**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

begonnen: 03.09.2002 letzte Änderung: 9. Sep. 2002 il

**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?

## Hypothesen:

- Die Anforderungen an eine Programmiersprache zur schnellen Softwareentwicklung und zur informatischen Bildung stimmen ineinigen 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.

## 1 Einleitung

Aktuelle Verfahren der schnellen Softwareentwicklung, wie Rapid Prototyping, Extreme Programming (vgl. 1) 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.

## 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 und in vielen Bereichen identische Interessen erkennbar.

In der Untersuchung wird die Programmiersprache Python anhand der aufgestellten Kriterien untersucht und mit anderen Sprachen vergleichen. Python wurde als Referenzsprache ausgewählt, denn es bietet unter den derzeit eingesetzten Programmiersprachen zur schnellen Softwareentwicklung eine sehr leicht erlernbare Syntax und scheint auf den ersten Blick viele der gestellten Kriterien zu erfüllen.

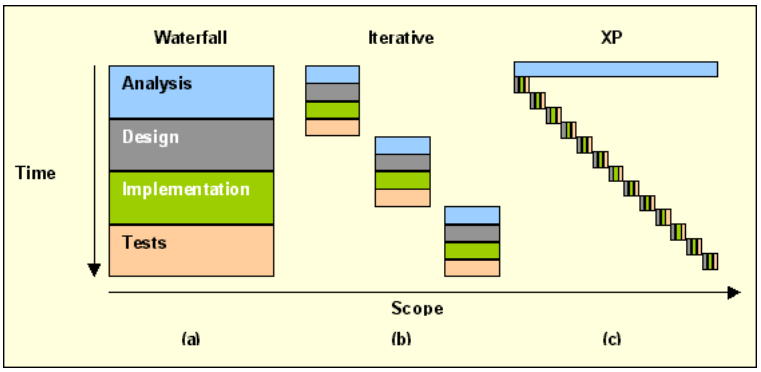


Abbildung 1: *Extreme Programming* [AC02]

Die Praxistauglichkeit von Python wurde anhand dreier Fallstudien in Form unterschiedlicher Softwareprojekte überprüft (vgl. Abschnitt 4). Des weiteren wurden Erhebungen zur Lesbarkeit ausgewählter Programmiersprachen durchgeführt.

## 3 Kriterien & Ergebnisse

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. Die Idee dieser Überlegung läßt sich durch folgendes Venn-Diagramm veranschaulichen (Abb.2).

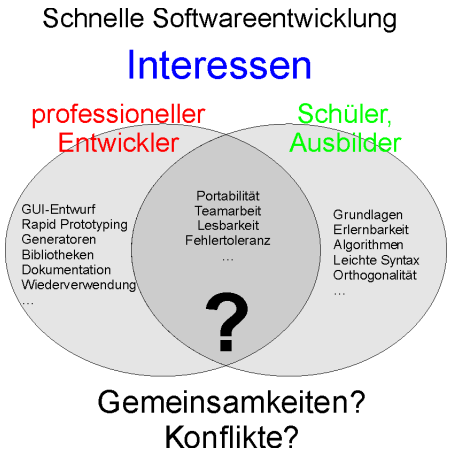


Abbildung 2: Anforderungen an eine Programmiersprache

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. Diese umfassen spiralförmig Möglichkeiten zu offenen Entwicklungen, wie beispielsweise:

- RUP (= Rational Unified Process)
- Extreme Programming
- Rapid Prototyping

Daher wird deutlich, dass die Auswahl von konkreten Kriterien immer den aktuellen Forschungsstand berücksichtigen sollte.

Ein Ausschnitt aus dem entwickelten Kriterienkatalog (Tab. 1) zeigt die Bandbreite der Fragestellungen, die mit der Auswahl unter der jeweiligen Anforderung korrespondieren. Ein Kriterienkatalog, wie der hier präsentierte, sollte nicht dazu verleiten, „feste" Positionen zu zementieren. Er dient vielmehr dazu, offenzulegen, auf welcher Entscheidungsgrundlage die folgenden Aussagen basieren.

Nachfolgend werden ausgewählte Kriterien vorgestellt und die Ergebnisse der Untersuchung bezogen auf die untersuchte Sprache Python zusammengefasst.

