Seminararbeit im Seminar Trends der Softwaretechnik
Sommersemester 2025

Vor- und Nachteile verschiedener Typsysteme im Kontext von Web-Anwendungen

Nils Derenthal (7217566)
nils.derenthal002@stud.fh-dortmund.de
Informatik Dual

Inhaltsverzeichnis

1 Einleitung

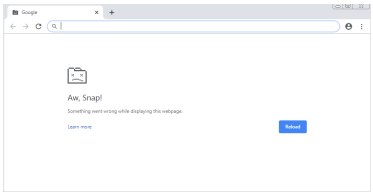
Die Verlässlichkeit von Web-Anwendungen hat in der heutigen Zeit eine so hohe Relevanz wie noch nie zuvor und die Wirtschaftlichkeit von Dienstleistenden IT-Unternehmen hängt stark von der Qualität ihrer Software ab. Ein wichtiger Aspekt in Bezug auf die Verlässlichkeit von Software sind Typsysteme, mit ihrem Versprechen eine Vielzahl von Fehlern auszuschließen, bevor die Software überhaupt läuft. Die folgende Arbeit richtet sich an Softwarearchitekten sowie Technologieentscheider und soll bei der Entscheidungsfindung von Backendtechnologien von Webanwendungen helfen, indem das Konzept der Typsysteme sowie deren Eigenschaften erläutert werden. Hierbei wird durch anschauliche Beispiele und realen Szenarios aus der Welt der Webentwicklung eine Praxisnahe Gegenüberstellung verschiedener Paradigmen vorgenommen.

1.1 Grundlagen

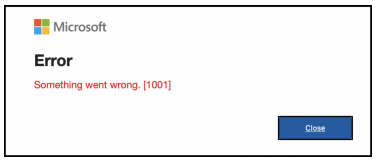
Für die Abeit wird im folgenden angenommen, dass Leser*innen ein Basiswissen in der Webentwicklung besitzen, auch wenn keine Details diesbezüglich notwendig sind. Es ist desweiteren hilfreich, wenn Kenntnisse über eine Sprache mit typisierung vorhanden sind (Beispielsweise Klassen und Objekttypen in Java, Types in TypeScript oder Enums in Rust). Die Typentheorie muss nicht bekannt sein und alles was in diesem Bezug relevant ist wird im Laufe der Arbeit erläutert.

1.2 Problemstellung

Fehler in der Softwareentwicklung sind auch heutzutage noch sehr präsent. Ob in einem Studienprojekt, der Website einer Behörde, oder dem Betriebssystem eines Millardenkonzerns: Im Alltag trifft man nicht selten auf solche Fehler.



(a) Ein Crash im Chrome-Browser



(b) Ein Fehler während des Microsoft Logins

Abbildung 1: Fehler in alltäglichen Anwendungen

Kaum eine Software is frei von Fehlern, welche je nach schwere ein großes Betriebsrisiko darstellen, weswegen die Vermeidung von diesen eine hohe Bedeutung in der Entwicklung hat. Ein Ansatz, welcher in den letzen Jahren zunehmend an Bedeutung gewonnen hat sind stärkere Typsysteme, welche vermeintlich dafür sorgen, dass Fehler bereits in der Entwicklung abgefangen werden

können. Beispiele für jüngere Entwicklungen in diesem Bereich sind Technologien wie TypeScript, welches versucht JavaScript zu typisieren, oder Rust, als eine Sprache die einen Fokus auf ein ausdrucksstarkes Typsystem legt.

