LIESERS CLEAN CODE

# KISS oder YAGNI?

Mach es so einfach wie möglich und lass weg, was nicht benötigt wird.

Verwirrung gibt es immer wieder bei der Frage, worin denn eigentlich der Unterschied zwischen den Prinzipien KISS und YAGNI besteht. KISS steht für „Keep It Simple, Stupid". Manchmal lese ich auch „Keep It Simple and Stupid". Ich bevorzuge die erste Übersetzung, die ungefähr lautet: „Halte es einfach, du Depp!" Wobei ich mich selber in die Reihe der Deppen einschließe. Gemeint ist, dass man bei Code, der unnötig kompliziert geschrieben ist, selber nach kurzer Zeit wie ein Depp davor sitzt und sein eigenes Werk nicht mehr versteht. Daher sagt das KISS-Prinzip: Gestalte den Code so einfach wie möglich, sodass ihn jeder Depp leicht verstehen kann. Aber lassen wir die Deppen ab jetzt mal raus aus der Betrachtung. Es geht um einfachen Code.

Das meint eine Einfachheit, die sich an den Anforderungen orientiert. Denn es darf natürlich nicht darum gehen, eine zu einfache Lösung zu realisieren. Würde etwa die Aufgabenstellung lauten, eine Million Datensätze zu sortieren, würde die Verwendung des „Bubble Sort"-Algorithmus dem KISS-Prinzip nicht entsprechen. Dieser Sortieralgorithmus ist sehr einfach, aber für große Datenmengen ungeeignet. Es geht also darum, eine Lösung zu wählen, die einerseits einfach ist, gleichzeitig aber die Anforderungen umsetzt.

Das KISS-Prinzip steht somit noch mit einem weiteren Prinzip in Verbindung: „Beware of premature optimization", zu Deutsch: Vorsicht vor Optimierungen. Reflexartig immer den Algorithmus mit dem besten Laufzeit- und Speicherverhalten zu implementieren würde das KISS-Prinzip verletzen. Doch darum soll es ein anderes Mal gehen.

Schauen wir als Nächstes auf YAGNI: „You Ain't Gonna Need It". Hierbei geht es um die Sicht auf die Anforderungen. Das Prinzip drückt aus, dass nur umgesetzt werden soll, was sich aus den Anforderungen ergibt. Letztlich muss irgendwer den Aufwand bezahlen, den wir als Entwickler treiben. Da eine zu umfangreiche Umsetzung häufig nicht hinterfragt wird, tauchen die zusätzlichen Kosten nirgendwo explizit auf. Dennoch fallen sie an. Bei YAGNI geht es also um eine passgenaue Umsetzung im Sinne der Anforderungen.

Als Entwickler haben wir während der Implementation oft gute Ideen zu weiteren Features. Dagegen wendet sich das YAGNI-Prinzip ausdrücklich nicht. Die guten Ideen müssen allerdings vom Product Owner „abgesegnet" werden, denn in seiner Verantwortung liegt es, zu entscheiden, welche Anforderungen umgesetzt werden sollen. Für uns als Entwickler bedeutet YAGNI, dass wir uns strikt daran halten, nur solche Anforderungen umzusetzen, die der Product Owner sich wünscht – und für die er bereit ist, Zeit und Geld zu investieren.

Nun gibt es zwischen KISS und YAGNI offensichtlich eine gewisse Ähnlichkeit. Im Web findet man zu YAGNI auch die Aussage, dass eine Lösung nicht zu generisch erstellt werden solle. Nun tendieren wir als Entwickler aber generell dazu, Lösungen zu schaffen, die generischer sind. So möchten wir bestmöglich vorbereitet sein auf spätere Anforderungen – die dann aber oft gar nicht eintreffen. Oft kommt die generische Lösung nie zum Einsatz oder war am Ende sogar hinderlich. In der Abgrenzung von KISS und YAGNI ist durch den Code, der zu generisch ausgelegt ist, das KISS-Prinzip verletzt: Die Lösung setzt ausschließlich die derzeit geforderten Anforderungen um, auch wenn sie generischer ausfällt als benötigt. Ein Mehr an Generik führt nicht zwingend zu einer umfangreicheren Umsetzung von Anforderungen. Insofern ist hier das KISS-Prinzip verletzt. Die Lösung ist aufgrund des generischen Ansatzes eben nicht so einfach wie möglich.

YAGNI wäre verletzt, wenn die Implementation zusätzliche Features bietet. Diese müssen sich allerdings an der Oberfläche für den Product Owner bemerkbar äußern. Es genügt nicht, dass „unter der Haube" eine generische Lösung arbeitet, die theoretisch mehr könnte, sondern es muss tatsächlich ein Mehr an Anforderungen umgesetzt sein.

## Fazit

Als Entwickler müssen wir uns weiterhin damit auseinandersetzen, dass manche Begriffe unscharf verwendet werden. Die Abgrenzung von KISS und YAGNI wird oft nicht klar benannt. Man könnte einwenden, dass es in der Hauptsache darum geht, den Code wandelbar zu gestalten. Ob ich bei Code, der dem Wert der Wandelbarkeit noch nicht entspricht, eine KISS- oder eine YAGNI-Verletzung benenne, ist letztlich gleichgültig, solange der Code daraufhin vereinfacht wird.

Doch für die weitere Entwicklung der Informatik ist es wesentlich, uns auf gemeinsame Begriffe zu verständigen, um präziser diskutieren zu können. Daher sei hier noch mal knackig auf den Punkt gebracht: YAGNI betrachtet die Frage, ob der Code eine konkrete Anforderung umsetzt, während es bei KISS um die Frage geht, ob die Anforderungen so einfach wie möglich umgesetzt sind. ■

**Stefan Lieser**
sucht ständig nach Verbesserung und neuen Wegen, um die innere Qualität von Software zu optimieren. Gemeinsam mit Ralf Westphal hat er die Clean Code Developer Initiative (https://clean-code-developer.de) ins Leben gerufen.
**https://ccd-akademie.de**

**dnpCode** A2202CleanCode