hochschule mannheim
Fakultät für Informatik

Programmierung 2

Objektorientierung verstehen

Prof. Dr. Oliver Hummel

Was erwartet Sie heute?

1. Wiederholung der Grundlagen der Objektorientierung
2. Überblick über objektorientierte Modellierung und Architektur
3. Vereinfachtes Vorgehen beim Entwurf von OO-Programmen

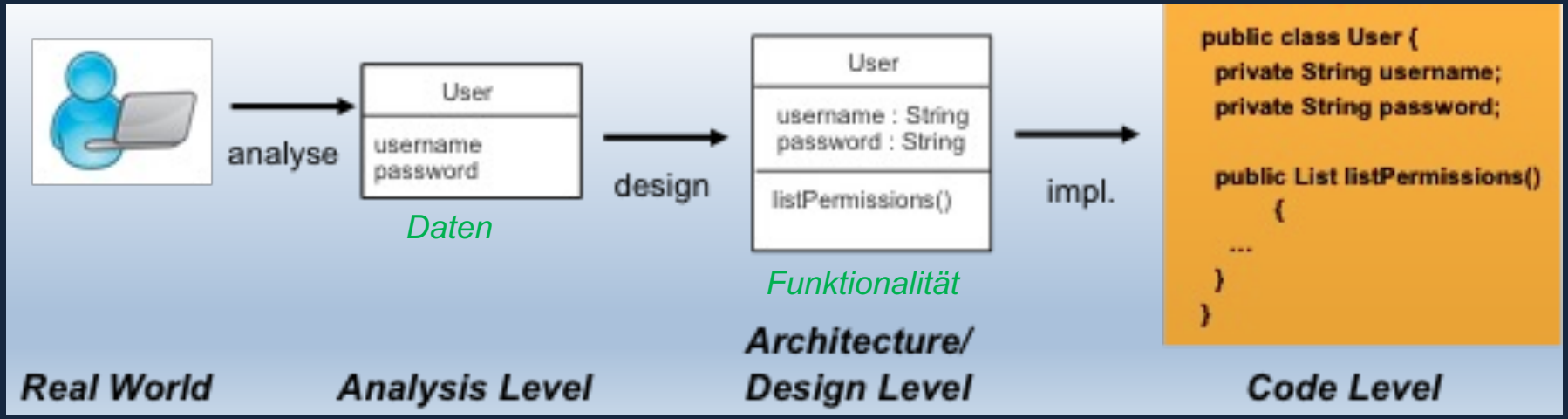
Wegweiser PR2-Inhalte

- Objektorientierung / Klasse Object (Wiederholung)
- Polymorphismus (Vertiefung)
- JUnit und Software-Qualität
- Innere Klassen und Lambdas
- Input/Output (Grundlagen)
- Exception Handling
- Grundlagen der Nebenläufigkeit (Threads)
- Generics (Grundlagen)
- Collection Framework
- Wichtige Datenstrukturen
  - Hash(maps), Bäume, Tries, Graphen
- Sortieren

Die Idee der **Objektorientierung** ist, die **reale Welt** im Softwaresystem „**nachzubauen**“

Nutzung eines Gemeinsamen Vokabulars über alle Entwicklungsstufen hinweg

- ausgehend von den Datenstrukturen



Blitz-Abfrage OO-Grundlagen

Sind Ihnen folgende Punkte geläufig?

1. Unterschied Klasse und Objekt
2. Geheimnisprinzip
3. Punktnotation für Methoden und Attribute von Objekten
4. Nutzen und Nutzung der toString-Methode
5. wie sich Objekte vergleichen lassen (equals-Methode)
6. Unterschied von Stack und Heap und Zusammenhang mit Objekten
7. Konstruktoren
8. Überladen von Methoden (und Konstruktoren)
9. this
10. static
11. Wrapper-Klassen für primitive Datentypen und Autoboxing

1. Objektorientierung heißt, dass zusammenhängende Daten und zugehörige Funktionalität in einer Klasse zusammengefasst werden

2. Software-Objekte sind wie Sachbearbeiter, die gemeinsam eine Firma (-> das Programm) am Laufen halten

Wdh.: Grundlegendes Entwurfsvorgehen

Aus Gründen der Änderbarkeit und Testbarkeit von Programmen empfiehlt es sich folgende (erste) **Heuristiken** zu befolgen:

