1 Komplexe Systeme führen zu Fehlern

»Das einzige Mittel, den Irrtum zu vermeiden, ist die Unwissenheit.«[1] Für uns bietet diese Erkenntnis kaum eine Möglichkeit, Fehler zu vermeiden. Also wollen wir uns der Problematik von Fehler und Test aktiv stellen. Wir sind beileibe nicht chancenlos bei unserem Kampf für bessere Software.

Komplexe Systeme sind analytisch in begrenzter Zeit nur unvollständig erfassbar. Fehler sind damit zwangsläufig die Folge. Ganz allgemein betrachtet gibt es dafür vier Gründe:

- Kommunikationsprobleme
  - extern, also zwischen Menschen
  - intern, also im internen Kommunikationsprozess bei der Transformation von Sprache in Verständnis
- Gedächtnisprobleme
- Fachliches Problemverständnis
- Hohe Komplexität der Softwarelösung

Betrachten wir die vier Bereiche ruhig noch etwas genauer.

1.1 Kommunikation

Ein einfaches Kommunikationsmodell beschreibt Kommunikation als Folge von Transformationen [87]. Dabei kann es bei jeder Transformation zu Verlusten im Informationsgehalt kommen. Dies erfolgt sowohl zwischen Personen wie auch innerhalb eines Menschen (Abb. 1.1).

Jede Wahrnehmung ist subjektiv. Dazu kommt das Ausfiltern von Informationen aufgrund der physikalischen Einschränkungen unserer Wahrnehmung. Das Wahrgenommene wird dann transformiert und als Erinnerung im Gehirn abgelegt. Bei dieser Transformation helfen uns unsere bisherigen Erfahrungen. Außerdem sind wir begrenzt durch unsere individu-

[1] Jean-Jacques Rousseau (1712–1778), Schriftsteller und Kulturphilosoph, aus: *Emilie*, ca. 1762.

elle Auffassungsgabe. Beides hilft uns beim schnellen Erfassen, filtert aber erneut Informationsgehalt aus.

Bei der Retransformation aus unserem Gehirn in extern Kommunizierbares, also z. B. Sprache, bleibt erneut einiges auf der Strecke. Besonders bewusst wird uns dies, wenn wir nicht in unserer Muttersprache, sondern in einer Fremdsprache kommunizieren müssen. Wir spüren förmlich, wie Informationsgehalt liegen bleibt. Bei unserem Gesprächspartner spielen sich natürlich dieselben Prozesse ab.

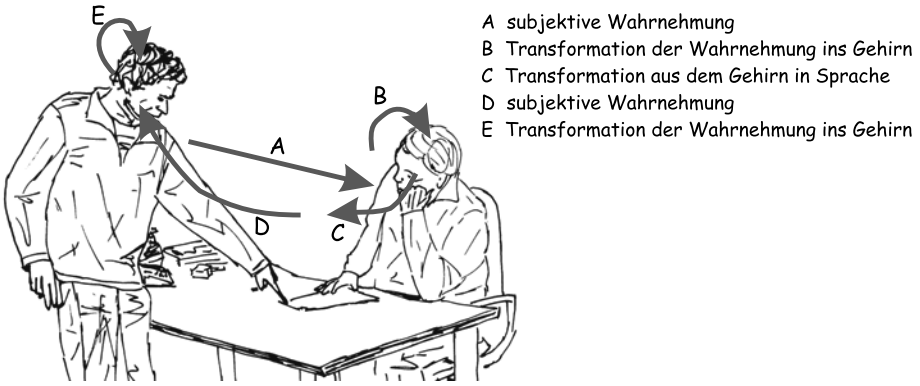

Abbildung 1.1: Ein einfaches Kommunikationsmodell nach Shannon und Weaver [87, 94]

Wir technisch geprägten Menschen reduzieren Kommunikation häufig auf den reinen Informationsgehalt. Dies wird als inhaltlich-sachliche Ebene der Kommunikation bezeichnet. Daneben gibt es aber noch drei weitere Ebenen der Kommunikation, die der Geschäftsordnung, der sozialen Beziehungen und des Unbewussten (Abb. 1.2) [2, 76]. Dieses Kommunikationsmodell wird grafisch in Form eines Eisbergs dargestellt und entsprechend genannt. Schauen wir uns die Ebenen des Eisbergmodells genauer an.

Sachebene: fachlicher Inhalt, Ziele, Verstand, Aufgaben. Wir transportieren hier die Antworten nach dem *was*.

Geschäftsordnung: Befugnisse, Entscheidungsverfahren, Standards oder Regeln. Mit der Geschäftsordnung beantworten wir die Frage nach dem *womit*.

Soziale Beziehungen: Gefühle, Erwartungen, Ängste, Anerkennung, Offenheit, Vertrauen.

Unbewusstes: Träume, Visionen, unbewusste Ängste oder tief sitzende Verhaltensmuster.

