Wissenschaftliche Softwareentwicklung

Bausteine einer rechnergestützten Datenauswertung, die den Kriterien der Wissenschaftlichkeit genügt

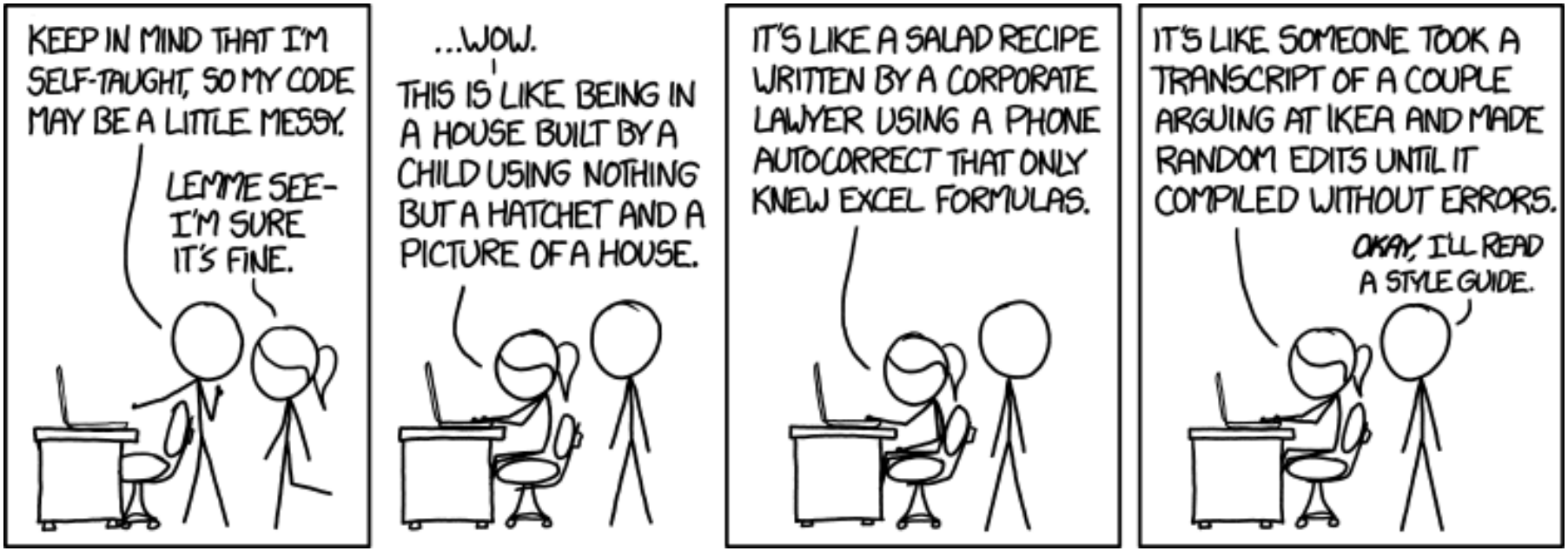
2. Motivation: Softwareentwicklung und Codequalität

Till Biskup
Physikalische Chemie
Universität Rostock
17.10.2024

Zentrale Aspekte

- Abstraktion ist der Schlüssel zur Beherrschung komplexer Fragestellungen.
- Intellektuelle Beherrschbarkeit ist ein wesentliches Kriterium. Sie erfordert Disziplin bei der Softwareentwicklung.
- Wiederverwertbarkeit, Zuverlässigkeit und Überprüfbarkeit erfordern modularen, getesteten und *lesbaren* Quellcode.
- Codequalität (insbesondere Lesbarkeit mit allen Konsequenzen) ist nicht optional für wissenschaftliche Auswertungssoftware.
- Softwareentwicklung hat in den Naturwissenschaften trotz ihrer Bedeutung einen zu geringen Stellenwert.

Code Quality



„I honestly didn't think you could even USE emoji in variable names.
Or that there were so many different crying ones."

© Randall Munroe, CC-By-NC 2.5, Quelle: https://xkcd.com/1513/

These

Code von Naturwissenschaftlern ist in der Regel viel zu schlecht, als dass er sinnvoll weiterverwendet werden kann.

