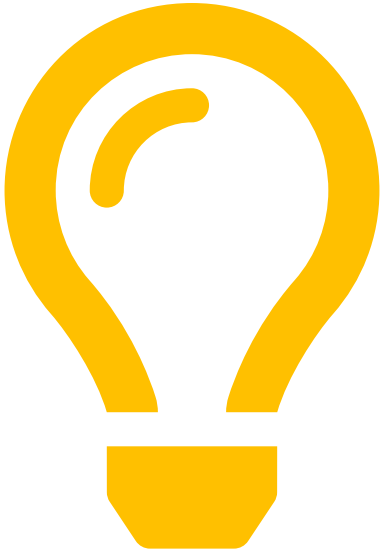
Hochschule Konstanz
Fakultät Informatik

Software-Prozesse

Software-Prozess

- Abstrakte Vorgehensweise zur Entwicklung eines Softwaresystems
- definiert durch Menge von erforderlichen Aktivitäten:
  - In der **Spezifikation** legt man fest, was das System tun soll;
  - Im **Entwurf und der Implementierung** legt man die Aufteilung des Systems fest und implementiert es;
  - Bei der **Validierung** überprüft man, ob das System das tut, was der Kunde will;
- Prozessmodelle bieten manchmal auch Aktivitäten für die Software-**Weiterentwicklung** an, der Änderung des Systems als Antwort auf sich ändernde Markt- oder Kundenbedürfnisse.
- Der **Betrieb** wird normalerweise nicht behandelt.
- Es gibt viele verschiedene Software-Prozesse.

Plangesteuerte und agile Prozesse



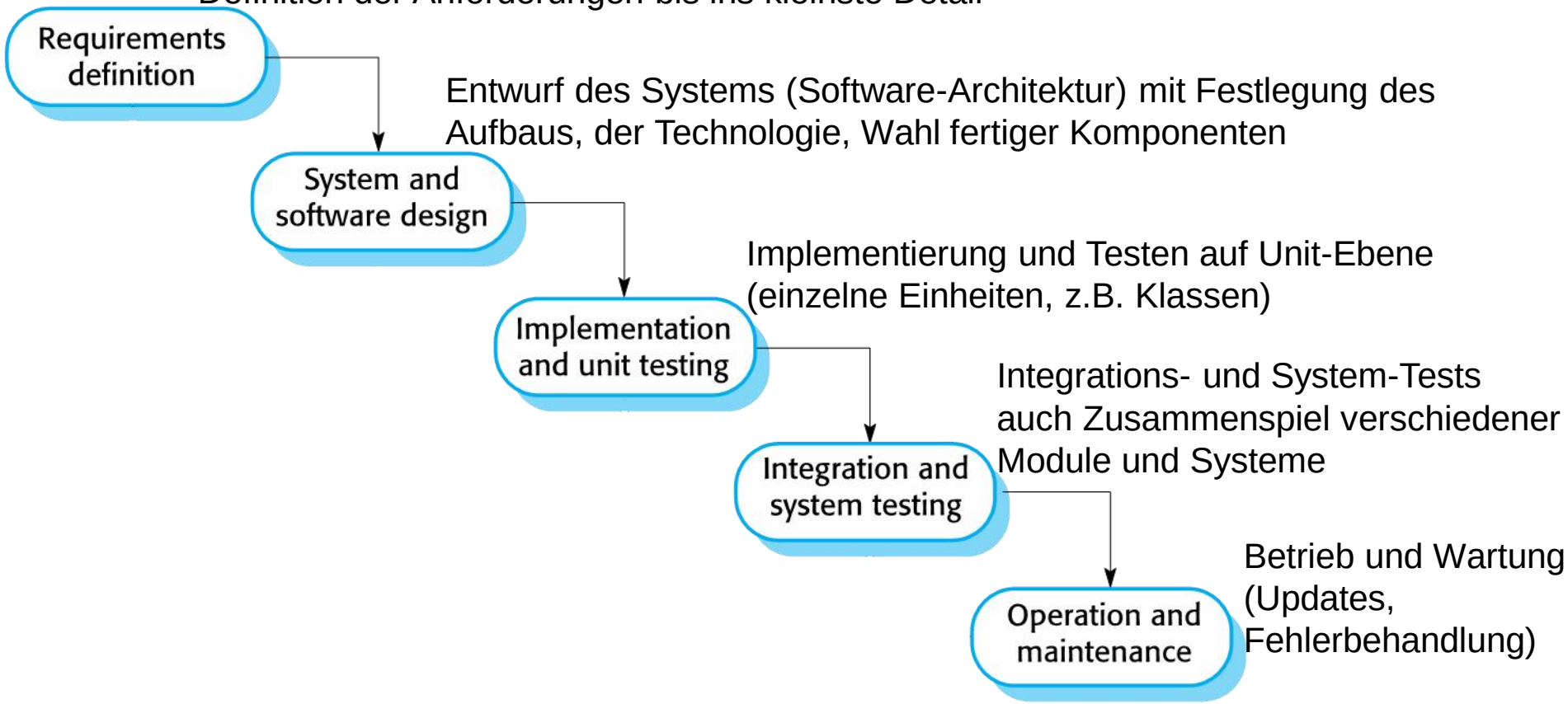
Von Ian Sommerville:

- **Plangesteuerte Prozesse** sind Prozesse, bei denen alle Prozessaktivitäten im Voraus geplant werden und der Fortschritt anhand dieses Plans gemessen wird.
- Bei **agilen Prozessen** erfolgt die Planung **inkrementell**, und es ist einfacher, den Prozess an sich ändernde Kundenanforderungen anzupassen.
  - Die Prozesse der Spezifikation, des Entwurfs und der Implementierung sind ineinander verschachtelt.
  - Es gibt keine detaillierte Systemspezifikation, und die Entwurfs-Dokumentation ist auf ein Minimum reduziert.
  - Das Dokument mit den Benutzeranforderungen beinhaltet grobe Definition der wichtigsten Merkmale des Systems.
- Die meisten in der Praxis angewendeten Verfahren enthalten Elemente sowohl planorientierter als auch agiler Ansätze.
- Es gibt keine richtigen oder falschen Softwareprozesse.

