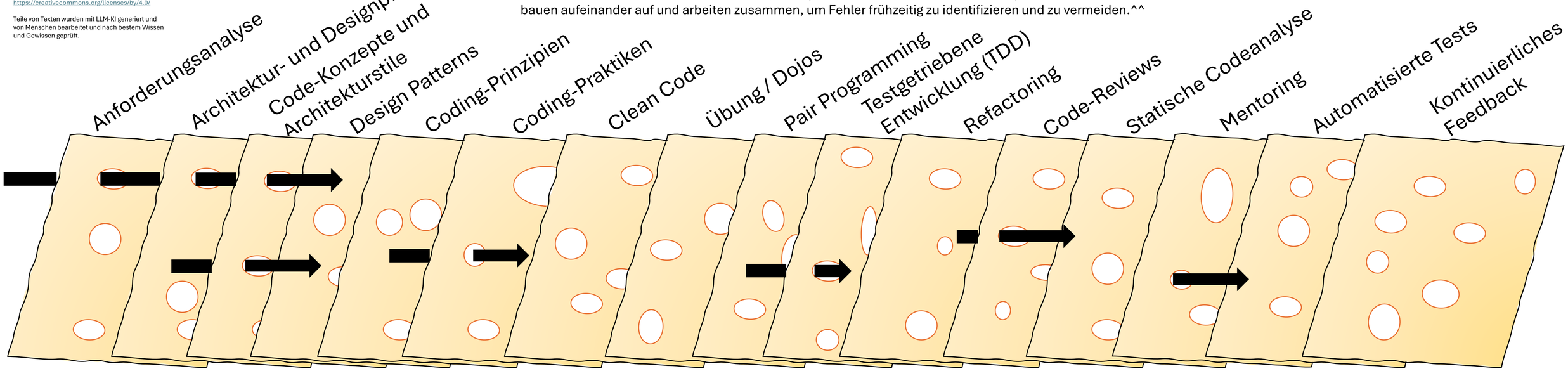
Schweizer Käse Schichtenmodell der Software Entwicklung

Jede Käsescheibe stellt eine Schicht dar, die spezifische Maßnahmen zur Fehlerprävention umfasst. Die Schichten bauen aufeinander auf und arbeiten zusammen, um Fehler frühzeitig zu identifizieren und zu vermeiden.^^



Eine klare und vollständige **Anforderungsanalyse** ist der erste Schritt, um Fehler in der Softwareentwicklung zu vermeiden. Durch die sorgfältige Erfassung der Anforderungen können Missverständnisse vermieden und teure nachträgliche Änderungen reduziert werden.

In der **Architektur- und Designphase** werden die grundlegenden Strukturen und Komponenten der Software festgelegt. Sie bilden das Fundament für die Implementierung und entscheiden über die Flexibilität und Wartbarkeit der Software.

Mit **Code-Konzepte und Architekturstile** können Entwickler die Struktur und Organisation ihres Codes verbessern. Durch die Verwendung bewährter Praktiken und Muster können sie die Lesbarkeit, Wartbarkeit und Erweiterbarkeit ihrer Software erhöhen.

Design Patterns ermöglichen es Entwicklern, wiederkehrende Probleme in der Softwareentwicklung effizient zu lösen. Sie bieten bewährte Lösungen für häufig auftretende Herausforderungen und fördern die Wiederverwendbarkeit und Skalierbarkeit des Codes. Andere Entwickler können die Muster leicht erkennen und verstehen, was die Zusammenarbeit erleichtert.

Coding-Prinzipien fördern die Konsistenz und Qualität des Codes. Sie umfassen Regeln und Richtlinien, die Entwickler dabei unterstützen, sauberen und wartbaren Code zu schreiben.
Z.B. SOLID, DRY, KISS, YAGNI

Mit **Coding-Praktiken** können Entwickler ihre Effizienz und Produktivität steigern. Sie umfassen bewährte Methoden und Techniken, die Entwickler dabei unterstützen, qualitativ hochwertigen Code zu schreiben.

Clean Code stellt die Grundlage für eine gute Softwarequalität dar. Es bezeichnet gut strukturierten, verständlichen und wartbaren Code, der leicht zu lesen, zu verstehen und zu erweitern ist. Clean Code ist ein zentraler Bestandteil der Softwareentwicklung und hilft dabei, Fehler frühzeitig zu vermeiden und die Qualität der Software zu verbessern.

Übung / Dojos bieten Entwicklern die Möglichkeit, ihre Fähigkeiten zu verbessern und neue Techniken zu erlernen. Durch regelmäßige Übungen können sie ihre Programmierfähigkeiten schärfen und sich mit anderen Entwicklern austauschen. Wiederholungen und Feedback helfen dabei, kontinuierlich zu lernen und zu wachsen.

Pair Programming ist eine Praktik, bei der zwei Entwickler zusammen an einem Code arbeiten. Durch die gegenseitige Überprüfung und Zusammenarbeit können sie Fehler frühzeitig erkennen und die Qualität des Codes verbessern.

Testgetriebene Entwicklung (TDD) ist eine Methode, bei der Entwickler zuerst Tests schreiben und dann den Code implementieren, um die Tests zu bestehen. Dadurch wird der Entwickler gezwungen zuerst über die Funktionalität nachzudenken und kleine Methoden zu schreiben, die dann getestet werden.

Refactoring ist der Prozess, bei dem der Code verbessert wird, ohne das Verhalten der Software zu ändern. Code wird selten allen Regeln gerecht, und Refactoring hilft dabei, den Code den stetig den Anforderungen von lesbarem und wartbarem Code anzupassen.

Code-Reviews ermöglichen durch das Vier-Augen-Prinzip eine Überprüfung des Codes durch andere Entwickler. Je nach Insentität des Reviews können nicht nur Verletzungen von Coding-Regeln gefunden werden, sondern auch Fehler oder nicht umgesetzte Anforderungen.

Die **statische Codeanalyse** ist ein automatisierter Prozess, bei dem der Code auf Fehler, Sicherheitslücken und Codestil überprüft wird. Durch den Einsatz von Tools können Entwickler potenzielle Probleme frühzeitig erkennen und beheben.

Mit **Mentoring** können erfahrene Entwickler ihr Wissen und ihre Erfahrung weitergeben. Durch den Austausch von Best Practices und Feedback können Entwickler ihre Fähigkeiten verbessern und schneller lernen.

Automatisierte Tests sind ein wesentlicher Bestandteil der Softwareentwicklung. Sie ermöglichen es Entwicklern, die Funktionalität und Qualität ihres Codes zu überprüfen und sicherzustellen, dass Änderungen keine unerwünschten Nebenwirkungen haben.

Kontinuierliches Feedback ist entscheidend für die kontinuierliche Verbesserung der Softwarequalität. Durch regelmäßige und frühzeitige Rückmeldungen von Entwicklern, Testern und Nutzern können Probleme frühzeitig erkannt und behoben werden.