Wieso ein gutes API Design wichtig ist

Salame, Ahmed
Hochschule Mannheim
Fakultät für Informatik
Paul-Wittsack-Str. 10, 68163 Mannheim

Zusammenfassung—**TODO–**
Dieses Paper soll aufzeigen was eine Application Programming Interface (API) ist und was daran wichtig ist für einen Entwickler. Dabei zeigt dieses Paper Kriterien für ein gutes API Design. Es zeigt ebenso was schlechtes Design ausmacht anhand von Beispielen. Es werden verschiedene Ansätze aufgezeigt, wie ein nutzerfreundlicheres API entworfen werden kann.

Inhaltsverzeichnis

1. Einleitung

Application Programming Interfaces(API) sind kaum wegzudenken. So gut wie alle Programmbibliotheken, Frameworks und Software Development Kit (SDK) nutzen eine. Auf programmableweb[1] werden über 18.000 APIs aufgelistet.
Für die Entwicklung einer API gibt es ein immer größer werdender Markt für Firmen, die aktiv bei der Entwicklung der APIs bei anderen Unternehmen aushelfen, wie Apigee[2].
Firmen haben auch unterschiedliche Strategien, wie sie mit ihren APIs umgehen [9, siehe Seite 11]. So hat der Blogging-Dienst Twitter seine Popularität unter anderem durch die gute API zu verdanken. Es bot die notwendige Infrastruktur, so dass Twitter auf unzähligen Endgeräte genutzt werden konnte. Facebook setzt mit ihrer API auf ausfallsicher und Skalierbarkeit. Oder die Firma Best Buy entwickelte eine API, um ihren Online-Dienst auszubauen.

2. Application Programming Interface

2.1. Die Definition einer API

Eine API besitzt keine einheitliche Definition. Die unterschiedlichen Erklärungsansätze weichen an bestimmten Punkten voneinander ab. Allgemein kann eine API als etwas beschreiben, dass dem Nutzer zur Verfügung gestellt wird, so dass dieser darüber mit einem Softwaresystem kommunizieren kann. Laut Bloch [1] handelt es sich um eine API, wenn sie folgende Kriterien erfüllt:

1) Eine API bietet eine Menge an Operationen an, welche durch ihre Eingaben und Ausgaben definiert sind.

2) Es ist möglich die Schnittstellen neu zu Implementieren, ohne dass der Nutzer seinen Code anpassen muss.

Im folgenden Paper wird diese Definition zu Grunde gelegt.

2.2. Die Arten einer API

Es gibt unterschiedliche Arten einer API [6]. Es ist möglich eine API grob in drei Kategorien einzuteilen.

1. https://www.programmableweb.com/
2. https://apigee.com/api-management/

Interne APIs sind nicht für die Öffentlichkeit gedacht. Sie werden intern verwendet und sind daher nicht auf die Akzeptanz der Community angewiesen. Dies sorgt dafür, dass sie nicht den strengen Regeln befolgen müssen, die eine API ausmacht. Es ist möglich noch Änderungen vorzunehmen an der Schnittstelle. Sie helfen dabei, interne Strukturen umzusetzen, wie das arbeiten in verschiedenen Teams.
Öffentliche APIs sind der Öffentlichkeit zugänglich. Sobald eine API öffentlich zugreifbar ist, ist es kaum möglich diese nachhaltig so abzuändern, ohne dass all die Nutzer dieser API auch ihren Code anpassen müssen. Zu einer guten Öffentlichen API gehört auch eine dazu passende Dokumentation. Beispiele hierfür wäre das Java Development Kit (JDK) von Java oder die Standard Template Library (STL) von C.
Neben der Öffentlichen und Internen API gibt es noch die Remote APIs. Hier werden die Schnittstellen über das Netzwerk zugänglich gemacht. Für den Entwickler soll dabei das Gefühl entstehen, dass die Nutzung der API lokal geschieht. Beispiel hierfür wäre die Remote API von Java.

