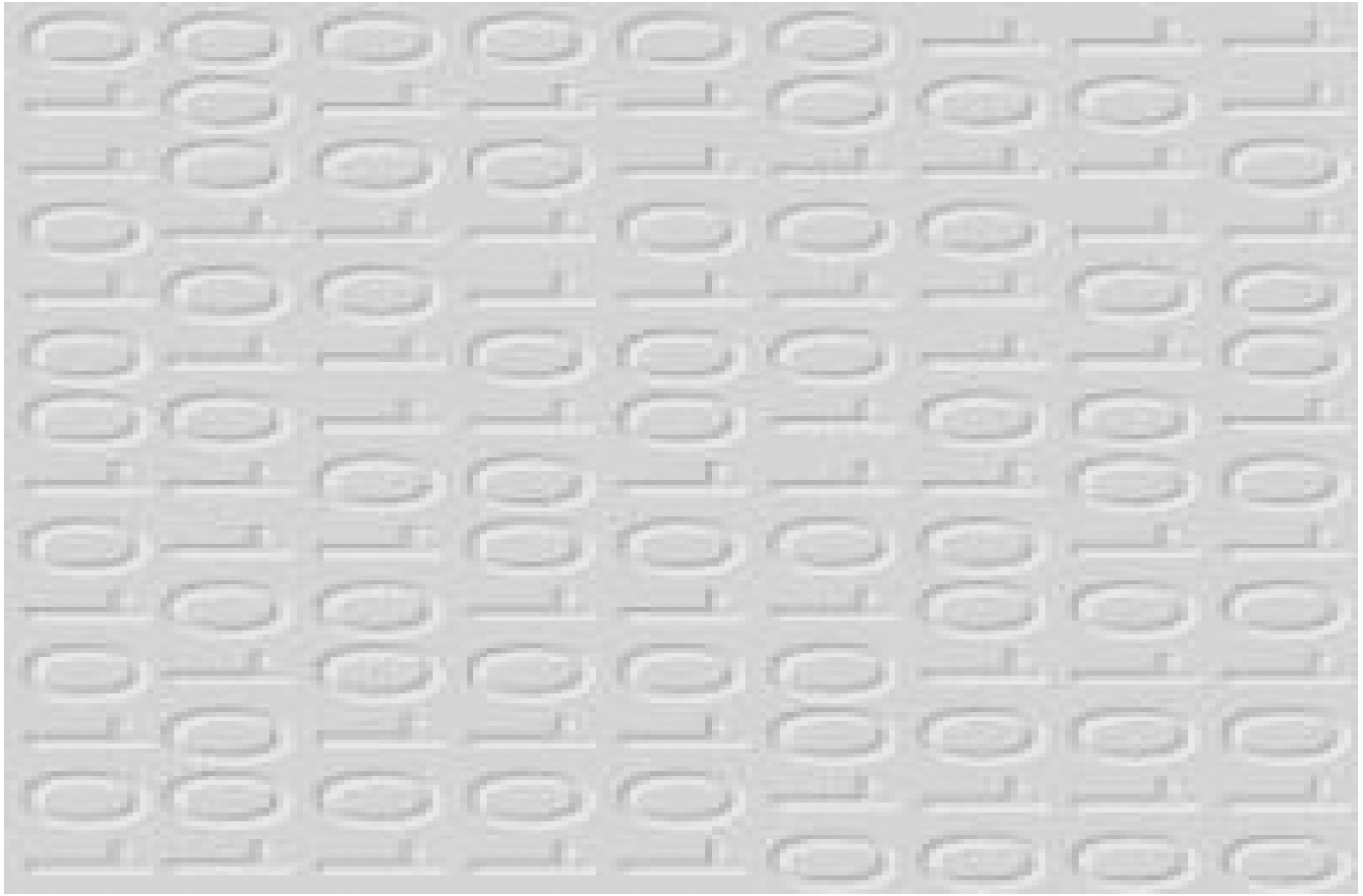


Kapitel 1

Kapitel 1

Symmetrie und Software: Die Suche nach Entwurfsmustern [1]

Symmetrische Software?

Ist es nicht vermessen, einen gehaltvollen griechischen Begriff einer anglo-amerikanischen Worthülse gegenüber zu stellen? „Symmetrisch" bedeutet *ebenmäßig* und *schön* – Software ist *häßlich* und *kraus!* Den Gehalt von „Software" beschreibt seit mehr als dreißig Jahren ein anderer griechischer Begriff: „Krisis".[2] Das Dauerphänomen Softwarekrise deutet auf fehlende Symmetrie, denn Symmetrie bezeichnet ein universelles Erfolgsprinzip in Natur, Technik und Kunst. Wie entwirft man nun *symmetrische Software*, will heißen: ebenmäßige und schöne? Der folgende Beitrag skizziert eine neue Hoffnung in der Informatik: Entwurfsmuster; sie sind die lang gesuchten Kandidaten für eine Didaktik des guten Entwurfs.

---

1 Veröffentlicht in der Zeitschrift „Diagonal"; Abdruck mit freundlicher Genehmigung der Redaktion [Quibeldey-Cirkel, 1996].

2 Die Disziplinbezeichnung „Software-Engineering" wurde 1967 von Friedrich L. Bauer geprägt und war ursprünglich *provokativ* gemeint: Die Softwarekrise geht gerade auf die mangelnde Ingenieurmäßigkeit (Engineering) im Software-Entwurf zurück, siehe [Naur & Randell (Hrsg.), 1969, S. 13].

Der Reihe nach: Was bedeutet Symmetrie eng und weit gefaßt? Warum ist Software kraus? Wie verhält sich die Symmetrie zum Schönen und Guten im Entwurf? Und wie sollte man den guten Software-Entwurf lehren?

1 Erscheinungsformen der Symmetrie

Spiegelgleichheit

Von der Flügelzeichnung eines Schmetterlings über Eisblumen und Schneeflocken bis zum Wechsel der Jahreszeiten: immer und überall erscheinen uns belebte und unbelebte Dinge *symmetrisch* – in Raum und Zeit. In der Umgangssprache assoziieren wir mit Symmetrie gewöhnlich die seitenumkehrende *Spiegelung*, die sogenannte *bilaterale* Symmetrie, von der Bild 1 ein Beispiel gibt. [3]

Bild 1:
Frühe Prägung durch bilaterale Symmetrie: „C'est la dissymétrie, qui crée le phénomène." [4]

Symmetrie versus Asymmetrie

Das Beispiel zeigt weit mehr: Der ästhetische Reiz der Symmetrie liegt nicht allein in ihr selbst, sondern entsteht erst im Verbund mit ihrem Widerpart, der *Asymmetrie*. Perfektes Ebenmaß wirkt langweilig und monoton (Plattenbau); der Turm von Pisa ist im Volksmund gerade wegen seiner Asymmetrie zur Umgebung bekannt. Die bilaterale Symmetrie ist deshalb so prägend, weil sie unsere erste Begegnung mit der Welt ist: Sie manifestiert sich im Gesicht der Mutter. Bilateral steht für das ausgewogene Paar *rechts-links*, und es ist kein Zufall der Evolution, wenn mehr als 95 % aller Tiere und wir selbst

3 Abdruck mit freundlicher Genehmigung unserer vierjährigen Laura

4 Pierre Curie, zitiert nach [Brandmüller, 1985, S. 221]

im wesentlichen eine Rechts-Links-Symmetrie besitzen. Die Schwerkraft diktiert *oben* und *unten*, der Fortbewegungstrieb *vorne* und *hinten* (hin zur Nahrung, weg von den Freßfeinden), aber rechts und links sind in der Fortbewegung gleichwertig. Hier erhalten wir auch einen ersten Hinweis auf die Zweckgebundenheit der Symmetrie. So viel zur alltagssprachlichen Einengung auf die „Spiegelgleichheit". Der Begriff hat aber längst eine viel größere Bedeutung erfahren.

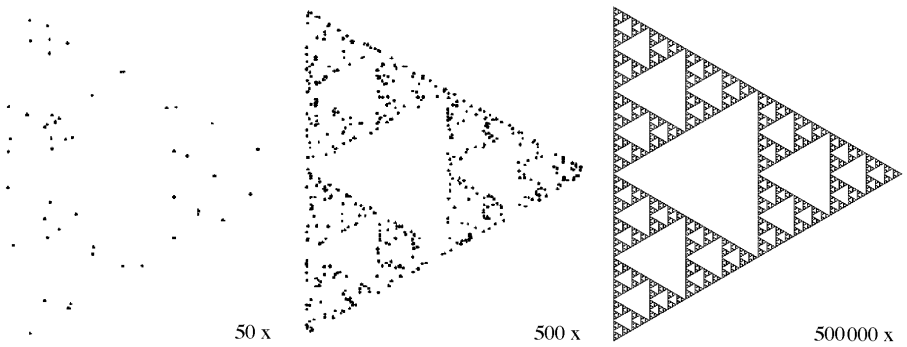


Bild 2:
Symmetrisches Fraktal
nach Wroclaw Sierpiński:[5]
„Plus ça change,
plus c'est la même chose."