Abbildung 1.2: Eisbergmodell: Kommunikation spielt sich parallel auf vier Ebenen ab [94].

Das Besondere am Eisbergmodell ist weniger die Tatsache der Existenz der vier Ebenen, sondern deren Anteil an der Wichtigkeit für den Transport von Information. Die Sachebene spielt mit ca. 10 % nur eine untergeordnete Rolle, der Anteil unterhalb der Wasserlinie macht dagegen über die Hälfte aus. In diesem Zusammenhang gibt es zwei Regeln, die uns im täglichen Leben von Nutzen sein können.

1. Offensichtliche Kommunikationsprobleme auf einer Ebene haben ihre versteckte Ursache häufig in der darunter liegenden Ebene.
2. Kommunikationsprobleme müssen auf der Ebene gelöst werden, auf der sie verursacht werden.

Natürlich könnte man ein eigenes Buch über das Eisbergmodell schreiben.[2] Wichtig ist mir hier, dass wir ein erstes Gefühl bekommen, warum es so schnell zu Kommunikationsproblemen kommen kann, obwohl inhaltlich doch eigentlich alles gesagt wurde. Durch das Verletzen von Konventionen und Regeln oder z. B. durch arrogantes Verhalten verderben wir uns die Möglichkeit, andere Menschen zu überzeugen. Der unbewusste Anteil ist dabei erheblich; manchmal können wir eine bestimmte Person einfach nicht erreichen und nur, weil wir sie durch unser Aussehen und unseren Habitus an ihren alten Chef erinnern, von dem sie nach lautem Streit entlassen wurde.

[2] In [94] und z. T. in [95] gehe ich gemeinsam mit meinen Co-Autoren auf diese Aspekte der Soft Skills tiefer ein.

1.2 Gedächtnis

Es gibt Techniken, das Gedächtnis zu stärken oder durch Mnemotechniken zu verbessern. Trotzdem können wir uns nicht alles merken. Leider gelingt es uns aber auch nur begrenzt, alles aufzuschreiben und so vor dem Vergessen zu retten. Wir können nur die wichtigsten Dinge dokumentieren.

Dabei sollten wir auch immer einen pragmatischen Kompromiss finden zwischen Aufwand, Umfang und Inhalten, wobei ein besonderes Augenmerk auf der Wartung von Dokumenten liegen muss, um nicht entweder andauernd unsere Dokumente ändern zu müssen oder aber schnell veraltende Texte vorzufinden, die keinen relevanten Praxiswert mehr haben. Die Kunst besteht also im Rahmen der Softwareentwicklung darin, durch den Code, im Code und durch codierte Testfälle die lokale Dokumentation durch die Arbeitsergebnisse selbst zu erreichen und in externen Dokumenten die übergreifenden Zusammenhänge und langfristig gültigen Aspekte zu behandeln. Selbst wenn uns das gelingt, brauchen wir die Detailinformationen in unseren Köpfen. Und dort verhält es sich wie mit einer Festplatte, auf der von jemand anderem Dateien gelöscht werden und wir kein Backup haben. Es passieren Fehler.

1.3 Fachlichkeit

Wir sind technische Experten, stecken in den Tiefen unserer Programmiersprachen, Tools und Klassenbibliotheken. Methodisches Arbeiten in den Bereichen der Softwarearchitektur und des Designs sind uns geläufig, wir modellieren mit der Unified Modeling Language (UML) und können Tage bzw. Nächte damit verbringen, Code zu optimieren, die Performance eines Systems zu verdoppeln oder einen vertrackten Fehler durch Analyse des Stackdumps zu finden. Dafür sind wir ausgebildet, und darin haben wir Erfahrung.

Programmierung und die Erstellung von Software sind kein Selbstzweck. Auftraggeber verfolgen fachliche Ziele, und die Anwender und Fachabteilungen, mit denen wir es zu tun haben, um an die Anforderungen zu kommen, die wir umsetzen sollen, haben ganz andere Sorgen. Auch sie sind Fachexperten, nur leider auf ganz anderen Gebieten. So lange, wie wir uns bereits mit der Umsetzung von Entwurfsmustern oder der Implementierung von Mehrschichtarchitekturen befasst haben, haben sie in ihrer eigenen fachlichen Welt gearbeitet.

