Vorwort

Bei uns in Dänemark zählt Ga-Jol zu den beliebtesten Süßigkeiten. Sein starker Lakritzduft ist ein perfektes Mittel gegen unser feuchtes und oft kühles Wetter. Der Charme, den Ga-Jol für uns Dänen entfaltet, liegt auch an den weisen oder geistreichen Sprüchen, die auf dem Deckel jeder Packung abgedruckt sind. Ich habe mir heute Morgen eine Doppelpackung dieser Köstlichkeit gekauft und fand darauf das folgende alte dänische Sprichwort:

Ærlighed i små ting er ikke nogen lille ting.

Ehrlichkeit in kleinen Dingen ist kein kleines Ding. Dies war ein gutes Omen. Es passte zu dem, was ich hier sagen wollte. Kleine Dinge spielen eine Rolle. In diesem Buch geht es um bescheidene Belange, deren Wert dennoch beträchtlich ist.

Gott steckt in den Details, sagte der Architekt Ludwig Mies van der Rohe. Dieses Zitat erinnert an zeitgenössische Auseinandersetzungen über die Rolle der Architektur bei der Software-Entwicklung und insbesondere in der Agilen Welt. Bob und ich führen gelegentlich heiße Diskussionen über dieses Thema. Und ja, Mies van der Rohe war sehr an der Nützlichkeit und der Zeitlosigkeit der Formen des Bauens interessiert, auf denen großartige Architektur basiert. Andererseits wählte er auch persönlich jeden Türknopf für jedes Haus aus, das er entworfen hatte. Warum? Weil kleine Aufgaben eine Rolle spielen.

Bei unserer laufenden »Debatte« über TDD haben Bob und ich festgestellt, dass wir beide der Auffassung sind, dass die Software-Architektur eine wichtige Rolle bei der Entwicklung spielt, obwohl wir wahrscheinlich verschiedene Vorstellungen davon haben, was genau das bedeutet. Doch solche Differenzen sind relativ unwichtig, weil wir davon ausgehen können, dass verantwortungsbewusste Profis am Anfang eines Projekts eine gewisse Zeit über seinen Ablauf und seine Planung nachdenken. Die Vorstellungen der späten 1990er-Jahre, dass Design nur durch Tests und den Code vorangetrieben werden könnte, sind längst passé. Doch die Aufmerksamkeit im Detail ist ein noch wesentlicherer Aspekt der Professionalität als Visionen im Großen. Erstens: Es ist die Übung im Kleinen, mit der Profis ihr Können und ihr Selbstvertrauen entwickeln, sich an Größeres heranzuwagen. Zweitens: Die kleinste Nachlässigkeit bei der Konstruktion, die Tür, die nicht richtig schließt, die missratene Kachel auf dem Fußboden oder sogar ein unordentlicher Schreibtisch können den Charme des größeren Ganzen mit einem Schlag ruinieren. Darum geht es bei sauberem Code.

Dennoch bleibt die Architektur nur eine Metapher für die Software-Entwicklung und insbesondere die Phase der Software, in der das anfängliche Produkt etwa in derselben Weise entsteht, wie ein Architekt ein neues Gebäude hervorbringt. In der heutigen Zeit von *Scrum* und *Agile* geht es hauptsächlich darum, ein Produkt schnell auf den Markt zu bringen. Die Fabrik soll mit höchster Kapazität Software produzieren. Nur: Hier geht es um menschliche Fabriken, denkende und führende Programmierer, die eine Aufgabenliste abarbeiten oder sich bemühen, anhand von Benutzer-Stories ein brauchbares Produkt zu erstellen. Ein solches Denken wird von der Metapher der Fabrik dominiert. Die Produktionsaspekte der Herstellung von Autos in Japan, also einer von Fließbändern dominierten Welt, haben viele Ideen von *Scrum* inspiriert.

Doch selbst in der Automobilindustrie wird der Hauptteil der Arbeit nicht bei der Produktion, sondern bei der Wartung geleistet – oder bei dem Bemühen, sie zu vermeiden. Bei der Software werden die 80 oder mehr Prozent unserer Tätigkeit anheimelnd als »Wartung« bezeichnet, ein beschönigendes Wort für »Reparatur«. Statt uns also auf typische westliche Weise auf die *Produktion* guter Software zu konzentrieren, sollten wir anfangen, eher wie ein Hausmeister bei der Gebäudeinstandhaltung oder wie ein Kfz-Mechaniker bei der Reparatur von Autos zu denken. Was haben japanische Managementweisheiten dazu zu sagen?

