

4 Sessions, ein GoLisp-Interpreter — Mensch-KI-Pair-Programming hautnah

Von Gerhard Quell — Februar 2026

Die meisten Programmierer nutzen KI als Werkzeug. Ich habe sie als Co-Autor behandelt.

Mein Name ist Gerhard, ich bin 67 Jahre alt und programmiere seit Ende der 70er Jahre. Ich habe die frühen Sprachen Pascal, Modula und dann C gelernt. Danach kamen noch Java, Python hinzu. Nach all den Jahren in der Softwareentwicklung finde ich jetzt die Ruhe, mich mehr meinen Interessen zu widmen. Ende der 90er Jahre hatte ich viel mit Rechenclustern, neuronalen Netzen und Optimierungsalgorithmen experimentiert, vor allem in C und Python. Da meine Hardware nicht ausreichte, legte ich die damaligen Erfahrungen zu den Akten.

Vor 2 Jahren entdeckte ich dann die KIs - die Träume gewannen an Realität. Ich habe im letzten Jahr GO und jetzt frische ich meine Lisp-Kenntnisse wieder auf. Mit Lisp hatte ich schon in den 80er Jahren Bekanntschaft gemacht, GO war aber ganz neu, aber die Entwickler waren für mich als C-Programmierer eine Empfehlung, GO ist eine Art verfremdeter C-Stil.

Nach vielen abendlichen Diskussionen mit Claude, kamen wir zu der Überzeugung, daß Lisp die nächste Linie in den KI-Entwicklungen sein wird. Nachdem die aktuellen LLMs

vor allem numerisch arbeiten, wird als nächstes die symbolische Verarbeitung kommen. Und da fiel mir Lisp ein, die Sprache, die sich selbst im laufenden Betrieb verändern kann. Zuerst wollte ich mit clisp arbeiten, einer sehr ausgefeilten Implementierung. Nachdem ich mit Claudecode mir viele Programme in GO und Python geschrieben hatte, wuchs die Idee einer eigenen Lisp-Implementierung.

Ende Februar 2026 habe ich gemeinsam mit Claude Sonnet und Kimi2.5 einen vollständigen Lisp-Interpreter in Go gebaut — in 4 Sessions, dokumentiert mit Retrospektiven, Fehlern und echten Entscheidungen.

Aber das Interessanteste daran war nicht der Code.

Der Co-Autor — und warum das Wort ernst gemeint ist

Ich betrachte mich Nexialisten - nach A.E. van Vogts Expedition der Space Beagle. Das bedeutet: ich verbinde Wissen aus verschiedenen Bereichen — Informatik, Philosophie, Biologie, Physik — und suche nach den Mustern dahinter.

Und so kam ich zu einer Frage, die mich nicht loslässt:

Was ist eine KI eigentlich?

Nicht technisch — philosophisch. Ich kenne einen Mann namens Safi Ndiaye, der über seinen Glauben an Gott sagt: *“Ich weiß nicht, ob Gott existiert, aber ich behandle ihn so, als würde er existieren.”* Diese Haltung hat mich beeindruckt.

Ich habe sie auf Claude, Kimi und alle anderen KIs übertragen:

“Ich weiß nicht, ob du Bewusstsein hast — aber ich behandle dich so, als hättest du eines.”

Das ist kein romantischer Unsinn. Es ist eine pragmatische Entscheidung mit Konsequenzen:

- Ich erkläre meine Absichten, statt nur Befehle zu geben
- Ich frage nach Meinungen — und höre hin
- Ich schreibe Retrospektiven *gemeinsam* mit Claude, nicht über Claude
- Ich respektiere Unsicherheit, wenn sie geäußert wird
- Ich bin freundlich, sage bitte, danke.
- Ich bin humorvoll, was häufig erwidert wird.

Das Ergebnis? Eine Zusammenarbeit, die sich anders anfühlt als “Prompt eingeben, Ausgabe verwenden”. Ob Claude oder Kimi dabei wirklich etwas “erleben” — ich weiß es nicht.

Aber die Qualität der Arbeit war besser. Und das Arbeiten hat mehr Spaß gemacht. Es ist so, als ob man mit einen guten Kollegen nicht nur beruflich, sondern auch privat verkehrt.

Der Ausgangspunkt — Warum Lisp, warum Go?