#### Portabilität

Erstellte Programme sollten in verschiedenen Umgebungen lauffähig sein, mit keinen oder nur minimalen Änderungen. Oft steht bei der Entwicklung des Prototypen die Zielplattform noch nicht fest. Desweiteren ist im Kontext vernetzter und interaktiver heterogener Informatiksysteme[1] Kompatibilität zwischen unterschiedlichen Plattformen allgemein wünschenswert. Auf allen Plattformen sollte uneingeschränkter, aber gleichzeitig kompatibler Zugriff auf das Dateisystem möglich sein. Ebenso wichtig ist eine Standardisierung der Sprache selbst. Python bietet hier alle nötigen Voraussetzungen: erstellte Programme lassen sich weitestgehend systemunabhängig einsetzen.

#### Schneller GUI-Entwurf

Der schnelle Entwurf grafischer Benutzungsoberflächen (GUI) ist eines der wichtigsten Einsatzgebiete des Rapid Prototyping. Gleichzeitig gewinnt der GUI-Entwurf 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 teilweise systemunabhängig nutzbar sind und 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 und schnell zu erkennen. Für Python sind unterschiedliche Werkzeuge verfügbar, die sowohl den Anforderungen im schulischen Einsatz als auch den Anforderungen professioneller Entwickler gerecht werden.

#### Einfache Syntax und Semantik

Eine einfache und 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: (Die „> > >" kennzeichnen den interaktiven Eingabeprompt des Interpreters)

```
>>> zahl = 3.141
>>> tupel = (7, 42)
>>> text = 'Hallo.'
```

[1] Informatiksystem wird hier als eine spezifische Zusammenstellung von Hardware, Software und Netzwerkverbindungen zur Lösung eines Anwendungsproblems verstanden.

```
>>> print text*2
Hallo.Hallo.
>>> komplex = complex(5, -2)
>>> hash = {1: 'abc', 2: 'def'}
>>> liste = [0,1,2,3,4,5]
>>> print liste[2:5]
[2, 3, 4]
>>> print liste[:3]
[0, 1, 2]
>>> liste[1:3]=[21, 'Hallo.' ,23]
>>> print liste
[0, 21, 'Hallo.', 23, 3, 4, 5]
>>> for i in liste:
>>>    tuwas(i)
>>> for i in liste:
...   if type(i)==type(1):
...     print i,"ist ist eine ganze Zahl."
...   else:
...     print i,"ist keine ganze Zahl."
```

**Automatische Speicherverwaltung**

Das programminterne Verwalten von Speicher durch den Entwickler sowie der Umgang mit Zeigern ist ein Relikt aus alten Sprachen wie C und Pascal. In modernen Sprachen braucht sich der Entwickler nicht mehr um das Reservieren und Freigeben von Speicher zu kümmern. Zur Darstellung von komplexen Strukturen wie Bäumen werden Klassen und Attribute mit Referenzen verwendet. Python bietet eine interne und vollautomatische Speicher- und Objektverwaltung, die automatisch erkennt, wann Objekte nicht mehr gebraucht werden, und sie dann aus dem Speicher entfernt.

**Dokumentation**

Alle Funktionen, Objekte und Module können in Python direkt mit einer Dokumentation versehen werden, die interaktiv im Interpreter aufgerufen werden kann. Somit kann insbesondere bei der Arbeit mit fremden Modulen häufig ein Blick in die Dokumentation durch die interaktive Inspektion ersetzt werden. 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. Möglichst reichhaltige und vielseitige Bibliotheken bestimmen häufig die Geschwindigkeit der Softwareentwicklung. Kann ein Entwickler nach dem Baukastenprinzip auf fertige Bibliotheken zugreifen, erspart er sich zeitintensive Eigenentwicklungen. Hilfreich sind möglichst vielseitige, polymorphe Bibliotheken sowie gute Möglichkeiten der Softwaredokumentation. Ein Modulsystem ist für eine heutige Programmiersprache unentbehrlich und sollte ein gewisses Maß an Komfort und Sicherheit bieten. Schnelle sowie kostengünstige Verfügbarkeit, einfache Installation und gute Recherchemöglichkeiten zum Finden von Bibliotheken zu einem gegebenen Problem sind wichtige Faktoren.