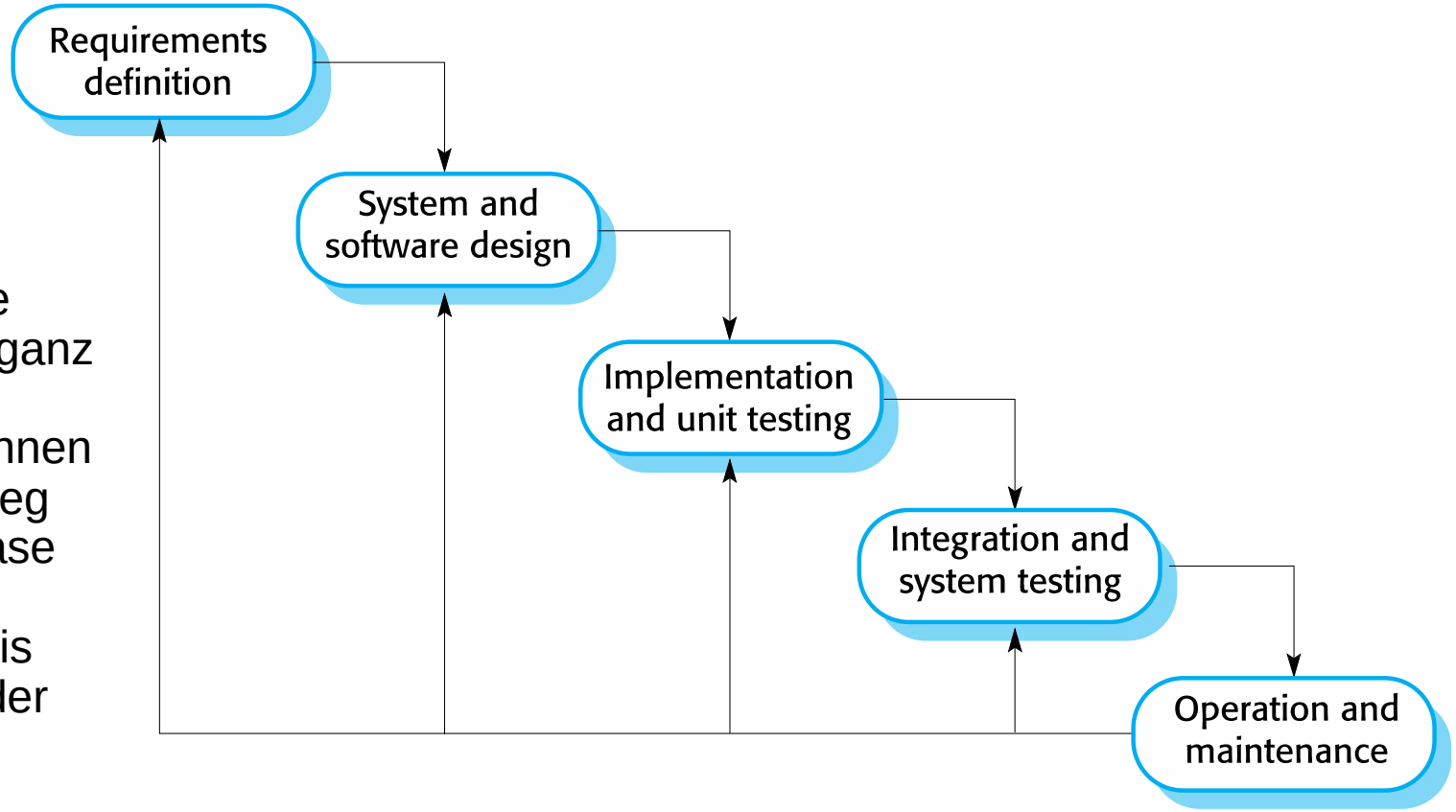
Software-Prozessmodelle

- **Das Wasserfallmodell**
  - "das" typische plan-getriebene Modell.
  - alle verschiedenen Aktivitäten werden getrennt und nacheinander ausgeführt
- **Inkrementelle Entwicklung**
  - Software wird nicht als ein einziges großes Produkt entwickelt, sondern als eine Abfolge von einer ersten Minimalversion, gefolgt von Inkrementen, die weitere Funktionen hinzufügen
  - Spezifikation, Entwicklung und Validierung werden mit Blick auf das nächste Inkrement durchgeführt
  - … ist die übliche Methode in agilen Prozessen
- **Integration und Konfiguration**
  - Das System wird aus vorhandenen konfigurierbaren Komponenten zusammengesetzt (oft als COTS, Commercial off-the-shelf products, bezeichnet)
  - Vermeidet oder minimiert zumindest die eigene Entwicklung.
- In größeren Systemen können verschiedene Teile mit unterschiedlichen Prozessmodellen entwickelt werden