Beharrende Form

Der Gruppentheoretiker argumentiert folgendermaßen: Das *Nichts* ist reine Symmetrie, das *Chaos* nicht viel weniger, und schließlich hat jedes Objekt zumindest eine, die *triviale* Symmetrie. Symmetrie ist auch nicht Gestalt, sondern *Bewegung:* Gewisse Eigenschaften eines (geometrischen) Objekts bleiben invariant bezogen auf gewisse Transformationen. Jeder Symmetrie ist eigen, daß eine bestimmte Form trotz Änderungen erhalten bleibt. Also: Das Nichts genügt allen Symmetrietransformationen in Raum und Zeit; es bleibt, was es ist – trotz Spiegelungen, Translationen, Drehungen und Kombinatio-

---

5 Der polnische Mathematiker Wroclaw Sierpiński veröffentlichte 1915 eine regelmäßige Figur, die aus einem *Zufallsprozeß* hervorgeht: Zeichne in einem gleichseitigen Dreieck einen beliebigen inneren Punkt. Wähle nun eine der drei Ecken *zufällig* aus und bewege den inneren Punkt zur Streckenmitte in Richtung dieser Ecke. Wiederhole den Zufallsprozeß mit dem neuen Punkt ad infinitum (im Bild ist die Zahl der Iterationen angegeben). Nach einem vorübergehenden Verhalten entsteht immer das Sierpiński-Dreieck; jedes der drei inneren Dreiecke besteht wiederum aus drei verkleinerten Kopien seiner selbst: sie sind alle *selbstähnlich* und unendlich reproduzierbar, siehe auch [Field & Golubitsky, 1993].

nen daraus. Das Chaos ist gar nicht so, wie es heißt, zumindest in der Mathematik birgt es in den symmetrischen Fraktalen eine phantastische Ordnung und einen ästhetischen Reiz: Bild 2. Und die triviale Symmetrie bezieht sich auf eine beliebige Transformation eines Objekts, wenn dieses seine Ursprungslage wieder einnimmt. Im Mittelpunkt unserer Aufmerksamkeit steht immer das, was sich bei Veränderungen nicht ändert – *die beharrende Form*.

Urbedeutung der Symmetrie

Lassen wir einen namhaften Vertreter des Symmetriegedankens den Begriff taxieren, den deutschen Mathematiker und theoretischen Physiker Hermann Weyl:

> „Symmetrie, ob man ihre Bedeutung weit oder eng faßt, ist eine Idee, vermöge derer der Mensch durch die Jahrtausende seiner Geschichte versucht hat, Ordnung, Schönheit und Vollkommenheit zu begreifen und zu schaffen."
>
> [Weyl, 1955, S. 13]

Das *Schaffen* von Ordnung und Schönheit setzt uns wieder in den Kontext von „Symmetrie und Software". Daß unser Schöpfer symmetrisch denkt, steht für viele Mathematiker, Biologen und Physiker außer Frage: Der interessierte Leser möge die umfangreiche Literatur betrachten, zum Beispiel [Golubitsky & Stewart, 1993]. Dort ist Platz für die Symmetriebeispiele in der belebten und unbelebten Natur. Zwei herausragende Wissenschaftler dürfen aber auch hier nicht fehlen: Albert Einstein und Eugene P. Wigner.

Physiker zum Symmetriebegriff

Der Physiker sucht die Antwort auf das *Warum* komplexer Phänomene im *Wie;* nicht: „Warum fällt der Apfel auf die Erde?", sondern: „Wie fällt er?" Dieses Wie verhalf Isaac Newton zu seinem Gravitationsgesetz. Albert Einstein fragte weiter: „Wie verhalten sich die Newtonschen Gesetze in Raum und Zeit, in ruhenden und bewegten Bezugssystemen?" Und Eugene P. Wigner faßte in seinem Nobelvortrag [Wigner, 1963] alles zusammen: Die naturwissenschaftliche Erkenntnis ist *hierarchisch:* auf der untersten Stufe die Naturgesetze, sie spiegeln die Ordnung der physikalischen Phänomene wider; auf einer höheren Stufe die Symmetriegesetze, in Form der Erhaltungssätze spiegeln sie die invarianten Naturgrößen wider, wie Energie, Impuls und Drehimpuls. Der Physiker bevorzugt einfache und schöne Modelle, um in der Flut der Phänomene *die beharrende Form* zu erkennen.

Symmetrie in Kunst & Technik

Wenn der Bauplan der Natur stets Symmetrieprinzipien genügt, wie sieht es dann mit uns irdischen Entwerfern aus? Entwerfen wir symmetrisch? Bedingt. Ein klares Ja, wenn es um Artefakte in der Architektur und der allgemeinen Technik geht (Fahr- und Flugzeuge, rotierende und translatorische Antriebe). Ein Ja-Nein, wenn es um Symmetrie in der Kunst geht (streng symmetrische Kompositionen wirken feierlich und lebensfremd, vielleicht mit Ausnahme der regelmäßigen Füllmuster in den Werken von M. C. Escher: Bild 3). Ein zögerndes Nein, wenn es um Artefakte aus Software geht – aber ein klares Ja im Fall der Hardware! Bekanntlich teilen die Informatiker die künstliche Welt in *soft* und *hard*. Der *technische* Informatiker (der Schreiber ist ein solcher) entwirft beides: entweder getrennt als hardware-nahe Software oder software-nahe Hardware oder aber gemeinsam als Hardware-Software-*Codesign*. Warum nun habe ich eingangs Software als *kraus* geschmäht? Fragen wir zunächst, warum Hardware so regelmäßig ist.

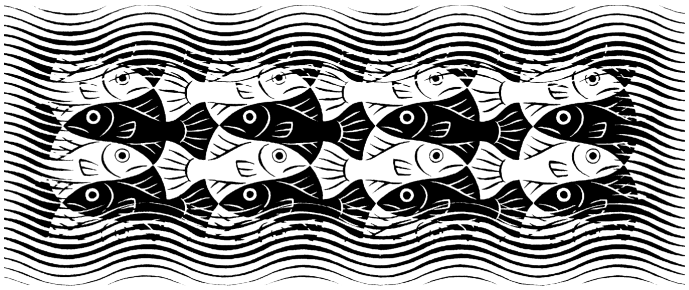


Bild 3:
M. C. Escher:
„regular division with fish“

[Escher, 1958, S. 234]

1.1 Ebenmäßige Hardware – Krause Software

Gibt es ein Moore-Gesetz des Software-Entwurfs?

Im Chipentwurf versucht der Hardware-Entwerfer, größere Strukturen durch Vervielfältigen kleinerer *additiv* aufzubauen. So kann er Mega-Bit-Speicherstrukturen aus feinstkörnigen Speicherelementen der Größe eines Bits *rapportierend* entwerfen. Gordon E. Moore, Mitbegründer der Firma Intel, stellte hierfür schon 1975 die Produktivitätsregel auf: Die Integrationsdichte der Halbleiterschaltungen verdoppelt sich alle zwei Jahre [Moore, 1975]. Seine Prognose hat sich bis zum neuesten Intel-Prozessor ausnahmslos bestätigt. Diese extreme Produktivitätssteigerung bei gleichzeitigem Preisverfall wurde jahrzehntelang den Software-Entwerfern vorgeführt, die dem nichts entgegensetzen konnten. Das hat seine Gründe in der Natur der „weichen

Ware“, die sich gänzlich unterscheidet: Software kann *nicht* additiv entworfen werden; sie ist stets ein *Geflecht* aus Datenstrukturen, Algorithmen und Prozeduraufrufen. Es gibt keine Standardprobleme, die allein mit konfektionierten Bausteinen nach dem Lego-Prinzip gelöst werden könnten. Auch läßt sich kein Programm auf der Grundlage einer minimalen Strukturgröße und eines Satzes von Entwurfsregeln *skalieren*.

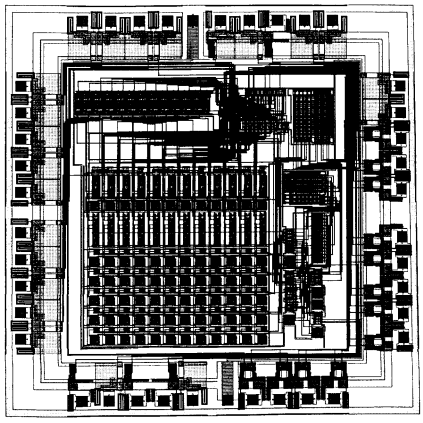


Bild 4: Erscheinungsformen der Hardware: Rhythmische Wiederholung ähnlicher Formen!

[Corbin & Snapp, 1988, S. 426]

Symmetriebruch: *krause* Software

Verliert ein Objekt an Symmetrie, gewinnt es an Struktur; so die biologische Zelle, die mit jeder Differenzierung an Symmetrie verliert, was sie an individueller Struktur gewinnt. Ähnlich entwickelt sich Software aus einem bescheidenen Repertoire von Symmetrie-Elementen: *Sequenz, Iteration* und *Selektion*. Sie sind die elementaren Ablaufstrukturen in einem Von-Neumann-Rechner, wie Sie ihn auf dem Schreibtisch stehen haben. Die Sequenz meint die *Reihung* von Rechneranweisungen: erst a_1, dann a_2; Iteration die *Schleife:* Solange die Bedingung b erfüllt ist, rechne Prozeß p; und Selektion meint die *Auswahl:* falls b erfüllt ist, rechne p_1, sonst p_2.

Es gibt sicherlich weitere Strukturelemente in den Hunderten von Programmiersprachen, *sprachspezifische*, wie Zeichenketten in Snobol, Listen in Lisp, Felder in APL, Fakten und Regeln in Prolog oder Klassen in Smalltalk. Man könnte auch *artspezifisch* unterscheiden nach Funktionen in funktionalen Sprachen, Prädikaten in logischen oder Objekten in objektorientierten Sprachen. Setzt man aber Tausende und Abertausende von Programmzeilen aus diesen Sprachelementen zusammen, *bricht* die Symmetrie, und die Strukturphänomene brechen durch, was jedem Programmierer bestens vertraut ist:

Die Software wird kraus. Im Gegensatz zur Hardware (Bild 4) wiederholen sich keine grobkörnigen Strukturelemente – und vor allem nicht *rapportierend,* das heißt Muster und Motive bildend.