Etwa 1951 erschien ein Qualitätsansatz namens Total Productive Maintenance (TPM; »Umfassendes Management des Produktionsprozesses«, der Ausdruck wird nicht ins Deutsche übersetzt) in der japanischen Szene. Er konzentrierte sich weniger auf die Produktion, sondern mehr auf die Wartung. Eine der fünf Säulen des TPM ist der Satz der so genannten 5S-Prinzipien. 5S bezieht sich auf einen Satz von Disziplinen oder Tätigkeitsbereichen. Tatsächlich verkörpern diese 5S-Prinzipien die Bedeutung von *Lean* – einem anderen Modewort in westlichen Produktionskreisen, das zunehmend auch in der Software-Entwicklung verwendet wird. Diese Prinzipien sind nicht optional. Wie Uncle Bob auf der Titelseite sagt, erfordert die Software-Praxis eine gewisse Disziplin: Fokus, Aufmerksamkeit und Denken. Es geht nicht immer nur darum, aktiv zu sein und die Fabrikausrüstung mit der optimalen Geschwindigkeit zu betreiben. Die 5S-Philosophie umfasst die folgenden Konzepte:

- *Seiri* oder Organisation (Eselsbrücke: »sortieren«). Zu wissen, wo sich Dinge befinden, ist erfolgsentscheidend. Dazu zählen zum Beispiel Ansätze wie das Vergeben geeigneter Namen. Wenn Sie meinen, die Benennung Ihrer Bezeichner sei nicht wichtig, sollten Sie auf jeden Fall die folgenden Kapitel lesen.
- *Seiton* oder Ordentlichkeit (Eselsbrücke: »aufräumen«). Ein altes Sprichwort sagt: *Ein Platz für alles, und alles an seinem Platz*. Ein Code-Abschnitt sollte da stehen, wo Sie ihn zu finden erwarten; und falls er nicht da steht, sollten Sie ein Refactoring von Ihrem Code vornehmen, so dass er danach an der erwarteten Stelle steht.

- *Seiso* oder Sauberkeit (Eselsbrücke: »wienern«). Sorgen Sie dafür, dass der Arbeitsplatz frei von herumhängenden Drähten, Müllspuren, Einzelteilen und Abfall ist. Was sagen die Autoren hier über die Vermüllung Ihres Codes mit Kommentaren und auskommentierten Codezeilen, die den Verlauf der Entwicklung oder Wünsche für die Zukunft wiedergeben? Weg damit.
- *Seiketsu* oder Standardisierung. Die kommt zu einem Konsens darüber, wie der Arbeitsplatz sauber gehalten werden soll. Hat dieses Buch etwas über einen konsistenten Codierstil und gemeinsam in der Gruppe befolgte Praktiken zu sagen? Wo kommen diese Standards her? Lesen Sie weiter.
- *Shutsuke* oder Disziplin (*Selbst*-disziplin). Dies bedeutet, die Disziplin aufzubringen, den Praktiken zu folgen, seine Arbeit regelmäßig zu überdenken und bereit zu sein, sich zu ändern.

