Clean Code und Wartbarkeit in Softwareprojekten

Leon Palluch

© SEQIS GmbH, 2022

Wartbarkeit vs Softwarequalität

- Etablierte Aspekte der Softwarequalität
  - Sicherheit
  - Usability & User Experience
  - Automatisiertes funktionales Testen

- Wartbarkeit
  - Wird nicht vom Endnutzer wahrgenommen
  - Mittel- und langfristige Auswirkungen

© SEQIS GmbH, 2022

Technische Schulden

- Schulden ↑ → Probleme ↑
  - Frustration
  - Kein Vertrauen in die Codebasis
  - Legacy Code
  - Höhere Fehlerwahrscheinlichkeit
  - Höherer Zeitaufwand

© SEQIS GmbH, 2022

Raum für Qualität schaffen

- Qualität unter Zeitdruck ist nicht möglich
  - Rechtfertigt technische Schulden
  - Zerstört Motivation
- Qualität muss verdient sein
  - Priorisierung
  - Plan
  - Förderung

© SEQIS GmbH, 2022

Wie man ein Gespür für Wartbarkeit entwickelt

- Austausch mit Kollegen

- Die richtige Denkweise
  - Code soll für **andere** verständlich sein
  - Fehler verstehen, dann erst beheben
  - Boy Scout Regel
  - Refactoring
  - Sinnhaftigkeit hinterfragen

© SEQIS GmbH, 2022

1. Streben Sie ein qualitätsorientiertes Mindset an

- Schaffen Sie Zeit für Qualität
- Übernehmen Sie Verantwortung für Ihren Code

© SEQIS GmbH, 2022

Automatisierung

- Kontinuierlicher Check
  - Statische Codeanalyse
  - Beautifier beim Speichern

- Beim Einchecken von Code
  - Automatisierte Tests
  - Formatierung und Konventionen prüfen

© SEQIS GmbH, 2022

Automatisierung

- Doppelter Check
  - Lokal konfigurieren
  - In der CI/CD Pipeline

© SEQIS GmbH, 2022

Code Reviews

- Wann?
  - Vor dem Abschluss jedes Features
  - Bei Bedarf mehrmals

- Vorteile
  - Missverständnisse klären
  - Zweite Meinung
  - Wissen weitergeben

© SEQIS GmbH, 2022

Code Reviews

- Richtige Einstellung
  - Unklarheiten ansprechen
  - Respektvoll bleiben

© SEQIS GmbH, 2022

2. Integrieren Sie Wartbarkeit in den Entwicklungsprozess

- Automatisieren Sie Qualitätschecks
- Integrieren Sie Code Reviews in den Entwicklungsprozess

© SEQIS GmbH, 2022

Wozu automatisiert testen?

- Billig und zuverlässig auf Fehler prüfen

- Vertrauen in die Codebasis
  - Erleichtert Änderungen
  - Ermutigt zum Refactoring