- Die Wenigsten haben eine größere zeitliche Perspektive
  - insgesamt meist maximal fünf Jahre (BSc, MSc, PhD)
- Erfahrung mit Softwareentwicklung ist eine Zusatzqualifikation
  - hilft beim Finden eines Jobs
- Das Problem ist bekannt, bewährte Lösungen existieren
  - Motivation der Vorlesung: Lösungen vorstellen

> *Any fool can write code that a computer can understand.*
> *Good programmers write code that humans can understand.*
>
> – Martin Fowler

Codequalität ist direkt mit drei Aspekten verknüpft:

- Wiederverwendbarkeit
- Zuverlässigkeit
- Überprüfbarkeit

☛ Mangelnde Codequalität ist ein lange bekanntes Problem. . .

Fowler: *Refactoring*, Addison-Wesley Longman, Boston 1999

Übersicht

Ausgangspunkt: Die Softwarekrise

Lösungsansatz: Softwaretechnik (Software Engineering)

Problem der Wissenschaft: Softwareentwicklung nur Randthema

Schlüssel zum Erfolg: Konzepte und persönliche Disziplin

Ausgangspunkt: Die Softwarekrise

Der Bericht eines Augenzeugen

> *. . . so long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem. [. . .]*
>
> *The increased power of the hardware, together with the perhaps even more dramatic increase in its reliability, made solutions feasible that the programmer had not dared to dream about a few years before. And now, a few years later, he had to dream about them and, even worse, he had to transform such dreams into reality!*
>
> – Edsger Dijkstra

Dijkstra, *Commun. ACM* **15**:859, 1972

Softwarekrise

Erstmals Mitte der 1960er-Jahre auftretendes Phänomen, dass die Kosten für die Software die Kosten für die Hardware überstiegen, weil die vorhandenen Softwareentwicklungsstrategien der steigenden Komplexität der Programme, die durch die deutlich verbesserte Hardware möglich wurden, nicht gewachsen waren.

- zwei Phänomene
  - steigende Komplexität der Software
  - keine Strategien zum Umgang mit der Komplexität

☛ Parallele zur (aktuellen) Situation in den Naturwissenschaften

Ausgangspunkt: Die Softwarekrise

Parallele zur Situation in den Naturwissenschaften

Ansprüche an Software

- viele Auswertungen softwarebasiert
- Komplexität mitunter sehr erheblich

Programmierkenntnisse und Fähigkeiten

- meist nur Grundkenntnisse in einer/wenigen Sprachen
- fehlende Kenntnisse über grundlegende Konzepte

☛ Qualität der Software häufig ungenügend:
nicht wiederverwendbar, zuverlässig, überprüfbar

Ausgangspunkt: Die Softwarekrise

Parallele zur Situation in den Naturwissenschaften

These

Die meisten Wissenschaftler haben nicht das Wissen
und die (mentalen) Werkzeuge, um die Möglichkeiten,
die ihnen heutige Computer und Programmiersprachen bieten,
für ihre Zwecke sinnvoll und gewinnbringend einzusetzen.

- Es liegt nicht an den grundlegenden Fähigkeiten,
  sondern am fehlenden Wissen (und Bewusstsein der Wichtigkeit).
- rührt an die Grundlagen wissenschaftlichen Arbeitens
  (Nachvollziehbarkeit, Überprüfbarkeit, Reproduzierbarkeit)

Übersicht

Ausgangspunkt: Die Softwarekrise

Lösungsansatz: Softwaretechnik (Software Engineering)

Problem der Wissenschaft: Softwareentwicklung nur Randthema

Schlüssel zum Erfolg: Konzepte und persönliche Disziplin

Software engineering

1. the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software [. . .]