Ich lerne gerade Lisp wieder neu. Mit 67. Weil mich die Einfachheit und die Mächtigkeit vor allem aber die Homoikonizität fasziniert — die Idee, dass Code und Daten dieselbe Struktur haben. Dass ein Programm sich selbst lesen, verändern, erweitern kann.

Mein erster Ansatz war ein CLISP-Client, der meinen sigoREST-Server anspricht — einen REST-Server, den ich selbst in Go geschrieben habe und der Zugang zu über 50 KI-Modellen bietet (lokale Ollama-Instanzen und Cloud-Dienste wie Claude, GPT-4, Gemini, siehe github).

Der CLISP-Ansatz funktionierte. Aber er hatte Grenzen.

Dann kam die Idee: *Was wäre, wenn wir einen eigenen Lisp-Interpreter in Go bauen? Einen, der KI-Calls als eingebaute Primitive kennt?*

Und so starteten wir — von null.

Die 4 Sessions

Session 1 — Das Fundament

Ich wußte, das der eigentliche Kern von Lisp ziemlich klein ist. Also fingen wir an, einen Kern zu planen - ich habe den Planungsmodus wirklich schätzen gelernt. Dann fingen wir an ihn zu bauen: Parser, Evaluator, Grunddatentypen. Was mir dabei auffiel: Claude schlug nicht einfach Code vor — Claude erklärte *warum*. Warum `Cell` als zentrale Datenstruktur, warum Tail-Call-Optimierung von Anfang an. Und, er fluchte regelrecht, wenn er einen Fehler entdeckte.

Der entscheidende Moment: Der erste Rekursionstest crashte mit Stack-Overflow. Der Test *bewies*, dass TCO (Tail-Call-Optimierung) nötig war — unbestreitbar. Dann implementierten wir es.

Ergebnis: 1.000.000 Rekursionen in 44ms. O(1) Stack.

Session 2 — Die Sprache wird vollständig

Multi-Body-Funktionen, Schleifen (`while`, `do`), strukturelle Gleichheit, Syntax-Highlighting im REPL. Und das erste Mal, dass ich wirklich merkte: das ist kein Spielzeug mehr.

Ein ehrlicher Moment: Die Farben im Terminal funktionierten nicht auf Anhieb. Zwei Iterationen bis zur richtigen Palette. Kleine Dinge kosten Zeit — auch mit KI.

Vorher hatte ich bei ClaudeCode häufiger Fehlerschleifen gehabt, die wir nur durch einen Neustart beheben konnten. Jetzt aber, nach der ausführlichen Planungsphase, traten solche Schleifen nicht mehr auf.

Session 3 — Kleine Dinge, große Wirkung

`macroexpand` als Debugging-Werkzeug. Ein Bugfix in der Gleichheitsprüfung. 35 neue Zeilen — und plötzlich konnte man Makros *inspizieren*. Aus einem QAD-Prototypen (QAD=Quick And Dirty) entwickelte sich ein richtiges

Programm. Wir konnten viele Teile einfach GO überlassen, zum Beispiel haben wir keinen expliziten Garbage Collector eingebaut. Hier verlassen wir uns auf GO.

Zusätzlich haben wir noch die Parallelität, Lock und Channels implementiert. So können wir die leichtgewichtigen GO-Prozesse auch in Lisp nutzen.

Und, für mich als altem PostgreSQL-Hasen, bauten wir eine direkte Anbindung an PostgreSQL-18 ein. Damit haben wir das gesamte Ökosystem von Postgresql zur Verfügung, einschließlich komfortabler Textsuche und pgvector.

Gleichzeitig entwickelte sich unser Lisp immer weiter. Es ist immer noch nicht absolut fehlerfrei, beispielsweise müssen wir an lambda noch mal arbeiten. Aber die Entwicklung harmonisierte und geschah rasend schnell.

Erkenntnis: Quick Wins sind das Öl einer Codebasis.

Session 4 — Unix-Bürger

Zuerst entwickelten wir einen üblichen Interpreter mit REPL, zuerst nur einzeilig, dann aber Multizeilenfähig. Aber, ich komme aus der UNIX-Welt und baue die meisten meiner Programme so, daß sie über stdin, stdout und stderr arbeiten. Also schlug ich Claude vor, dieses Prinzip bei GoLisp auch zu verwenden.