1.2 Erscheinungsformen der Software

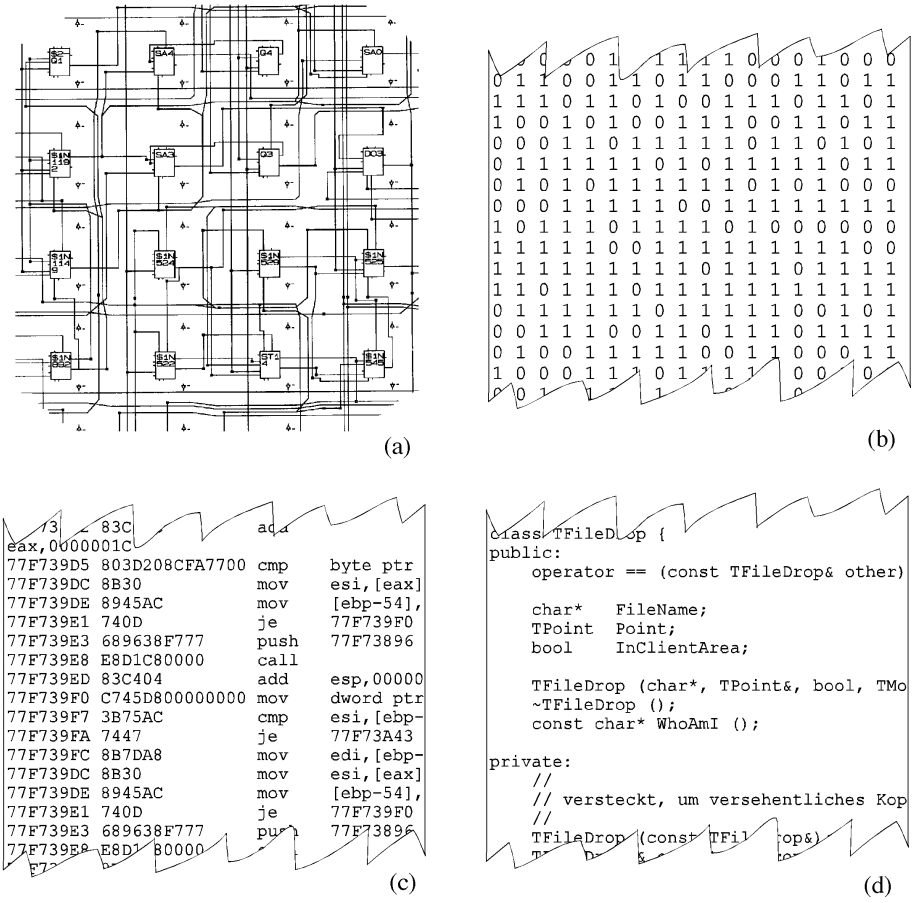
Gibt es symmetrische Software-Formen?

Herbert A. Simon (der einzige, der neben der höchsten Informatik-Auszeichnung, dem *Turing-Award,* auch den Nobelpreis erhielt) formulierte beredt die „verborgene Einfachheit" in der Natur; sein Credo lautet: „Complexity evolves from simplicity" [Simon, 1962]. Komplexe *stabile* Formen – ob biologisch, sozial oder unbelebt – entwickeln sich stets aus *einfachen* stabilen Zwischenformen. Wie aber sehen die Zwischenformen im Software-Entwurf aus? Je tiefer wir in die Artefakte aus Software dringen, desto komplexer und verwirrender sind die gesichteten Dinge, und von Stabilität kann kaum die Rede sein.

Auf der untersten Ebene der Programmierung codieren einfachste Software-Formen die binäre Schaltungslogik: Je nach Vereinbarung steht die 1 für eine elektrisch leitende Verbindung und die 0 für eine Unterbrechung. Die Ausschnitte (a) und (b) in Bild 5 auf der nächsten Seite illustrieren dies in Grafik und Text. Die Schaltungslogik ist in den regelmäßig angeordneten Kästchen von (a) verborgen, deren innere und äußere Verbindungen sind *frei programmierbar;* der Text aus Einsen und Nullen in (b) bestimmt die aktuelle „Verdrahtung". Das ist relativ neu und beschreibt einen Trend in der Schaltungstechnik: weg von den starren Strukturen, hin zu den flexiblen – Hardware wird *weich!* Die Grenzen zwischen der Flexibilität der langsamen Software und der Geschwindigkeit der inflexiblen Hardware verschwimmen.

Die Ausschnitte (c) und (d) spiegeln die traditionellen „höheren" Formen der Maschinenbefehle wider. Die Assembler-Form in (c) codiert die Ursprache aller Rechner *mnemotechnisch:* So werden aus Wörtern des binären Alphabets $\{0, 1\}$ sinnfällige Kürzel; und der Quelltext in (d) ist in einer Hochsprache der Moderne codiert.

Bild 5: Erscheinungsformen der Software: Rhythmische Wiederholung ähnlicher Formen?

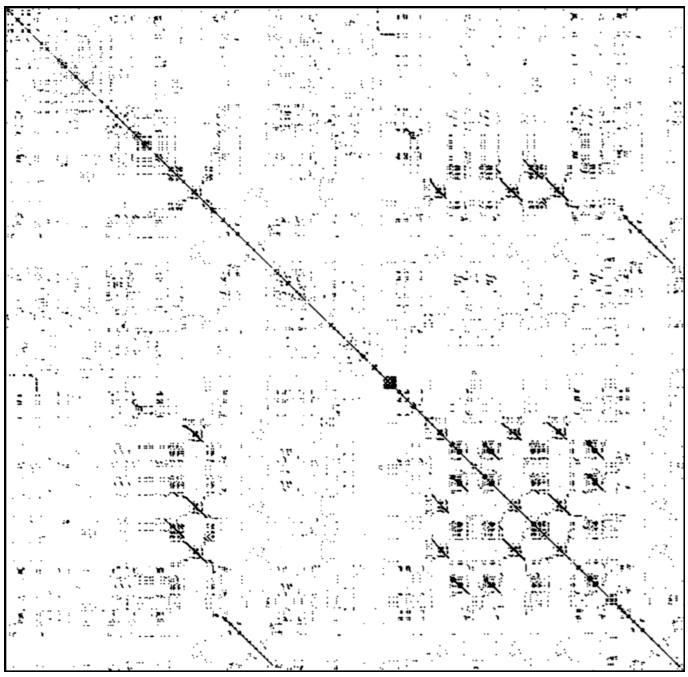


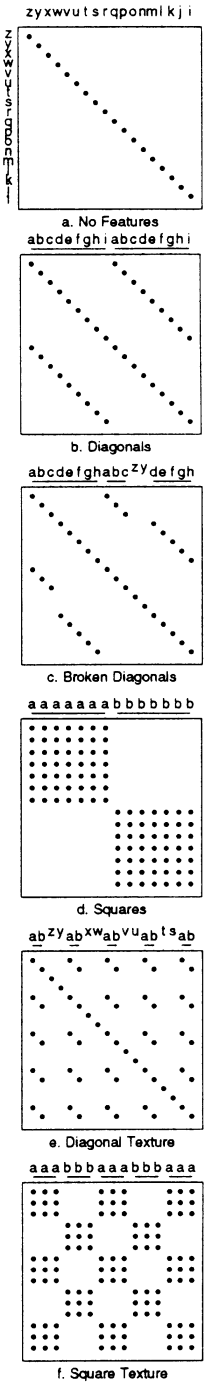
Gehen wir noch einen Schritt weiter und versuchen wir einmal, ein komplexes Programm als Ganzes zu visualisieren: Bild 6 zeigt als Beispiel das *Punktdiagramm*[6] eines 3.400zeiligen Programms aus der Telekommunikation [Church & Helfman, 1993]. Hier wird der Quelltext zeilenweise mit sich selbst verglichen und immer dann ein Punkt *(i, j)* gedruckt, wenn die Textzeile an der waagrechten Stelle *i* mit der Textzeile an der senkrechten Stelle *j* übereinstimmt. Die Legende zeigt anhand einfacher Buchstabenfolgen die Strukturen, die in einem Punktdiagramm prinzipiell auftauchen können. So entstehen Diagonale, Quadrate und Texturen, die auf ausgedehnte *selbstähn-*

6 Punktdiagramme *(dotplots)* sind eine bewährte Analysemethode in der Gentechnik, um gemeinsame Abschnitte in DNA-Folgen zu entdecken.

liche Strukturen deuten; „selbstähnlich“ meint hier aber nur einzelne Fragmente, nicht die Anordnung als Ganzes, wie wir es beim symmetrischen Fraktal von Sierpiński gesehen haben (siehe die Fußnote auf Seite 13).

Bild 6:
„Selbstähnliche“
Software-Strukturen





Auf all diesen Ebenen der Codierung werden wir den *symmetrischen* Anteil der Software – ihre *Proportionen* – schwerlich erkennen. Aber Software ist heute weitaus geordneter, als uns die Bilder suggerieren wollen. Wählen wir eine Strukturkörnigkeit, die heute üblich ist, nämlich das *Modul*, das eine Datenstruktur samt ihren Zugriffsoperationen kapselt, so treten anschaulichere Strukturphänomene auf: Bild 7 auf der nächsten Seite.

Wiederverwendbare Softwaremodule werden in *Bibliotheken* angeboten. Die Analogie zur Bücherverwaltung ist gewollt: Module können wie Bücher *recherchiert* werden, allerdings nicht nur über Signaturen, sondern auch und besonders über *Beziehungen*. Denn hier sind die Softwaremodule „Klassen“; sie beschreiben Objekte mit gemeinsamen Eigenschaften [Quibeldey-Cirkel, 1994b].