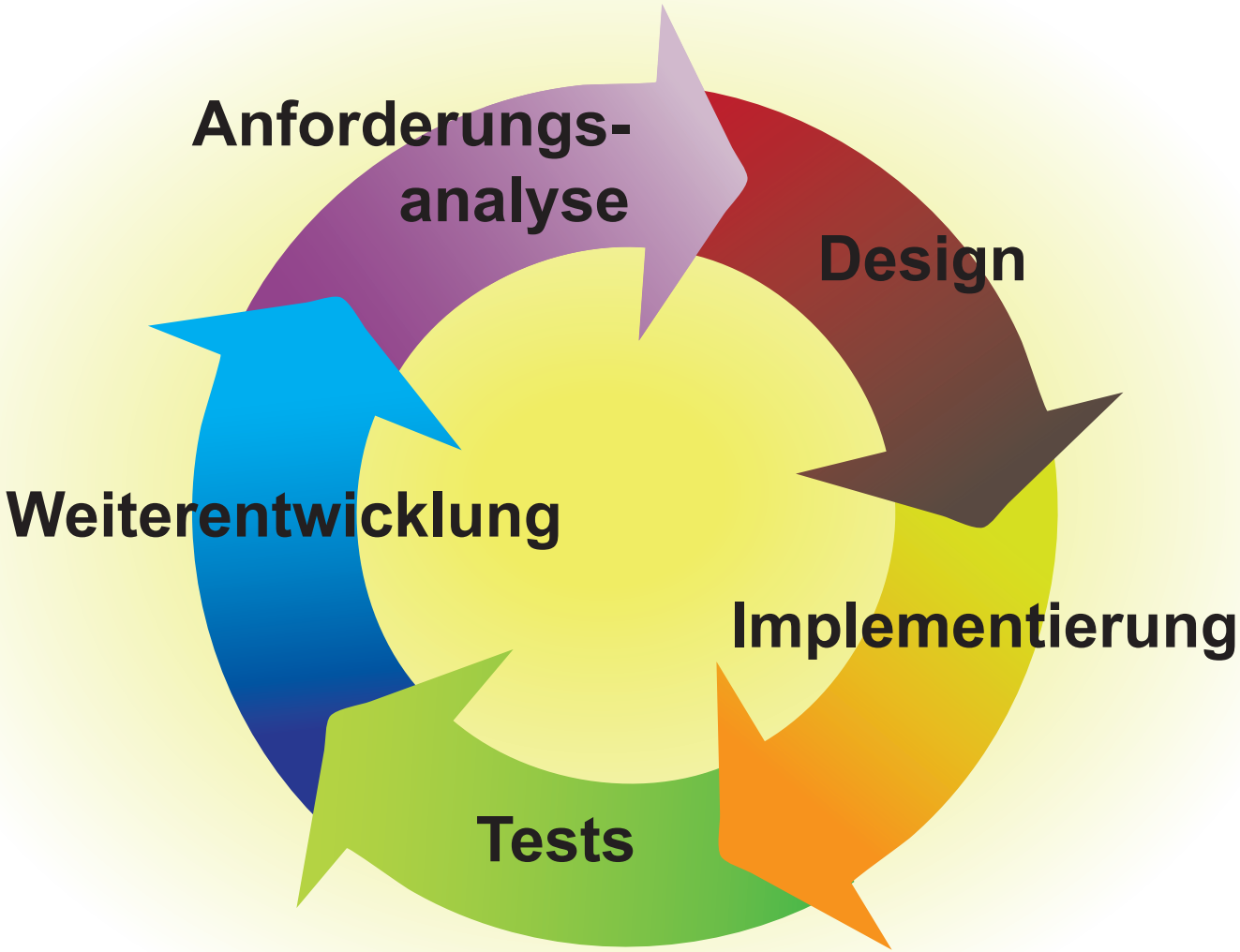
2. the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software

- Softwareentwicklung ist eher eine Ingenieursdisziplin.
- Es gibt etablierte Strategien für größere Projekte.

ISO/IEC/IEEE 24765:2010

Grundlegende Aspekte der Softwareentwicklung

- Projektmanagement
  - Parallele aus dem Ingenieurswesen
  - Formalisierung einiger Prozessabläufe
- Intellektuelle Beherrschbarkeit
  - Aufteilung in Unterprojekte
  - Trennung der Zuständigkeiten (*separation of concerns*)
- Abstraktion
  - Abbildung der Problemstellung in Software
  - Entwicklung entsprechender Sprachkonzepte
- Muster
  - Wiederverwendung etablierter Lösungen
  - Kenntnis der Muster, Erkenntnis der Einsatzmöglichkeiten


Anforderungs-
analyse
Design
Implementierung
Tests
Weiterentwicklung

Softwarentwicklung braucht Planung und Formalisierung

- Unterteilung in kleinere Abschnitte
  - Projekte werden schnell komplex.
  - Voraussetzung für parallele Bearbeitung
  - Voraussetzung für „nicht-hauptamtliche“ Bearbeitung
- „Schnittstellen“ zwischen einzelnen Abschnitten
  - Einzelne Abschnitte werden parallel bearbeitet.
  - erhöht potentielle Wiederverwertbarkeit einzelner Bereiche
- Erfolgskriterien
  - Erreichen einzelner Ziele
  - Spezifikationen von Teilprojekten/Abschnitten

☛ Projektmanagement ist ein eigener Themenkomplex . . .