Das Wasserfallmodell



Das Wasserfallmodell

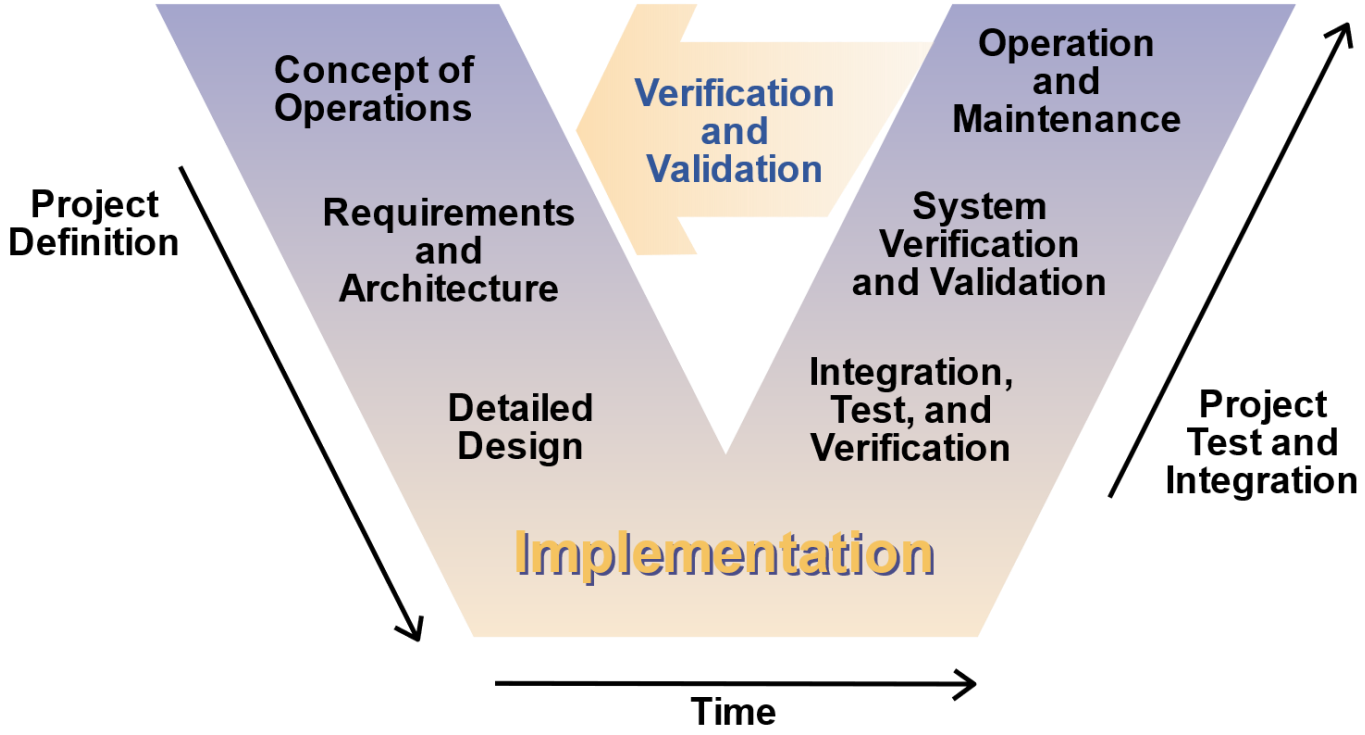


Achtung: Probleme werden meist erst ganz zum Schluss festgestellt und können einen Wiedereinstieg in eine frühere Phase erfordern, im schlimmsten Fall bis hin zur Änderung der Anforderungen.