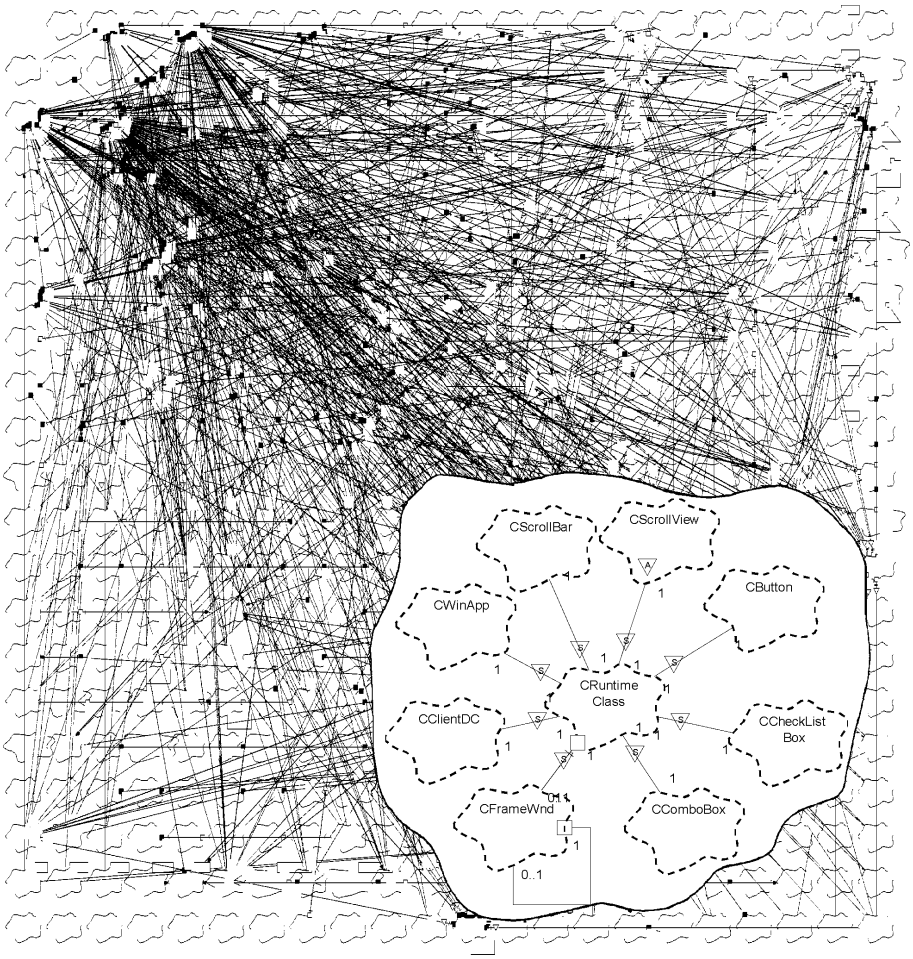


Bild 7: Organigramm einer Software-Bibliothek: Spuren von Symmetrie?

Für den Kenner: Zu „sehen" ist das Klassendiagramm der Microsoft-Bibliothek für grafische Oberflächen (MFC-Framework: ca. 500 Klassen, symbolisiert durch „Wölkchen"). Es wurde mit einem Reverse-Engineering-Werkzeug der Firma Rational generiert. Die Notation stammt von Grady Booch [Booch, 1994].

Symmetrien auf Modulebene?

Wie in allen Klassifikationssystemen gibt es Klassen, die allgemeiner sind als andere (Oberklassen); Klassen, die andere umfassen (Container), und assoziierte Klassen, die andere kennen, sich ihnen anbieten oder von ihnen profitieren (Client- und Server-Klassen). Solche Klassenbeziehungen nennt man entsprechend *Generalisierung*, *Aggregation* und *Dienstleistung*. Der vergrößerte Ausschnitt in Bild 7 zeigt einige der Beziehungstypen in einer konkreten Ausprägung. Können wir nun auf dieser, von der Ursprache der Rechner weitgehendst abstrahierten Formebene Symmetrien ausmachen? Wohl kaum. Es verdichtet sich hier lediglich das Beziehungsgeflecht, wobei die Verdichtung mit der jeweiligen Anordnung der Klassen variiert. Wenn nun

selbst auf der höchsten Organisationsstufe moderner Softwarekomponenten keine übergeordnete Symmetrie erkennbar ist, können wir dann überhaupt von „Invarianten" im Software-Entwurf sprechen, also von solchen anwendungsspezifischen Größen, die erhalten bleiben, auch wenn wir die Darstellungsform wechseln, wenn wir transformieren? Was ist denn invariant im Software-Entwurf, wenn es nicht einmal die *Vorstellung* vom zu gestaltenden Zweck ist? Bild 8 (ein zeitloses Klischee) erzählt hierzu „the same old story" der Software-Entwicklung.

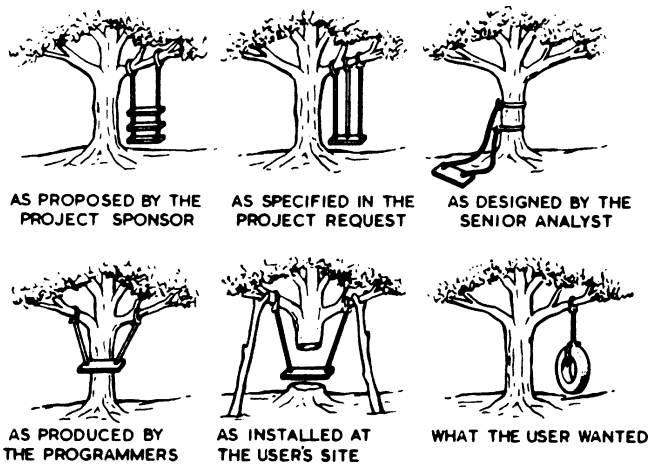


Bild 8:
Spiegelungen zwischen Wunsch und Wirklichkeit

[Brittan, 1980]

Sozio-organisatorisches Muster:
CONWAY'S LAW

Dieser Beitrag wäre nicht geschrieben worden, gäbe es da nicht invariante Aspekte, die sich im Sinne der Symmetrie auf die *beharrende Form* beziehen. Sie sind allerdings nicht geometrisch anschaulich wie die Beispiele aus den anderen Entwurfsdisziplinen. Wir lösen uns im folgenden von der geometrischen Anschaulichkeit und übernehmen den Symmetriebegriff des Mathematikers: Ein Objekt ist symmetrisch, wenn bestimmte Eigenschaften gegenüber bestimmten Transformationen invariant sind. In diesem Sinne gibt es eine erstaunliche Symmetrie zwischen der Organisation einer Entwicklungsabteilung und den aus ihr hervorgehenden Entwurfsprodukten: Melvin E. Conway hat diesen Symmetrie-Aspekt schon vor dreißig Jahren beschrieben [Conway, 1968]. Der gegenwärtige Trend zu „flachen Hierarchien und schlanken Organisationen" folgt unmittelbar Conways Symmetrie-Erkenntnis: Nur schlanke und flexible Organisationen schaffen schlanke und flexible Produkte. Beispiel: Eine Softwarefirma mit acht Programmierern sollte

Übersetzer für die Sprachen Cobol und Algol entwickeln. Nach Abschätzung der Komplexität und des Zeitaufwandes wurden fünf Programmierer für den Bau des Cobol- und drei für den Bau des Algol-Übersetzers zugewiesen. Symmetrie-Ergebnis: Der Cobol-Übersetzer lief in fünf Phasen, der Algol-Übersetzer in drei. Diese *Homomorphie* zwischen Organisation und Produkt liefert den Stoff, aus dem die Erfolgsgeschichte manch innovativer Softwarefirma geschrieben wurde, die als Garagenfirma begann und zum Konzern wuchs, Apple Computer zum Beispiel [Young, 1988].

Spezifikation der Software-Invarianten

Ein zweiter Symmetrie-Aspekt, der kurz erwähnt werden soll, ist die sogenannte *Invarianten*-Programmierung. Sie gewährleistet in der Sprache Eiffel von Bertrand Meyer, daß gewünschte Eigenschaften über *jede algorithmische* Transformation hinweg erhalten bleiben [Meyer, 1988]. Meyer nennt dies das „Zusicherungskonzept": In seiner Programmiersprache können Modulinvarianten explizit formuliert werden. Das sind formal-logische Aussagen über die Werte einer Datenstruktur, die stets wahr oder falsch sein sollen, egal welche Manipulationen (Rechenschritte) angewandt werden. Klassisches Lehrbeispiel: Das Gehalt des Abteilungsleiters soll stets höher sein als das Gehalt seines Mitarbeiters. Auch die Invarianten der Ein- und Ausgaben einer Prozedur können explizit formuliert werden: durch *Vor-* und *Nachbedingungen*. Die Prozedur wird nur ausgeführt, wenn die Vorbedingung erfüllt ist, sie löst eine Fehlerbehandlung aus, wenn sie die Nachbedingung verletzt. Die Invarianten eines Softwaremoduls werden zwischen Entwerfer und Anwender *vertraglich* vereinbart: „programming by contract".

Vom „guten Entwurf"

Es gibt sie also, die technischen Mittel, um Invarianz in der Software zu beschreiben (Eiffel avanciert wohl auch deswegen zur Lehrsprache der modernen Programmierung). Was aber sagt dem Software-Entwerfer, welche Eigenschaften er unbedingt invariant halten soll gegenüber allen algorithmischen Transformationen und in allen Phasen der Entwicklung – von der Analyse des Kundenwunsches über das Design der Software-Interna bis zur Laufzeit auf dem Rechner? Hier kommt der Begriff des „guten Entwurfs" ins Spiel! Der Kybernetiker Hans Sachsse lehrt uns den Zusammenhang zwischen Symmetrie und Ästhetik – Ästhetik, verstanden als die Wahrnehmung des Schönen und Guten [Sachsse, 1985].