1. Beginnen Sie mit der **Datenmodellierung**
   - der Klassen und ihrer Attribute
2. Fügen Sie die **Geschäftslogik** dort hinzu, wo die entsprechenden **Daten** sind
   - z.B. Methoden zur Bearbeitung von Geschäftsvorfällen
3. Befolgen Sie das **Geheimnisprinzip**
   - Geschäftsobjekte sollten innerhalb des Systems verbleiben
   - Halten Sie die Schnittstelle des Systems (die public-Methoden) so klein wie möglich
4. **Trennen** Sie Ein- und Ausgabe (**UI**) von der **Geschäftslogik** (Architektur!)
   - die UI liegt „außerhalb“ des Systems und nutzt dessen Schnittstelle
   - achten Sie auf Testbarkeit (insbes. keine Nutzeringaben in Geschäftsmethoden!)

Mini-Übung von Zuhause

Gibt es Rückmeldungen oder –fragen zu diesen Aufgaben?

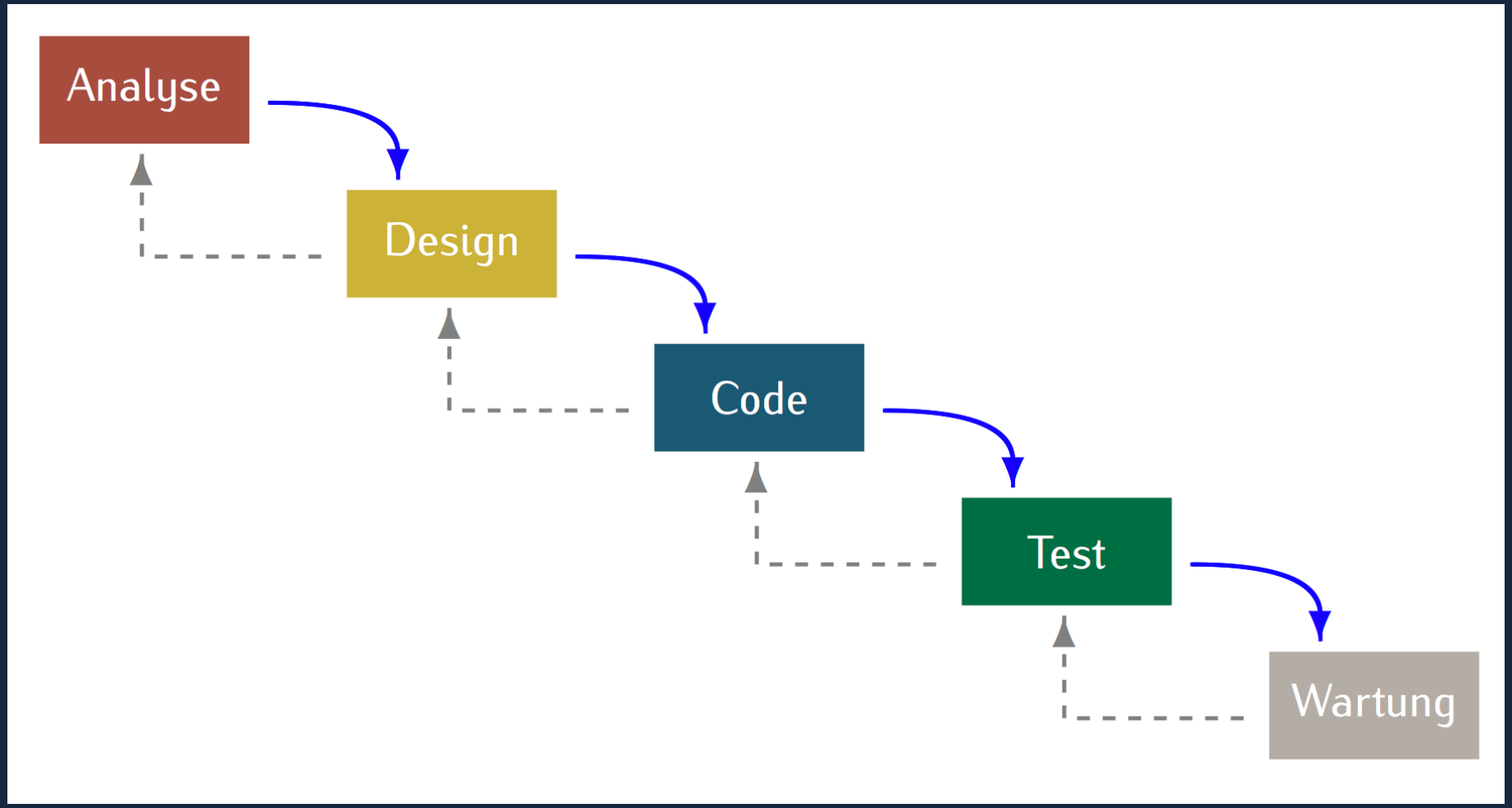
1. Kontext **Bank**: Implementieren Sie ein Bankkonto mit einem Inhaber(namen), einer Kontonummer sowie einem Kontostand. Ergänzen Sie dann Methoden zum Ein- und Auszahlen von Geldbeträgen.