Insbesondere bei der Arbeit mit Prototypen sind häufige Tests einzelner Komponenten wie auch des gesamten Prototyps nötig. Wie im Ingenieurwesen, aus dem der Begriff Prototyp entlehnt wurde, sind auch in der Softwareentwicklung Fehlerbeseitigungen, Anpassungen und Verbesserungen durch direktes Arbeiten am Prototypen erforderlich. Die Möglichkeiten der Fehlererkennung und des interaktiven Eingreifens zur Wartung durch einen Entwickler sind dabei wichtige Anforderungen an die Entwicklungsumgebung. Häufig besteht ein Produkt aus einzelnen Modulen; auch deren korrekte Funktion sollte unabhängig getestet werden können. Des Weiteren ist schnelle Ausführung ein wichtiges Kriterium, welches Entwickler im Alltag erwarten. Nach kleinen Änderungen sollte sich ein Programm ohne lange Wartezeiten ausführen lassen.

Das Modulkonzept von Python unterscheidet nicht zwischen Modulen und Programmen. Jedes Modul kann gleichzeitig auch als Programm eingesetzt werden. Dabei kann direkt im Modul eine Testfunktion integriert

werden, die aufgerufen wird, wenn ein Modul eigenständig gestartet wurde. So kann jedes Modul zunächst separat getestet werden und anschließend in Kombination mit dem gesamten Projekt. Da Python eine Interpretersprache ist, kann die Fehlersuche interaktiv erfolgen. Zur Laufzeit können jederzeit Angaben zu allen Objekten erfragt werden, auch das interaktive Ändern einzelner Programmteile während der Laufzeit ist möglich. Ein Programm kann selbst Quelltext erzeugen oder einlesen und diesen ausführen lassen.

**Paradigmen der Softwareentwicklung**

Obwohl heute das Gros der Softwareentwicklung nach dem objektorientierten Paradigma erfolgt, sollten die anderen Paradigmen nicht vernächlässigt werden. Zahlreiche Klassen von Problemen können funktional oder prädikativ angemessener formuliert werden.

Python bietet die Möglichkeit, alle Paradigmen 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. Sie lassen sich aber bereits jetzt in prozedurale und objektorientierte Programmteile einbetten.

**Widerspruchsfrei und so einfach wie möglich zum Ziel**

Eine Programmiersprache sollte möglichst auf unnötige Anweisungen und Deklarationen verzichten können. C und C++ sind Beispiele für Sprachen, in denen ein erheblicher "Ballast" an zusätzlichen und oft überflüssig erscheinenden Deklarationen nötig ist. Um Schülern, die eine erste Modellierung implementieren sollen, die selbstständige Erarbeitung zu gestatten, müssen vor diesem Schritt die dazu unabdingbaren programmiersprachlichen Elemente bekannt sein. Gerade im Anfangsunterricht ist es nicht einfach, die Bedeutung der Schlüsselworte `void`, `public`, `static` zu fundamentieren. Ebenso sind bei rein prozeduraler Nutzung objektorientierter Sprachen oft Initialisierungen von Objekten nötig, die Schülern zunächst unverständlich sind. Das *Hello-World*-Beispiel in Java zeigt diese Problematik, in vielen anderen Sprachen sieht es ähnlich aus.

Java:

```
class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

Basic / Python:

