Funktionale Programmierung 1

Für die Einsendeaufgabe „Funktionale Programmierung 1“ wurde das Paper „Out of the Tar Pit“ von Ben Moseley und Peter Marks aus dem Jahr 2006, sowie der Artikel „The Nature of Lisp“ aus dem Jahr 2006 gelesen. Nachfolgend teile ich meine Gedanken und Feststellungen zu den beiden Literaturen.

Out of the Tar Pit

In diesem Paper gehen die Autoren Moseley und Marks auf die Komplexität großer Softwaresysteme ein und betrachten Lösungsansätze zur Verminderung dieser „ungewollten“ Komplexitäten. Die Komplexität der heutigen Softwareentwicklung wird direkt zu Beginn des Papers als größtes Problem erklärt. Zur Hauptaufgabe haben sich die Autoren gemacht den Hauptgrund der Komplexität zu finden und explizit zu benennen, um die Verständlichkeit der mächtigen Anwendungen und Systeme zu wahren. Ich persönlich finde es wichtig, komplexe Sachverhalte in einfachen Worten zu erklären, egal ob es hierbei um wissenschaftliche Arbeiten, politische Themen oder Software geht. Menschen haben unterschiedliche Ansichten, Wissensstände und Angewohnheiten – Und das ist auch gut so! Daher ist es für mich notwendig, komplexe Sachverhalte so einfach wie nur möglich darzustellen, um möglichst viele Leute „abholen“ zu können. Weniger fortgeschrittene Programmierer können ebenso vielversprechende Lösungsansätze liefern wie technisch begabtere Softwareentwickler, wichtig ist jedoch das der Zugang zur Software für alle Interessierten gegeben ist. Transparenz und Einfachheit helfen hierbei enorm.

Die Hauptgründe für Komplexität liegen nach Moseley und Marks in den verschiedenen Zuständen eines Systems, der Abfolge nach bestimmten Schemen und Regeln, sowie der bloßen Menge an geschriebenem Code. Diese Gründe sehe ich ebenfalls als größtes Problem der modernen Softwareentwicklung. Durch die vielen Möglichkeiten, welche moderne Technologien eröffnen, werden immer mächtigere Softwareprogramme „zusammengeschustert“. Dieser Prozess muss bereits in den frühen Phasen der Entwicklung genaustens kontrolliert und angepasst werden. Undurchdachte Programmabschnitte können durch auftretende Redundanz und Inkonsistenz zu komplexen Gebilden heranwachsen. Neben der enormen (unnötigen) Menge an geschriebenem Code führen diese Zusammensetzungen aus Redundanz und Inkonsistenz zu gravierenden Qualitätsverlusten des Produkts. Wie die Autoren beschreiben resultierenden „einfache“ Lösungen, wie beispielsweise ein Neustart, aus einer komplexen Verschachtelung von technischen Abfolgen, welche meiner Meinung nach zu unvorhersehbaren Zuständen des Systems führen können.

The Nature of Lisp

Der Artikel beschäftigt sich mit Lisp und ist von persönlichen Eindrücken und Erfahrungen des Autors/der Autorin geprägt. Bereits zu Beginn wird die hohe Einstiegshürde von Lisp kritisiert, da die Sprache sich grundlegend in ihrer Form und Syntax von den gängigen Programmiersprachen unterscheidet. Ein Großteil des Artikels beschäftigt sich daher mit der Herleitung von Lisp über XML, Ant. Diese Herleitung ist meiner Meinung nach sehr gelungen, da auf diese Weise klar wird, dass sich Lisp nicht direkt mit herkömmlichen Programmiersprachen wie C und Java vergleichen lässt. Betrachtet man Lisp als eigenständiges „Tool“, lassen sich die Vorteile unabhängig von gebrochenen Konventionen der gängigen Sprachen erkennen. In meinen Augen der Kern des „Problems“.

Ich habe bisher keine Berührungspunkte zu Lisp gehabt und folgere meine Schlüsse daher lediglich aus diesem einen Artikel. So wie ich es verstehe liegt der Unterschied in Lisp in der Freiheit, welche die Sprache im Vergleich zu Java oder C bietet. Beispielsweise ist Lisp durch die Symbolfreiheit weniger limitiert und behält dabei trotzdem die Möglichkeit mit Funktionen zu arbeiten. Die festgeschriebene, wenn auch einfacher verständlichere, Syntax der gängigen Programmiersprachen hingegen schränkt die Variationen eher ein. Lisp stellt viele kleine Bausteine zur Verfügung und bedient sich an ähnlichen technisch-gestalterischen Methoden, wie Domain Specific Languages. Lisp erhöht die Flexibilität durch eine vermeintlich komplexere Syntax, welche jedoch den unumstößlichen Konventionen der „einfacheren“ Programmiersprachen geschuldet ist.