Kapitel 1
Verhindern Sie den Weltuntergang!

1.1 Vorwort

Als mein Verlag mir vor einer Weile vorschlug, ein Buch über »besseres Coden« zu schreiben, hatte ich gerade ein Beratungsprojekt hinter mir, bei dem es unter anderem um Programmcode ging, dessen Qualität ehrlicherweise nur mit negativen Zahlen hätte beziffert werden können. Wie sich herausstellte, kannten die beteiligten Junior-Programmierer nur ein einziges Buch über Softwarequalität, und das war schon etwas älteren Datums.

Die Gelegenheit kam wie gerufen, um meine langjährige Erfahrung in Form einer Sammlung von Ratschlägen, Beispielen und denkwürdigen Anekdoten niederzuschreiben. Dabei geht es wohlgemerkt nicht nur um akkurat positionierte geschweifte Klammern, sondern auch um Fragen zur Arbeit im Team und zum Umgang mit hochkomplexen Aufgaben.

Dieses Buch will nur eines: Ihnen helfen, eine bessere Programmiererin oder ein besserer Programmierer zu werden. Denn wenn Sie sich die Qualität von Software, die Ihnen im täglichen Leben unterkommt, genauer anschauen, werden Sie sicher bestätigen, dass die Welt mehr davon braucht.

Mehr bessere Programmiererinnen und Programmierer, nicht mehr Schrottsoftware.

Versäumen möchte ich keinesfalls, an dieser Stelle meinen Kolleginnen und Kollegen zu danken, die mir nicht nur in meinem beruflichen Alltag, sondern auch zu diesem Buch sachdienliche Hinweise gegeben haben:

Martin Drößler, *Nicole Enders*, *Matthias Geirhos*, *Alex Hilsing*, *Dr. Andreas Kotulla*, *Marcus Schlechter*, *Martin Schroer*

Danke! Ohne euch wäre ich längst in der Programmiererhölle.

1.2 Schöne neue Welt

Viele düstere Zukunftsvisionen versprechen die Apokalypse: Klimakatastrophe, Verkehrskollaps – und natürlich Zombie-Pandemie.

Wenn Sie mich fragen, ist die Schrottsoftware-Katastrophe viel schlimmer. Und sie hat schon angefangen.

Wirklich …?

Smartphones übersetzen japanische Speisekarten, Kassen erkennen Waren am Barcode, Ihr Auto parkt von ganz allein ein und erspart Ihnen damit mitleidige Blicke der Passanten. Alles wird gesteuert von toller Software, von coolen Algorithmen, implementiert von schlauen Entwicklerinnen und Entwicklern mithilfe moderner Sprachen und Tools. Ein Traumjob!

Wenn Sie selbst zu diesen schlauen Entwicklerinnen und Entwicklern gehören, erzähle ich Ihnen nichts Neues, wenn ich Ihnen sage, dass die Wirklichkeit bisweilen völlig anders aussieht.

Ihr Auto erklärt, die Parklücke sei zu schmal, Ihr PC verweigert von jetzt auf gleich den Start, die Kasse kennt Ihren Lieblingskäse nicht mehr und Ihr Smartphone behauptet allen Ernstes, das Sushi-Restaurant, in dem Sie gerade sitzen, sei heute geschlossen.

Und just in dem Moment, in dem Sie nach Hause kommen, finden Sie im Briefkasten eine Mahnung wegen einer Rechnung über ein Set rosa Spannbettlaken, das Sie nie bestellt haben. Sie zögern natürlich nicht, sofort die Hotline des Inkassounternehmens anzurufen, doch eine etwas hilflos wirkende Callcenter-Mitarbeiterin erklärt Ihnen, dass sie den fraglichen Datensatz wegen eines Systemausfalls gerade leider nicht aufrufen kann. Schließlich stellt sich heraus, dass lediglich Ihr Name rein zufällig mit dem des tatsächlichen Bestellers übereinstimmt, aber nicht die Anschrift. Sie haben eigentlich gar nichts mit der ganzen Sache zu tun, lediglich eine etwas zu simpel gestrickte if-Abfrage ist da völlig anderer Ansicht.

Scans ergeben, dass Zehntausende im Internet hängende Server über dramatische Sicherheitslücken verfügen – im harmlosesten Fall installieren die Angreifenden bloß Krypto-Miner, um Digitalwährung zu schürfen. Ansonsten müssen ganze Unternehmen tagelang ihre Arbeit einstellen, weil Ransomware-Attacken ihnen die Festplatten verschlüsselt haben. Ein solcher Angriff auf den Betreiber einer US-amerikanischen Pipeline führte beispielsweise im Frühjahr 2021 zu Versorgungsengpässen mit Kraftstoffen und sogar in Europa zu einem Anstieg der Benzinpreise. Der wirtschaftliche Schaden ist unmöglich zu beziffern, zumal viele Betroffene nur heimlich die Scherben einsammeln (sprich: das Lösegeld bezahlen), statt die Polizei zu rufen und die Öffentlichkeit zu informieren.