Wenn Sie die Herausforderung – jawohl, die Herausforderung – annehmen, dieses Buch zu lesen und anzuwenden, werden Sie den letzten Punkt verstehen und schätzen lernen. Hier kommen wir schließlich zu den Wurzeln einer verantwortlichen professionellen Einstellung in einem Beruf, der sich um den Lebenszyklus eines Produktes kümmern sollte. So wie Automobile und andere Maschinen unter TPM vorbeugend gewartet werden, sollte eine Wartung im Falle eines Zusammenbruchs – also darauf zu warten, dass die Bugs sich zeigen – die Ausnahme sein. Stattdessen sollten Sie eine Ebene höher ansetzen: Inspizieren Sie die Maschinen jeden Tag und tauschen Sie Verschleißteile aus, bevor sie kaputtgehen, oder lassen Sie den sprichwörtlichen Ölwechsel vornehmen, um die Abnutzung des Motors möglichst zu verringern. Für Ihren Code bedeutet das: Nehmen Sie ein gnadenloses Refactoring vor. Sie können mit der Verbesserung noch eine Ebene höher ansetzen, wie es die TPM-Bewegung innovativ vor über 50 Jahren vorgemacht hat: Bauen Sie Maschinen, die von vornherein wartungsfreundlicher sind. Code lesbar zu machen, ist genauso wichtig, wie ihn ausführbar zu machen. Die ultimative Praxis, die in TPM-Kreisen etwa 1960 eingeführt wurde, besteht darin, die alten Maschinen vorbeugend durch vollkommen neue zu ersetzen. Wie Fred Brooks anmahnt, sollten wir wahrscheinlich Hauptteile unserer Software etwa alle sieben Jahre von Grund auf erneuern, um die schleichende Ansammlung von *Cruft* (Jargon: Müll, Staub, Unrat) zu beseitigen. Vielleicht sollten wir die von Brooks genannte Zeitspanne drastisch reduzieren und nicht von Jahren, sondern von Wochen, Tagen oder gar Stunden sprechen. Denn dort finden sich die Details.

Details haben eine mächtige Wirkungskraft. Dennoch hat dieser Ansatz etwas Bescheidenes und Grundsätzliches, so wie wir es vielleicht stereotyp von einem Ansatz erwarten, der japanische Wurzeln für sich in Anspruch nimmt. Aber dies ist nicht nur eine köstliche Art, das Leben zu sehen. Auch die westliche Volksweisheit enthält zahlreiche ähnliche Sprichwörter. Das Seiton-Zitat von oben floss auch aus der Feder eines Priesters in Ohio, der Ordentlichkeit buchstäblich »als Gegenmittel für jede Art von Bösem« sah. Was ist mit Seiso? *Sauberkeit kommt gleich nach Gött-*

lichkeit. So schön ein Haus auch sein mag, ein unordentlicher Tisch raubt ihm seinen ganzen Glanz. Was bedeutet Shutsuke in diesen kleinen Dingen? *Wer an wenig glaubt, glaubt an viel*. Wie wär's damit, das Refactoring des Codes rechtzeitig durchzuführen, um seine Position für folgende »große« Entscheidungen zu stärken, als die Aufgabe aufzuschieben? *Ein Stich zur rechten Zeit erspart dir neun weitere. Wer früh kommt, mahlt zuerst. Was du heute kannst besorgen, das verschiebe nicht auf morgen.* (Dies war die ursprüngliche Bedeutung des Ausdrucks »der letzte tragbare Moment« bei Lean, bevor er in die Hände von Software-Beratern fiel.) Was ist mit der Ordnung eines Arbeitsplatzes im Kleinen im Vergleich zum großen Ganzen? *Auch die größte Eiche wächst aus einer kleinen Eichel.* Oder wie ist es damit, einfache vorbeugende Arbeiten in den Alltag einzubauen? *Vorsicht ist besser als Nachsicht. Ein Apfel am Tag hält den Doktor fern*. Sauberer Code anerkennt die verwurzelte Weisheit unserer Kultur im Allgemeinen, oder wie sie einmal war, oder wie sie sein sollte, oder wie sie sein *könnte*, indem er den Details die schuldige Aufmerksamkeit schenkt.

Selbst in der Literatur über große Architektur greifen Autoren auf Sprichwörter über die Bedeutung von Details zurück. Denken Sie an die Türgriffe von Mies van der Rohe. Das ist Seiri. Das bedeutet, sich um jeden Variablennamen zu kümmern. Sie sollten Variablennamen mit derselben Sorgfalt auswählen wie den Namen eines erstgeborenen Kindes.

