Zusammenfassung: Pragmatic Programmer

Group Assignment SEP2
Luzia Kündig

April 15, 2022

Contents

1 Intro

Aus dem Vorwort:
Das Buch "Pragmatic Programmer" fokussiert sich neben verschiedenen technischen Best Practices vor allem auf auch den sogenannten "Common Sense", also den gesunden Menschenverstand. Es fokussiert sich auf Themen, die jeden Programmierer beschäftigen, wie z.B.

- die Zukunft mit heutigen Entscheidungen "schmerzfreier" gestalten
- Dinge für seine Teammitglieder einfacher gestalten
- Fehler zu machen und damit umzugehen
- positive Gewohnheiten zu entwickeln
- sein Toolset als Programmierer zu verstehen und zu beherrschen

Saron Yitbarek, der das Vorwort schreibt, vergleicht den Nutzen dieses Buches für Programmier-Einsteiger mit den "freundlichen Nachbarn", die dir in einer fremden Stadt Dinge zeigen, die wichtig sind: die effizientesten Pendlerstrecken, die besten Cafés, Kniffe und Tricks die man kennen sollte. Dieser Vergleich trifft für mich absolut ins Schwarze.

Da dieses Buch in 9 Kapiteln insgesamt 53 "Topics" quer durch den Alltag als Programmierer präsentiert, stellt es sich als eher schwierig heraus, dies alles in eine Zusammenfassung zu pressen. Hiermit möchte ich versuchen, die für mich hilfreichsten, interessantesten oder einfach amüsantesten Themen hervorzuheben.

2 Die Philosophie

Die Grundeigenschaften eines Pragmatic Programmers, beschrieben unter dem Abschnitt *Pragmatic Philosophy*.

- *Think about your work*: If this sounds like hard work, then you're exhibiting the realistic characteristic.
- *Care about your craft*: We who cut mere stones must always envision cathedrals.
- *Kaizen (japanisch)*: Das Konzept, jeden Tag ganz kleine Verbesserungen vorzunehmen, um schlussendlich durchgehend hohe Qualität zu erreichen.
- *Take responsibility*: It's your life.

2.1 Software Entropie

Entropie in Software wird als *Amount of Disorder* oder *Software Rot* beschrieben. Software altert. Um die gegebene Qualität über eine längere Zeit zu erhalten, sollten Probleme (sog. *Broken Windows*) wie zum Beispiel

- schlechtes Design
- falsche Entscheidungen
- unschöner Code

möglichst bald behoben werden. So kann dazu beigetragen werden, dass auch zukünftige Arbeiten am Code auf gewissenhafte und saubere Weise ausgeführt werden.

2.2 Veränderung

Veränderung und der richtige Umgang damit ist das zweite grundlegende Konzept in diesem Buch. Passiert diese langsam und in kleinen Schritten, wird sie viel weniger wahrgenommen als wenn auf einen Schlag etwas komplett anders ist. Im positiven kann man sich dies zu Nutze machen, indem man Verbesserungen in kleinen Schritten einführt und den Menschen Zeit gibt.

Kleine, stetige Veränderungen in der Aussenwelt oder den Voraussetzungen in einem Projekt sind aber ebenso einfach zu verpassen, wie sie im positiven anzunehmen sind. Deshalb sollte man stets den Blick vom aktuellen, spezifischen Problem auch wieder aufs grosse Ganze richten und hinterfragen, ob man immernoch auf dem richtigen Weg ist.

3 Wissen, Technologien, Beherrschen der Tools

3.1 Knowledge Portfolio

Das beste und wichtigste Asset, das ein Programmierer in seinen Job mitbringt, ist Wissen. Unendlich viele verschiedene Technologien, die sich extrem schnell verändern, machen es unumgänglich, dass man sich stetig weiterbildet, und wenn es nur einige Minuten am Tag oder in der Woche sind. Neue Technologien ausprobieren, Kurse besuchen, News und Publikationen lesen sollte im Zeitplan einen fixen Platz haben und wird im Idealfall auch vom Arbeitgeber geschätzt und vergütet, da es vor allem auch diesem zu Gute kommt.