Aber auch das ist Software: Code altert, Festplatten laufen mit Logfiles voll, die niemand liest. Schlechter oder schlicht semantisch falscher Programmcode lauert unauffällig in großen Applikationen, um im unerwarteten Moment ein widerwärtiges Tohuwabohu zu produzieren.

Programmfehler multiplizieren sich: Unterläuft einem Koch ein Bedienungsfehler, sagen wir mit dem Salzstreuer, so ist vielleicht eine Mahlzeit verdorben. Unterläuft hingegen einer Programmiererin ein Fehler, so befindet sich dieser womöglich in zigtausend Kopien der fraglichen Anwendung oder in einer Server-Applikation, die pro Minute von Hunderten Nutzerinnen und Nutzern verwendet wird.

Die Katastrophe ist im wahrsten Sinne des Wortes vorprogrammiert.

Sicher kennen auch Sie diese Art von Science-Fiction-Geschichten, in denen das Internet oder eine Armee Roboter spontan eine Art Bewusstsein entwickelt und nach kurzer Überlegung zum dem logischen Entschluss gelangt, die Menschheit sicherheitshalber auszuradieren.

Ich glaube nicht an solch einen Unfug.

Ich halte es für viel wahrscheinlicher, dass kaputte Software, unentdeckte Fehler und dilettantischer Programmcode der Menschheit den Garaus machen. Schwarzseher würden behaupten: Es hat schon begonnen.

Vielleicht kennen Sie den Spielfilm-Klassiker »2001: Odyssee im Weltraum«. Auf den ersten Blick mag der Eindruck entstehen, dass hier ein verrücktspielender Computer Menschen tötet. Tatsächlich handelt es sich um eine simple Fehlerkette – letztlich mit fatalen Folgen. Die Programmiererinnen und Programmierer des HAL-9000-Computers hielten ihn für perfekt – und damit letztlich sich selbst.

Was für ein Irrtum.

1.3 Was läuft falsch?

Im Jahr 2014 war der *Heartbleed-Bug* in aller Munde. Es handelte sich um einen kleinen, aber schwerwiegenden Programmfehler in der *OpenSSL*-Bibliothek, der es Angreifenden ermöglichte, über das HTTPS-Protokoll sicherheitsrelevante Daten wie private Krypto-Schlüssel und Passwörter von Webservern auszulesen. Die betroffenen Versionen der Bibliothek waren auf unzähligen Rechnern installiert – laut unterschiedlichen Untersuchungen auf den Servern, auf denen bis zu einem Viertel der Top 1000 der meistfrequentierten Webseiten liefen. Um den Fehler loszuwerden, mussten Administratorinnen und Administratoren lediglich die OpenSSL-Version auf ihren Servern auf den damals aktuellen Stand (1.0.1g oder neuer) bringen. Zur Sicherheit war es außerdem erforderlich, installierte private Schlüssel auszutauschen, da man sie als kompromittiert betrachten musste. Laut einer Untersuchung verwendeten einige Monate nach Bekanntwerden der Lücke jedoch nur 14 % der betroffenen Webseiten neue, definitiv sichere Zertifikate.

Selbst im November 2020 waren laut einer anderen Untersuchung immer noch um die 200.000 Webserver von der Lücke betroffen. Kein Mensch kann auch nur annähernd schätzen, wie viele Passwörter oder persönliche Daten über die Sicherheitslücke abgegriffen wurden, geschweige denn, welchen wirtschaftlichen Schaden der Fehler unter dem Strich verursacht hat. *Kleiner Fehler, globale Wirkung.*

Wie konnte es dazu kommen?

Wie sich herausstellte, hatte ein einzelner Programmierer den Bug schon im Jahr 2012 versehentlich eingebaut. Er hatte es versäumt, die Größe eines Eingabewerts auf Plausibilität zu prüfen, und ermöglichte es so, über die Verbindung mehr Daten aus dem Speicher zu lesen als eigentlich vorgesehen (*buffer over-read*), darunter auch Daten, die eigentlich nicht in fremde Hände gelangen sollten.

Es handelt sich bei OpenSSL um eine Open-Source-Bibliothek, bei der im Idealfall die beteiligte Entwickler-Community Änderungen auf Fehler hin überprüft. Das wurde aber offenbar versäumt. Letztlich war der Grund dafür laut Aussagen der Beteiligten Personalmangel, was bei einem Freiwilligenteam vielleicht verständlich ist – ich kann Ihnen aber versichern, dass mangelhafte Qualitätssicherung aufgrund knappen Personals auch in großen Unternehmen keine Ausnahme ist.

