# Kurz und langfristige Ziele unterscheiden

Da es in der angegebenen Literatur nicht gut herauskommt, versuche Ich es mit eigenen Worten. Am ehesten kommt es aus dem Buch von Robert Cecile Martin Clean Architecture heraus, als er das Eisenhower Prinzip beschreibt.

Siehe auch <https://de.wikipedia.org/wiki/Eisenhower-Prinzip>

Eisenhower war amerikanischer Präsident und unterschied zwischen wichtigen und dringenden Zielen. Die dringenden Ziele sind selten wichtig, und die wichtigen Ziele sind meist nicht dringend.

Manager fokussieren sich meist auf die dringenden Ziele und vernachlässigen die wichtigen Ziele.

Ein Softwaresystem droht mit der Zeit zu degenerieren. Deshalb wird regelmäßiges Refactoring empfohlen. Refactoring ist ein Prozess, der an der Funktionalität der Software nichts ändert, aber die Wartbarkeit erhöht. Der Kunde hat zumindest nicht sofort etwas davon. Ganz im Gegenteil. Die Umstrukturierung des Codes sorgt häufig für (meist minimale) Performance Verluste.

Ein häufiges Problem ist das Vorhandensein von totem Code. Toter Code sind Softwareteile, die nicht mehr ausgeführt werden. Eine regelmäßige Coverage Analyse ist ein typisch wichtiger, aber nicht dringender Prozess. Der größte Anteil der Softwareentwicklung ist nun einmal die Fehlersuche und das Analysieren des Codes. Ein Mitarbeiter, der toten Code analysiert ist am Ende unproduktiv.

Ein weiteres typisches Problem ist doppelter Code. Doppelter Code sorgt dafür, dass bei Änderungen, diese doppelt nachvollzogen werden müssen.

Ein weiteres Problem liegt in der Nichtbeachtung der Abhängigkeiten. Besteht ein Code aus 5 Modulen und ist jedes Modul von jedem abhängig, müssen bei Hinzufügen neuer Funktionalität (in einem neuen Modul) die Auswirkungen in allen neuen Modulen untersucht werden. Kommt dann wieder ein neues Modul hinzu, steigt der Aufwand exponentiell – denn wenn alles von allem abhängig ist, müssen die Auswirkungen vom gesamten Code betrachtet werden. Dies ist der Grund warum Software Projekte anfangs relativ zügig voranschreiten, aber mit zunehmender Größe immer schwerer zu warten wird. Am Ende dauern kleine Änderungen Ewigkeiten und die Entwickler haben Angst Änderungen einzubauen, weil das Vertrauen, dass der Code noch nach der Änderung funktioniert verloren geht.

Es ist allerdings kein Naturgesetz, dass Code degeneriert. Beachtet man von Anfang an die Abhängigkeiten und stellt und überwacht Architekturregeln, sollte mit zunehmender Zeit es einfacher werden neue Funktionalität einzufügen. Die Investitionen in Architektur, sollten sich langfristig auszahlen.

# Die Ziele einer Softwarearchitektur

Ich möchte den Nutzen und die Ziele von Softwarearchitektur nochmals zusammenfassen:

* Qualitätsanforderungen und -ziele erfüllen: Explizite konstruktive und technische Strategien, Maßnahmen, Taktiken oder Praktiken stellen die Erreichung der notwendigen Qualität sicher.
* Konzeptionelle Integrität (Konsistenz): Ähnliche Aufgaben werden im System durchgängig ähnlich gelöst – Ausnahmen sind begründet.
* Verständlichkeit des Systems und seiner grundlegenden Entwurfsentscheidungen, Strukturen, und Konzepte/Prinzipien: Angemessene Verwendung von Abstraktionen, Modellen, Dokumentation zur Erläuterung von Entscheidungen, Strukturen, Konzepten und Quellcode.
* Starker Fokus auf Langfristigkeit – daher Betonung von (langfristigen) Architekturzielen, gegenüber (meist kurzfristigen) Projektzielen.
* Unterstützung des gesamten Lebenszyklus: Von der Problem- und Anforderungsanalyse über Konstruktion, Entwicklung, Implementierung bis zum praktischen Einsatz und Betrieb von Systemen, sowohl bei Neuentwicklung wie auch Wartung/Änderung von Systemen.
* Vereinfachung der Wiederverwendung von Systembestandteilen oder Konzepten

Quelle: Effektive Softwarearchitekturen, 8. Auflage, Kapitel 2.1.1