Andere wasserfallbasierte Modelle

- V-Modell

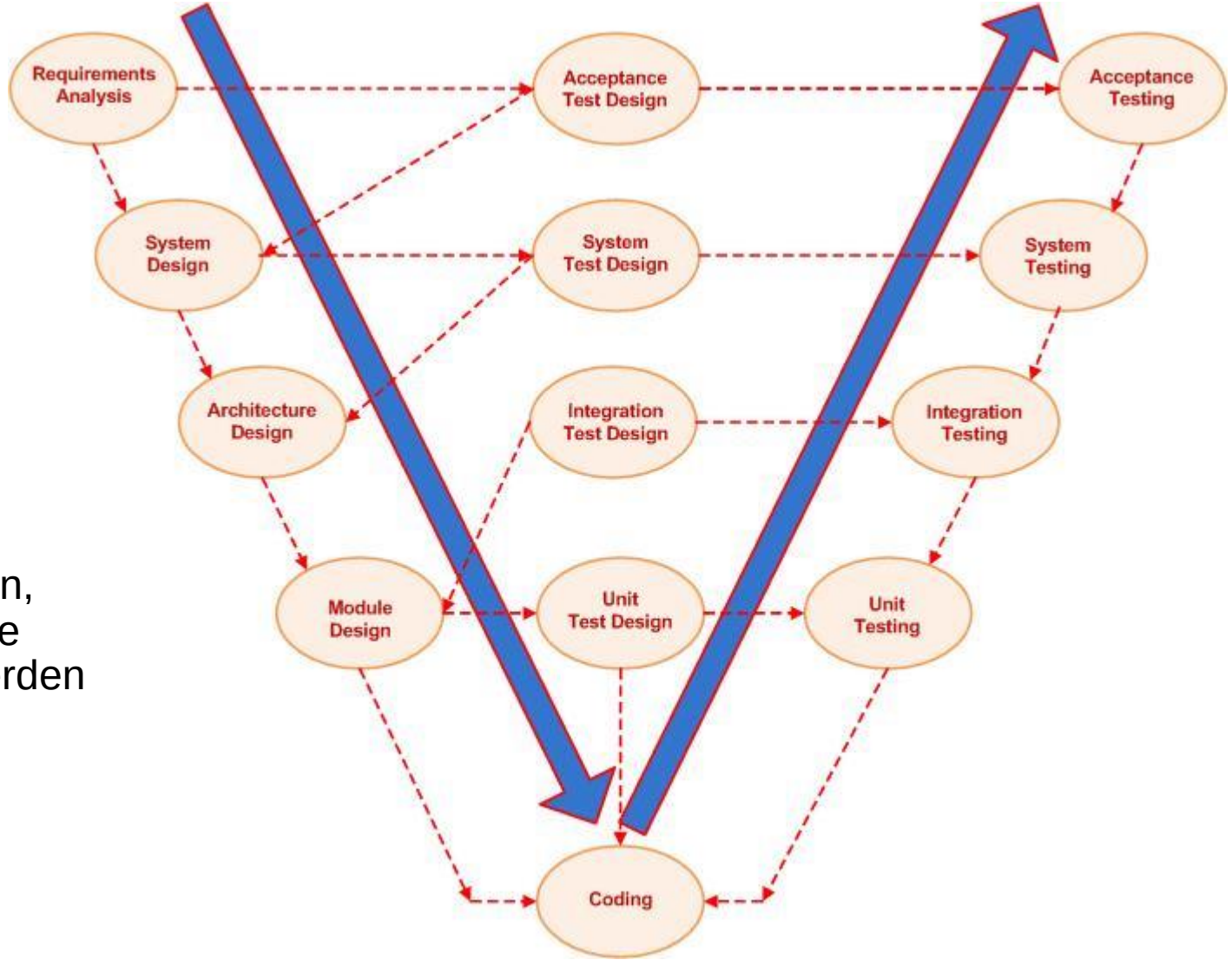
- Das V-Modell ist eine Abwandlung des Wasserfall-Modells, welches die notwendige Validierung der jeweiligen Phase gegenüberstellt.



Andere wasserfallbasierte Modelle

- V-Modell, ausführlicher

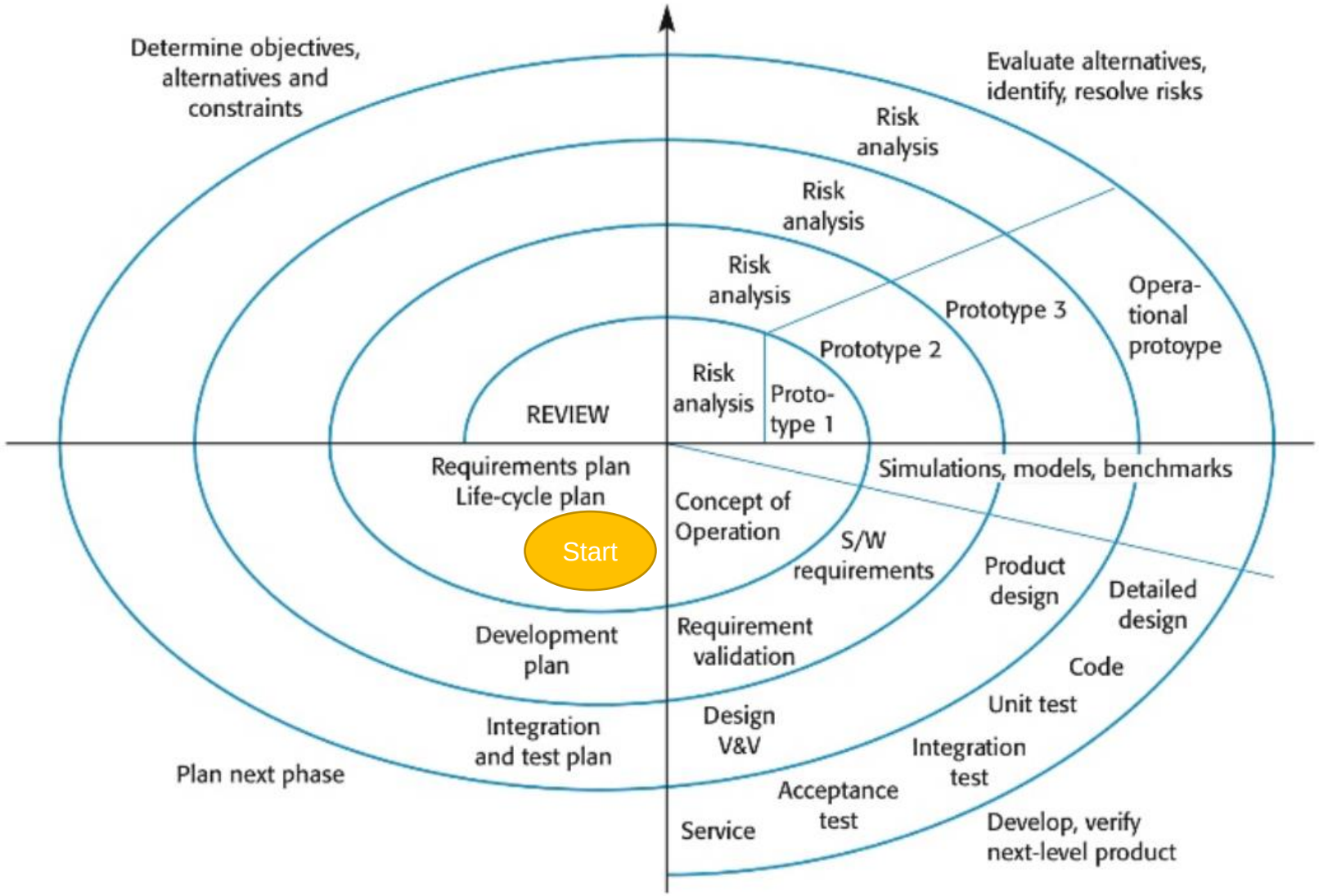
- Das V-Modell ähnelt daher dem Wasserfallmodell, legt aber den Schwerpunkt auf die Ausgabe geeigneter Testspezifikations-dokumente für jede Phase.
- In dieser Darstellung hier sieht man, dass in jeder Phase entsprechende Test-Spezifikationen produziert werden sollen.