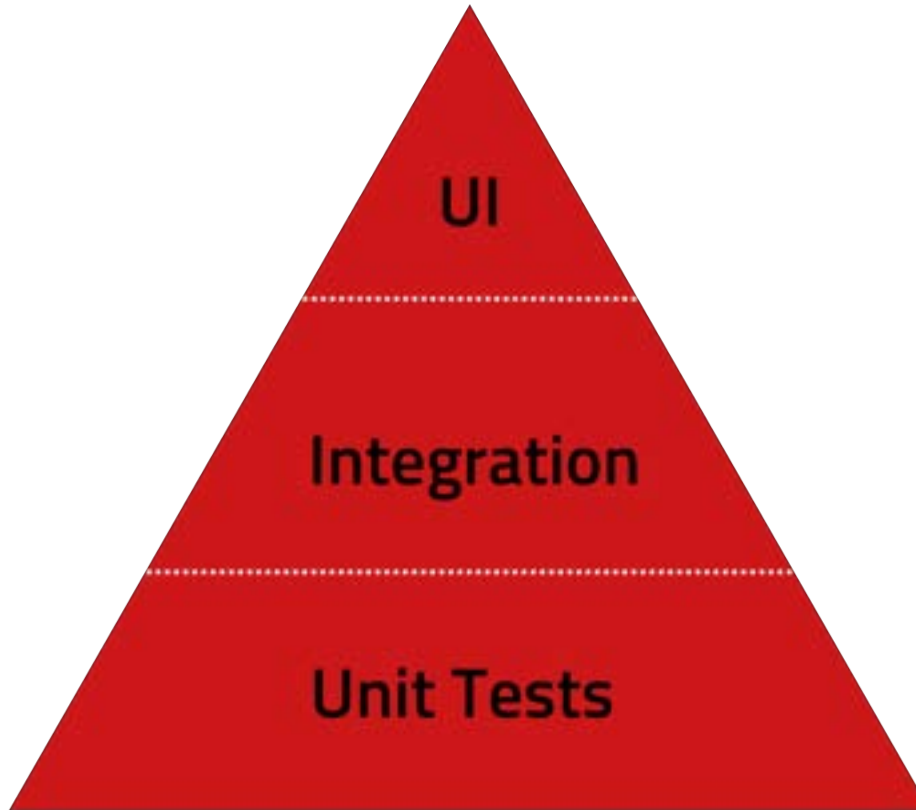
© SEQIS GmbH, 2022

Testen aus Sicht der Wartbarkeit

- Just Enough
  - Keine 3rd Party Logik
  - Brüchige Tests vermeiden
  - Verhalten statt Implementation

© SEQIS GmbH, 2022

Die Testpyramide



© SEQIS GmbH, 2022

3. Nutzen Sie automatisierte Tests

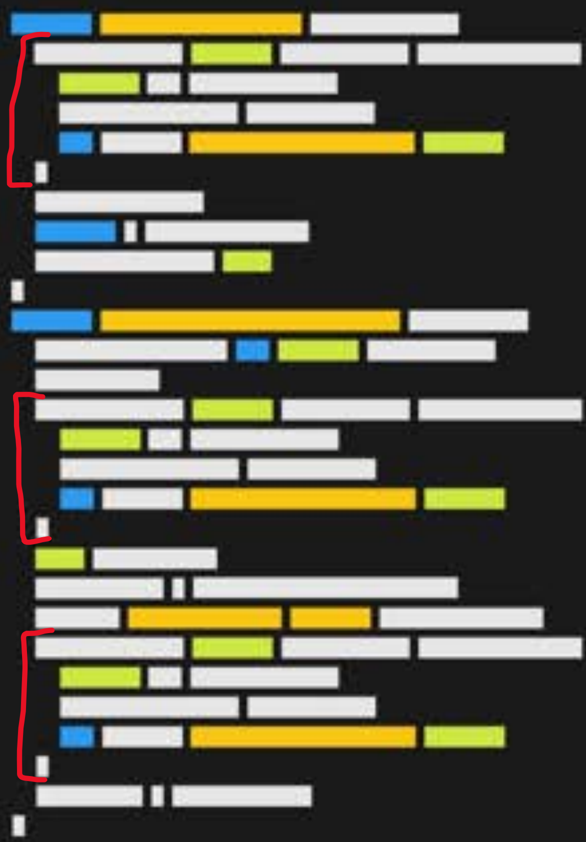
- Um Entwicklern Sicherheit zu geben

© SEQIS GmbH, 2022

Don‘t Repeat Yourself (DRY)

- Keinen Code kopieren!

© SEQIS GmbH, 2022



Don‘t Repeat Yourself (DRY)

- Generalisierung
  - Bei ähnlicher Funktionalität
  - Beim Refactoring beachten
  - ABER: Übermaß erhöht Kopplung und Komplexität

© SEQIS GmbH, 2022

Don‘t Repeat Yourself (DRY)

- Wiederholten Designaufwand vermeiden
  - An bestehenden Lösungen orientieren
  - Konventionen für Designentscheidungen

© SEQIS GmbH, 2022

4. Vermeiden Sie Wiederholungen

- Kopieren Sie keinen Code
- Generalisieren Sie bei ähnlicher Funktionalität
- Vermeiden Sie wiederholten Designaufwand

© SEQIS GmbH, 2022

Konsistenter Code

- Ähnliche Konzepte gleich behandeln
  - Ordnerstruktur
  - Namensgebung
  - Implementation
  - …

© SEQIS GmbH, 2022

Konsistente Namensgebung

```
class MyAwesomeClass {
  String userName;
  String companyName

  MyAwesomeClass(String user, String company) {
  }

  String getNameOfUser() {

  }

    void renameUser(String name) {

  }

  String getCompanyName() {

  }

  void setCompanyName(String newNameOfCompany) {

  }
}
```