3.2 Shell Games, Power Editing, Text Manipulation

- version control
- debugging

- engineering daybook

4 The Essence of Good Design: Entscheidungen treffen

4.1 ETC - Easier to change

Ein System hat ein gutes Design, wenn es sich an die Menschen, die es nutzen, anpassen kann.

Easier to change ist deshalb das grundlegende Prinzip, auf dem fast alle weiteren Designprinzipien aufbauen. Es soll keine feste Vorgabe sein, sondern eine Hilfe. Immer wenn eine Entscheidung zwischen Vorgehen A oder B getroffen werden muss, sollte man sich hieran orientieren.

Um diese Denkweise zu verinnerlichen kann man sich diese Frage bewusst immer wieder stellen. Beim Speichern einer Datei, beim Schreiben eines Tests oder beim Beheben eines Bugs. Ist der Code, den ich soeben geschrieben habe *easy to change*?

4.2 DRY - Don't repeat yourself

Jedes Stück an Information braucht eine einzige, eindeutige und authoritative Repräsentation in einem System.

Das Duplizieren von Informationen führt unweigerlich zu Inkonsistenz. Sei dies im Code, in Dokumentationen oder anderswo. Weitere, nicht ganz so offensichtliche Arten von Duplikation:

Duplikation der Repräsentation

Interne/Externe APIs: Das Wissen über die ausgetauschten Daten bzw. deren Struktur muss grundsätzlich auf beiden Seiten der Kommunikation vorhanden sein. Abhilfe schaffen können hier Sprachen zur neutralen Spezifikation von APIs, die idealerweise zentral abgelegt werden und zur Erstellung von automatisierten Tests und Test-Clients verwendet werden können.

Datenquellen: Falls möglich sollten Objekte aus ihrer Dantenbankrepresentation automatisiert erstellt werden können. Andernfalls müssen Änderungen an der Struktur an verschiedenen Orten durchgeführt werden.

Duplikation zwischen Entwicklern

Diese Art von Duplikation ist wahrscheinlich am schwierigsten zu entdecken und adressieren. Ein wichtiges Mittel hierzu ist die Kommunikation von Teams zu stärken, innerhalb sowie übergreifend. Weiter muss es einfacher

sein, bereits erstellte Funktionalität wiederzuverwenden, als sie neu zu implementieren.

4.3 Orthogonalität

Wenn in der Geometrie zwei Vektoren orthogonal zueinander stehen, sind sie voneinander komplett unabhängig.

In der Softwareentwicklung wurde diese Idee als wichtiges Grundprinzip übernommen: Wenn an einem Element eine Änderung vorgenommen wird, sollen davon möglichst wenige oder keine anderen Elemente beeinflusst werden. Code soll möglichst modular aufgebaut sein, aufgeteilt in kleine Einheiten die klar definierte Funktionalität bieten.

Erreichen können wir dies mittels verschiedenen Grundsätzen.
Decoupling: Wenige Abhängigkeiten bilden zwischen verschiedenen Modulen, resultiert meist in *high cohesion*, was einen guten Zusammenhalt innerhalb eines Moduls beschreibt.
Globale Daten vermeiden: Globale Daten verbinden alle Module miteinander, die darauf zugreifen. Abhängigkeiten entstehen, Testing wird schwieriger, der Code wird schwieriger verständlich.
Unit Tests als Gradmesser: Wenn für einzelne Unit Tests verschiedenste Elemente aus dem Code eingebunden werden müssen, ist das ein schlechtes Zeichen.