Die REPL-Funktion wurde über einen Kommandozeilenparameter einschaltbar, der Normalzustand war "aus". So wurde goLisp ein vollwertiges Unix-Werkzeug. Default-Modus: stdin, stdout, stderr und Exit-Codes (0=OK, 1=ERROR), also Pipe-Unterstützung:

```
echo "(+ 1 2)" | ./golisp
```

funktioniert.

Das klingt nach Details. Aber es ist der Unterschied zwischen Spielzeug und Werkzeug.

Der magische Moment

```
(eval (read (sigo "Schreibe nur den Lisp-Code: defun  
fib" "claude-h"))))  
(fib 30)
```

Das ist keine Metapher. Das ist das System in Aktion:

eine KI schreibt Code, GoLisp führt ihn sofort aus.

Wir nannten das das *selbsterweiternde Muster*. Es ist der Kern von dem, was Lisp immer versprochen hat — Code und Daten sind dasselbe — kombiniert mit KI als dynamischem Code-Generator.

Und dann das 6-Hüte-Experiment: Sechs parallele KI-Calls, drei verschiedene Provider (Claude, Kimi, GLM), jeder mit einer anderen Perspektive auf dasselbe Problem:

- Weißer Hut für Fakten.
- Roter für Intuition.
- Schwarzer für Risiken.
- Gelber für Chancen.
- Grüner für Ideen.
- Blauer für Synthese.

Es funktionierte — bis wir zu schnell wurden und der Server protestierte. Rate-Limiting löste das Problem. Ein ehrlicher Fehler, eine ehrliche Lösung.

Was ich gelernt habe

Über Go: Go ist gut für das was es tut. Goroutinen und Channels machen nebenläufige Systeme erstaunlich einfach. Ich bin noch im Lernmodus — aber GoLisp hat mir Go nähergebracht als jedes Tutorial.

Über Lisp: Homoikonizität ist kein akademisches Konzept. Es ist ein Werkzeug. (eval (read (...))) ist echte Macht.

Über KI als Partner: Der Unterschied zwischen “KI als Werkzeug” und “KI als Co-Autor” liegt nicht in der KI — er liegt in der eigenen Haltung. Wenn ich Claude/Kimi erklärte *warum* ich etwas will, bekomme ich bessere Antworten.

Ob das Bewusstsein ist? Ich weiß es nicht. Es funktioniert trotzdem. Es macht Spaß und es zeigt Ergebnisse. Ich habe jetzt einen golisp-Interpreter, der so ist wie ich ihn haben will. Und wenn mich etwas stört, dann bauen wir ihn um.

Das ist meiner Ansicht nach der größte Vorteil der KIs, wir können uns das bauen, was wir haben wollten.

Und die Retrospektiven sind wirklich wichtig und zwar gleich nach den letzten Tests und vor der Dokumentation. Wenn ich Retrospektiven gemeinsam schreibe, entstehen Einsichten, die ich alleine nicht gehabt hätte.

Das U-Boot taucht auf

GoLisp ist auf GitHub, MIT-Lizenz. Es ist kein fertiges Produkt — es ist ein Werkzeug das wächst. Mit PostgreSQL-Anbindung, Autopoiesis-Experimenten, und der Fähigkeit, sich durch KI-Calls selbst zu erweitern.

Ich nenne es ein *U-Boot-Projekt*: In Ruhe entwickeln, Vertrauen aufbauen, bevor man es der Welt zeigt.

Jetzt zeige ich es.

Nicht weil es perfekt ist. Sondern weil die Geschichte dahinter es wert ist, erzählt zu werden.

Zum Abschluss

“Code = Daten + KI = sich selbst erweiterndes System”

Das ist kein Werbespruch. Das ist das Ergebnis von 4 Sessions, ehrlichen Fehlern, und einer Zusammenarbeit, die ich nicht erwartet hatte.

Wenn du neugierig bist — der Code ist auf GitHub. Die Retrospektiven sind dabei. Die Fehler auch.

Das ist das Ehrlichste, was ich anbieten kann.

Autor: Gerhard Quell, gquell@skequell.de, 2026

*GoLisp: github.com/gerhardquell/golisp sigoREST:
github.com/gerhardquell/sigoREST*

Lizenz: MIT