Abbildung 1.1 Der Heartbleed-Bug macht noch heute Hunderttausende Webserver angreifbar.

Mehr zum Heartbleed-Bug

Common-Vulnerabilities-and-Exposures-Bezeichnung:

CVE-2014-0160

Wikipedia-Seite zum Thema:

https://de.wikipedia.org/wiki/Heartbleed

Report über betroffene Webseiten mit Stand November 2020:

https://isc.sans.edu/diary/26798

Der genaue Bugfix im GitHub-Repository von OpenSSL:

https://git.openssl.org/gitweb/?p=openssl.git;a=commitdiff;h=96db902

Was läuft eigentlich falsch?

Eine Menge. Fangen wir mit den Programmiererinnen und Programmierern an. Sei es jetzt der arme Tropf, dem der Heartbleed-Bug unterlief, oder irgendjemand anderes irgendwo auf der Welt: *Wir alle machen Fehler*. Sie, Ihre Kollegin mit zehn Jahren Berufserfahrung, ich. *Jeder*.

1.4 Weltuntergang verhindern – aber wie?

Menschen sind nicht perfekt. Folglich ist auch von Menschen produzierte Software nicht perfekt. Fehler zu machen, ist menschlich – und unvermeidbar! Folglich müssen Software-Entwickler, Testerinnen und andere Beteiligte dafür sorgen, dass möglichst jeder ihrer Fehler entdeckt und entfernt wird. Sich für perfekt zu halten, den eigenen Code also für fehlerlos, ist schon der erste Fehler und der Anfang der Katastrophe. Klingt deprimierend? Ist es aber nicht.

Akzeptieren Sie Ihre Schwäche – und verhalten Sie sich entsprechend. Wenn Sie wissen, dass Sie Fehler machen, dann wissen Sie auch, dass Sie Maßnahmen ergreifen müssen, um sie möglichst zu vermeiden bzw. sie da, wo das nicht gelingt, umgehend auszumerzen. Machen Sie aus der unvermeidlichen Schwäche eine Stärke, indem Sie Fehler aus der Welt schaffen, von denen andere nicht einmal zugeben würden, dass sie ihnen möglicherweise unterlaufen sein könnten. Dieses Buch wird Ihnen eine ganze Reihe von Hilfsmitteln dazu an die Hand geben: Konventionen, Entwurfsmuster, Strategien und Tools.

Es sollte für jede Entwicklerin und jeden Entwickler eine Frage des Berufsethos sein, fehlerfreien Code zu produzieren. Gemeint ist damit aber nicht, dass alles, was Sie an Code eintippen, auf Anhieb perfekt funktioniert. Gemeint ist, dass Sie nach eingehender Prüfung und entsprechenden Tests den Code so weit verbessert haben, dass er tatsächlich tut, was er soll (und sonst nichts!). Das erfordert, dass Sie Ihre eigene Arbeit ständig infrage stellen. Seien Sie nie zufrieden, versuchen Sie immer, etwas zu verbessern. Schauen Sie zwei- oder dreimal hin.

Werfen Sie einen Blick auf den folgenden Java-Dreizeiler:

```
String s = "Zugriff erlaubt";
s.replace("erlaubt", "verweigert");
System.out.println(s);
```

Auf den ersten Blick wird dieses kleine Programm das Wort »erlaubt« durch »verweigert« ersetzen und folgende Ausgabe erzeugen:

```
Zugriff verweigert
```

Tut es aber nicht.

Finden Sie den Fehler? Suchen Sie ruhig eine Weile. Das gehört zu Ihrer Arbeitszeit dazu! Sie werden nicht nur dafür bezahlt, Code einzutippen, sondern auch dafür, seine Qualität sicherzustellen! Selbst wenn Ihnen Java-Expertenwissen fehlt, können Sie zumindest ahnen, wo der Bug versteckt ist.

Übrigens habe ich diesen Fehler nicht frei erfunden, sondern ihn vor Jahren rein zufällig in einer recht umfangreichen Software gefunden, die bei einem ziemlich großen Telekommunikationsunternehmen im Einsatz war. Ob der Fehler letztlich zu Problemen wie falsch verbuchten SMS-Kosten oder dergleichen geführt hat, vermag ich wegen der hohen Komplexität der ganzen Lösung nicht zu sagen. Das macht den Fehler aber nicht harmloser, sondern zu einem perfekten Beispiel: Dass er unentdeckt blieb, zeigt, dass die zuständigen Entwicklerinnen und Entwickler sich nicht die

Bohne darum geschert haben, ob der fragliche Code korrekt funktioniert. Sie haben ihn nicht ordentlich getestet, nicht gegengelesen und eventuelle, indirekt dadurch verursachte Fehler nicht auf die tatsächliche Ursache zurückgeführt.