2. Kontext **Autovermietung**: implementieren Sie eine PKW-Klasse, mit passenden Attributen und der Funktionalität um gefahrene Kilometer einzugeben, die auf den Kilometerstand addiert werden

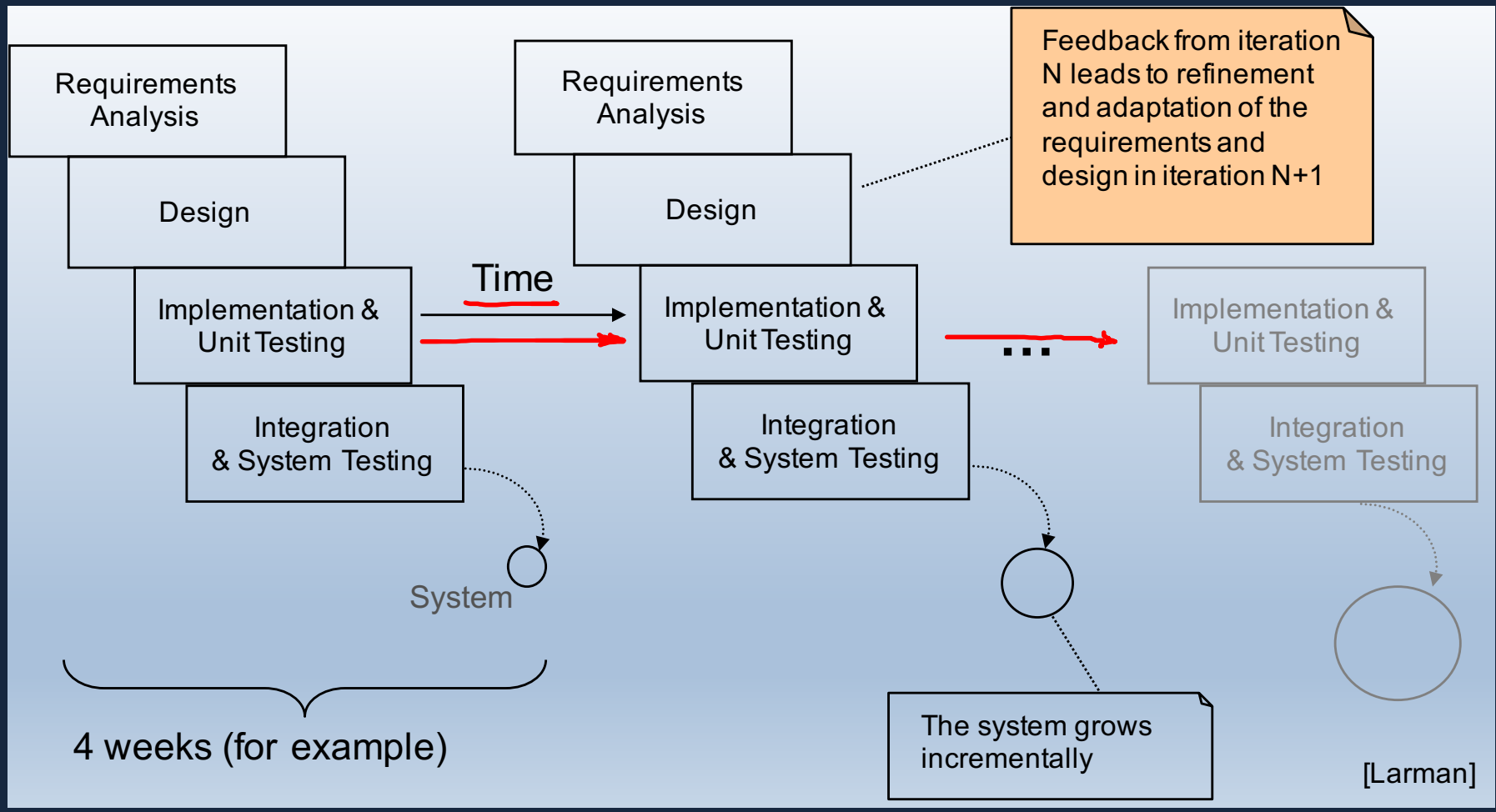
3. Kontext **Weinhandlung**: Implementieren Sie eine Klasse für einen Wein, die die in einer Flasche enthaltene Alkoholmenge berechnen kann

