Let’s Take Esoteric Programming Languages Seriously

Geschrieben von J. Singer & S. Draper

Silas A. Kraume

Heinrich-Heine-Universität Düsseldorf

Dynamische Programmiersprachen - Präsentation

Inhaltsverzeichnis

- Unpraktisch und unnötig Komplex
- Dennoch überraschend populär in der Informatik-Community

Kernfrage

Warum gibt es sie und welchen Nutzen haben sie?

Was ist eine esoterische Programmiersprache ("Esolang")?

Konventionelle Definition

Eine Sprache, die für Experimente mit seltsamen Ideen, schwierige Programmierung oder als Scherz gedacht ist.

- Oftmals absichtlich unbrauchbar
- Ungewöhnliche oder obskure Syntax/Semantik

Unterschied zu Mainstream-Sprachen: Nicht für praktische Anwendung gedacht.

Das Wesen von Esolangs

- **Low-Level Semantik:** Meist primitive arithmetische Operationen, begrenzter I/O
- **Keine Abstraktion:** Keine komplexen Datentypen, keine Modularität
- **Unkonventionelle Syntax:** Der Code sieht oft gar nicht aus wie Code

Wir teilen Esolangs in drei Kategorien (nicht exklusiv):

1. Natürliche Sprache (Natural Language)
2. Exotische Syntax (Quirky Syntax)
3. Exotisches Rechenmodell (Quirky Computational Model)

Kategorie 1: Natürliche Sprachen - Shakespeare

- Programme sehen aus wie Theaterstücke Von William Shakespeare
- Variablen sind Charaktere (Romeo, Julia)
- Wertezuweisung durch Dialoge

Beispiel

```
[Enter Hamlet and Romeo]
Hamlet:
You lying stupid fatherless big smelly half-witted coward!  You are as
stupid as the difference between a handsome rich brave hero and thyself!
Speak your mind!
...
```

Kategorie 1: Natürliche Sprachen - Chef

- Code sieht aus wie ein Kochrezept
- Variablen sind "Zutaten"
- Kochanweisungen sind Methodiken

Beispiel

```
Ingredients.
72 g haricot beans
101 eggs

Method.
Put eggs into the mixing bowl.
Put haricot beans into the mixing bowl.
Liquefy contents of the mixing bowl.
Pour contents of the mixing bowl into the baking dish.
Serves 1.
```

Kategorie 2: Exotische Syntax - Whitespace

- Syntax besteht nur aus Leerzeichen, Tabs und Zeilenumbrüchen
- Für das menschliche Auge unsichtbar
- Alle anderen Zeichen werden als Kommentare ignoriert

Beispiel

Kategorie 2: Exotische Syntax - Piet

- Code ist ein Bitmap-Bild (ähnlich Mondrian-Kunst)
- **Program Flow:** Ein Zeiger bewegt sich 2-dimensional über die Farbfläche
- Befehle entstehen beim Wechsel von Farben/Helligkeiten

Figure 1: Ein "Hello World" Programm in Piet

Kategorie 3: Exotische Rechenmodelle - Brainfuck

- Minimalistische Sprache, simuliert Turing-Maschine
- Nur 8 Befehle: `> < + - . , [ ]`
- Extrem begrenzter Speicher (typischerweise 30KB Array)

- **Jedes "interessante" Programm wird extrem lang und unleserlich**

Beispiel

```
++++++++[>++++[>++>+++>+++>+<<<-]>+>+>->>+[<]<-]>>.>---.+++++++..+++.>>.<-.<.+++.
------.--------.>>+.>++.
```

Das extremste Beispiel: Malbolge

- Benannt nach dem 8. Kreis der Hölle in Dantes Inferno
- **Designziel:** Die schwerstmögliche Sprache überhaupt
- Selbstmodifizierend: Befehle ändern sich nach Ausführung ("Verschlüsselung")
- Erstes Programm wurde erst Jahre später durch einen Such-Algorithmus generiert

Beispiel

```
(=BA#9"=<;:3y7x54-21q/p-,+*)"!h%B0/.
~P<
<:(8&
66#"!~}|{zyxwvu
gJ%
```

Warum sind Esolangs so beliebt?