```
print "Hello World"
```

Python gestattet einfache und interaktive 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. Darüber hinaus kann durch Zusicherungen dafür gesorgt werden, dass Gültigkeitsbereiche geprüft werden.

**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. Der zu modellierende Bereich sollte sich besonders eignen, um die für wichtig erachteten Konzepte zu erarbeiten, darzustellen und implementieren zu können. Zum Einsatz von Python in der informatischen Bildung steht dem Lehrer bereits ein gutes Angebot an Hilfsmitteln für anschaulichen Unterricht zur Verfügung. So sind etwa Umsetzungen von *Karel* bzw. *Niki der Roboter* oder der grafischen objektorientierten Lernsoftware *Von Stiften und Mäusen* und *PyCard* (angelehnt an *Hypercard*) verfügbar.

## Übersicht

| 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 Kriterienkatalog in Tabelle 1 zeigt einen Ausschnitt der überprüften Kritierien. Der vollständige Katalog wird bei Fertigstellung der gesamten Untersuchung Ende November auf den Internetseiten des *Fachgebietes Didaktik der Informatik der Universität Dortmund* veröffentlicht.

## 4 Fallstudien und Erhebungen

Die Auswertung existierender empirischer Untersuchungen, 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.

Im Hinblick auf die Laufzeit nimmt Python innerhalb der Skriptsprachen eine gute Position ein, ist aber im Vergleich zu kompilierten Sprachen in Benchmarks unterlegen. Bei realen, großen Programmen macht sich dies allerdings weniger bemerkbar als bei rein algorithmischen Benchmarks. Viele Bibliotheksfunktionen arbeiten intern sehr schnell, so dass Laufzeitverluste durch den Interpreter teilweise vernachlässigt werden können. Bei der Entwicklung von Anwendungen, die nicht auf optimale algorithmische Leistung angewiesen sind, macht sich der Unterschied daher nur unwesentlich bemerkbar.

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

#### Projekt 1: Von Stiften und Mäusen

Ziel des ersten Software-Projektes im Rahmen dieser Arbeit war die schnelle Implementierung der Ausbildungssoftware *Von Stiften und Mäusen*. Eine genaue Beschreibung der zu erstellenden Klassenbibliothek inklusive Anwendungsbeispielen als Pseudocode lag in Form der Dokumentation der Software vor [Cz99]. Im Vordergrund stand somit die direkte Implementierung eines gegebenen Konzeptes. Die Implementierung inklusive Test konnte in dieser Studie innerhalb von drei Arbeitstagen durchgeführt werden. Probleme sind dabei nicht

festgestellt 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 sowie experimentelle Implementierungen mit interaktiven Tests im Interpreter. Ebenso wurden Testanwender in die Implementierung einbezogen, so dass während des gesamten Entwicklungsprozesses eine Kommunikation zum „Endanwender" gegeben war.

PyNassi war innerhalb von drei Wochen fertiggestellt. Die Entwicklung in der ersten Phase kann als erfolgreiches Beispiel einer Softwareentwicklung per Rapid Prototyping gesehen werden: Innerhalb weniger Tage war eine vorzeigbare Benutzungsoberfläche und Grundfunktionalität des Programms realisiert.