> *We live in a world of changing requirements, and our job is to make sure that our software can survive those changes.*
>
> – Robert Martin

- Problem
  - Die Anforderungen an Software ändern sich häufig.
  - Starres Projektmanagement kann problematisch werden.
- (mögliche) Lösung
  - flexible Entwicklungskonzepte („*agile*“)
  - Code von Anfang an auf Veränderungen auslegen.

Martin: *Agile Software Development*, Prentice Hall, Upper Saddle River 2003

Software wird schnell sehr komplex

- Problem komplexer Zusammenhänge
  - menschliche Fähigkeit, mehrere Aspekte gleichzeitig zu erfassen, ist sehr eingeschränkt
- Ziel der Softwareentwicklung
  - intellektuelle Beherrschbarkeit (*intellectual manageability*)
- Aspekte intellektueller Beherrschbarkeit
  - Anzahl der Aufgaben eines Programmabschnittes
  - Länge eines zusammenhängenden Programmabschnittes
  - Klarheit der Formulierungen

☛ Konsequenzen für die Programmierung

Bausteine intellektueller Beherrschbarkeit

- KISS
  - *Keep it simple, stupid*
  - Parallele: Sparsamkeitsprinzip in den Wissenschaften
- Modularität
  - Aufteilung der Gesamtaufgabe in kleine Abschnitte
  - iterativer Prozess
- Unix-Philosophie
  - *Do one thing, and do it well.*
  - Jeder Programmabschnitt erfüllt genau eine Aufgabe.
- Trennung der Zuständigkeiten (*separation of concerns*)
  - (mentale) Fokussierung auf jeweils einen Aspekt
  - ohne Vernachlässigung der intrinsischen Gesamtkomplexität

Schlüssel zur intellektuellen Beherrschbarkeit: Abstraktion

> *the only mental tool by means of which a very finite piece of reasoning can cover a myriad of cases is called "abstraction" . . .*
>
> *. . . the purpose of abstracting is not to be vague, but to create a new semantic level in which one can be absolutely precise.*
>
> – Edsger Dijkstra

- ☛ entscheidend für die Lösung eines realen Problems mit Hilfe von Software
- ☛ erfordert tiefgreifendes Verständnis der Problemstellung

Dijkstra, *Commun. ACM* **15**:859, 1972

Konsequenzen fehlender Abstraktion:

> *Cut and paste may be useful text-editing operations, but they can be disastrous code-editing operations. [. . .]*
>
> *When the same code appears over and over again, in slightly different forms, the developers are missing an abstraction. Finding all the repetition and eliminating it with an appropriate abstraction may not be high on their priority list, but it would go a long way toward making the system easier to understand and maintain.*
>
> – Robert Martin

- DRY: *Don't repeat yourself*

Martin: *Agile Software Development*, Prentice Hall, Upper Saddle River 2003

Das Rad wurde bereits erfunden...

- Viele Probleme sind nicht grundlegend neu:
  Es gibt allgemeine und erprobte Lösungen.
  - Es gibt ganze Kataloge mit Mustern.
- Parallele aus der Architektur
  - Christopher Alexander (1977):
    A Pattern Language. Towns, Buildings, Construction
- Muster erleichtern die Kommunikation.
  - Namenskonventionen für viele Muster

☛ Kenntnis der entsprechenden Entwurfsmuster
und korrekte Benennung

Ansprüche

Wiederverwendbarkeit, Zuverlässigkeit,
Überprüfbarkeit

Werkzeuge

Projektmanagement, intellektuelle Beherrschbarkeit,
Abstraktion, Muster

Ergebnis

modularer, lesbarer, getesteter Code

Zusammenhang mit den Themen der Vorlesung

- Infrastruktur
  - formalisiert Abläufe, sorgt für Struktur und Überprüfbarkeit
  - erleichtert die Arbeit des Programmierers/Softwareentwicklers
  - Voraussetzung: einfach bedienbar, schnell
- Sauberer Code
  - intellektuelle Beherrschbarkeit, Abstraktion
  - „*Clean Code*“:
    KISS; DRY; Unix-Philosophie, Modularisierung, ...
- Architektur
  - Aufteilung in Unterprojekte
  - Definition von Schnittstellen/Anforderungen

Übersicht

Ausgangspunkt: Die Softwarekrise

Lösungsansatz: Softwaretechnik (Software Engineering)

Problem der Wissenschaft: Softwareentwicklung nur Randthema

Schlüssel zum Erfolg: Konzepte und persönliche Disziplin

Geschichte wiederholt sich...