4. Kontext **Lauf-App**: Implementieren Sie eine Klasse für einen Lauf, die an Hand der gespeicherten Strecke und der dafür benötigten Zeit die durchschnittliche Zeit pro Kilometer berechnet

Lebenszyklus eines Features



System wächst durch Hinzufügen von Features über die Zeit...



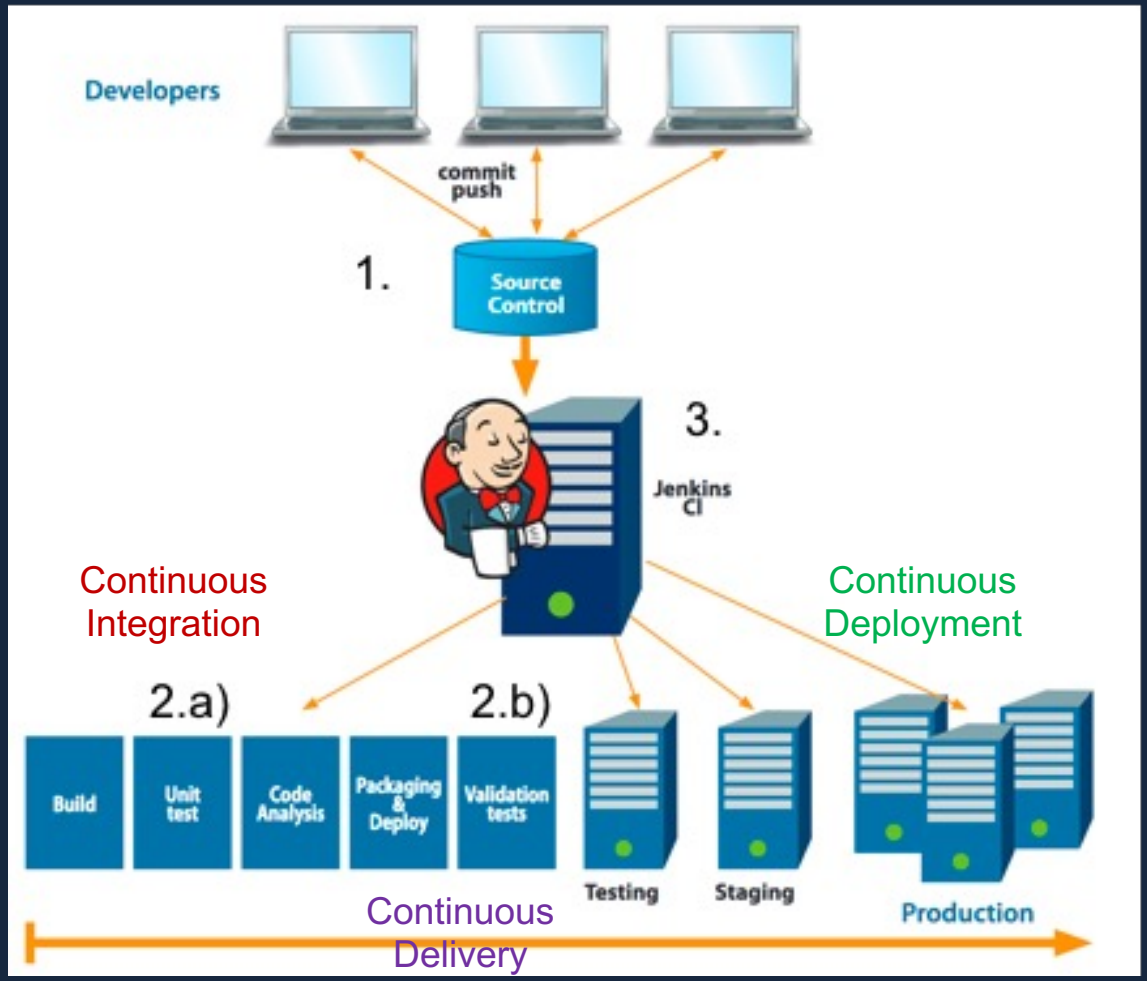
C. Larman: Applying UML and Patterns

„Big-Bang-Integration“ hat sich als nicht durchführbar erwiesen

Heute gilt **Continuous Integration** als Best Practice:

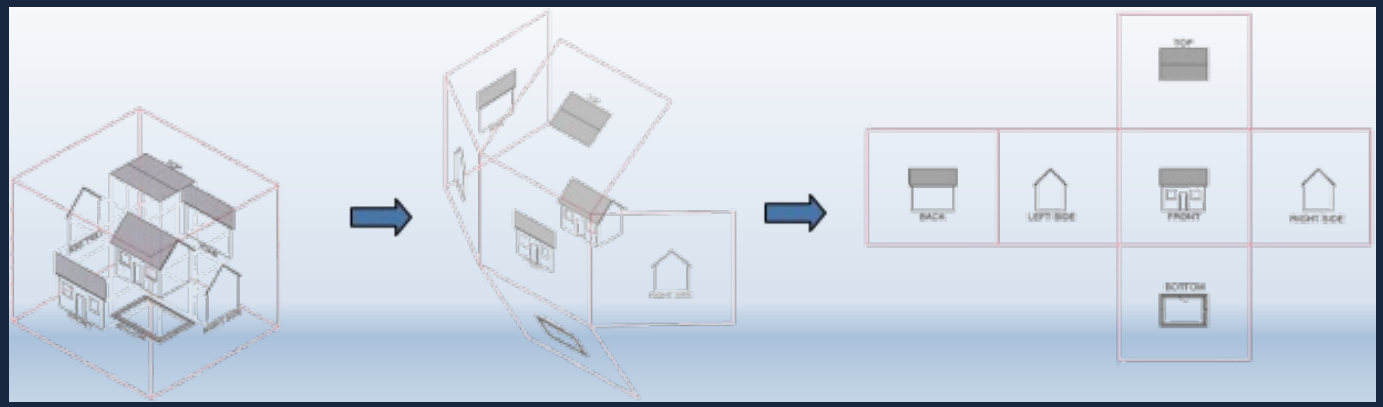
- jeder Entwickler comittet häufig in das gemeinsame (Git-)Repository
- → das System wird jedes Mal automatisch gebaut und getestet