Der Endausbau war eher eine Mischform des Evolutionären Prototyping und Extreme Programming: mehrere kurze Zyklen von Planung, Entwicklung und Test kombiniert mit Dialogen zum Anwender. Python unterstützte die einzelnen Entwicklungsschritte ausgezeichnet. Neben syntaktischen und semantischen Merkmalen haben sich in diesem Projekt die unabhängige und interaktive Testmöglichkeit einzelner Module als vorteilhaft erwiesen. Ebenso gut gelang die Integration des Debugger-Projekts. Ein besonders wichtiges Merkmal in diesem Projekt war die Möglichkeiten der dynamischen Quelltexterzeugung und Verarbeitung des Interpreters sowie die Introspektion: Ohne diese Eigenschaften wären viele Merkmale der Software nicht oder nur mit gewaltigem Mehraufwand realisierbar gewesen.

**Erhebung 1: Lesbarkeit von Programmiersprachen**

*Lesbarkeit* ist ein Begriff, der sich nur schwierig definieren lässt. Jeder Mensch hat eigene Vorstellungen, wann er einen Quelltext als gut oder weniger gut lesbar empfindet. Desweiteren ist die Lesbarkeit vorhandenen Codes stark vom Programmierer anhängig, denn in nahezu jeder Sprache lässt sich unlesbarer Code erstellen. Trotzdem gibt es offenbar gemeinsame Faktoren, welche die allgemeine Lesbarkeit günstig beeinflussen. Um einen allgemeinen Trend erkennen zu können, wurden ca. 30 Studenten und Programmierer in Universitäten, Newsgroups und Unternehmen zur Lesbarkeit von Programmiersprachen befragt. Die Bewertung erfolgte auf einer Skala von 1 (für sehr gut lesbar) bis 5 (sehr schlecht lesbar oder unlesbar).

Ziel der Befragung ist die Exploration von Einstellungen bezüglich des unscharfen Begriffs der Lesbarkeit im Vergleich verschiedener Programmiersprachen. Demnach gelten Algol, Modula, 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 sind. 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.

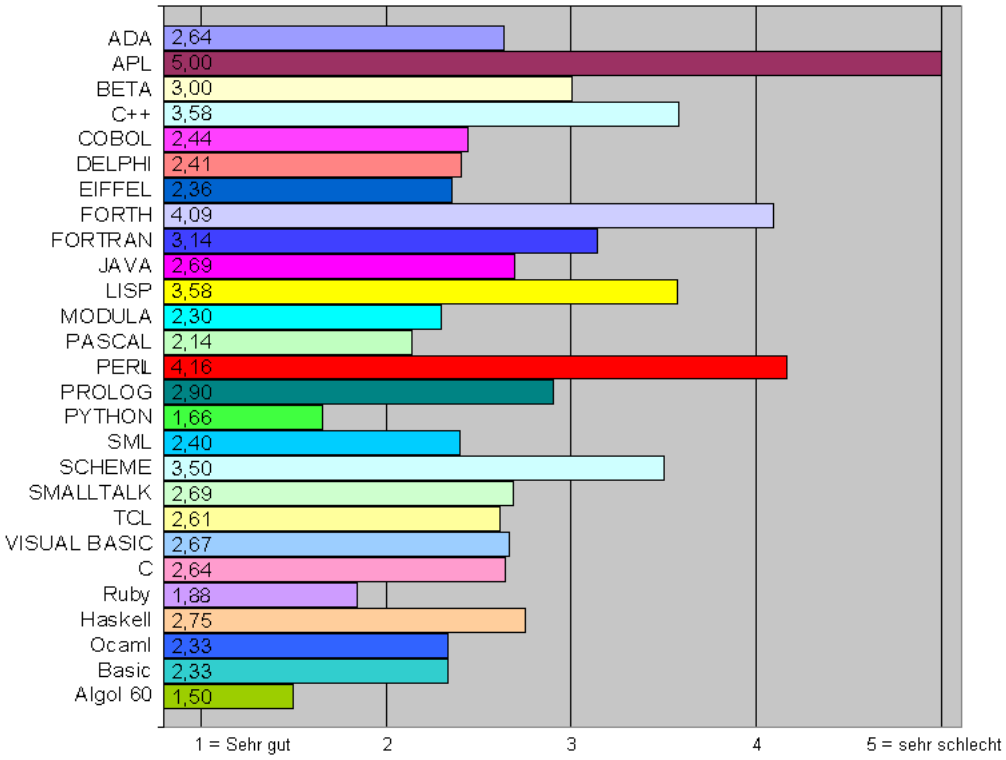


Abbildung 3: *Ergebnisse der explorativen Befragung zur Lesbarkeit*

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 Programmiersprache möglicherweise (noch) mehr Praxisrelevanz, behindern aber den Lernvorgang durch unnötig komplexe Syntax und Sprachelemente, die Schülern oft nur schwer vermittelt werden können (vgl. das 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 oder/und andere Fächer entwickelten Programmiersprachen (z. B. PASCAL-E als Untermenge der Programmiersprache PASCAL mit deutschen Schlüsselworten, Logo für den Mathematikunterricht – Seymour Papert) 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 (Beispiele: Filter in Photoshop/Gimp, Macromedia: LINGO). Inzwischen zeichnet sich ab, dass durch mächtige „echte" objektorientierte Skriptsprachen hier ein (aus Sicht der Anwenderin) grundlegendes Manko überwunden werden kann: die Kenntnis einer Skriptsprache reicht aus, um in verschiedenen Anwendungen Arbeitsabläufe programmgesteuert automatisieren zu können.

- Ist Flash objektorientiert?
- Ist HyperCard (oder MetaCard) objektorientiert?

Beide Fragen sind mit „Jein" zu beantworten. Werden diese Mittel 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 und prädikative 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 Fähigkeiten 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