Andere wasserfallbasierte Modelle

- Spiral-Modell

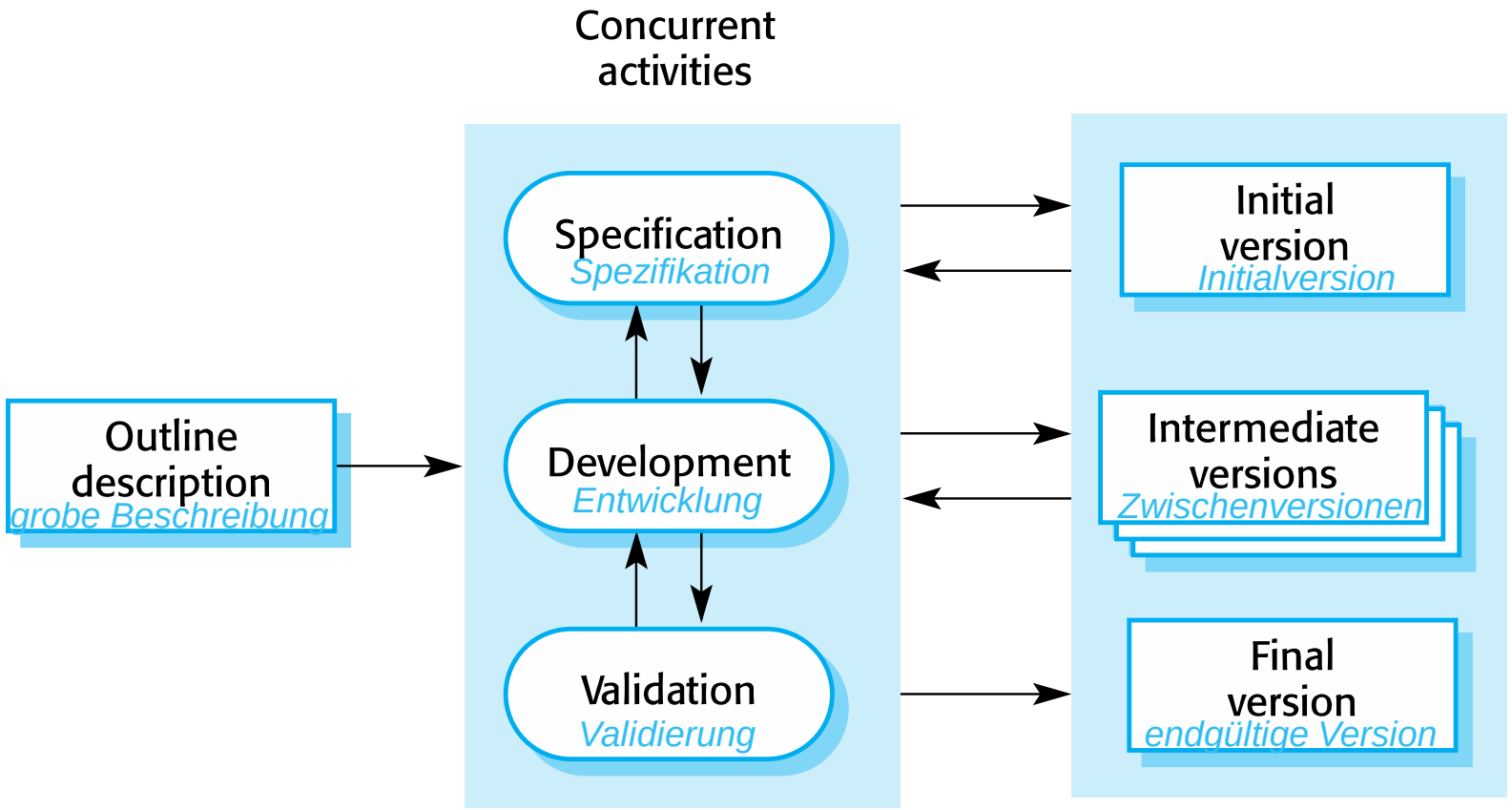
- ähnlich wie Wasserfall, bietet aber Zwischen-prototypen zur Überprüfung
- das "echte" Endprodukt jedoch, das mit der endgültigen Technologie entwickelt wurde, ist erst ganz zum Schluss prüfbar