© SEQIS GmbH, 2022

Konsistente Namensgebung

```
class MyAwesomeClass {
  String userName;
  String companyName

  MyAwesomeClass(String userName, String companyName) {

  }

  String getUserName() {

  }

  void setUserName(String name) {

  }

  String getCompanyName() {

  }

  void setCompanyName(String name) {

  }
}
```

© SEQIS GmbH, 2022

Konsistente Implementation

```
Date getPreviousMonth() {
    Date previousMonth = this._currentDate.subtract(months: 1);
    return previousMonth;
}

int getNextMonth() {
    return this._currentDate.month + 1;
}
```

© SEQIS GmbH, 2022

Coding Conventions

- Konkrete Richtlinien
  - Formatierung und Struktur
  - Namensgebung
  - Programmierpraktiken
  - Architekturelle Vorgaben

© SEQIS GmbH, 2022

Coding Conventions

- Universelle Konventionen
  - Allgemein
  - Sprachenspezifisch

- Vom Team erstellt und aktualisiert

© SEQIS GmbH, 2022

5. Bleiben Sie konsistent

- Behandeln Sie ähnliche Konzepte konsistent
- Definieren Sie Coding Conventions

© SEQIS GmbH, 2022

Readability > Everything

- Readability > Performance
  - Kleine Unterschiede haben keine Auswirkung auf UX
  - Wichtige Einflüsse sind selten Low Level
- Readability > Brevity
  - Weniger Code ist nicht gleich besser
  - Nur wenn Übersichtlichkeit besser wird

© SEQIS GmbH, 2022

Kompakte Schreibweise

```
if ((something > somethingElse) && (somethingIsEnabled)) {
  foo = service.getThingA();
}
else if (something > 0){
  foo = service.getThingB();
}
else {
  foo = defaultFoo;
}
```

© SEQIS GmbH, 2022

Kompakte Schreibweise

```
if (something > somethingElse && somethingIsEnabled) foo = service.getThingA();
else if (something > 0) foo = service.getThingB();
else foo = defaultFoo;
```

© SEQIS GmbH, 2022

Kompakte Schreibweise

```
foo = something > somethingElse && somethingIsEnabled ? service.getThingA() : something > 0 ? service.getThingB() : defaultFoo;
```

© SEQIS GmbH, 2022

6. Priorisieren Sie Lesbarkeit

- Bevorzugen Sie Lesbarkeit gegenüber Performance
- Bevorzugen Sie Lesbarkeit gegenüber Brevity

© SEQIS GmbH, 2022

Verständlicher Code

- Logik eindeutig machen
  - Sprechende Namen

© SEQIS GmbH, 2022

Sprechende Namen

```
double updateV(double vOld, double a, double f, double max) {
    double v = vOld + a;
    v = v * f;

    if (v > max) {
        return max;
    }
    return v;
}
```

© SEQIS GmbH, 2022

Sprechende Namen

```
double getUpdatedVelocity(double previousVelocity, double acceleration, double frictionPercentage, double maxVelocity) {
    double acceleratedVelocity = previousVelocity + acceleration;
    double velocityAfterFriction = acceleratedVelocity * frictionPercentage;

    if (velocityAfterFriction > maxVelocity) {
        return maxVelocity;
    }
    return velocityAfterFriction;
}
```

© SEQIS GmbH, 2022

Verständlicher Code

- Logik eindeutig machen
  - Sprechende Namen
  - Negative Conditionals vermeiden
  - Doppelte Verneinung vermeiden
  - „Magic Numbers“ vermeiden

© SEQIS GmbH, 2022

Komplexität niedrig halten

- Keep It Simple, Stupid (KISS)
  - Single Responsibility Principle
  - Side Effects vermeiden
  - Verschachtelung vermeiden

© SEQIS GmbH, 2022

Verschachtelung

```
ProcessedEntity processEntity(Entity entity) {
    if (entityIsValid(entity)) {
        if (entity.type == "foo") {
            List<AddOn> addOns = AddOnService.getAddOns();
            for (int i = 0; i < addOns.length; i++) {
                AddOn addOn = addOns[i];
                if (entity.addOnID == addOn.id) {
                    if (addOn.isValid) {
                        return new ProcessedEntity(entity: entity, addon: addOn);
                    }
                    else {
                        throw new InvalidAddOnException();
                    }
                }
            }
        }
        else if (entity.type == "bar") {
            //process "bar" entity
        }
        else if (entity.type == "baz") {
            //process "baz" entity
        }
        //...
    } else {
        throw new InvalidEntityException();
    }
}
```