2 Symmetrie und Ästhetik

Techno-philosophische Betrachtung zur Zweckgebundenheit der Symmetrie

Der griechische Begriff Ästhetik meint ursprünglich „Wahrnehmung". Wenn wir heute die Lehre vom *Schönen* mit diesem Begriff bezeichnen, sollten wir uns fragen, wie sich unsere Wahrnehmung zum Schönen, das heißt zur Symmetrie verhält. Um nicht zu sehr ins Philosophieren abzugleiten, beschränke ich mich auf den biologischen Ansatz: Die Wahrnehmungsorgane haben sich parallel zur Mobilität entwickelt, denn gemeinsam erlauben Mobilität und Wahrnehmung die ortsungebundene *Orientierung* nach besseren Umweltbedingungen. Die Wahrnehmung ist dabei stets selektiv; ihr Zweck ist nicht die Abbildung der Umwelt, sondern desjenigen Ausschnitts, der für die Befriedigung arterhaltender Bedürfnisse notwendig ist, wie Nahrung, Schutz und Fortpflanzung. Hans Sachsse nennt dies die „Prägnanzleistung" der Sinneserfahrung:

> „Die Abschirmung von dem Überflüssigen ist für das Wachstum und die Entwicklung der Individuen ebenso wichtig wie der Kontakt mit dem Notwendigen. Es gibt Tiere, die nach vollzogenem Geschlechtsakt den Partner nicht mehr wahrnehmen."
>
> [Sachsse, 1985, S. 7]

Aufgrund der Zweckgebundenheit unserer Wahrnehmung empfinden wir alles *Anziehende* als schön, alles *Abstoßende* als häßlich. Dem biologischen Ansatz zufolge

> „ist das Schöne das mit den äußeren Sinnen wahrnehmbare Merkmal des Guten; es zeigt den Weg, der zum Guten führt."

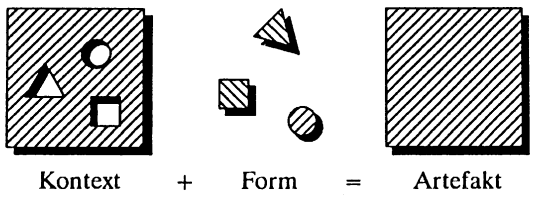
Warum ist die Symmetrie ein Merkmal des Schönen und somit ein „Merkmal des Guten"?

> „Weil das Regelmäßige, Maßhaltige, sich Wiederholende das Mittel für jede Orientierung ist. Ob unbewußt und rein instinktiv oder bewußt überlegt, immer benutzt die Orientierung diesen Schluß von der Vergangenheit auf die Zukunft, die durchhaltenden, konstanten Strukturen, die Invarianten. Die gesamte evolutionäre Entwicklung, diese aktive Anpassung durch Ausnutzung von Nischen, beruht auf

> Bedürfnisbefriedigung durch Einpassung in bestehende Ordnungen. Daher erleben wir das Regelmäßige, das die Erwartungen bestätigt, als erwünscht, vertraut und beruhigend und das Unregelmäßige als Verwirrung, die sich bis zum Schrecken und zur Panik steigern kann."

Ich zitiere hier Hans Sachsse ausführlich, weil er sich zeitlebens der Technikphilosophie verschrieben hat und daher für den technischen Entwerfer besonders glaubwürdig ist und weil er die Zweckgebundenheit der Symmetrie wie kaum ein anderer herausstellt. Zweckgebunden ist auch der allgemeine technische Entwurf, worauf schon der Architekt Christopher Alexander hingewiesen hat: Die „Synthese der Form" ist ökonomisch, wenn sie aus einer „Symmetrie der Form" hervorgeht. Symmetrie bedeutet für den technischen Entwurf in erster Linie *Ökonomie* – die Wiederverwendung von *Orientierungswissen*. Bild 9 zeigt symbolisch die Zweckgebundenheit zwischen Synthese und Symmetrie.[7]

Bild 9:
Symmetrie
als Orientierung:
„Fitting Form & Context"



Alexander umschreibt das Gemeinsame aller Syntheseprobleme mit den Worten „to achieve fitness between two entities: the form in question and its context" [Alexander, 1964]. Der *Kontext* definiert das Entwurfsproblem; die Lösung wird durch die *Form* repräsentiert. Entwerfen ist folglich ein *adaptiver* Vorgang, der dann ökonomisch ist, wenn er sich an Symmetriemerkmalen orientiert: „We want to put the context and the form into effortless contact or frictionless coexistence." [Alexander, 1964, S. 19].

Erstes Fazit

Fassen wir zusammen: Symmetrie ist zweckgebunden; wie der Zweck des Schönen in unserer Wahrnehmung zeigt sie uns den Weg zum Guten, ihre

---

7 Die Analogie zum Formenspiel geht auf eine Anregung unserer zweijährigen Linda zurück. Sie half ihrer nulljährigen Schwester Lotta bei den ersten entwurfstechnischen Gehversuchen.

zugrundeliegende Ordnung schafft Orientierung. Als Entwerfer widmen wir uns primär der Zweckgestaltung. Um in der Vielfalt der zweckvollen Formen den Überblick zu bewahren, sollten wir uns möglichst an Symmetrien orientieren. Was aber sind die Orientierungssymmetrien im Entwurf?

2.1 Die Invarianten des guten Entwurfs

Entwerfen = Suchen

Zunächst müssen wir uns im klaren sein, daß es den *optimalen* Entwurf im allgemeinen nicht geben kann; Entwerfen ist primär ein Suchproblem und kein Optimierungsprozeß. Gäbe es eine globale Zielfunktion, läge kein Entwurfsproblem, sondern eine *algorithmisch* lösbare Aufgabe vor. Einzelne Entwurfsschritte im globalen Suchraum können dagegen sehr wohl eine Aufgabe der Optimierung sein. Insgesamt aber bedeutet Entwerfen das *exploratorische* Suchen nach einer *zufriedenstellenden* Lösung, die der Menge von selbst- und fremdbestimmten Entwurfskriterien genügt. Die gefundene Lösung erhebt keinen Anspruch, „perfekt" oder „optimal" zu sein; Herbert A. Simon nennt ihre Eigenschaft „satisficing" – für die Suchprobleme der KI (Künstliche Intelligenz) ein bezeichnendes Kunstwort: „satisfice = satisfy + suffice".

gut = zufriedenstellend

Die Suche im Entwurfslabyrinth, der Menge aller theoretisch denkbaren Lösungsschritte, kann allein schon wegen der begrenzten Ressourcen nicht erschöpfend sein. Zum einen sind unsere *kognitiven* Möglichkeiten begrenzt: Einen exponentiellen Zustandsraum gedanklich aufzuspannen und systematisch sämtliche Lösungspfade abzuschreiten, übersteigt unsere Verarbeitungskapazität, siehe George Millers „Magische Zahl 7 ± 2" [Miller, 1956, auch auf der Buch-CD]. Zum anderen sind das Projektbudget und die Rechenleistung harte Randbedingungen; sie diktieren den Kompromiß nach dem Kosten-Nutzen-Verhältnis. Anstelle einer kombinatorisch erschöpfenden Suchstrategie leiten *Intuition* und *Heuristik* den Entwerfer: In der Entwurfstechnik nennt man diese Vorgehensweise „Versuch und Irrtum", in der Denkpsychologie „Schema und Korrektur" und in der Wissenschaftsphilosophie „Vermutungen und Widerlegungen" (Karl Popper). Stets gibt sich der Entwerfer mit einer Lösung zufrieden, die den vorgegebenen Beschränkungen – Suchraum, Suchzeit und Suchmittel – *genügt*. Im folgenden setze ich also „gut" mit „zufriedenstellend" gleich.

Streben nach „Wiedergabetreue“

Will man die Invarianten des guten Entwurfs finden, muß man deutlich zwischen *Essenz* und *Akzidens* unterscheiden. Frederick P. Brooks tat dies eloquent für die Softwaretechnik und prägte ein geflügeltes Wort der Informatik: „No Silver Bullet“ [Brooks, 1987]. Die Essenz umfaßt die im Wesen der Software liegenden, unveränderlichen Eigenschaften, das Akzidens lediglich die mit der Software-Entwicklung einhergehenden, zufälligen und veränderlichen Eigenschaften. Diese können im Projektverlauf zu einem *Monster* mutieren, gegen das es kein Mittel zu geben scheint.[8] Suchen wir die Invarianten des guten Entwurfs, sollten wir uns natürlich auf die Software-Essenz beschränken, um fündig zu werden.

Ontologie?

Die Essenz eines Sachverhalts beschreibt man wissenschaftlich objektiv und formal mit *ontologischen* Begriffen [Bunge, 1979]; sie sind unabhängig von der technischen Anwendung. Da *Informationssysteme* (mit deren Entwurf beschäftigt sich die Informatik hauptsächlich) *reale* Anwendungen modellieren sollen, so wie der Anwender sie *wahrnimmt*, ist die Ontologie prädestiniert für die Suche nach Invarianten; denn Ontologie ist die Wissenschaftsphilosophie von der Begriffs- und Ordnungsbestimmung der Dinge in der realen Welt. Um nun die interessierenden Merkmale eines Weltausschnitts einzufangen, bedarf es eines Maßes für *Wiedergabetreue:* Das Informationssystem (das ist Software!) soll die Essenz des wahrgenommenen Weltausschnitts *getreu* wiedergeben. Ein wiedergabegetreues Informationssystem, wenn es seinem Anspruch genügt, repräsentiert die Invarianten unabhängig von der angewandten Entwurfsmethode und unabhängig von der technischen Realisierung.

