Softwaredesign

Eine Einführung

Geistesverwandtschaft: Recht & Code

Die Welt der Juristen

- Gesetze & Verträge
- Logische Definitionen
- Strukturierung von Realität
- Handlungsanweisungen

Die Welt der IT

- Algorithmen & Quellcode
- Klassen & Objekte
- Abstraktion von Realität
- Prozesssteuerung

Code & Recht

Beide sind abstrakte Systeme zur Steuerung der Realität.

Ein Gesetz ist ein Algorithmus für die Gesellschaft.

Quellcode ist ein Gesetz für den Computer.

Abstraktion: Die Kunst des Weglassens

Wie bildet man ein Mandat ab?

Nicht jedes Detail zählt (die Krawattenfarbe des Mandanten ist irrelevant, sein Geburtsdatum für die Verjährung nicht).

Beispiel: Die "Akte" als Objekt in der Software vs. die physische Akte.

Abstraktion: Modelle der Realität

Definition: Reduktion von Komplexität auf das Wesentliche.

Beispiel "Mandat":
Physisch: Ein Mensch, ein Aktenstapel, Gespräche.
Digital: Eine ID, ein Status, verknüpfte Dokumente, Fristen.

Gefahr: Wenn die Abstraktion nicht zur Realität passt, entstehen "Bugs" (Fehler im System).

Gutes Design bedeutet:
Die richtigen Informationen zur richtigen Zeit an der richtigen Stelle.

Abstraktion: Die Karte ist nicht das Gebiet

Objekt-Orientierte Analyse & Design

Klasse: Das "Gesetz" oder die Schablone (z.B. Allgemeines Mandat).

Objekt: Der konkrete Fall (z.B. Mandat Müller vs. Meier).

Eigenschaften: Daten (Aktenzeichen, Streitwert).

Methoden: Handlungen (Abrechnen, Frist setzen).

Analogie:
Ein Formular im BGB ist die Klasse, das ausgefüllte Dokument das Objekt.

Benennung: Warum Präzision alles ist

In der IT: Unklare Variablennamen und Bezeichner führen zu Wartungs-Alpträumen.
Schlecht: `data_1` | Gut: `verjaehrungs_datum_mahnung`

Im Recht: Unbestimmte Rechtsbegriffe ohne Definition führen zu Rechtsunsicherheit.

Ubiquitous Language: Entwickler und Juristen müssen dieselbe Sprache sprechen.
Ein "Termin" muss für beide dasselbe bedeuten.

Schlechte Bezeichnungen sind die technische Schuld ("Technical Debt") von morgen.

ORM: Die Beziehungen zwischen Objekten

Object-Relational Mapping
Objekt-orientierte Software denkt in Objekten (Dingen) und deren Beziehungen zueinander, Datenbanken in Listen/Tabellen und deren Beziehung zueinander.

Die Herausforderung: Wie bildet man Beziehungen ab?

Ein Mandant hat viele Akten (1:n).

Ein Anwalt arbeitet an vielen Akten, eine Akte hat viele Anwälte (m:n).

Ein schlechtes Datenmodell ist wie ein falsch sortiertes Archiv mit falscher Ordnungsstruktur: Man findet nichts wieder, wenn man es braucht, die Datenhaltung ist doppelt, dreifach, mehrfach und die Datenkonsistenz ist immer in Gefahr.

Conways Law: Die Kommunikationsstruktur einer Organisation prägt die Softwarearchitektur

Wenn die Kanzlei-Abteilungen (z.B. Buchhaltung vs. Anwalt) Silos sind, wird auch die Software Silos bilden.

Konsequenz: Software-Probleme sind nicht selten ungelöste Probleme in der Organisation oder Kommunikation.

Gute Software unterstützt eine bereits bestehende gesunde Organisation mit hoher Kohärenz und loser Kopplung.

Beispiel: Ident/Auth/Auth in neuen Setup

Anti-Pattern: Der Goldene Hammer

Der Goldene Hammer: "Wenn man nur einen Hammer hat, sieht jedes Problem wie ein Nagel aus."

➔ Beispiel: Kalender mit Excel bauen, anstatt mit einer Kalendersoftware, Analysen in einer Präsentationssoftware erstellen statt in einer Tabellenkalkulation

Anti-Pattern: Ambivalente Gesichtspunkte

Ambivalente Gesichtspunkte / Ambiguous Viewpoints: Verschiedene Aspekte der Software (z.B. Datenarchitektur und Präsentation) werden vermischt und durcheinandergebracht.

Software wird für "alle" gebaut, aber niemandem wird sie gerecht.

➔ Partner wollen Dashboards, Associates wollen effiziente Eingabe. Ohne Fokus scheitert das Tool.

Gegenentwurf: Software sollte eine Sache machen, die aber sehr gut. (Unix Prinzip)

Anti-Pattern: Cargo Cult Programming

Toolkits werden eingesetzt, weil sie *cool* sind, ohne das man ihren Sinn versteht.

Beispiel: ***React*** für eine einfache Website, ***Keycloak*** für ein einfaches Ident/Auth/Auth.
Entwicklung ist Buzzword-getrieben und nicht anforderungsgetrieben.

Lösungsansätze für die Praxis

Klein anfangen (MVP): Nicht das "Gesetzbuch für alles" schreiben, sondern mit einer "Verordnung" beginnen.

Interdisziplinäre Teams: Juristen müssen beim Design am Tisch sitzen, nicht erst beim Testen. Entwickler müssen bei der Anforderungsanalyse dabei sein, nicht erst beim Projektstart.

Mut zum "Nein": Jedes zusätzliche Feature macht das System komplexer und fehleranfälliger.

My biggest job is to say “No.” to new features. - Linus Torwalds

Fazit

Software ist kein Fremdkörper, sondern digitale Logik.

Gutes Design spart langfristig Zeit (Vermeidung von Technical Debt).

Modularität und saubere Schnittstellen sind eine gute Sache - in der Software und in der Kanzlei.

Juristen sind durch ihr Studium prädestiniert für logisches Software-Design – wenn sie die Konzepte verstehen.

Passende verbindliche Bezeichnungen und klar definierte Beziehungen (Relationen) sind ein Schlüsselmerkmal für gutes Softwaredesign. Die Parallelen zum guten Gesetzen und Rechtsverordnungen sind erkennbar.

Danke. Fragen?