4.4 Good Enough

Die Qualität eines Produktes hängt von vielen Faktoren ab. Kosten, Zeit und Umfang eines Projektes spielen eine Rolle. Am wichtigsten ist es dabei, diese Faktoren auszubalancieren, sodass die Qualität *für den vorgesehenen Zweck* ausreichend ist und den Endbenutzer zufriedenstellt. Sind die Kosten am Schluss viel höher oder der Umfang viel kleiner als erwartet, ist bessere Qualität der erbrachten Leistung meist keine Rechtfertigung.
Know when to stop.

4.5 Reversibility

Es gibt keine finalen Entscheidungen.

4.6 More..

- Naming
- don't trust anyone (not even yourself)
- programming by coincidence
- listen to your lizard brain
- Algorithm Speeds

5 Code

Wenn es um effektive Techniken geht, wie *pragmatische Projekte* umgesetzt werden können, werden folgende Punkte diskutiert.

5.1 Tracer Bullets

Leuchtende Projektile sollen im Militär anzeigen, ob man die Zielscheibe auch wirklich trifft oder nicht. Genauso können beim Entwickeln kleine Einheiten von Funktionalität verwendet werden, um die End-zu-End Kommunikation quer durch alle Layers einer Architektur zu testen und zu garantieren. Sollte das Ziel verfehlt werden, können Anpassungen in der Strategie oder im Toolset umgehend vorgenommen werden.

5.2 Prototypes

Prototypen sind lauffähige Systeme, die gebaut werden um bestimmte Aspekte eines Projektes auszuloten, neue Technologien zu testen oder Risiken zu minimieren. Punkte wie Korrektheit, Komplettheit, Robustheit und Coding-Stil können vernachlässigt werden, da ein Prototyp nicht dazu da ist, später in der Produktion verwendet zu werden.

5.3 Domain Languages

¿¿ todo ¡¡
Interne:
Externe: (Bsp. Ansible) Muss von einer Software / einem Tool werden.

5.4 Methodik: You can't write perfect Software

- Design by Contract
- semantic invariants
- crash early
- outrunning your headlights
- resource balancing
- testing
- refactoring

5.5 Bend or Break

- test
- test

5.6 Concurrency

- Temporal Coupling: Zeitliche Abhängigkeit - Shared State equals Incorrect State
- actors and processes
- blackboards

6 Kommunizieren

Kommunikation als Programmierer kann sein wie man sich im Code ausdrückt (Prinzipien wie DRY, ETC, ..), dasselbe gilt aber auch für die sprachliche Kommunikation. Meetings mit Kunden, Kollegen und Vorgesetzten machen einen wichtigen Teil der Tätigkeit aus und entscheiden meist über Erfolg oder Misserfolg eines Projekts. Deshalb sollten auch hier die eigenen Fähigkeiten gepflegt und gestärkt werden.

- Kenne dein Publikum
- Wisse, was du hinüberbringen willst
- Wähle den richtigen Moment
- Wähle eine passende Ausdrucksart
- Beziehe dein Publikum mit ein
- Höre zu
- Bleibe keine Antworten schuldig
- Bringe Optionen statt Ausreden
- **Ja sagen**, wenn man sich sicher ist, dass etwas in der gewünschten Zeit oder auf die gewünschte Weise umsetzbar ist.
- **Nein sagen**, wenn eine Deadline nicht sicher einzuhalten ist, man mehr Zeit zum abklären oder einschätzen braucht oder auf Unterstützung angewiesen ist.
- **Dokumentation** dort platzieren, wo sie gelesen wird.
  Das *wie* soll durch den Code selbsterklärend sein, das *warum* kann als Kommentar eingefügt werden.
  Speziell bei APIs

7 Projekte Planen, Schätzen, Erfahrungswerte

- how accurate is accurate enough
- what to say when asked for an estimate
- the requirements pit

8 Working in a (pragmatic) Team on (pragmatic) Projects

- solving impossible puzzles
- working together
- pragmatic teams
- coconuts don't cut it - cargo cult

9 *agile thinking*

Hinterfragen der Situation, der Entscheidungen und der Methoden