Vier Hauptgründe für die Popularität:

1. Verspieltheit (Playfulness)
2. Nostalgie
3. Zugehörigkeitsgefühl (Cultic initiation)
4. Künstlerischer Ausdruck

> *"Deep down, we like confusing people who are stuffier and less mentally agile than we are [...]" [RayOO]*

- Anti-autoritäre Haltung: Verwirrung stiften mit "snazzy special effects"
- **Advent of Code:** Viele lösen Rätsel in Esolangs als zusätzliche Herausforderung

2. Nostalgie ("Das Goldene Zeitalter")

3 Der Reiz von Esolangs

- Sehnsucht nach der Zeit, als Programmieren noch "schwer" war
- **Das "Haarhemd" tragen (Peyton Jones):** Stolz darauf, unter schwierigen Bedingungen Code zu produzieren
  - "Technomasochismus"
- Gegenbewegung zu modernen IDEs, Autocomplete und KI-Hilfen

3. Zugehörigkeit & Kult

- Sprache als soziales Bindemittel (wie Klingonisch)
- **Initiationsritus:** Wer Brainfuck lesen kann, gehört zum "Inner Circle"
- Wissen wird oft mündlich oder in versteckten Wikis weitergegeben, nicht in Lehrbüchern

4. Künstlerischer Ausdruck

3 Der Reiz von Esolangs

- Code als Artefakt vs. ausführbare Entität
- Esolangs betonen das Artefakt (das Bild bei Piet, das Rezept bei Chef)
- **Knuth:** "Literate Programming" – Code wird für Menschen geschrieben
- Steganographie: Code versteckt in Bildern oder Texten

Konzept von "Hard Fun"

Lernen ist besonders effektiv, wenn es herausfordernd, aber lohnend ist

Variationstheorie

Dasselbe Konzept in verschiedenen Kontexten sehen fördert das Verständnis

- Wer Schleifen in Brainfuck versteht, hat das Konzept "Schleife" **wirklich** verstanden

1. Die Gefahr von "GOTO"

- Dijkstras Kritik an *GOTO* wird greifbar
- Beispiel Shakespeare: *Let us proceed to Scene N* erzeugt schnell Spaghetti-Code
- Als Anfänger erkennt man dies sofort als problematisch

2. Das Turing-Tarpit

- *"Beware of the Turing tar-pit in which everything is possible but nothing of interest is easy"* [Per82]
- Turing-Vollständigkeit allein reicht nicht für eine gute Sprache

- Esolangs entfernen den Komfort
- Man lernt das zugrunde liegende Rechenmodell (Stack, Speicherzellen, Pointer) direkt kennen
- Entmystifizierung dessen, was "unter der Haube" passiert

Empathie für Lehrende

4 Pädagogischer Wert

- Erfahrene Programmierer fühlen sich in Esolangs wieder wie Anfänger
- Hilft Dozenten, die Frustration von Erstsemestern wieder nachzuvollziehen

Wer entwickelt Esolangs?

- Oft Informatik-Studenten (z.B. INTERCAL in Princeton, Whitespace in Durham)
- Oft als Projekt, um Kompilerbau zu lernen
- Demografisch oft männlich dominiert
- Kultur die "Härte" und "Technomasochismus" glorifiziert

Warum entwickelt man eine Esolang?

1. **Comedy:** Parodie auf existierende Sprachen (INTERCAL vs. Fortran)
2. **Herausforderung:** Sich selbst extreme Beschränkungen auferlegen
3. **Kreativität:** Eine "eigene" Sprache haben ("A room of one’s own")

Die 5 Motive für Sprachdesign (Die 5 E's)

Warum werden Sprachen generell entwickelt?