Probleme mit allen wasserfallbasierten Modellen

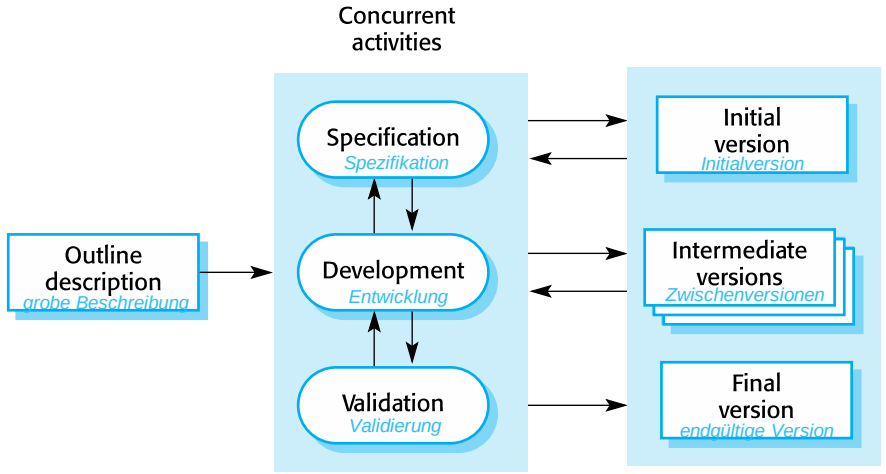
- Das eigentliche Softwareprodukt (**laufende Software**) wird erst nach einem **langen Zeitraum** (manchmal mehrere Jahre) erstellt.
- Zu diesem Zeitpunkt
  - können erst echte Benutzertests durchgeführt und kritische Designfehler oder missverstandene Anforderungen aufgedeckt werden
  - können sich die Anforderungen wieder geändert haben
  - können Technologien bereits veraltet sein
- Aber am wichtigsten ist: Zu diesem Zeitpunkt **erfordern alle Änderungen einen großen Aufwand**!

Inkrementelle Entwicklung



Inkrementelle Entwicklung

- Inkrementelle Entwicklung: Man beginnt mit einer groben Beschreibung des Systems und einzelnen Meilensteinen (im agilen Kontext oft Epics).
- Die Software wird dann nicht als ein einziges großes Produkt entwickelt, sondern als eine Abfolge von Versionen.
  - Zunächst eine erste Minimalversion, auf die viele Inkremente folgen, die weitere Funktionen hinzufügen.
  - Spezifikation, Entwicklung und Validierung erfolgen mehr oder weniger gleichzeitig mit Blick vor allem auf das nächste Inkrement.



Vorteile der inkrementellen Entwicklung

- Die **Kosten** für die Anpassung an sich **ändernde Kundenanforderungen** sind **geringer**,
  - denn der Umfang der Analyse und der Dokumentation, die neu erstellt werden müssen, ist viel geringer als beim Wasserfallmodell.
- Es ist **einfacher, Kundenfeedback** zum aktuellen Stand der Entwicklung **zu erhalten**,
  - denn die Kunden können diesen vorgeführt bekommen und direkt Feedback geben.
- **Eine schnellere Lieferung und Bereitstellung** von Software ist möglich, da man bereits Zwischenversion installieren kann.
  - Die Kunden können auch schon viel früher die Software nutzen als dies bei einem Wasserfallverfahren möglich ist.
  - Die Kunden erhalten Software, die viel direkter an Ihre Bedürfnisse angepasst wurde.

Probleme der inkrementellen Entwicklung

- Der **Prozess** ist **nicht sichtbar und messbar**.
  - Wenn Systeme schnell entwickelt werden, ist es nicht kosteneffizient, ausführliche Dokumentation zu jeder Version des Systems zu erstellen.
  - Aber: Manager:innen brauchen regelmäßige Ergebnisse, um den Fortschritt zu messen.
  - Manager:innen können nicht einfach einen Plan abhaken und die Erstellung gewisser Dokumente und Spezifikationen überprüfen
- **Die Systemstruktur neigt dazu, sich zu verschlechtern**, wenn **neue Inkremente** hinzugefügt werden.
  - Wenn nicht viel Zeit und Geld in die Verbesserung der Software durch Refactoring investiert wird, erhält die Software eine schlechte Wartbarkeit. Weitere Softwareänderungen werden immer schwieriger, aufwändiger und kostspieliger.

  → Wir brauchen **Architekturarbeit**, um **das große Ganze** im Auge zu behalten, z.B. die Einheitlichkeit des Systems

Integration und Konfiguration

- englisch *Integration and configuration*
- Basiert auf der **Wiederverwendung von Software**, wobei Systeme aus bestehenden Komponenten oder Anwendungssystemen (manchmal auch COTS - Commercial-off-the-shelf systems genannt) zusammengesetzt werden.
- Wiederverwendbare Elemente können so **konfiguriert werden**, dass ihr Verhalten und ihre Funktionalität an die Anforderungen des Nutzers angepasst werden
- Die Wiederverwendung ist heute der Standardansatz für den Aufbau vieler Arten von **Geschäftssystemen**, z.B. SAP-Systeme

Arten von wiederverwendbarer Software

- **Eigenständige Anwendungssysteme** (manchmal auch COTS, commercial off-the-shelf systems, genannt), die für den Einsatz in einer bestimmten Umgebung konfiguriert sind.
- **Webservices**, die nach Standards entwickelt werden (z.B. SOAP oder REST) und über das Netzwerk aufgerufen werden.
  - oft Cloud-Dienste
- **Komponenten-Frameworks** wie .NET, Spring Framework (Java),
- **Software-Bibliotheken,** z.B. für Netzwerkaufrufe, Bildverarbeitung, Datenstrukturen

Die beiden letztgenannten sind Teil fast jedes modernen Softwareprojekts. Ein Teil von "Integrations- und Konfigurationsarbeit" muss also in jedem Projekt geleistet werden!