2.3. Die Vorteile einer API

Es gibt viele gute Gründe APIs zu nutzen. Nach Reddy [8] sprechen insbesondere folgende Gründe für das nutzen einer API:

Robustheit

> Eine API sorgt für robusteren Code. Da die Nutzer nur über eine Schnittstelle mit den System Kommunizieren, besteht Information Hiding. Es sorgt dafür, dass die eigentliche Implementation ausgetauscht werden kann, ohne das es beim Nutzer angepasst werden muss. So lässt sich ganz leicht die bestehende Implementierung austauschen, ohne das der Nutzern dieser API gezwungen ist, seinen Code Anpassung zu müssen.

Parallele Entwicklung

> Wenn ein Entwickler ein Programm schreiben möchte und dazu den Code von einem anderen Entwickler braucht, dessen Code allerdings nicht fertig ist, wäre es verschwendete Zeit, falls einer auf die Fertigstellung des Codes warten muss. Es wäre klüger wenn stattdessen sich beide einigen würden, wie die Schnittstellen auszusehen haben. So könnte man von der Implementierung der Schnittstelle ein Mockup erstellen und in ihrem Inhalt einfache Werte zurückgeben, so dass sich der Code zumindest Compilieren lässt. Die eigentliche Implementierung kann so später stattfinden, so dass der Nutzer dieser Schnittstelle nur wenige Änderung, bis keine Änderung vornehmen muss.
> So lässt es sich realisieren, dass mehrere Teams parallel Entwickeln können, ohne jedes Detail von den Aufgaben der anderen Teams zu kennen.

Wiederverwendbarkeit

> Eines der größten Vorteile einer API ist die Wiederverwendbarkeit von bisher existierenden Code. Das Entwickeln von eigenen Schnittstellen kostet sehr viel Zeit und Geld. Durch das wiederverwenden von API, ob nun eigene oder von anderen, kann man sich dies einsparen. Dabei stehen sowohl kostenpflichtige, als auch kostenlose zur Auswahl. Es existieren viele unterschiedliche APIs für die unterschiedlichsten Probleme.
> Es gibt als Beispiel viele APIs, die sich mit dem laden von Bildern beschäftigen[3]. Viele solcher Bibliotheken haben bereits einen gewissen Reifeprozess durchgemacht. Sie wurden ausgiebig getestet und haben womöglich auch ein breites Spektrum an Informationen. Als Beispiel ist hier die Spiele-Bibliothek LibGDX zu nennen. Es hat eine große Community, an die sich Entwickler wenden können, falls sie fragen haben[4].

3. Beispiele vom Schlechtem API Design

Es gibt sehr viele Programmbibliotheken, die sehr viele Schnittstellen anbieten, auf welche die Entwickler zugriff haben. Bei einer Erweiterung der Bibliothek kommen zudem immer mehr Schnittstellen hinzu. Wo in dem JDK aus Java 8 sich noch 4240 Klassen befinden[5], hat dessen Nachfolger Java 9 bereits 6005 Klassen[6]. Bei einer solch wachsenden Anzahl an Klassen ist es wichtig sich ein Konzept auszudenken, damit der Entwickler bei der Bibliothek keine Schwierigkeiten bekommt. Wenn einzelne Punkte nicht eingehalten werden, kann der Entwickler schnell die Überblick verlieren. Der Erfolg einer Bibliothek kann unter anderem dadurch gemessen werden, wie häufig sie von Verschiedenen Entwicklern genutzt wird. In diesem Kapitel werden Punkte aufgezählt, die den Erfolg einer Bibliothek bestimmen können.

3.1. Mangelnde Dokumentation

Die Dokumentation einer Bibliothek beschreibt ihre Funktionalität, auf die der Entwickler zugreifen kann. So werden die Schnittstellen der Bibliothek hier definiert, sowie mit einer zusätzlichen Beschreibung versehen. Zusätzlich kann eine Dokumentation noch weitere Punkte beinhalten, wie Namens Konventionen. Ein bekanntes Beispiel hierfür ist die Java Dokumentation von Oracle. Hier werden alle Klassen auf die ein Entwickler Zugriff hat angezeigt. Neben einer Beschreibung der Klassen, gibt es in der Java API Dokumentation auch eine Beschreibung für die Methoden einer Klasse.