1. **E**xpressivity (Ausdrucksstärke)
2. **E**fficiency (Effizienz)
3. **E**ducation (Bildung)
4. **E**conomy (Wirtschaftlichkeit)
5. **E**xploration (Forschung/Neugier)

Motiv 1: Expressivity

Ziel

Komplexe Konzepte kürzer ausdrücken

Beispiele: Lisp, Prolog
Esolangs: Das Gegenteil. Sie fügen Komplexität hinzu

Motiv 2: Efficiency

Ziel

Nähe zur Hardware, Geschwindigkeit

Beispiele: C, Rust, Zig
Esolangs: Ineffizient, da das Rechenmodell oft abstrakt und umständlich ist

Motiv 3: Education

Ziel

Leicht zu lernen

Beispiele: BASIC, Scratch, Logo
Esolangs: Explizit nicht für Anfänger
(INTERCAL Manual: "besser geeignet, Anfänger dazu zu bringen, den Beruf zu wechseln" [WL73])

Motiv 4: Economy

Ziel

finanzieller Gewinn, Marktdominanz

Beispiel: C# (als Rivale zu Java)
Esolangs: Kein kommerzieller Nutzen (außer ggf. Buchverkäufe)

Motiv 5: Exploration

Ziel

Intellektuelle Übung, Grenzen testen

Beispiel: Haskell (ursprünglich)

Hier leben die Esolangs. Sie sind reine Exploration.

KI und Esolangs - Codegenerierung

- LLMs (wie ChatGPT) sind gut in Python, aber schlecht in Esolangs
- **Grund:** Mangelnde Trainingsdaten (Es gibt kaum Brainfuck oder Piet Repositories)
- Experimente zeigen: Generierter Code für Piet oder Shakespeare enthält meist fundamentale Syntaxfehler

KI und Esolangs - Spracherfindung

- Kann eine KI eine neue Esolang erfinden?
- Versuche ergaben: Sprachen waren oft nicht Turing-vollständig oder plagiierten existierende Ideen (z.B. Musik-Sprachen)
- **Kreativität und "Witz" fehlen der KI hier noch**

Zusammenfassung

7 Fazit

- Esolangs sind Kunst, Kult und pädagogisches Werkzeug
- Sie stellen sich den Mainstream-Sprachen entgegen
- Sie lehren uns kritisches Denken über Design und Konzepte

Abschließendes Zitat

Alan Perlis (Epigramm 19)

"A language that doesn't affect the way you think about programming, is not worth knowing." [Per82]

Esolangs verändern definitiv, wie wir denken – also sind sie es wert, gekannt zu werden.

Vielen Dank!

Vielen Dank für Eure Aufmerksamkeit!

Quellenverzeichnis I

8 Quellen

[Per82] Alan J. Perlis, *Special feature: Epigrams on programming*, SIGPLAN Not. **17** (1982), no. 9, 7–13.

[Ray00] Eric Raymond, *Guest editorial: World domination*, Linux Journal (Jan. 2000).

[SD25] Jeremy Singer and Steve Draper, *Let's take esoteric programming languages seriously*, Proceedings of the 2025 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (New York, NY, USA), Onward! '25, Association for Computing Machinery, 2025, p. 213–226.

[WL73] Donald R. Woods and James M. Lyon, *The intercal programming language reference manual*, 1973.