Ontologische Betrachtung zur Wiedergabetreue

Yair Wand bezeichnet den Aspekt der essenztragenden Entwurfsinvarianten als „Tiefenstruktur“ und den implementierungsbedingten akzidenstragenden Aspekt als „Oberflächenstruktur“ [Wand, 1989]. Mit Hilfe der ontologischen Begriffe „Zustand“, „Gesetz“ und „Ereignis“ können wir die Invarianten der Tiefenstruktur formal beschreiben: Hierzu notieren wir das Schema eines beliebigen Systems als Tripel $\langle Z, G, E \rangle$. Z steht für die Menge der Zustände, die das System und seine Teilsysteme einnehmen können. Ein Zustand wird durch die *aktuellen* Werte der Systemeigenschaften beschrieben. G steht für die Menge der Systemgesetze; sie umfassen zwei Arten der Information: eine *Bedingung*, ob ein Zustand stabil ist, und eine *Regel*, wie das Sy-

8 Über Illusionen und Ernüchterungen in der Softwaretechnik siehe „The Mythical Man-Month“ [Brooks, 1975]. Ein Bonmot daraus: „Adding man power to a late project makes it later.“

stem aus einem instabilen Zustand in einen stabilen wechselt. „Stabil" heißen Zustände $z \in Z$, wenn sie Fixpunkte eines Gesetzes sind $G(z) = z$; für „instabile" Zustände gilt: $G(z) \neq z$. Und E steht für die Menge der bedeutsamen externen Ereignisse. Das Adjektiv „bedeutsam" schränkt die Menge der möglichen Ereignisse so weit ein, daß die instabilen Zustände, hervorgerufen durch externe Ereignisse, mit wenigen Systemgesetzen in stabile Zustände überführt werden können. Ein Ereignis $e \in E$ löst einen Zustandswechsel aus: $\langle z_1, z_2 \rangle$ mit $z_i \in Z$, wobei z_1 den Zustand vor dem Wechsel bezeichnet und z_2 den Zustand danach. Mit diesen ontologischen Grundbegriffen können wir vier notwendige und hinreichende Bedingungen für Wiedergabetreue definieren:

1. Es existiert eine Abbildung *a* zwischen dem Zustandsraum Z_R des realen Systems und dem Zustandsraum Z_I des Informationssystems: $a : Z_R \rightarrow Z_I$. — Mapping

2. Für jeden Zustand des realen Systems $z_R \in Z_R$ gilt: $a(G_R(z_R)) = G_I(a(z_R))$. Die beiden ersten Bedingungen implizieren eine Strukturähnlichkeit zwischen den Zustandsräumen Z_R und Z_I. Mit anderen Worten: Das Informationssystem „weiß", wie es die Struktur des realen Systems nachbilden soll. Es weiß allerdings noch nicht, wie es dessen *Verhalten*, das heißt die Zustandsfolgen aufgrund externer Ereignisse, wiedergeben soll. Dazu müssen zwei weitere Bedingungen erfüllt sein: — Tracking

3. Sei $e_R \in E_R$ ein externes Ereignis des realen Systems: $e_R = \langle z_R, \tilde{z}_R \rangle$. Der Zustand $z_I \in Z_I$ des Informationssystems sei $z_I = a(z_R)$. Das reale System wird ein externes Ereignis $e_I \in E_I$ für das Informationssystem erzeugen, so daß gilt: $e_I = \langle z_I, \tilde{z}_I \rangle = \langle a(z_R), a(\tilde{z}_R) \rangle$. In diesem Fall spiegelt das externe Ereignis im Informationssystem das externe Ereignis im realen System wider. Mit der dritten Bedingung ist noch nicht sicher, daß das Informationssystem *zeitgleich* dem realen System folgt; die vierte Bedingung spezifiziert den Zeitbezug: — Reporting

4. Seien $\{ e_{R_i} \}$ die Menge der Ereignisse im realen System und $\{ e_{I_i} \}$ die Menge der zugeordneten Ereignisse im Informationssystem, dann müssen die Folgen $e_{R_1}, ..., e_{R_n}$ und $e_{I_1}, ..., e_{I_n}$ elementweise übereinstimmen. — Sequencing

Nehmen wir nun an, alle vier Bedingungen seien erfüllt. Das Informationssystem sei zurückgesetzt, um mit dem Anfangszustand des realen Systems übereinzustimmen *(Mapping)*, bevor dieses beginnt, die Ereignisse im Informationssystem zu beeinflussen. Wenn nun Ereignisse im realen System auftreten, wird es Ereignisse für das Informationssystem erzeugen *(Reporting)*. Diese Ereignisse werden in der gleichen Folge auftreten wie die ursprünglichen Ereignisse im realen System *(Sequencing)*. Die Natur dieser Ereignisse ist derart, daß aufgrund der *Tracking*-Bedingung das Informationssystem eine Zustandsmenge durchqueren wird, die der Zustandsmenge entspricht, die auch das reale System durchquert. Damit garantieren die vier genannten Bedingungen die Wiedergabetreue im Zustandsverhalten und damit die invariante Transformation der Tiefenstruktur aus der realen Anwendung in das Informationssystem.

Zweites Fazit

Mapping, Tracking, Reporting und *Sequencing* sind also Mechanismen, die das Laufzeitverhalten des Informationssystems *kongruent* mit dem gewünschten Verhalten der Anwendung halten: „What the user wanted“ (Bild 8 auf Seite 21). Sie sind somit die gesuchten *Symmetrietransformationen* des guten Entwurfs. Wie manifestieren sich die Invarianten des guten Entwurfs – die Tiefenstruktur aus Zuständen, Gesetzen und Ereignissen – in der Praxis? Letztlich beruht alles auf *Erfahrungen*, auf Entwurfsentscheidungen, die sich über die Anwendungen hinweg als erfolgreich und wiederverwendbar erwiesen haben. Wir kommen zur Kernfrage:

2.2 Wie entwirft man gute Software?

Blick zurück

Der Software-Entwurf wurde schon immer als *Handwerkskunst* verstanden – im guten Sinne als Kunsthandwerk: „Die Kunst des Programmierens“ [Knuth, 1974], im schlechten Sinne als Spezialistentum: ausgeklügelte Spaghetti-Programme, die kein anderer als der „Künstler“ selbst warten kann. Wie dokumentiert sich nun die *gute* Kunst des Handwerks?

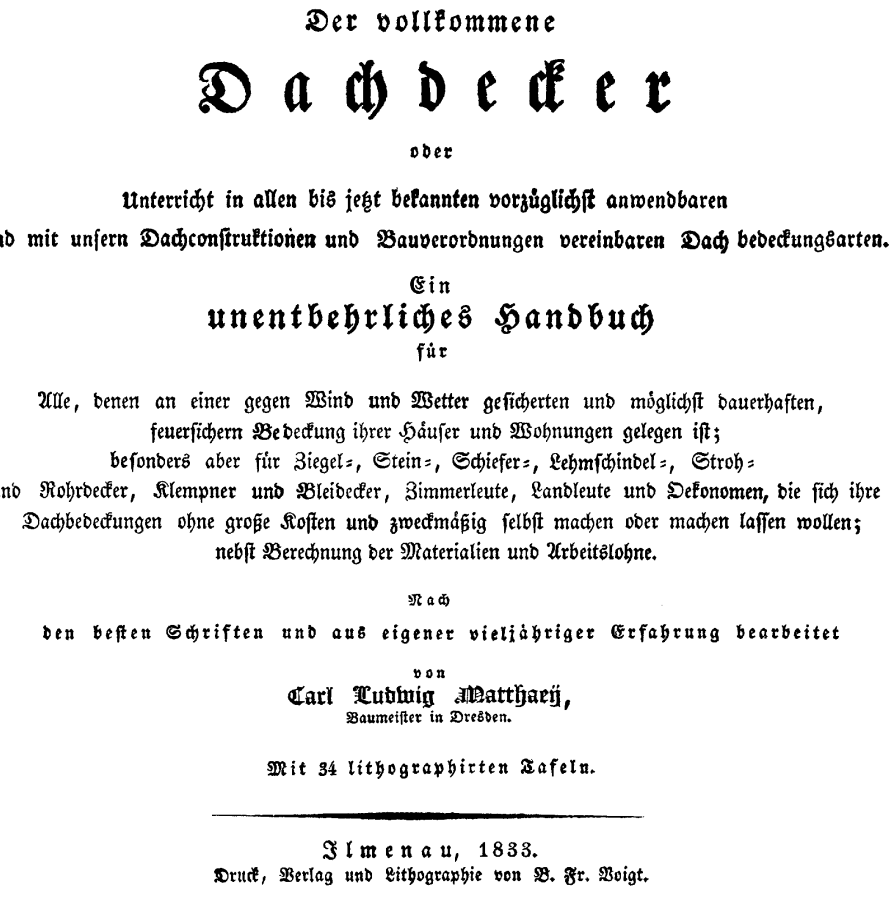
Neuer
Schauplatz der Künste
und Handwerke.
Mit
Berücksichtigung der neuesten Erfindungen.
Herausgegeben
von
einer Gesellschaft von Künstlern, Technologen und Professionisten.
Mit vielen Abbildungen.