Bildquelle: zend.com

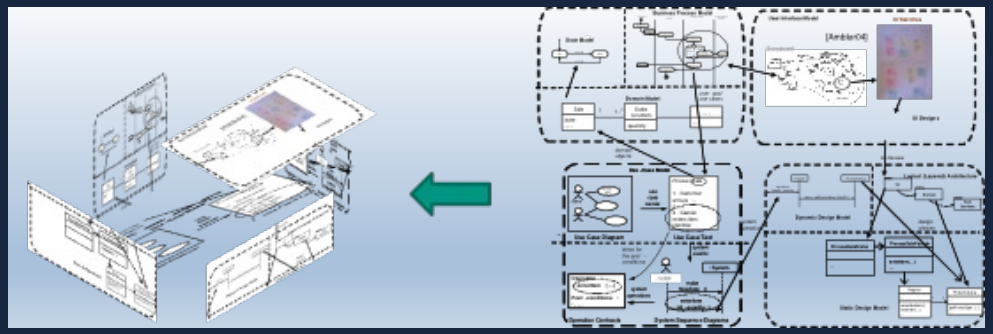


UML-Modelle sind verschiedene (unvollständige) Sichten auf das Gesamtsystem

- *nur der Code (+ Deployment-Skripte) enthält wirklich alle Informationen*

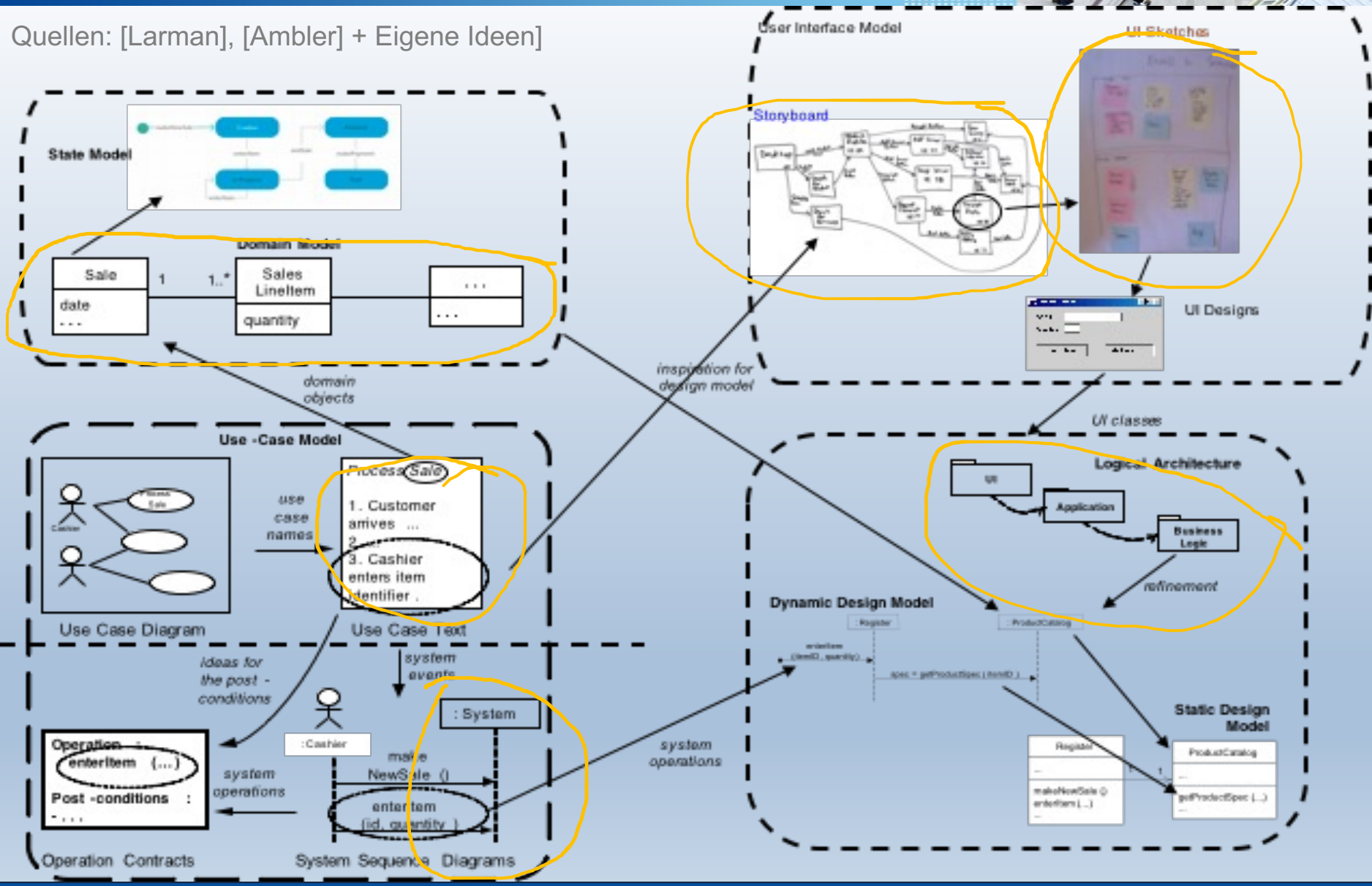


Courtesy of Prof. Atkinson, Uni MA



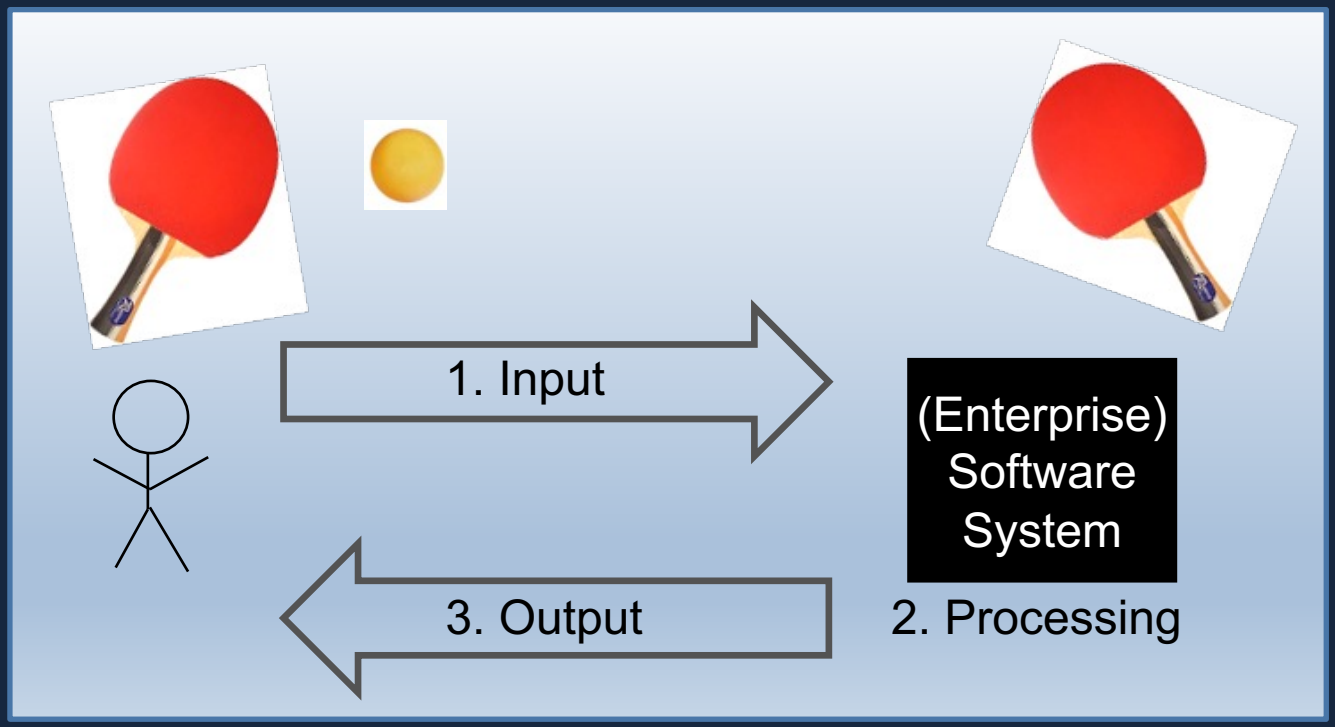
Agile Modelling als mentales Modell für OOP

Quellen: [Larman], [Ambler] + Eigene Ideen]



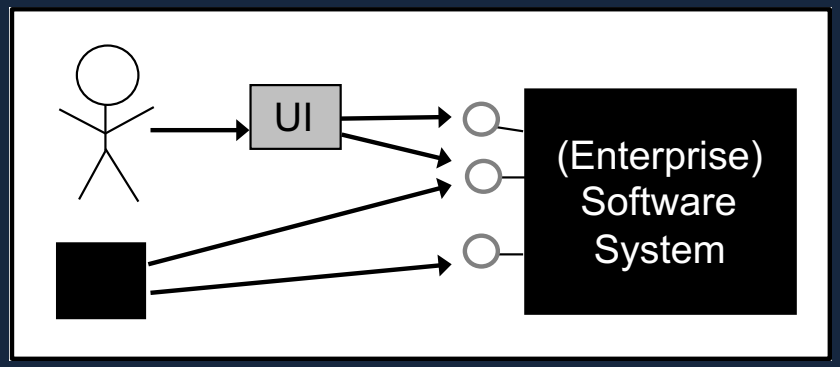
Wichtig für den Moment I