Um zu verstehen, was wir eigentlich tun sollen, müssten wir eigentlich selbst eine Banklehre gemacht, Versicherungs- oder Speditionskaufmann gelernt, ein Baustatik-, Jura- oder BWL-Studium absolviert haben. Haben wir aber nicht! Entwickler, die lange im selben Umfeld arbeiten, erarbeiten

sich nebenbei einen Großteil dieses Wissens, aber eben nicht alles. Warum ist das überhaupt ein Problem? Wir reden doch miteinander und die Anforderungsgeber sagen uns schon, was sie wollen. In der Praxis treffen wir dabei primär auf drei Probleme:

- Die Anforderungsgeber können nur schwer vermitteln, was sie wollen, da sie sich selbst darüber nur diffus im Klaren sind. Die Abstraktions- und Analysefähigkeit ist leider sehr unterschiedlich verteilt. Häufig muss dies auf Seiten der Entwicklung im Rahmen der Analyse geleistet werden. Jetzt ist aber die Gefahr groß, dass wir aus Unkenntnis fachlicher Zusammenhänge in der Abstraktion wichtige Details übersehen bzw. Zusammenhänge zu einfach sehen.
- Wir sprechen nicht die gleiche Fachsprache (Abb. 1.3). Es ist zwar immer noch Deutsch, aber wir verstehen es trotzdem nur begrenzt. Es findet also ein Transfer statt, bei dem Informationsgehalt verloren gehen kann. Diese Transferverluste können sich später als Fehler rächen. Gemindert werden kann dieser Verlust nur durch einen expliziten Übersetzer. Zusätzlich erfolgt im Rahmen der Analyse eine Abstraktion, die auch zu Fehlern führen kann.
- Die Anforderungsgeber wissen nicht, was technisch möglich ist, und können daher nur im Rahmen ihres technischen Horizonts Vorschläge machen. Um aber von unserer Seite aus Alternativen vorschlagen zu können, müssen wir erstmal verstanden haben, was die Anforderungsseite eigentlich will bzw. braucht. Da wir das aber nur schwer verstehen, drehen wir uns leicht im Kreis.

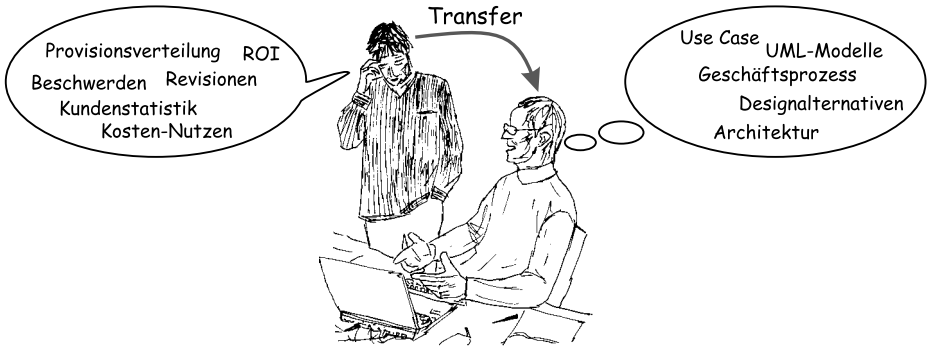

Abbildung 1.3: Bei der Kommunikation zwischen Fachabteilung und Entwicklung findet ein Transfer aus einer Fachsprache in eine andere statt.

1.4 Komplexität

Selbst wenn wir die drei zuvor behandelten Bereiche in den Griff bekommen, spielen uns unsere Aufgabenstellungen immer noch einen Streich. Softwareprojekte gehören zu den komplexesten Dingen, die Menschen versuchen. Leider sind wir nur begrenzt in der Lage, Komplexität zu überblicken. Wir versuchen unsere Aufgaben zu zerlegen oder zu modellieren und so die Gesamtkomplexität zu reduzieren. Dennoch werden uns die Vielfalt und die Abhängigkeiten der Anforderungen wie auch unsere technischen Möglichkeiten immer wieder Fehler bescheren. Durch die Zerlegung schaffen wir eben nur den Anschein, die Komplexität im Griff zu haben.

1.5 Erstes Fazit

Es bleibt uns also nichts anderes übrig, als zu akzeptieren, dass Fehler immer wieder gemacht werden, ja geradezu gemacht werden müssen! Oder anders formuliert: Wenn wir keine Fehler machen, treten wir auf der Stelle und kommen inhaltlich nicht voran.

Akzeptieren wir Fehler als notwendig in unserem Projekt-Lern-Prozess, so können wir sie bewusst dazu nutzen:

- Über Fehler können wir die kritischen Bereiche identifizieren, um sie dann genauer zu betrachten.
- Unser Entwicklungsprozess sollte sich aktiv und zu jeder Zeit den Fehlern stellen, sie suchen, finden und beheben.

Leider stellt sich uns das Problem der Fehler und ihrer Entdeckung noch wesentlich differenzierter dar. Fehler weisen Strukturen auf. Nicht jeden Fehler können wir zu jedem Zeitpunkt finden. Es muss daher differenzierte Testarten geben. Die grundsätzliche Einstellung ist dabei aber der Schlüssel zum Erfolg!

Wir akzeptieren Fehler und nutzen sie. Fehler sind nicht notwendigerweise ein Zeichen von Schwäche, sondern systemimmanent. Wenn wir das leugnen, werden wir scheitern. Die Einstellung gegenüber Fehlern, die *Fehlerkultur*, ist so wichtig, dass sie später noch differenzierter betrachtet wird.