www.lib.uchicago.edu/keith/crisis/ - geprüft: 07/2002.

[WP02] Wang, Lingyun; Pfeiffer, Phil: A Qualitative Analysis of the Usability of Perl, Python, and Tcl. East Tennessee State University - http://www.python10.com/p10-papers/14/index.htm - geprüft: 07/2002.