1) Ein Gefühl für die Anforderungen, den Workflow im System sowie die Systemgrenze zu entwickeln

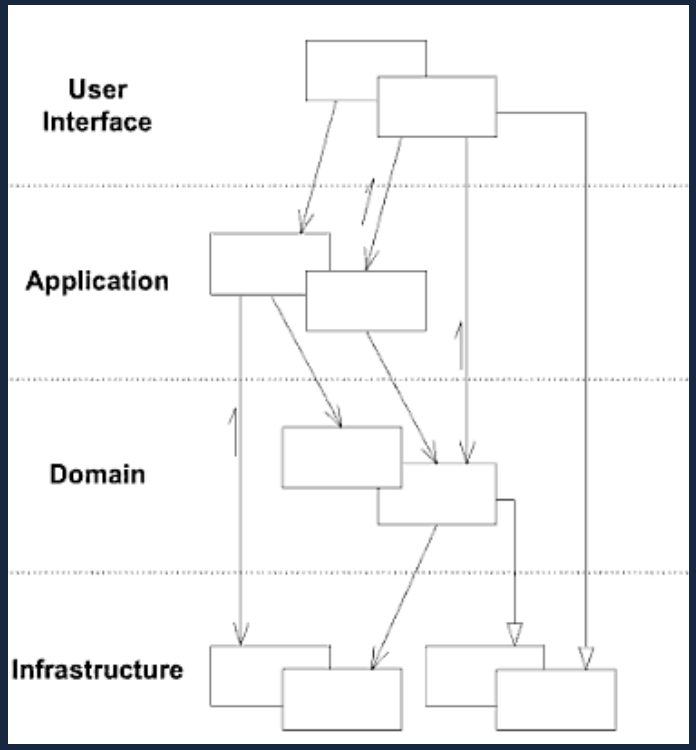


Wichtig für den Moment II

2) UI vom eigentlichen System (Blackbox) trennen



3) Das System über Layer strukturieren



Wichtig für den Moment III

4) Die Zuweisung von Funktionalität zu Datenklassen verstehen

Im Domain Layer ist die eigentliche **Geschäftslogik** enthalten

- hier muss jetzt das eigentliche **objektorientierte Design (OOD)** passieren
  - inspiriert von den Use Cases, den Contracts und dem Datenmodell finden wir:

- **doing reponsibilities**
  - Sale muss SaleLineItems erzeugen

- **knowing responsibilities**
  - Sale muss seinen Gesamtpreis (total) kennen

Diskussion / Übung

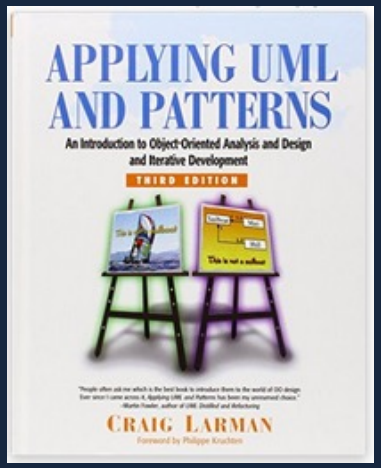
Betrachten wir uns folgende Aufgabenstellung:

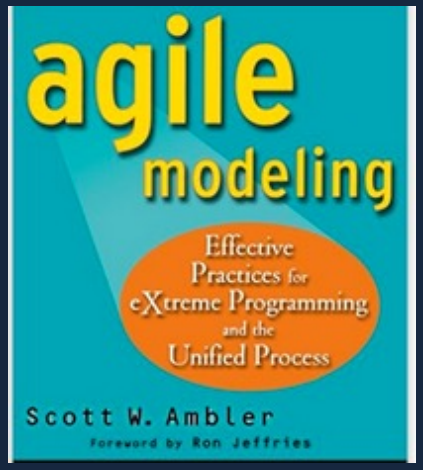
Als Kunde möchte ich im Online-Shop einkaufen können, um mir Produkte bequem nach Hause liefern zu lassen.

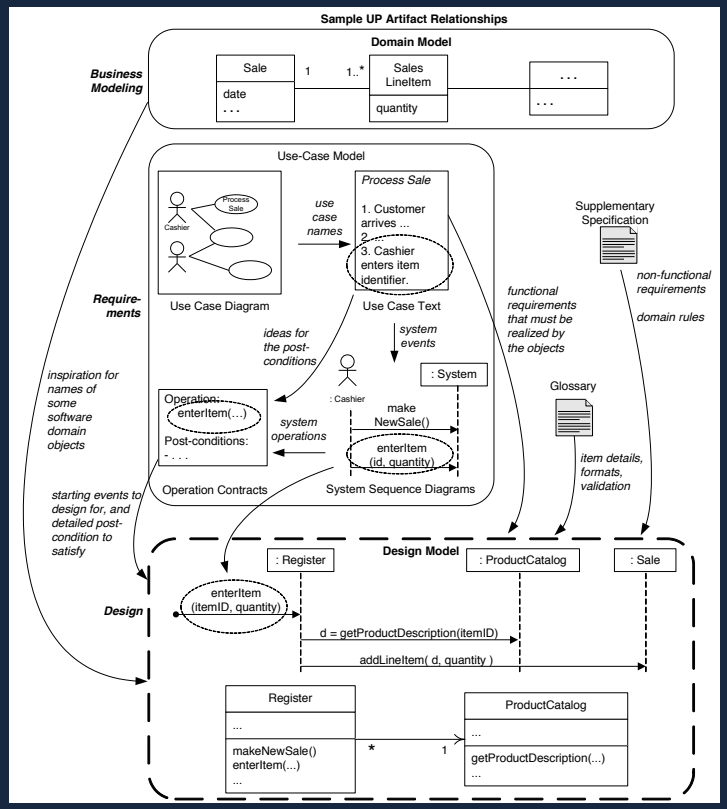
= RUP + Agile

Begriff geprägt von Scott Ambler

- etwa modelbasierte agile Entwicklung
- verbreitet auch von Craig Larman







→ macht die "Modellierungsgedanken" der Software-Entwicklung explizit