Jeder Hausbesitzer weiß, dass eine solche Pflege und fortwährende Verbesserung niemals zu Ende ist. Der Architekt Christopher Alexander – Vater der Patterns und Pattern-Sprachen – betrachtet jeden Design-Akt selbst als einen kleinen, lokalen Akt der Reparatur. Und er betrachtet die Anwendung des Könnens auf feine Strukturen als den einzigen legitimen Arbeitsbereich des Architekten; die größeren Formen können den Patterns und deren Anwendung durch die Bewohner überlassen werden. Design geht immer weiter. Es betrifft nicht nur den Anbau neuer Räumlichkeiten, sondern auch profanere Aufgaben wie ein neuer Anstrich, das Ersetzen eines abgelaufenen Teppichs oder die Modernisierung der Spüle in der Küche. In den meisten Künsten werden ähnliche Überzeugungen vertreten. Bei unserer Suche nach anderen, die das Haus Gottes in den Details sehen, stießen wir beispielsweise auf den französischen Autor Gustav Flaubert aus dem 19. Jahrhundert. Der französische Dichter Paul Valery sagt uns, ein Gedicht wäre niemals fertig und müsse laufend überarbeitet werden; mit dieser Arbeit aufzuhören wäre vergleichbar damit, dieses Gedicht zu verwerfen. Eine solche Besessenheit von Details ist allen Bemühungen gemeinsam, die auf Exzellenz, also auf herausragende Leistungen gerichtet sind. Also: Vielleicht gibt es hier wenig Neues, aber wenn Sie dieses Buch lesen, werden Sie herausgefordert, die guten Disziplinen wieder aufzunehmen, die Sie vor langer Zeit aus Apathie, dem Wunsch nach Spontanität oder einfach deswegen aufgegeben haben, weil Sie »etwas anderes« machen wollten.

Leider betrachten wir solche Überlegungen normalerweise nicht als Grundbausteine der Kunst der Programmierung. Wir entlassen unseren Code früh aus unse-

rer Obhut, nicht, weil er fertig ist, sondern weil unser Wertesystem mehr auf die äußere Erscheinung als auf die Substanz des gelieferten Produkts gerichtet ist. Diese fehlende Aufmerksamkeit kommt uns letztendlich teuer zu stehen: Irgendwann machen sich die Defekte immer bemerkbar. Weder in akademischen Kreisen noch in der Industrie begibt sich die Forschung in die niedrigen Ebenen hinunter, Code sauber zu halten. Früher, als ich noch bei der Bell Labs Software Production Research Organization (also wirklich Produktion!) beschäftigt war, machten wir die beiläufige Entdeckung, dass ein konsistenter Stil beim Einrücken von Klammern der statistisch signifikanteste Indikator für eine geringe Fehlerhäufigkeit war. Wir wollen, dass die Architektur, die Programmiersprache oder irgendein anderes hoch angesiedeltes Konzept die Ursache für Qualität sein soll. Als Entwickler, deren vorgebliche Professionalität auf der Meisterung von Werkzeugen und abgehobenen Design-Methoden basiert, fühlen wir uns von dem Mehrwert beleidigt, den diese Maschinen aus der Fabrikhalle, pardon!, Codierer, erzielen, indem sie einfach einen bestimmten Stil der Einrückung von Klammern konsistent anwenden. Um mein eigenes Buch zu zitieren, das ich vor 17 Jahren geschrieben habe: Ein solcher Stil unterscheidet Exzellenz von bloßer Kompetenz. Die japanische Weltsicht versteht den entscheidenden Beitrag jedes gewöhnlichen Arbeiters und mehr noch, wie Entwicklungssysteme von den einfachen gewöhnlichen Aktionen dieser Arbeiter abhängen. Qualität ist das Ergebnis eine Million selbstloser Akte der Sorgfalt – nicht nur das einer großartigen Methode, die vom Himmel gefallen ist. Dass diese Akte einfach sind, bedeutet nicht, dass sie simplizistisch (einfältig) sind. Es bedeutet auch nicht, dass sie leicht sind. Dennoch sind sie der Stoff, aus dem die Größe und mehr noch die Schönheit menschlicher Anstrengungen gemacht ist. Wer diese Akte ignoriert, hat noch nicht sein volles menschliches Potenzial realisiert.