3. http://cimg.eu/
4. http://www.badlogicgames.com/forum/
5. https://docs.oracle.com/javase/8/docs/api/
6. https://docs.oracle.com/javase/9/docs/api/

Für das erstellen von Dokumentation gibt es viele Möglichkeiten. Die Programmiersprache Java bietet für die Dokumentation des Quellcodes das Javadoc an[7]. Bei Javadoc handelt es sich um ein Tool für das Generieren von API Dokumentation. Dabei braucht ein Entwickler z.B. über einer Methode nur ein Javadoc Kommentar zu schreiben, so dass das Javadoc Tool es in HTML code umwandeln kann. Beim Javadoc Kommentar lassen sich auch die Abhängigkeiten als Referenzen zu anderen Klassen und Methoden herstellen, das erleichtert dem Entwickler zusätzlich die Suche nach der benötigten Klassen. Die Java API Dokumentation z.B. wurde so erstellen. Auch andere Bibliotheken, die in Java geschrieben worden sind, haben so ihre Dokumentation erstellt, wie die Bibliothek Lightweight Java Game Library (LWJGL)[7]. Das Javadoc nimmt beim erstellen des Quellcodes viel Arbeit für eine externe Dokumentation ab.
Ein Ähnliches Tool für das erstellen für die Dokumentation ist Oxygen[4]. Oxygen steht dabei nicht nur für Java zur Verfügung, sondern auch für andere Programmiersprachen wie C, C++ oder Python.

3.2. Eindeutige Bezeichnungen von Schnittstellen

Falls eine Dokumentation die API nicht gut genug Dokumentiert, so dass z.B. eine Methode nicht genau beschrieben wird wie sie intern Funktioniert, kann es dazu führen, dass ein Entwickler eine Schnittstelle der API falsch nutzt.

Die Java Klasse java.util.Calendar[8] besitzt folgende Methode:

```
public final void set(int year, int month, int date)
```

Listing 1. Methode set(...)

Anhand der Parameter wäre es naheliegend, das beim Aufruf mit dem folgenden Werten die Methode das Datum auf den 22 August 2017 setzt:

```
Calendar cal = Calendar.getInstance();

cal.set(2017, 8, 22);
```

Listing 2. Setzten eines Datums

Allerdings sieht man anhand der Beschreibung der Methode das bei der Eingabe vom Monat, der Januar nicht mit der Eins sondern mit der Null beginnt. Also wäre das Datum, das im obigen Beispiel gesetzt wurde, der 22 September 2017. Vorgesehen ist es, es folgendermaßen zu machen:

```
cal.set(2017, Calendar.AUGUST, 22);
```

Listing 3. Setzten eines Datums mit Konstante

Es wird keine Zahl der Methode als Monat übergeben, sondern eine Konstante. Auch wenn die Methode set(...) dokumentiert ist, so liegt die Benennung ihrer Parameter nahe dass der Entwickler, beim nutzen dieser Methode, ihr einfach Zahlen übergeben kann, und er so die Dokumentation nicht zu lesen braucht. Tatsächlich soll laut Bloch[3] eine Methoden so funktionieren, dass es einfach ist sie zu nutzen, aber schwierig, die Methode falsch zu nutzten. Idealerweise verhindert man das Falsche nutzen einer Methode komplett. Zudem soll eine Methode einer API nicht zwangsweise Dokumentiert sein, wenn der Bezeichner der Methode und ihre Parameter das Nutzen nahelegen.

Ein anderes Beispiel für eine Methode, die zum Missverständnis führen kann[2, ab Minute 7], ist folgende Methode:

```
public static boolean getBoolean(String name);
```

Listing 4. getBoolean methode aus java.lang.Boolean

Sie stammt aus der Klasse java.lang.Boolean[9]. In Java gibt es Wrapperklassen, die Objekte der primitiven Datentypen darstellen, wie int, double oder auch boolean. Daher gibt es jeweils für jeden Primitiven Datentyp eine Wrapperklasse, für den Datentyp int wäre es z.B. java.lang.Integer. Jede Klasse besitzt eine Methode, dessen Name mit set anfängt, so wie setInteger oder wie im Beispiel setBoolean. Es liegt nahe, dass viele Entwickler zunächst annehmen würden, das beim Aufruf der Methode getValue der Wert true geliefert wird.

```
public boolean getValue(){
  boolean bol = getBoolean("true");
  if(bol == true){
    return true;
  }else{
    return false;
  }
}
```

Listing 5. Beispiel mit getValue

Allerdings wird hier false geliefert. In der Dokumentation dieser Methode ist zu entnehmen, dass die Methode getBoolean nur dann true liefert, falls es ein System Property gibt, welches den Name hat, den diese Methode als String bekommt. Diese Vorgehensweise ist irreführend, das kann für Frust sorgen und für Programmfehlern.

