Uwe Post

Besser coden

Guter Code, gutes Design, guter Stil, gute Sicherheit, gutes Refactoring, gute Doku, gute Tests, gute Tools – alles auch asynchron, continuous, agil und im Team: So geht's!

Rheinwerk Computing

von den eigenen Gewohnheiten abweichen. Die besten Konventionen sind solche, für die sich möglichst wenige Kollegen umgewöhnen müssen.

Konventionen festzulegen ist nur der erste Schritt. Man muss sie auch konsequent einhalten, sonst gerät man ins Stolpern. Missverständnisse z. B. durch ungewohnte Schreibweisen kosten Zeit, Nerven (Ihre und die der Anwender Ihrer Software) und letztlich Geld.

Aber wie sklavisch müssen sich die Kolleginnen und Kollegen an die Konventionen halten? Kostet es nicht extra Aufwand, dauernd auf deren Einhaltung zu achten?

Versuchen Sie, das richtige Maß zu finden. Eine Konvention, an die sich niemand hält, ohne dass dies zu nennenswerten Problemen führt, mag verzichtbar sein. Hält das Team eine neue Konvention für angebracht – etwa im Zuge der Einführung einer neuen Technologie –, ziehen Sie auch dieses Feedback in Erwägung. Kommt es häufig zu Problemen aufgrund vermeintlich einfacher Fehler, prüfen Sie, ob die Konventionen in Ihrem Team hinreichend sind oder ob sie schlicht in Vergessenheit geraten sind.

Halten Sie die offizielle Liste der Regeln schriftlich fest, und sorgen Sie dafür, dass jeder weiß, wo sie zu finden ist.

2.1.1 Erlaubte und verbotene Abweichungen

Besonders selbstbewusste Programmierer halten ihren speziellen Stil für Ausdruck ihrer Individualität. Sie haben eigene Regeln erfunden und gebrauchen sie zusätzlich zu oder anstelle jener des Teams. Sie haben sogar möglicherweise gute Gründe dafür. Abweichungen sind aber nur zu tolerieren, wenn sie keine Missverständnisse verursachen.

Als positives Beispiel sei die Array-Deklaration in Java genannt. Die sieht üblicherweise so aus:

```
String[] strings;
```

Syntaktisch erlaubt ist allerdings auch – genau wie in C – diese Variante:

```
String strings[];
```

Semantisch ist beides identisch: Beide Codezeilen erzeugen eine Variable namens `strings`, dessen Typ ein Array aus `String`-Objekten ist. Kein Kollege wird über die Zeile stolpern, egal wie sie aussieht.

Verfechter der Konvention »Keine Array-Deklarationen im C-Stil«, die die zweite Variante verbietet, argumentieren, der Typ der Variable sei ein Array, folglich gehörten die Klammern zu `String` und nicht zu `strings`. Das ist durchaus plausibel – aber die »falsche« Schreibweise wird selten zu Missverständnissen führen. Also muss das nicht unbedingt durch eine Regel festgelegt werden. Denn weniger ist wie so oft mehr. Je mehr Konventionen Entwickler beachten müssen, umso leichter passiert es, dass sie mal eine übersehen. Vor allem dann, wenn sie selten zum Einsatz kommen und in der Praxis kaum von Belang sind. Übertreiben Sie es also nicht mit dem Regelwerk.

Wie auch immer: Wenn die Konventionen einmal feststehen, fangen Sie keine Diskussionen an.

»Die öffnende geschweifte Klammer `{` soll also immer am Ende der Zeile stehen?«

»Ja.«

»Warum?«

»Ist Konvention.«

»Das ist ein blöder Grund.«

»Mag sein, aber …«

Solche Gespräche sind Zeit- und Energieverschwendung. Wenn das Team einmal Konventionen festgelegt hat, ist es sinnlos, sie dauernd zu hinterfragen. Nachträglich sehr viele Stellen im Code zu ändern, würde erheblichen Aufwand bedeuten, erst recht, wenn es an sich keinen besonders plausiblen Grund für die neue Vorgabe gibt.

»In England fahren die Autos im Gegensatz zum Rest Europas links?«

»Ja.«

»Warum?«

»Ist Konvention.«

»Das ist ein blöder Grund.«

»Mag sein, aber es wäre offensichtlich viel zu aufwendig, diese Konvention zu ändern.«

Sehen Sie, was ich meine? Natürlich können Sie trefflich diskutieren, sei es über die Array-Deklaration oder über die Sprachqualität in Kommentaren. Würden Sie

diese Zeit stattdessen investieren, um den kompletten Code auf einen einheitlichen Stand zu bringen, der die Konventionen ordentlich reflektiert, wären Sie wahrlich besser beraten. Vorausgesetzt, der Aufwand ist überschaubar.

Eine weitere Konvention definiert daher den Umgang mit entdeckten Verstößen. Sofort ändern? Was bei Kommentaren unproblematisch ist, kann bei echten Codeänderungen Probleme verursachen. Wenn Sie einen Bezeichner umbenennen (und sei es nur, weil er einen banalen Rechtschreibfehler enthält), müssen Sie das in allen Codedateien tun, die darauf zugreifen. Unter Umständen fassen Sie damit Code an, den gerade ein anderer Entwickler bearbeitet, oder übersehen gar eine Stelle.