Deep Dive: Shakespeare

- Titel frei wählbar, wird als Kommentar behandelt
- Variablen sind Charaktere und müssen in Shakespeare-Dramen vorkommen (152 insgesamt)
- Variablen müssen zunächst deklariert werden (deren Beschreibung ist verpflichtend aber wird ignoriert)

```
1 The Infamous Hello World Program
2
3 Romeo, a young man with a remarkable patience.
4 Juliet, a likewise young woman of remarkable grace.
5 Ophelia, a remarkable woman much in dispute with Hamlet.
6 Hamlet, the flatterer of Andersen Insulting A/S.
```

- Folgend werden Akte und Szenen definiert
- Diese dienen als Sprungziele für "Goto"-artige Befehle
- Zwischen Akten kann nicht gesprungen werden, nur zwischen Szenen innerhalb eines Akts

```
8 Act I: Hamlets insults and flattery.
9 Scene I: The insulting of Romeo.
```

- Es können immer nur zwei Variablen gleichzeitig angesprochen werden
- Variablen werden über "Enter" und "Exit" ein- und ausgeführt

```
11 [Enter Hamlet and Romeo]
```

- Variablen/Charaktere werden durch Dialoge manipuliert
- Die beiden aktiven Charaktere sprechen sich gegenseitig Werte zu
- Die Anzahl der Adjektive und das Nomen bestimmen den Wert
- Adjektive haben den Wert 2, Nomen haben den Wert 1 (nette Worte) oder -1 (beleidigende Worte)
- Die Werte werden multipliziert (big smelly coward = 2*2*-1 = -4)
- Im Folgenden entsteht der Wert 2*2*2*2*2*2*-1 = -64

```
13 Hamlet:
14 You lying stupid fatherless big smelly half-witted coward!
```

- Arithmetik durch "as" über Schlüsselwörter "difference", "sum", "product", "quotient"
- Im Folgenden entsteht 2*2*2*1 - thyself = 8 - (-64) = 72

```
15 You are as stupid as the difference between a handsome rich brave hero
       and thyself!
```

- Eingabe durch "Open your mind" (als Char) oder "Listen to your heart" (als Zahl)
- Ausgabe durch "Speak your mind" (als Char) oder "Open your heart" (als Zahl)
- Im Folgenden wird das ASCII-Zeichen 72 = ’H’ ausgegeben

```
16 Speak your mind!
```

- Aussagenlogik durch "X as good as Y" (X=Y), "X better than Y" (X>Y), "X worse than Y" (X<Y)
- Sprünge durch "Let us proceed to Scene N" oder "Let us return to Scene N"
- bedingte Sprünge durch "If so" und "If not"

```
Juliet:

 Am I better than you?

Hamlet:

 If so, let us proceed to scene III.
```

- Rezeptname und -beschreibung frei wählbar, werden als Kommentare behandelt
- Variablen sind "Zutaten" und müssen zunächst deklariert werden (mit Startwert)
- Es gibt trockene (kg, g, lb) und flüssige (l, ml, cup) Zutaten
- trockene Zutaten werden als Ganzzahlen, Flüssige als ASCII Characters gespeichert

```
1 Factorial and Chips.
2 This recipe calculates the factorial of a number.
3
4 Ingredients.
5 1000 g fish
6 1 kg Chips
7 500g breading
```

- Es folgt die "Method." Sektion mit Kochanweisungen / Programminstruktionen
- "Take X from refrigerator.": X wird ein Wert durch Eingabe zugewiesen (User Input)

```
9  Method.
10 Take fish from refrigerator.
```

- Schüsseln dienen als Stapelspeicher / Stack (LIFO)
- "Put X into the mixing bowl.": Wert von X auf den Stack legen
- "Add X to mixing bowl.": Wert von X zum obersten Stack-Wert addieren
- "Combine X into the mixing bowl.": Wert von X zum obersten Stack-Wert multiplizieren

```
11 Put chips into the mixing bowl.
```

- Schleifen durchlaufen solange Variable > 0 ist (solange bis X = 0)
- "_ the X." beginnt eine Schleife
- "_ X until _." endet eine Schleife

```
12 Bread the fish.
13     Combine fish into the mixin bowl.
14 Coat fish until breaded.
```

- "Pour contents of the mixing bowl into the baking dish.": Wert(e) vom Stack nehmen und für Ausgabe vorbereiten
- "Serves X.": Ausgabe der letzten X Werte (als Zahl oder Char, je nach Zutatentyp)

```
15 Pour contents of the mixing bowl into the baking dish
16
17 Serves 1.
```