© SEQIS GmbH, 2022

```
Response processFooEntity(Entity entity) {
    if (entityIsValid(entity) == false) {
        throw new InvalidEntityException();
    }

    List<AddOn> addOns = AddOnService.getAddOns();

    AddOn addOn;
    for (int i = 0; i < addOns.length; i++) {
        addOn = addOns[i];
        if (entity.addOnID == addOn.id) {
            break;
        }
    }

    if (addOn.isValid == false) {
        throw new InvalidAddOnException();
    }
    return new ProcessedEntity(entity: entity, addon: addOn);
}

Response processBarEntity(Entity entity) {
    if (entityIsValid(entity) == false) {
        throw new InvalidEntityException();
    }

    //process "bar" entity
}

Response processBazEntity(Entity entity) {
    if (entityIsValid(entity) == false) {
        throw new InvalidEntityException();
    }

    //process "baz" entity
}
```

© SEQIS GmbH, 2022

7. Sorgen Sie für Übersichtlichkeit

- Schreiben Sie eindeutige Logik
- Halten Sie die Komplexität niedrig

© SEQIS GmbH, 2022

Strukturierter Code

- Nach Konzepten sortieren
  - Imports, Konstanten, Members, Methoden
  - Abhängige und ähnliche Funktionen
  - Zusammengehörige Logik
  - Variablen deklarieren wo sie gebraucht werden

© SEQIS GmbH, 2022

Strukturierter Code

- Übersichtliche Formatierung
  - Einrückung
  - Abstände
  - Visuelle Trennung von Konzepten

© SEQIS GmbH, 2022

```
Response processFooEntity(Entity entity) {
    if (entityIsValid(entity) == false) {
        throw new InvalidEntityException();
    }

    List<AddOn> addOns = AddOnService.getAddOns();

    AddOn addOn;
    for (int i = 0; i < addOns.length; i++) {
        addOn = addOns[i];
        if (entity.addOnID == addOn.id) {
            break;
        }
    }

    if (addOn.isValid == false) {
        throw new InvalidAddOnException();
    }
    return new ProcessedEntity(entity: entity, addon: addOn);
}

Response processBarEntity(Entity entity) {
    if (entityIsValid(entity) == false) {
        throw new InvalidEntityException();
    }

    //process "bar" entity
}

Response processBazEntity(Entity entity) {
    if (entityIsValid(entity) == false) {
        throw new InvalidEntityException();
    }

    //process "baz" entity
}
```

© SEQIS GmbH, 2022

8. Strukturieren Sie Ihren Code übersichtlich

- Gruppieren Sie zusammengehörige Konzepten
- Achten Sie bei der Formatierung auf Übersichtlichkeit

© SEQIS GmbH, 2022

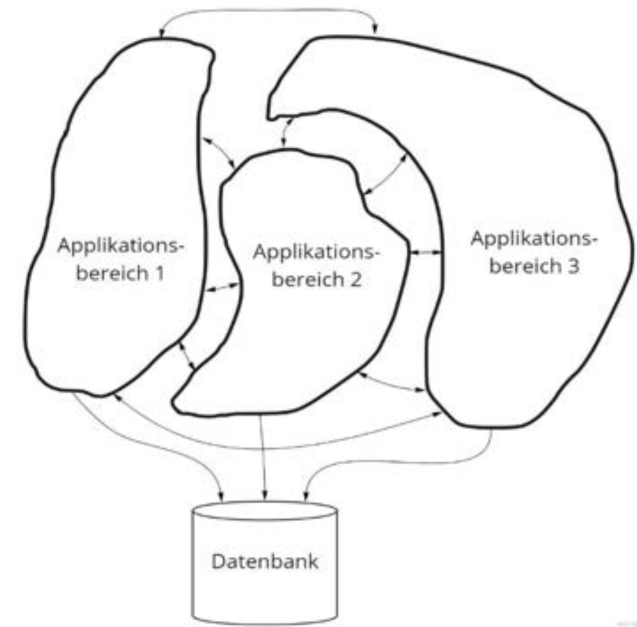
Softwarearchitektur planen

- Just Enough
  - Big Upfront Design ist schlecht
  - Gar nicht planen ist noch schlechter
  - Evolutionärer Prozess