Sobald Mechanismen wie etwa *Reflection* in Java im Spiel sind, reicht die Kette der Abhängigkeiten möglicherweise noch weiter. Gewisse Codeteile könnten *erwarten*, dass ein Bezeichner genau so heißt, wie er heißt, und auf Änderungen mit fatalen Fehlern reagieren. Nicht jede Abhängigkeit ist offensichtlich. Denken Sie an automatisch erzeugte Webservice-Endpoints, die von Clients benutzt werden, auf die Sie keinen Einfluss haben. Hier sind Änderungen schlicht verboten, selbst wenn noch so peinliche Schreibfehler in Bezeichnern damit bis in alle Ewigkeit zementiert sind.

2.1.2 IDE, Formatierung und Code Style

Ob sich geschweifte Klammern { am Ende einer Zeile oder vorn in der nächsten öffnen – Geschmacksache. Wie weit Sie Codeblöcke einrücken – alles Geschmacksache. Formatierungsstil, oder *Code Style*, ähnelt Vorlieben für Bekleidung. Im Gegensatz dazu ist Einheitlichkeit innerhalb eines Teams allerdings äußerst hilfreich.

IDEs wie *Eclipse*, *NetBeans*, *Visual Studio* oder *IntelliJ IDEA* erlauben es Ihnen, in gewissen Grenzen einen bestimmten Formatierungsstil zu etablieren, und enthalten zumeist auch bereits eine Vorgabe. Es ist möglich, beim Speichern (oder *Committen*) einer Codedatei den Stil automatisch anzuwenden. Nutzen Sie ruhig diese Funktion, um Einheitlichkeit zu gewährleisten. *Welchen* Stil Sie dabei anwenden, ist zweitrangig – entscheidend ist, dass alle Entwickler denselben verwenden. Dazu können Sie die betreffende Stilvorlage einfach auf allen Arbeitsplatzrechnern installieren.

Tabulator oder Leerzeichen?

Einrückung dient generell der Lesbarkeit. Wenn ein Quellcode aber auf der Maschine des Kollegen (oder in einem Texteditor oder auf einer Webseite) anders aussieht als auf Ihrer, nur weil die Tabulatorweite unterschiedlich eingestellt ist, ist Stirnrunzeln vorprogrammiert. Stellen Sie also Ihre IDE so ein, dass sie beim Drücken der ⇥-Taste Leerzeichen erzeugt. Das Tab-Steuerzeichen selbst hat in Quellcodedateien heutzutage nichts mehr zu suchen.

Wenn Sie in einer vorhandenen Codebasis teamweit einen Stil einführen, kann es passieren, dass auf einen Schlag eine Menge Quellcode verändert wird. Das kann zu Irritationen führen, wenn Ihr Quellcodeverwaltungssystem plötzlich Hunderte Änderungen protokolliert – ohne semantische Änderung an der Software. Mehr zum Thema Quellcodeverwaltung finden Sie in Kapitel 3.

Für Java-Entwickler existieren verschiedene Style Guides, die über Formatierungsregeln weit hinausgehen. So schreibt die verbreitete Richtlinie von *Google* beispielsweise nicht nur vor, dass Blöcke um genau zwei Leerzeichen (und nicht etwa einen, auf Breite 2 eingestellten Tabulator) einzurücken sind, sondern auch, dass nie mehr als eine Anweisung in einer Zeile stehen darf. Auch solche Guides können Sie als Konvention vereinbaren, wenn Sie keine eigene erfinden möchten. Warum nicht Bewährtes übernehmen?

Code Style Guides im Netz

- Google/Java: *https://google.github.io/styleguide/javaguide.html*
- Google/C++: *https://google.github.io/styleguide/cppguide.html*
- Microsoft/C#: *https://msdn.microsoft.com/de-de/library/ff926074.aspx*

Um einheitlichen Code zu produzieren, hilft es ungemein, die zugehörigen Regeln nicht unterschiedlichen Entwicklungssystemen eintrichtern zu müssen. Es verringert deutlich den Administrationsaufwand, wenn alle Entwickler dieselbe IDE verwenden. Einheitlichkeit geht hier über Individualität. So können beispielsweise alle Entwickler gleichzeitig größere Updates installieren. Gibt es dabei Probleme, können sie für das ganze Team zugleich gelöst werden und nicht peu à peu. Murphy's Law schreibt bekanntlich vor, dass ein Update genau dann die IDE eines beteiligten Entwicklers lahmlegt, wenn ein Projektabgabetermin bevorsteht.

Ihre Konventionen legen Sie schriftlich nieder, und zwar so, dass alle Entwickler jederzeit Zugriff darauf haben. Sogar im 21. Jahrhundert kann da ein Schnellhefter – am besten in einer Leuchtfarbe und an einer zentralen Stelle im Büro deponiert – mit einem darin abgehefteten Ausdruck der Regeln nicht schaden. Dann kann zumindest niemand behaupten, er hätte von nichts gewusst.