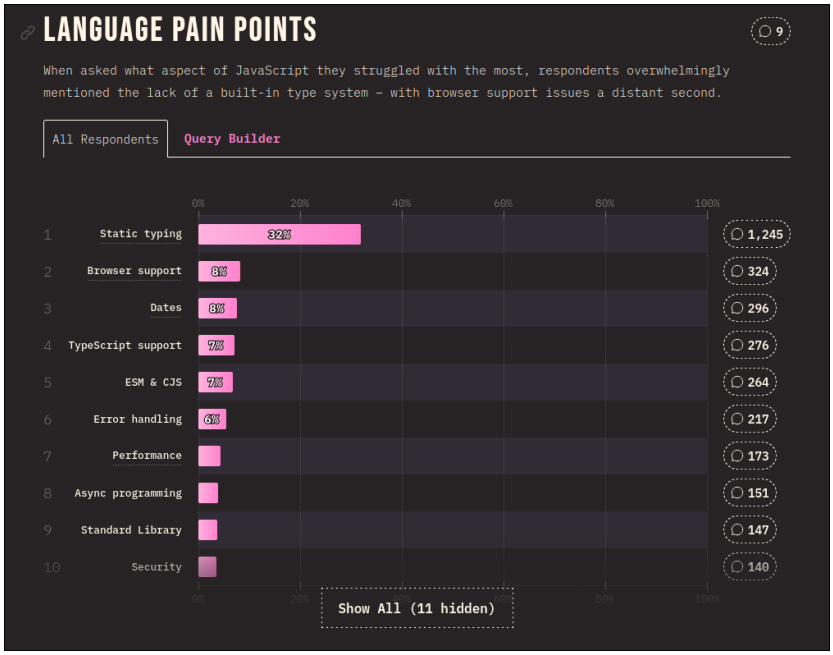


Abbildung 2: 32% der Befragten in der „State of JavaScript 2024“ Umfrage gaben an das (fehlende) statische Typisierung einer ihrer größten Probleme mit der Sprache sei [GB24]

1.3 Ziel der Arbeit

Im Laufe der Arbeit sollen den Leser*innen Eigenschaften und Vor- wie auch Nachteile von Typsystemen soweit vorgestellt werden, sodass aufgrund dieser eine informierte Technologieauswahl für neue Projekte mit einem Web-Backend getroffen werden kann. Die Erlernten Grundlagen von Typsystemen sollen die Begrifflichkeiten und Unterschiede von dynamischen und statischen sowie starken und schwachen Typsystemen umfassen.

Des weiteren können sich Leser*innen durch die Arbeit weiterführende Literatur zu Typensystemen besser verstehen und sich selber weiterbilden. Hierdurch können sie auch für weitere Anwendungsfälle Kriterien bestimmen welche für die Auswahl einer Sprache in Bezug auf deren Typsystem relevant sind.

2 Systematische Literaturrecherche

3 Hauptteil

3.1 Begriffserläuterungen

3.1.1 Typen und Typsystem

Die Grundsätzliche Idee eines Typsystems ist die verhinderung von verhinderung von Fehlern zur Laufzeit eines Programmes. Typischerweise wird hierfür ein Algorithmus eingesetzt welcher überprüft, dass alle Typen die innerhalb eines Programmtextes verwendet werden untereinander stimmig sind.

Typen können als eine Menge von möglichen Werten gesehen werden. So ist beispielsweise die Menge für den Typen von Ganzzahlen (oft `integer` oder `int`) im Fall von 32 bit $[-2^{31}, 2^{31} - 1]$. Einer der simpelsten Typen sind Wahrheitswerte (oft `boolean` oder `bool`) welche aus der Menge $\{\text{true}, \text{false}\}$ bestehen.

Oft stellen Sprachen Möglichkeiten bereit aus diesen, wie auch weiteren, als *primitiv* bezeichneten Typen, komplexere Strukturen zu bilden um kompliziertere Anwendungsfälle abzudecken. Das einfachste Konzept stellt hierbei ein *struct* da, welches einen Zusammengesetzen Datentypen aus verschiedenen anderen Zusammenfast. Dies wird in 3.2.1 genauer dargestellt.

Das Typesystem einer Sprache ist der Oberbegriff dafür wie Typen gebildet werden können und wie die richtigkeit dieser überprüft wird, falls das der Fall ist.

3.1.2 Statische und dynamische Typisierung

3.1.3 Starke und schwache Typisierung

3.2 Unterschiede der Typsysteme diverser Sprachen

Im folgenden werden die Typsysteme von C, Python und Haskell betrachtet und ihre Unterschiede dargestellt.

3.2.1 Strukturelle Typen (C)

3.2.2 Dynamische Typen (Python)

3.2.3 Abhängige Typen (Haskell)

Algebraische Datentypen?

3.3 Relevanz für die Webentwicklung

3.3.1 Schnittstellendefinition

- Rest API?
  - JSON?

Literatur

[GB24] Sacha Greif und Eric Burel. *State of javascript 2024*. 2024. URL: https://2024.stateofjs.com/en-US/features/#language_pain_points.

Suchergebnisse	**Notizen**
N. H. Madhavji und I. R. Wilson. „Cray Pascal“. In: *Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction*. SIGPLAN '82. Boston, Massachusetts, USA: Association for Computing Machinery, 1982, S. 1–14. ISBN: 0897910745. DOI: 10.1145/800230.806975. URL: https://doi.org/10.1145/800230.806975	Notizen

Tabelle 1: Literaturliste zum Rechercheprotokoll