> *Many scientists and engineers spend much of their lives writing, debugging, and maintaining software, but only a handful have ever been taught how to do this effectively: after a couple of introductory courses, they are left to rediscover (or reinvent) the rest of programming on their own.*
> *The result? Most spend far too much time wrestling with software, instead of doing research, but have no idea how reliable or efficient their programs are.*
>
> – Greg Wilson

in: Baxter *et al.*, *PLoS Comput. Biol.* 2:e87, 2006

Problem: mangelndes Wissen

- Softwareentwicklung wird selten gelehrt –
  (nicht nur) in der Physikalischen Chemie.
- Softwareentwicklung ist aber essentiell für viele Aspekte
  (nicht nur) der Physikalischen Chemie.

Motivation für die Vorlesung

- Softwareentwicklung ist gar nicht so schwer zu lernen.
- Es gibt von Profis viele einfache und gute Hinweise.
- Hinweise und Tipps für guten Code
  (gut = den Kriterien der Wissenschaftlichkeit entsprechend)

Problem: mangelnder Stellenwert

- Softwareentwicklung ist (fast) immer Mittel zum Zweck
  - hilft nicht beim Weiterkommen bzw. Einwerben von Geld
- Oftmals nur ein Hauptentwickler
  - Vorteile des Teams (*pair programming*, *code review*) fehlen
  - keine Motivation, für andere nutzbaren Code zu schreiben
- Mangelnder Zeithorizont
  - maximale Aufenthaltsdauer an einem Ort typischerweise 5 Jahre
  - Fokus auf Qualifikation, nicht auf Nachhaltigkeit und Qualität
- Problembewusstsein fehlt
  - Mangelnde Softwarequalität hat direkten Einfluss auf die Qualität der Wissenschaft.
  - Reproduzierbarkeit rechnergestützter Datenauswertung im Regelfall nicht gegeben

Übersicht

Ausgangspunkt: Die Softwarekrise

Lösungsansatz: Softwaretechnik (Software Engineering)

Problem der Wissenschaft: Softwareentwicklung nur Randthema

Schlüssel zum Erfolg: Konzepte und persönliche Disziplin

These

„Versprechen“ der Vorlesung: Kenntnis einiger Konzepte
und persönliche Disziplin haben einen großen Einfluss
auf die Codequalität und die Art der Softwareentwicklung.

Schlüssel zum Erfolg

- Kenntnis von Konzepten
- persönliche Disziplin

Ergebnis (idealerweise)

- viel größere Produktivität
- größere Freude an der Arbeit

Disziplin ist bei der Softwareentwicklung essentiell

- Softwareentwicklung ist eine systematische Tätigkeit
  - Zusammenhacken von Quellcode und ± erratische Fehlerbehebung helfen nicht beim Verständnis und sind langfristig gefährlich.
- Erlernen und konsequentes Verwenden etablierter Strategien und Muster
  - nicht das Rad neu erfinden . . .
  - verwendete Muster und Strategien beim Namen nennen
- Befolgen von Konventionen
  - Der konkrete Inhalt (der Konventionen) ist weniger wichtig.
  - Wichtig ist die konsequente Umsetzung.

> *Man kann nicht nicht kommunizieren*
>
> – Paul Watzlawick

Softwareentwicklung ist eine Art der Kommunikation

- „*Code for people, not computers*“
  - Kommunikation mit denjenigen, die das Programm lesen (und verstehen wollen) – oft genug der Entwickler selbst
- *Lesbarkeit* von Code ist zentral.
  - Viele Ansätze seit den 1960er Jahre haben ein Ziel: Lesbarkeit/Kommunikationsfähigkeit von Quellcode
- Große Freiheiten erfordern entsprechende Disziplin.
  - (fast) alles ist möglich – aber nicht immer sinnvoll
  - Verantwortung des Softwareentwicklers

Watzlawick, Beavin, Jackson (Hg.): *Menschliche Kommunikation*, Hogrefe, Bern 2017

These

Qualität von Code ist eine Frage der persönlichen Einstellung.