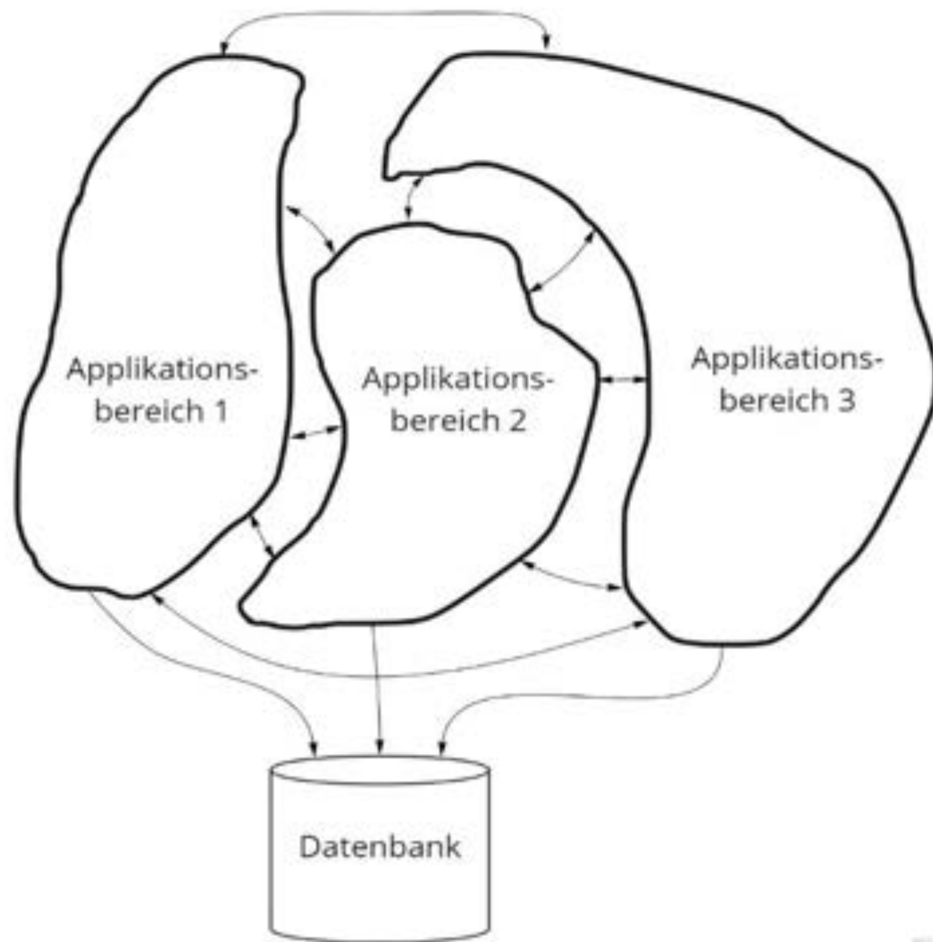
© SEQIS GmbH, 2022

Softwarearchitektur planen

- Just Enough
  - Big Upfront Design ist schlecht
  - Gar nicht planen ist noch schlechter
  - Evolutionärer Prozess