Integration und Konfiguration

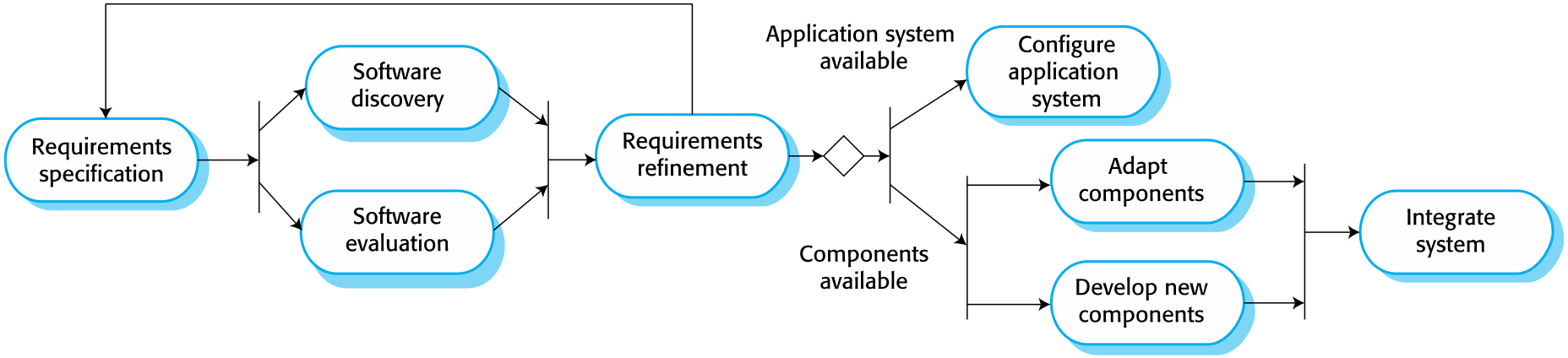
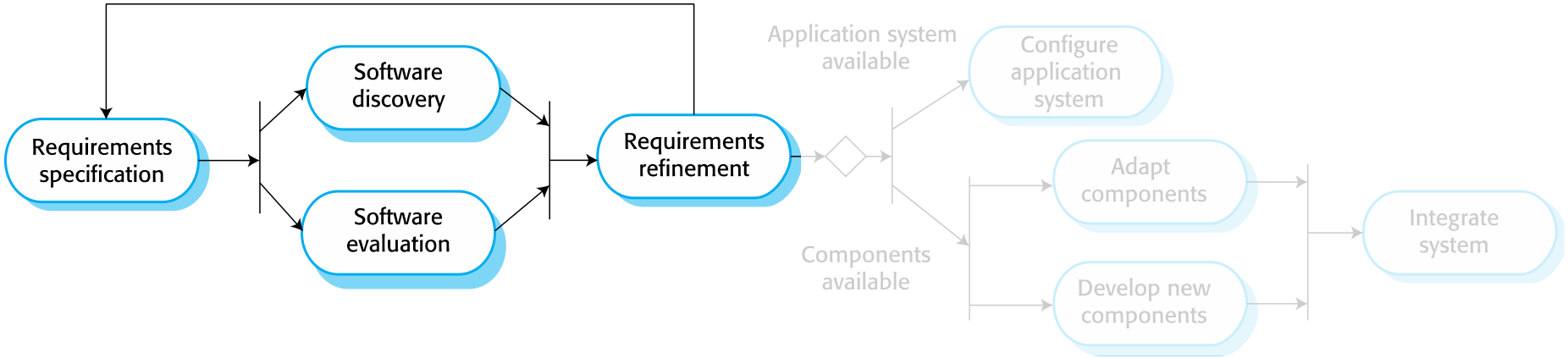


Abbildung von Ian Sommerville

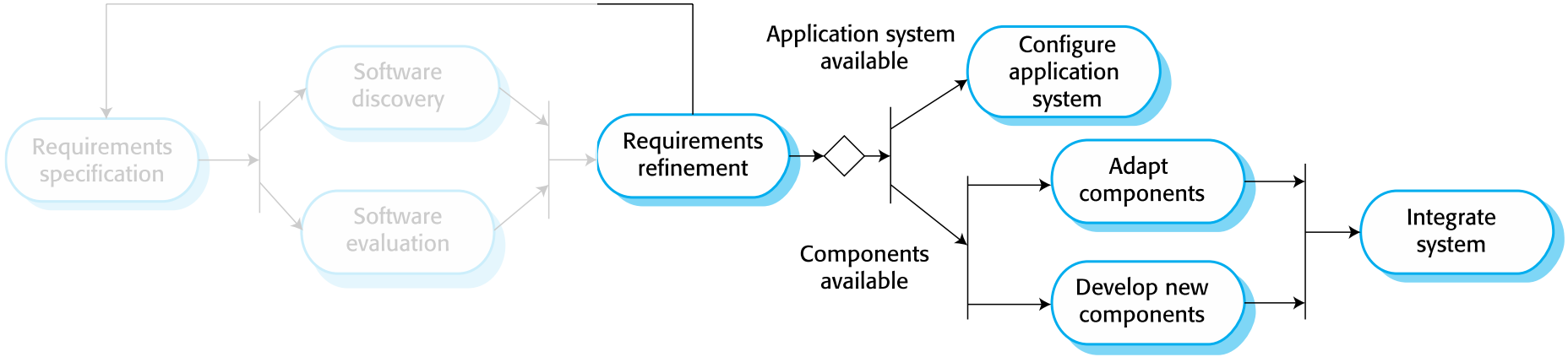
- auch bekannt als *Wiederverwendungsorientierte Softwareentwicklung*
- Typischer Prozess für "Make or Buy"-Entscheidungen

Integration und Konfiguration



- Möglichst vollständige Anforderungen sind wichtig.
- Man führt dann eine Marktanalyse durch und sucht nach vorhandener, passender Software, die man gleichzeitig evaluiert, um festzustellen, wie gut sie die Anforderungen umsetzt.
- Anforderungen müssen hierbei oft noch einmal verfeinert werden, weil einem oft erst bei näherer Betrachtung weitere Detailfragen auffallen.