3.3. Keine Konsistente Benennung der Bezeichner

Für einen Entwickler ist es wichtig, dass dieser sich in der API zurechtfindet.

4. Ansätze zur Verbesserung allgemeiner APIs

Eine API bietet viele Möglichkeiten für eine stetige Verbesserung. Eine API, die öffentlich genutzt wird, hat meist viele Schritte zur Verbesserung durchgemacht. Die Schritte zur vermeintlichen Verbesserung einer API kann sich dabei auch über Jahre strecken. Jedoch muss das Resultat nicht zwingend in jedem Fall Positiv sein[9, siehe Seite 25].
Folgend werden einige Vorgehensweisen beschrieben, die Aufzeigen sollen, wie dafür gesorgt werden kann, das eine API auch eine Konsequente Verbesserung widerfährt.

7. https://javadoc.lwjgl.org/
8. https://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html
9. https://docs.oracle.com/javase/9/docs/api/java/lang/Boolean.html

4.1. Umdenken der Menschen

Hier soll es einen Ansatz geben, der sich damit befasst, dass man die usability einer API nicht unterschätzen sollte. Viele nehmen das nicht ernst und entwickeln nicht sorgfältig genug.

4.2. Nutzer freundlicheres Design

Hier soll es eine erklärung geben, was genau die meisten Nutzer etwas als besonders benutzerfreundlich betrachten. Dabei soll auch Beispiele aufgezeigt werden, wie man sowas ermittelt

4.3. APIs Testen

5. Ausgewählte API Entwurfs Beispiele

Bloch beschreibt die Eigenschaft einer Gute API folgendermaßen:

Good APIs create long-term customers; bad ones create long-term support nightmares[3].

Eine API, die Nutzerfreundlich ist, erspart einem Entwickler, der diese API nutzt, sehr viel Stress und vor allem Zeit.

–TODO– Hier soll es einen Auszug geben, welche möglichen Lösungen es zu ausgewählten Probleme gibt. Hier soll es unter anderem einige Beispiele geben, die typisch sind für clean code (Methodennamen, Parameter, namens Konventionen etc). Um zu verdeutlichen was genau gemeint ist, wird dabei Java/C++ code als Beispiel genommen. Dieser Abschnitt soll sich mehr mit der Technischen Seite befassen.

5.1. Sinnvolle Klassennamen wählen

5.2. Methodennamen und ihre Parameter

Der Name einer Methode sollten idealerweise ihren Zweck beschreiben. Einen Namen gut zu wählen kann viel Zeit kosten, allerdings erleichtert es dem Entwickler, welcher eine API nutzt, die Suche nach einer Passenden Methode[5].

Hier soll es erst mal ein Beispiel geben, welche Probleme auftreten könnten, wenn man sich bei dem schreiben einer Methode keine Gedanken macht, z.B. falsche Parameter übergaben oder zu viele Parameter.

5.3. Keine redundanten Methoden

Hier soll es aufzeigen, wieso man nicht für alles eine Methode schreiben sollte, wie z.B. beim einlesen von einem Images.

5.4. Refactoring

Hier soll aufgezeigt werden, das durch Refactoring bestehenden code ausbessern kann. Es gibt unzählige

6. Tools zur Unterstützung der Entwicklung einer API

Bei dem Entwickeln einer API kann es sehr langwierig sein diese zu Programmieren und zu Testen. Viele Bibliotheken haben mehrere tausende Schnittstellen, was zu sehr unübersichtlichen Code führen kann. Es gibt allerdings viele Tools, die einem das Arbeiten erleichtern. Im Folgenden werden einige Tools vorgestellt und dessen Vorteile dargestellt.

6.1. Entwicklungsumgebung

Es gibt sehr viele Entwicklungsumgebungen, mit dessen Hilfe Programmiert werden kann. Einer der Bekanntesten Vertreter wäre Microsoft Visual Studio[10], welches es in vielen Verschiedenen Versionen gibt, Als Kostenpflichtige Premium Edition oder als kostenlose Community Edition. Sie bietet Unterstützung für viele verschiedene Programmiersprachen, wie C, C++, C#, Javascript oder Python.