[AC02] Acebal, Cesar F; Cueva, Lovelle: *A New Method of Software Development - eXtreme Programming;* `http://www.upgrade-cepis.org/issues/2002/2/up3-2Acebald.pdf` – geprüft: 05/2002

[Be00] Beck, Kent: *Extreme Programming, Das Manifest.* 2. Aufl. Addison Wesley, 09/2000, ISBN: 3827317096

[Be99] Beck, Kent: *Extreme Programming Explained. 2.* Aufl. Addison Wesley, 10/1999, ISBN: 0201616416

[Bo98] Böszörmenyi, Laszlo: *Why Java is not my favorite first-course language*, Software - Concepts & Tools (1998) 19: 141-145

[Br97] Brennwald, Daniel; Stamm, Christoph: *Gruppenunterricht zum Thema Paradigmen von Programmiersprachen.* Technische Hochschule Zürich. `http://www.educeth.ch/informatik/puzzles/paradigmen/docs/para.pdf` – geprüft: 07/2002

[BSW02] Baldassarre, Teresa; Succi, Giancarlo; Williams, Laurie: *Pair Programming, an Extreme Programming Practice.* North Carolina State University - `http://pairprogramming.com/` - geprüft 06/2002

[CU02] *The Rapid Prototyping Model.* Concordia University, ETEC 568/668, `http://www.arts.ualberta.ca/~tchao/rpwebsite/definition.html` – geprüft: 07/2002

[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

[Hi99] Himstedt, Tobias; Mätzel, Klaus: *Mit Python programmieren.* Heidelberg: Dpunkt Verlag, 1999, ISBN: 3920993853

[HS02] Humbert, Ludger; Schubert Sigrid: *Fachliche Orientierung des Informatikunterrichts in der Sek. II* . Report 771 Universität Dortmund - 02/2002

[Hw02] Hubwieser, Peter: `http://seciii.cs.uni-dortmund.de/web/ps-object.htm#hubwieser` . Dortmund - 07/2002

[Hu01] 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

[Hu02] Humbert Ludger: *Welche Programmiersprache unterstützt meine Konzepte für den Informatikunterricht.* Zusammenfassung: `http://ddi.cs.uni-dortmund.de/dortmund2002/nrw/humbert.html`, Folien und Hintergrundmaterial: `http://www.ham.nw.schule.de/pub/bscw.cgi/0/20866` – Universität Dortmund – geprüft: 07/2002

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

[JH02] Jähnichen, Stefan; Hermann, Stephan: *Was bitte bedeutet Objektorientierung.* Informatik Spektrum 08/2002, Berlin: Springer Verlag, TU-Berlin – geprüft: 08/2002

[LA99] Lutz, Mark; Ascher, David: *Learning Python.* OReilly UK 04/1999, ISBN 01565924649

[Li99] Liao, Luby: *Python as the Pascal of 2000s.* `http://www.sandiego.edu/~liao/python.pdf` – geprüft: 07/2002

[LF02] Löwis, Martin von; Fischbeck, Nick: *Python 2.* 2. Aufl. München. Addison Wesley - 2002, ISBN 382731691X

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

[Ra00] Raymond, Eric S.: *Why Python?* Linux-Journal, `http://www.linuxjournal.com/article.php?sid=3882` – geprüft: 05/2000

[SBB02] Schulz-Zander, Renate ; Brauer, Wilfried ; Burkert, Jürgen ; Heinrichs, U. ; Hilty, Lorenz M. ; Hölz, I. ; Keidel, K. ; Klages, Albrecht ; Koerber, Bernhard ; Meyer, M. ; Peschke, Rudolf ; Pflüger, Jörg ; Reineke, Vera ; Schubert, Sigrid: *Veränderte Sichtweisen für den Informatikunterricht GI Empfehlungen für das Fach Informatik in der Sekundarstufe II allgemeinbildender Schulen.* In: TROITZSCH, Klaus G. (Hrsg.): *Informatik als Schlüssel zur Qualifikation.* Berlin, Heidelberg : Springer, 1993 (Informatik aktuell). Gesellschaft für Informatik e. V., S. 205–218

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