Integration und Konfiguration



- Anschließend gibt es zwei Möglichkeiten: Ein geeignetes System wurde gefunden und muss lediglich geeignet konfiguriert werden.
- Oder: Man setzt sein System aus verschiedenen Teilkomponenten oder -systemen zusammen und entwickelt etwaige fehlende Kompomenten neu.
  - Hier ist die Integration essenziell – es muss ein funktionierendes Gesamtsystem entstehen
  - Oft scheitert es bei großen Projekten daran.

Integration und Konfiguration: Vorteile und Nachteile

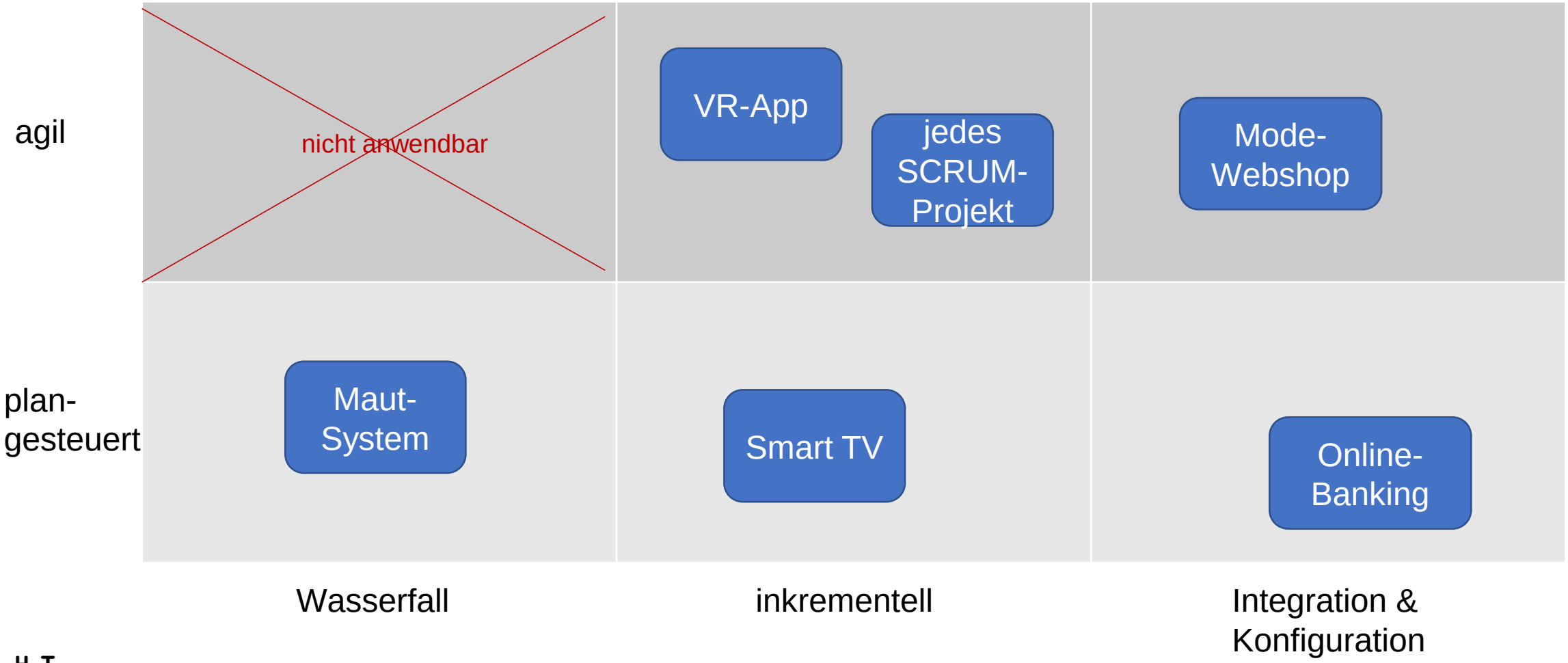
Vorteile:

- Geringere Kosten und Risiken, da weniger Software von Grund auf entwickelt werden muss
- Schnellere Lieferung und Bereitstellung des Systems

Nachteile:

- Kompromisse bei den Anforderungen sind unvermeidlich, so dass das System möglicherweise nicht den tatsächlichen Bedürfnissen der Nutzer entspricht
- Man hat keine Kontrolle über die Entwicklung von wiederverwendeten Systemelementen
  - Wann bekommen diese Updates?
  - Wie lange werden sie überhaupt weiterentwickelt?
  - Wie stark ändern sich die Systeme?
- Fertige Komponenten haben in der Regel viel mehr Funktionen als tatsächlich benötigt werden.

Zusammenfassung mit Beispielen



Frage: Zwei Beispielsysteme

System 1: Ein **Autoradio mit Amazon Alexa-Funktionalität** soll entwickelt werden.

System 2: Es soll eine neue **Dating-App** für Handys entwickelt werden.

Wie würden Sie vorgehen?

- Plangesteuert? Agil?
- Wasserfall? Inkrementelle Entwicklung? Integration und Konfiguration?