- "Stir for X minutes": Den obersten Stack-Wert um X nach unten verschieben

```
Input int to ascii converter.

Ingredients.
1 l water

Method.
Carbonate the water.
    Take water from refrigerator.
    Put water into mixing bowl.
    Stir for 999 minutes.
Filter until carbonated.
Pour contents of the mixing bowl into the baking dish.

Serves 1.
```

- "Mix the mixing bowl well": Stack random permutieren
- "Clean the mixing bowl": Stack leeren
- "Liquify contents of the mixing bowl": Alle Werte im Stack zu ASCII-Chars konvertieren

Deep Dive: Whitespace

- 3 Lexeme: Space (S), Tab (T), Newline (N), alles andere sind Kommentare
- Jede Instruktion hat einen Instruction Modification Parameter (IMP)
- Nach dem IMP folgt der Befehl selbst
- Nach dem Befehl folgt ggf. ein Parameter

IMP	Bereich
S	Stack Manipulation
TS	Arithmetik
TT	Heap-Zugriff
N	Flow Control
TN	I/O

Table 1: Whitespace IMPs

Deep Dive: Whitespace

- Zahl Parameter werden als binäre Zahlen kodiert (Space=0, Tab=1)
- Zahlen beginnen mit Vorzeichen (Space=+, Tab=-) und enden mit Newline
- Dadurch sind Stack Manipulationen möglich

Befehl	Parameter	Verhalten
S	Zahl	Push Zahl auf den Stack
NS	-	Dupliziere TOS (Top of Stack)
NT	-	Tausche TOS und NOS (Top und Next on Stack)
NN	-	Entferne TOS
TS	Zahl	Kopiere die n-te Zahl vom Stack
TN	Zahl	Schiebe die obersten n Zahlen vom Stack

Table 2: Whitespace Stack Manipulations Befehle

- Arithmetik verwendet die obersten zwei Stack-Werte
- Ergebnis von TOS op NOS (z.B. *TOS* ÷ *NOS*) wird wieder auf den Stack gelegt

Befehl	Parameter	Verhalten
SS	-	Addition
ST	-	Subtraktion
SN	-	Multiplikation
TS	-	Division (Int)
TT	-	Modulo

Table 3: Whitespace Arithmetik Befehle

- Heap verwendet Adressierung über Stack-Werte
- TOS = Adresse(, NOS = Wert)

Befehl	Parameter	Verhalten
S	-	Speichern
T	-	Auslesen

Table 4: Whitespace Heap Befehle

Deep Dive: Whitespace

- Flow Control steuert den Programmfluss
- Labels sind beliebige Kombinationen von Space und Tab, enden mit Newline

Befehl	Parameter	Verhalten
SS	Label	Erstelelt eine Sprungmarke
ST	Label	Ruft Subroutine auf
SN	Label	Sprung zu Label (ohne Bedingung)
TS	Label	Sprung zu Label (wenn TOS = 0)
TT	Label	Sprung zu Label (wenn TOS < 0)
TN	-	Beende Subroutine, übergib Kontrolle auf Aufrufenden
NN	-	Beendet ds Programm

Table 5: Whitespace Flow Control Befehle

Deep Dive: Whitespace

- Die Eingabe erfolgt über die Addressierung im Heap (TOS = Adresse)
- Die Ausgabe verwendet den obersten Stack-Wert
- TOS wird immer entfernt nach Ein-/Ausgabe

Befehl	Parameter	Verhalten
TS	-	Speichert Eingabe als ASCII an Heap Adresse (TOS)
TT	-	Speichert Eingabe als Zahl an Heap Adresse (TOS)
SS	-	Ausgabe von TOS als ASCII
ST	-	Ausgabe von TOS als Zahl

Table 6: Whitespace I/O Befehle