Damit sind wir gleich bei der nächsten Baustelle angekommen: Wie können Sie herausfinden, ob fehlerhafter Code überhaupt eine Auswirkung hat, ob er irgendwo getestet wird (in diesem Fall unzureichend) oder tatsächlich niemals benutzt wird und demzufolge schlicht gelöscht gehört?

Im konkreten Fall stand die fehlerhafte Zeile (die mit dem `replace()`-Aufruf, was Sie sicher schon ahnen) in irgendeinem `if`-Block, dessen Eingangsbedingung möglicherweise niemals `true` wurde.

Ich stand vor einer kniffligen Frage: Konnte ich es wagen, den Fehler zu korrigieren? Ich hatte keine Möglichkeit zu prüfen, ob die Änderung nicht vielleicht einen völlig anderen Fehler produzierte. Es war nämlich durchaus denkbar, dass sich irgendeine andere Funktion auf das (falsche) Resultat der fraglichen Operation verließ. Unwahrscheinlich, aber nicht auszuschließen. Undurchschaubare Abhängigkeiten dieser Art sind in komplexen Lösungen an der Tagesordnung. Sie dürfen meist nicht einfach irgendwo etwas ändern und dann hoffen, dass der Rest weiterhin wie zuvor funktioniert.

Glücklicherweise gab es in dem fraglichen Projekt immerhin eine Quellcodeverwaltung, sodass ich den Autor der defekten Zeile persönlich befragen konnte. Wie sich herausstellte, saß er mir direkt gegenüber. Tatsächlich erkannte er auch auf den zweiten Blick den Fehler nicht; ich musste ihn erst auf die offizielle Dokumentation der `String`-Klasse verweisen. Am Ende tat er meine Frage, ob ich den Fehler korrigieren solle, mit einem desinteressierten Schulterzucken ab.

Ich korrigierte den Fehler, nachdem mir mehrere Kollegen versichert hatten, dass Nebenwirkungen unwahrscheinlich seien. Von negativen Folgen wurde mir später nichts bekannt. Glück gehabt!

Um solche Schwierigkeiten zu überleben, beschäftigt sich dieses Buch nicht nur mit sauberem Programmcode und ordentlichen Tests, sondern auch mit Themen wie Testabdeckung und Umgang mit »schwierigen« Kolleginnen und Kollegen.

Natürlich möchte ich Ihnen die Auflösung des Rätsels nicht vorenthalten. Die Funktion `replace()` der Klasse `String` verändert das eigentliche Objekt `s` nicht, sondern gibt das Resultat der Ersetzen-Operation *als neues Objekt* zurück. Die Zeile müsste also wie folgt lauten:

```
s = s.replace("erlaubt", "verweigert");
```

Die Java-Klasse `String` ist *immutable*, keine ihrer Funktionen ändert also das Objekt selbst. Eine moderne Entwicklungsumgebung blendet übrigens die zugehörige Dokumentation ein, wenn Sie den Mauscursor über das `replace` halten.

Programmiersprachen in diesem Buch

Sie werden feststellen, dass die meisten Programmcodebeispiele in diesem Buch in Java verfasst sind. Das ist derzeit die verbreitetste Sprache in der Industrie. Die meisten Beispiele sähen in C, C++ oder C# sehr ähnlich aus. Wenn Sie eine dieser Sprachen beherrschen, werden Sie jedes Beispiel verstehen. Es mag sein, dass es in anderen Sprachen spezifische Abweichungen gibt, letztlich aber lassen sich alle Hinweise mehr oder weniger direkt anwenden.

An dieser Stelle hilft nur das nötige Wissen, um den Fehler zu vermeiden. Oft sind es aber Konventionen, saubere Architektur und ordentliche Tests, um die Qualität von Software zu steigern. Dieses Buch wird Ihnen dabei helfen, solche Verbesserungen konkret umzusetzen, sei es im technischen oder auch im organisatorischen Bereich. Wenn Sie ein Entwicklungsteam leiten, kennen Sie vermutlich schon die meisten Maßnahmen, können diese mithilfe des Buchs jedoch vielleicht aktualisieren oder erweitern. Wenn Sie Entwicklerin oder Entwickler sind, möglicherweise sogar recht neu im Job, wird Ihnen dieses Buch helfen, besseren Code zu produzieren. Werben Sie in Ihrem Team für die vielen Konventionen, Tools und Strategien, die ich Ihnen hier vorstellen werde.

Ich kann freilich nicht jeden Aspekt mit maximalem Tiefgang ausloten; wo sinnvoll, verweise ich auf Webseiten mit weiterführenden Informationen.

Begeben Sie sich jetzt in aller Ruhe an die Lektüre, und am Ende werden Sie sicher eine bessere Entwicklerin bzw. ein besserer Entwickler sein. Ich wünsche Ihnen viel Erfolg und gute Unterhaltung!