Natürlich befürworte ich immer noch, auch den größeren Rahmen zu betrachten und besonders den Wert von architektonischen Ansätzen zu berücksichtigen, die in einem tiefen Wissen sowohl des Problembereiches als auch der Software-Usability verwurzelt sind. Doch darum geht es in diesem Buch nicht – zumindest nicht vordergründig. Dieses Buch will eine subtilere Botschaft verbreiten, deren Gewichtigkeit nicht unterschätzt werden sollte. Sie passt zu dem gegenwärtigen Credo wirklich codebasierter Entwickler wie Peter Sommerlad, Kevlin Henney und Giovanni Asproni. Ihre Mantras lauten: »Der Code ist das Design« und »Einfacher Code«. Während wir darauf achten müssen, dass die Schnittstelle das Programm ist und dass ihre Struktur viel über die Struktur unseres Programmes aussagt, ist es von erfolgsentscheidender Bedeutung, dass wir uns laufend mit der bescheidenen Einstellung identifizieren, dass das Design tatsächlich nur im Code existiert. Und während eine Überarbeitung in der Metapher der fabrikmäßigen Produktion zu höheren Kosten führt, führt sie beim Design zu einem Mehrwert. Wir sollten unseren Code als den wunderschönen Ausdruck edler Design-Anstrengungen betrachten – Design als Prozess, nicht als statischer Endpunkt. Nur im Code lassen sich letztlich die architektonischen Metriken der Kopplung und Kohäsion nachweisen. Wenn Larry Constantine Kopplung und Kohäsion beschreibt, spricht er von Code

– nicht von abgehobenen abstrakten Konzepten, die man vielleicht in UML findet. Richard Gabriel rät uns in seinem Essay *Abstraction Descant*, Abstraktion sei böse. Code ist anti-böse, und sauberer Code ist vielleicht göttlich.

Zurück zu meiner kleinen Packung Ga-Jol: Ich glaube, dass es wichtig ist, anzumerken, dass uns die dänische Weisheit nicht nur rät, unsere Aufmerksamkeit auf die kleinen Aufgaben zu lenken, sondern auch, bei kleinen Aufgaben ehrlich zu sein. Dies bedeutet, wir müssen dem Code gegenüber ehrlich sein, wir müssen unseren Kollegen gegenüber über den Zustand unseres Codes ehrlich sein und vor allem, wir müssen uns selbst gegenüber über den Zustand unseres Codes ehrlich sein. Haben wir unser Bestes gegeben, um den »Campingplatz sauberer zu hinterlassen, als wir ihn gefunden haben«? Haben wir das Refactoring des Codes erledigt, bevor wir ihn eingecheckt haben? Dies sind keine nebensächlichen Belange, sondern Belange, die zum Kern agiler Werte gehören. Zum Beispiel empfiehlt *Scrum*, das Refactoring zu einem Bestandteil des Konzepts »Done« (»Fertig«) zu machen. Weder Architektur noch sauberer Code verlangt nach Perfektion, sondern nur nach Ehrlichkeit und dem Bemühen, unser Bestes zu geben. Irren ist menschlich, vergeben göttlich. Bei *Scrum* machen wir alles sichtbar. Wir zeigen unsere schmutzige Wäsche. Wir sind ehrlich über den Zustand unseres Codes, weil Code niemals perfekt ist. Wir werden menschlicher, werden würdiger, das Göttliche zu empfangen, und nähern uns der Größe in den Details.

In unserem Beruf brauchen wir verzweifelt alle Hilfe, die wir bekommen können. Wenn ein sauberer Fabrikboden die Anzahl der Unfälle reduziert und wohlgeordnete Werkzeugkästen die Produktivität verbessern, dann kann man dies nur befürworten. Was dieses Buch angeht: Es ist die beste pragmatische Anwendung von Lean-Prinzipien auf Software, die ich je in Druckform gesehen habe. Aber weniger hatte ich von dieser praktischen kleinen Gruppe denkender Individuen auch nicht erwartet, die sich nicht nur seit Jahren darum bemühen, immer besser zu werden, sondern auch ihr Wissen an die Branche weiterzugeben. Dieses Buch ist ein Ausdruck dieses Bemühens. Es hinterlässt die Welt ein wenig besser, als ich sie vorfand, bevor Uncle Bob mir das Manuskript schickte.

Doch genug von meinen abgehobenen Einsichten. Ich muss meinen Schreibtisch aufräumen.

James O. Coplien

Mørdrup, Dänemark