- Wer durch Code kommunizieren will . . .
  - . . . wird sich mühen,
    sich das notwendige Wissen anzueignen.
  - . . . wird immer wieder die notwendige Disziplin aufbringen,
    das Wissen anzuwenden.
- ☛ Ein Wort der Warnung:
  - Quellcode offenbart wie wenig sonst
    Herangehensweise und Charakter einer Person.

Ein Beispiel

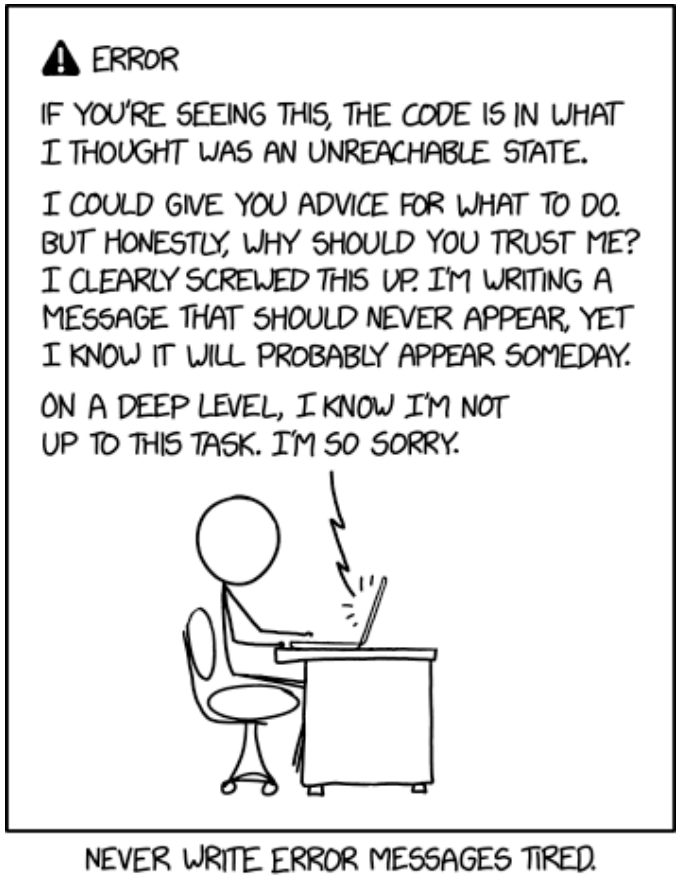
Listing 1: Schlecht formatierter (und fehlerhafter) Quellcode

```
x=[1:0.1:40*pi];
y=sin(5*x)*sin(0.1*x).*cos(2*x)
plot(x,y)
xlabel('time); ylabel('intensity');
tic; figure(); maxvalue = 100;
set(gca,'XLim',[0 maxvalue]); set(gca,'YLim',[0 1]);
hold on; plot([0 maxvalue],[.5 .5])
for k=1:maxvalue value = rand;
  if value<0.5 disp('Lower half'); else disp('Upper half'); end
plot(k,value,'rx'); pause(0.02);
  end
hold off
toc
tic; t = 0:0.01:2*pi; x =4 * cos(3*t); y = 4 * cos(4*t+pi/2);
figure; hold on; for k=1:length(x) plot(x(k),y(k),'r.'); pause(0.001); end
```

☛ Code fast unlesbar, Fehlerbehebung entsprechend mühsam

Persönliche Einstellung

Ehrlichkeit kann auch entwaffnend sein . . .



© Randall Munroe, CC-By-NC 2.5, https://xkcd.com/2200/

Zentrale Aspekte

- Abstraktion ist der Schlüssel zur Beherrschung komplexer Fragestellungen.
- Intellektuelle Beherrschbarkeit ist ein wesentliches Kriterium. Sie erfordert Disziplin bei der Softwareentwicklung.
- Wiederverwertbarkeit, Zuverlässigkeit und Überprüfbarkeit erfordern modularen, getesteten und *lesbaren* Quellcode.
- Codequalität (insbesondere Lesbarkeit mit allen Konsequenzen) ist nicht optional für wissenschaftliche Auswertungssoftware.
- Softwareentwicklung hat in den Naturwissenschaften trotz ihrer Bedeutung einen zu geringen Stellenwert.