Unter diesem Titel erschienen im Verlag Bernh. Friedr. Voigt, Weimar, von 1817 bis zum Ersten Weltkrieg über 300 Bände mit mehr als 600 Auflagen. Für die Entwicklung des Handwerks und der Industrie in Deutschland waren diese Werke von grundlegender Bedeutung; sie stellten ein Kompendium handwerklichen Wissens dar (meist *Hand-* oder *Musterbücher* genannt). In ihnen wurden die über die Jahrhunderte entwickelten und gesammelten Erfahrungen aus vielen Handwerksberufen festgeschrieben: „Man spürt, daß hier nicht Theorien verkauft wurden, sondern ein Praktiker seine Erfahrungen bekannt machte, und zwar so bekannt machte, daß sie wiederum leicht in die Praxis umzusetzen waren", kommentiert Manfred Gerner einen Band aus dem Jahr 1833, dessen Umschlagseite Bild 10 auf der nächsten Seite zeigt.

Historie der Musterbücher

Die literarische Gattung Musterbuch kann auf eine lange Tradition verweisen; sie hat ihre Anfänge in der Antike als technisches Traktat und im Mittelalter als annotiertes Bilderbuch. Musterbücher sind Sammlungen von Vorlagen, die dem bildenden Künstler des Mittelalters als *Exemplum* für seine eigenen Werke dienten. Ein Musterbuch enthält typisierte Einzelformen und Motive, die in einen beliebigen neuen Zusammenhang gebracht werden können. Das berühmteste Musterbuch schrieb der französische Baumeister Villard de Honnecourt um 1240: das *Bauhüttenbuch* (siehe Meyers Enzyklopädisches Lexikon). Es enthält Federzeichnungen von bedeutenden Kathedralen und kirchlichen Einrichtungsgegenständen, wie typisierten Figuren und Ornamenten, zusammen mit technischen Erläuterungen und Entwurfsnotizen.

Bild 10:
Umschlagseite
eines Musterbuchs
aus dem 19. Jahrhundert



Der vollkommene

Dachdecker

oder

Unterricht in allen bis jetzt bekannten vorzüglichst anwendbaren
und mit unsern Dachconstruktionen und Bauverordnungen vereinbaren Dach bedeckungsarten.

Ein
unentbehrliches Handbuch
für

Alle, denen an einer gegen Wind und Wetter gesicherten und möglichst dauerhaften, feuersichern Bedeckung ihrer Häuser und Wohnungen gelegen ist; besonders aber für Ziegel-, Stein-, Schiefer-, Lehmschindel-, Stroh- und Rohrdecker, Klempner und Bleidecker, Zimmerleute, Landleute und Oekonomen, die sich ihre Dachbedeckungen ohne große Kosten und zweckmäßig selbst machen oder machen lassen wollen; nebst Berechnung der Materialien und Arbeitslohne.

Nach
den besten Schriften und aus eigener vieljähriger Erfahrung bearbeitet
von
Carl Ludwig Matthaey,
Baumeister in Dresden.

Mit 34 lithographirten Tafeln.

Ilmenau, 1833.
Druck, Verlag und Lithographie von B. Fr. Voigt.

Auch die Gegenwart kennt Musterbücher, zum Beispiel das Buch von Maria Heller-Seitz: „Musterbuch für Seidenmalerei, Batik und Stoffdruck“ von 1986. Die Superlative im Bild 10 sind der Pragmatik gewichen: nicht der „vollkommene“, „vorzüglichst anwendbare“, sondern der *gute* Entwurf ist das Ziel. Auf der Suche nach dem guten Entwurf von Software hat nun ein Musterbuchschreiber der zeitgenössischen Architektur(!) den Software-Entwerfern die Richtung gewiesen: Christopher Alexander. Seine Werke „The Timeless Way of Building“ [Alexander, 1979] und „A Pattern Language“ [Alexander et al., 1977] werden derzeit von weitaus mehr Informatikern gelesen als Architekten. Die *Pattern-Bewegung* in der Informatik ist ein Phänomen ohnegleichen in ihrer Disziplin: Musterbücher für den Software-Ent-

wurf werden zu Bestsellern.[9] Jede Fachkonferenz der objektorientierten Softwaretechnik ist mit diesem Thema besetzt; eigene internationale Workshops gibt es seit Jahren, wie PLoP 94 …, EuroPLoP 96 …: „Pattern Languages of Programming". Die Fachgemeinschaft, die „Entwurfsmuster" oder „Design Pattern" zum Terminus technicus erhebt, kommuniziert via Internet; Brennpunkt ist die *Pattern-Homepage* an der University of Illinois.[10] Die theoretischen und praktischen Aspekte werden in einer Mailing-Liste diskutiert, die in der Frequentierung ihresgleichen sucht.[11]

2.3 Was nun sind Entwurfsmuster?

Beispiel: Strategiemuster

Blättern wir zurück auf die Seite 20: Bild 7 zeigt uns das Software-Inventar für den Entwurf einer interaktiven grafischen Benutzungsoberfläche. Zugegeben, die Darstellung ist verzerrt, nur der vergrößerte Ausschnitt gibt einige Modulbeziehungen lesbar wieder. Auch in dieser Ansicht wäre der Entwurf ohne Orientierung an Erfahrungswerten mühsam, ad hoc und fehleranfällig. Ein *mustergestützter* Entwurf könnte nun folgendermaßen verlaufen: Wir definieren zunächst unser Problem und suchen dann in einem Musterbuch nach geeigneten Vorlagen. Wir finden zum Beispiel ein *strategisches* Muster namens MODEL-VIEW-CONTROLLER und erfahren, daß es sich in zahlreichen interaktiven Grafikanwendungen bewährt hat. Wenige Seiten beschreiben das immer wiederkehrende Entwurfsproblem, ein bewährtes Lösungsschema und den typischen Anwendungskontext.

Die grafischen und textuellen Erläuterungen sind anschaulich und intuitiv. Wir erfahren, wie die Dreiteilung MODEL-VIEW-CONTROLLER die Grundprinzipien der Softwaretechnik umsetzt: das Modularisierungsprinzip *Information hiding* von David Parnas und das Strukturierungsprinzip *schwache Kopplung zwischen und starke Kohäsion in den Modulen* von Herbert A. Simon [Parnas, 1972; Simon, 1962]. Dazu werden wir auf *taktische* Muster verwiesen; sie ha-

---

9 Kultbuch der Software-Entwerfer ist das Musterbuch von Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides [Gamma et al., 1995].

10 http://hillside.net/patterns/patterns.html

11 http://hillside.net/patterns/Lists.html

ben dieselbe Ordnung: Musternamе, Synonyme und Einordnung, Zweck und Einsatzmotive, Anwendungs-Szenarien, Implementierungshinweise mit Vor- und Nachteilen, Beispiele und Nachweise über den erfolgreichen kommerziellen Einsatz, Querverweise und Abgrenzungen zu ähnlichen Mustern.

Beispiel: Taktikmuster

Ein solches Taktikmuster heißt PUBLISHER-SUBSCRIBER; es beschreibt die Wechselbeziehung zwischen den verschiedenen Sichten (VIEWS) und dem zentralen Datenbestand (MODEL). Das PUBLISHER-Objekt macht eine Datenänderung allen SUBSCRIBER-Objekten bekannt und veranlaßt deren Aktualisierung. Dabei sind die SUBSCRIBER-Objekte untereinander *entkoppelt* und somit austauschbar; sie beziehen sich ausschließlich auf das gemeinsame PUBLISHER-Objekt. Das Muster beschreibt also eine *1-zu-n*-Abhängigkeit zwischen Objekten. Darüber hinaus bindet es auch SUBSCRIBER-Objekte ein, die Daten visualisieren *und* ändern können, wie die Tabellenkalkulation. Der Bekanntmachungsmechanismus gewährleistet stets die Datenkonsistenz zwischen den Sichten.

Musterbücher des Software-Entwurfs unterscheiden sich hauptsächlich in ihrem *Abstraktionsgrad*. So gibt es Muster für die Analyse [Fowler, 1997], für den Programmentwurf (Bild 11) und für die Implementierung [Coplien, 1991]. Die Zukunft wird das Spektrum beträchtlich erweitern: Es werden bereits *sozio-organisatorische* Muster[12] diskutiert; sie beschreiben die unterschiedlichen Rollen, die Software-Entwerfer in großen Entwicklungsabteilungen einnehmen – auch Organisationen und ihre Arbeitsabläufe (work flows) werden entworfen.