Neben der Entwicklungsumgebungen von Microsoft gibt auch noch IntelliJ von JetBrains[11]. Ähnlich wie Microsofts Visual Studio bietet IntelliJ Sprachunterstützung für viele verschiedene Programmiersprachen an. Auch gibt es hier ähnliche Modelle für den Entwickler, mit denen er Programmieren kann, von einer Kostenpflichtige als auch einer Kostenlosen. Im Gegensatz zu Microsoft Visual Studio ist IntelliJ auch außerhalb von Microsoft Windows Verfügbar.

Die beliebteste Entwicklungsumgebung laut PopularitY of Programming Language (PYPL) ist das in Java geschriebene Programm Eclipse von Eclipse Software[12][13]. Auch Eclipse erlaubt mithilfe von Erweiterungen eine Sprachunterstützung von unterschiedlichen Sprachen, ursprünglich war sie dabei allerdings nur für die Programmiersprache Java Gedacht. All die genannten Entwicklungsumgebungen unterstützen das Refactoring5.4.

Entwicklungsumgebungen sind eine große Hilfe beim Entwickeln und verbessern einer API. Da die Entwickler meist über eine Entwicklungsumgebung auf eine API zugreifen, kann man leicht nachverfolgen, wie die natürliche Arbeitsweise eines Entwicklers ist der diese API nutzt.

In vielen Entwicklungsumgebungen haben die Entwickler die Möglichkeit die Texteingabe Automatisch zu vervollständigen. So werden zum Beispiel bei der Eingabe von "getïn Eclipse mithilfe von den Tasten STRG und LEERTASTE unter anderem sämtliche Methoden aufgelistet, auf die der Entwickler Zugreifen kann. So hat der Entwickler jederzeit die Möglichkeit nach einer Methode zu suchen, die sein aktuelles Problem lösen kann. Dies hat den Vorteil, dass so überprüft werden kann, wie der natürliche verlauf von vielen Entwickler beim Entwickeln ist und wie der Entwickler mit einer API zurechtkommt.

6.2. Das Programm Checkstyle

Das Programm checkstyle kann dazu helfen, dass man gewisse

10. https//www.visualstudio.com/
11. https://www.jetbrains.com/idea/
12. http://www.eclipse.org/
13. http://www.eclipse.org/

7. Fazit

Hier soll es nochmal einen Bezug auf die vorherigen Loesungsbeispielen geben und was für Vorteile es einem bringt.

Abkürzungen

API	Application Programming Interface
SDK	Software Development Kit
JDK	Java Development Kit
STL	Standard Template Library
LWJGL	Lightweight Java Game Library
PYPL	PopularitY of Programming Language

Literatur

[1] Joshua Bloch. *A Brief, Opinionated History of the API*. 19. Nov. 2017. URL: https://www.youtube.com/watch?v=ege-kub1qtk.

[2] Joshua Bloch. *Effective Java - Still Effective After All These Years*. 2014. URL: https://www.youtube.com/watch?v=V1vQf4qyMXg.

[3] Joshua Bloch. „How to Design a Good API and Why It Matters“. In: *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*. OOPSLA '06. Portland, Oregon, USA: ACM, 2006, S. 506–507. ISBN: 1-59593-491-X. DOI: 10.1145/1176617.1176622. URL: http://doi.acm.org/10.1145/1176617.1176622.

[4] Dimitri van Heesch. *Oxygen*. 4. Dez. 2017.

[5] Robert C. Martin. *Clean Code: Refactoring, Patterns, Testen und Technicken für sauberen Code*. 2009.

[6] Brad A. Myers und Jeffrey Stylos. „Improving API Usability“. In: *Commun. ACM* 59.6 (Mai 2016), S. 62–69. ISSN: 0001-0782. DOI: 10.1145/2896587. URL: http://doi.acm.org/10.1145/2896587.

[7] Oracle. *Javadoc Tool*. 4. Dez. 2017. URL: http://www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html.

[8] Martin Reddy. *API Design for C++*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. ISBN: 9780123850034.

[9] Kai. Spichale. *API-Design: Praxishandbuch für Java- und Webservice-Entwickler*. 2017.