12 http://www.bell-labs.com/people/cope/Patterns/Process/index.html

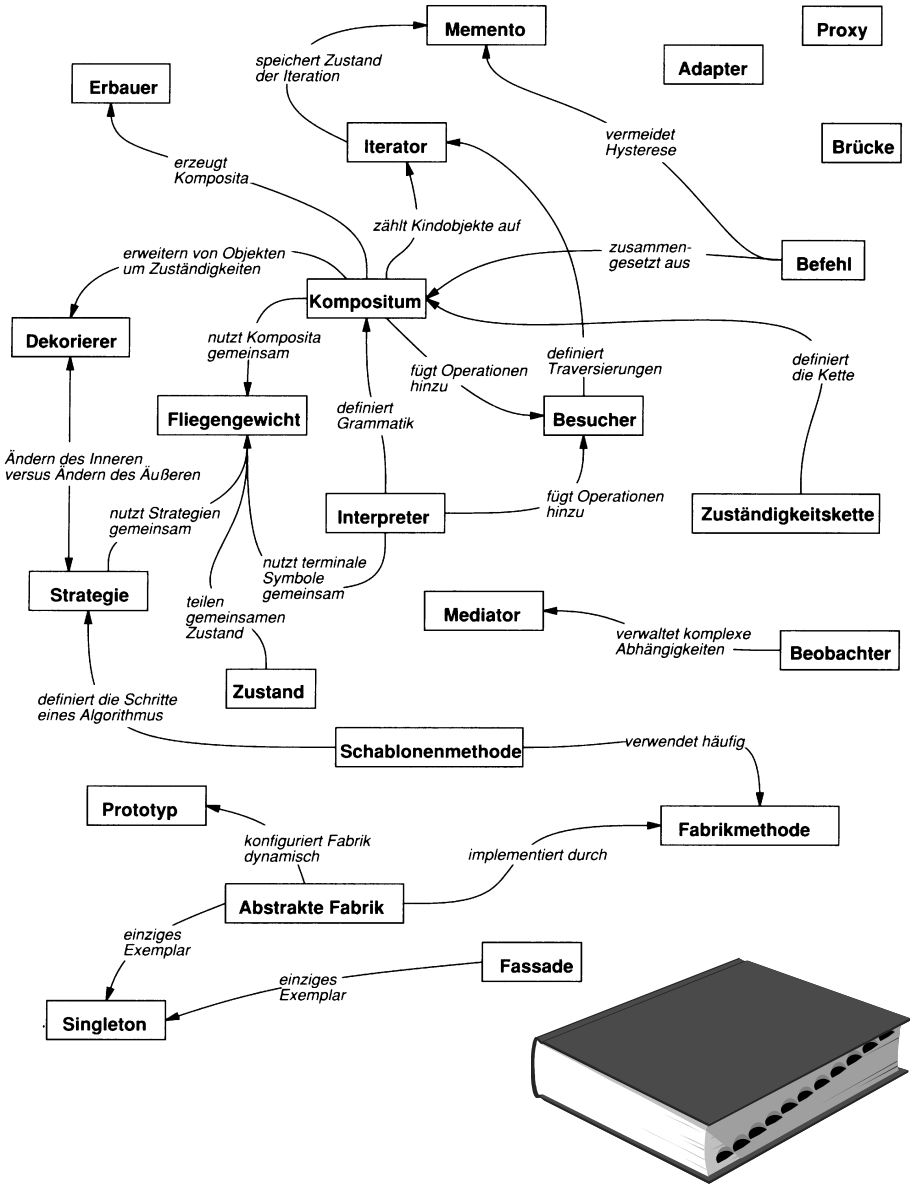


Bild 11:
Der gute Entwurf: *Pattern Map* eines Musterbuchs

[Gamma et al., 1996, Umschlagseite]

3 Epilog: Wie lehren wir gutes Entwerfen?

Entwurfsmuster tradieren Erfahrungswissen

Entwurfsmuster dokumentieren den Erfahrungsfundus einer Entwurfsdisziplin. Im Sinne des Symmetriebegriffs halten Muster die *guten* Praktiken über die unterschiedlichsten Anwendungen hinweg *invariant:* „nach den besten Schriften und aus eigener vieljähriger Erfahrung" (Bild 10 auf Seite 30). Entwurfsmuster sind Kopien ihrer selbst; sie manifestieren sich erst nach ungezählten Entwurfszyklen. Die Punkte im Sierpiński-Fraktal (Bild 2 auf Seite 13) stehen metaphorisch für die revidierten Fehlschläge; erst nach zahlreichen Redesigns entsteht das zeitlose Muster. Entwurfsmuster sind „kristallisiertes Wissen" aus langjähriger Tätigkeit: sie werden nicht erfunden, sondern *entdeckt* – von den Besten eines Faches.

Das Phänomen des „außergewöhnlichen Entwerfers" – *great designer* nach Frederick P. Brooks, *super programmer* nach Peter Molzberger [Brooks, 1987; Molzberger, 1984] – ist in der Mustererkennung angesiedelt. Der Experte entwirft musterorientiert: kreativ und zugleich effizient. Sein Fundus an Mustern bestimmt, wann der Entwurf in sich stimmig ist. Muster als *kognitive Skripte* sind ein Thema der Kreativitäts- und Gedächtnisforschung [Newell & Simon, 1972]. Softwaremuster dagegen sind ursprünglich kein akademisches Thema, vielmehr geht die Pattern-Bewegung von der Industrie aus: AT&T Bell Labs, IBM und Siemens [Beck et al., 1996]. Ziel ist die rationelle Vermittlung von Erfahrungswissen aus der industriellen Praxis. Novizen sollen – Zeit raffend – auf das Entwurfsniveau der Experten gebracht werden. Das ist natürlich das Ideal einer jeden Ausbildung, besonders der akademischen, womit ich zur Eingangsfrage zurückkomme: Wie lehrt man den guten Software-Entwurf?

Vokabular des Entwerfens

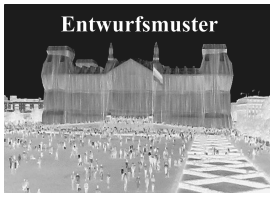
Schaffen wir in erster Linie Vertrauen – Vertrauen durch Orientierung! Symmetrie gibt Orientierung, sie zeigt das Konstante am Veränderlichen. Wir entwerfen das Neue in ständiger Differenzierung des Alten; orientieren wir uns daher an den Erfahrungen der Senior-Entwerfer. Wenn wir allgemein akzeptierte Entwurfsmuster in der Lehre verwenden, muß ein Jungingenieur nicht erst alt werden, bis er über die Erfahrung verfügt, um gut zu entwerfen. Schaffen wir ein beständiges *Vokabular* des Entwerfens, wie es

beispielsweise Bild 11 auf Seite 33 zeigt, und lehren wir dieses Vokabular in Übungen und Seminaren.[13]

Letztes Fazit: Entwurfsmuster sind die Invarianten des guten Entwurfs

Das Dilemma des guten Entwerfers beschreibt treffend ein Bonmot von Gerard de Nerval: „Ni tout à fait la même. Ni tout à fait une autre." [Meyer, 1987, S. 38]. Die Entwurfssituation ist nie ganz gleich: In jedem neuen Softwareprojekt trägt der asymmetrische Anteil die *Innovation*. Die Situation ist auch nie ganz anders: Die symmetrische Übernahme aus alten Projekten bewahrt die *Ökonomie* des neuen Projekts. Software-Symmetrie – weit gefaßt – meint die unverändert überlieferten Formen. Entwurfsmuster sind unverändert überliefertes Erfahrungswissen; sie dienen der raschen Orientierung, lassen aber genügend Spielraum für die Kreativität. Der Entwerfer „lebt im Zustand der Spannung, er liebt die Ordnung, weil sie Voraussicht ermöglicht und Sicherheit verspricht, *aber er liebt auch die Unordnung, weil sie Veränderung und Lebendigkeit bedeutet* ", Hans Sachsse mit seinen Hervorhebungen [Sachsse, 1985, S. 17].

Faszination Entwurfsmuster[14]



Das Randbild bündelt die Assoziationen, die mit der Suche nach Entwurfsmustern verknüpft sind. Die Symbolik ist vielfältig: so wie Software wirkt der unverhüllte Reichstag – mit Verlaub – *häßlich*, wenn auch nicht *kraus*. Die Verhüllung à la Christo & Jeanne-Claude ist ein Symbol für die im Detail verborgenen Strukturen, die sich erst durch eine ungewöhnliche Abstraktion als Muster offenbaren.[15] Daß hier ein Artefakt der Architektur verhüllt wurde, erinnert uns an den Ursprung der Musterbewegung in den Arbeiten des Gebäude-Architekten Christopher Alexander; und daß die Verhüllung einen künstlerischen Anspruch erhebt, steht wiederum für den Anspruch der Entwurfsmuster, den *Stand der Kunst* in der Softwaretechnik zu verdichten. Ein letztes: So wie der Zustrom auf das Reichstagsgebäude faszinieren Muster die Gemeinschaft der Software-Entwerfer.

---

13 Die Fachgruppe Technische Informatik der Universität Siegen veranstaltet regelmäßig das Seminar „Entwurfsmuster".

14 Die visuelle Analogie lag nahe: Meine Arbeitskreis-Initiative „Entwurfsmuster" im Rahmen der GI-Fachgruppe „Objektorientierte Systementwicklung" begann in Berlin im Jahre 1 nach Christos Reichstags-Verhüllung.

15 Siehe die Verhüllung und Enthüllung des Reichstags in stündlichen Bildern: http://www.zlb.de/projekte/kulturbox-archiv/archiv/

„Qualität ohne Namen“

Das Schlußwort gehört dem ersten Streiter für die Musterbewegung, Christopher Alexander. Was er als die Suche nach der „quality without a name“ im Entwurf moderner Architekturen umschreibt, steht auch für die Suche nach der *verborgenen Symmetrie:*

> „There is a central quality which is the root criterion of life and spirit in a man, a town, a building, or a wilderness. This quality is objective and precise, but it cannot be named.“
>
> [Alexander, 1979, S. ix]

Zwischen den Kapiteln

Christopher Alexander nutzte zahlreiche Chancen, um seine Mustersprache auszuprobieren – im großen: ein Klinikzentrum in Kalifornien, wie im kleinen: einzelne Bauten. Er beobachtete, wie andere seine Theorie in die Praxis umsetzten, und das Ergebnis? Seine Mustersprache scheiterte! Samt und sonders wirken die Bauten „funky“.[16]

In der Softwaretechnik hat die Geschichte der Muster erst begonnen; Erfolge gibt es zuhauf: [Beck et al., 1996; Brown, 1996; Budinsky et al., 1996; Schmidt, 1995; Schmidt & Stephenson, 1995; CACM 39 (Okt. 1996), S. 36-82]. Der praktische Wert der Softwaremuster steht außer Frage, kritisch sind Mittel und Verfahren, also Text und Textproduktion, – wahrlich ein nichttechnisches Thema. Das nächste Kapitel handelt davon.

---

16 Richard P. Gabriel – ein Software-Architekt – geht auf diesen Punkt ausführlich ein: „Critic-at-Large: The Failure of Pattern Languages“ [Gabriel, 1994].