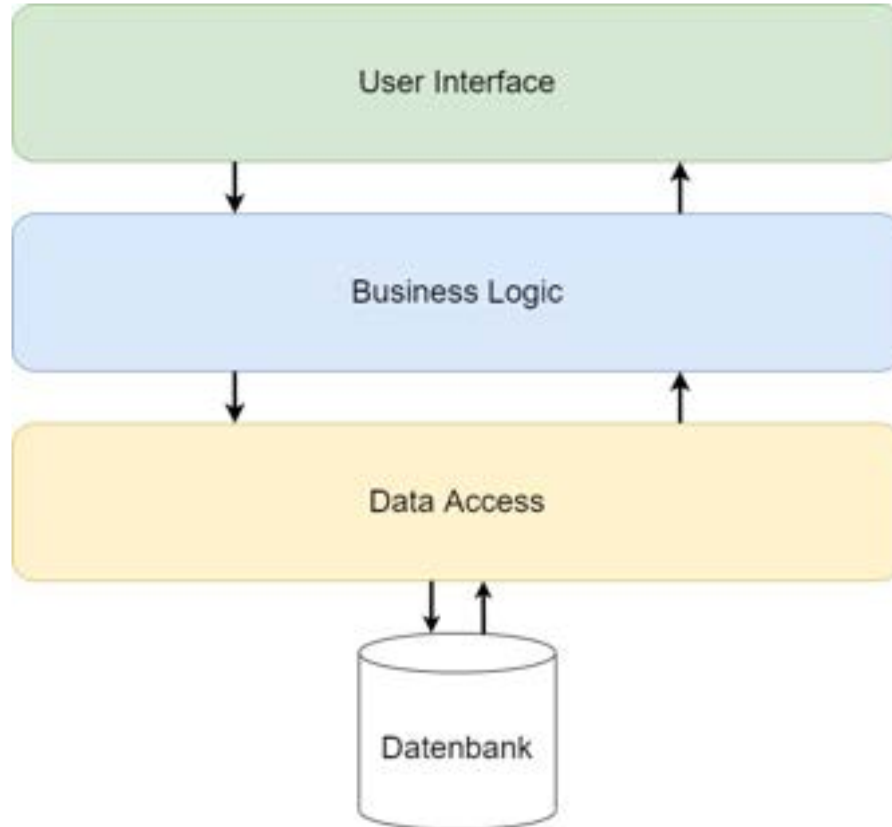


© SEQIS GmbH, 2022



© SEQIS GmbH, 2022

Schichtenarchitektur



© SEQIS GmbH, 2022

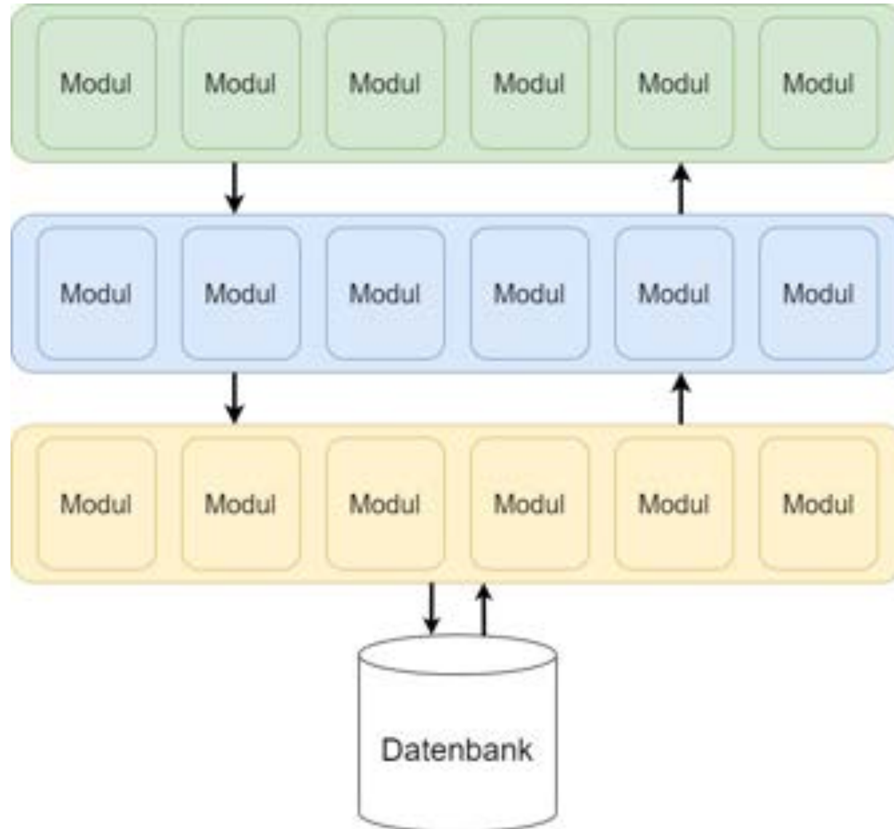
Modularität

- Ziel: Entkopplung

- Eigenschaften eines Moduls
  - Single Responsibility
  - Separation of Concerns
  - Agnostisch gegenüber anderen Modulen
  - Robuste Schnittstellen

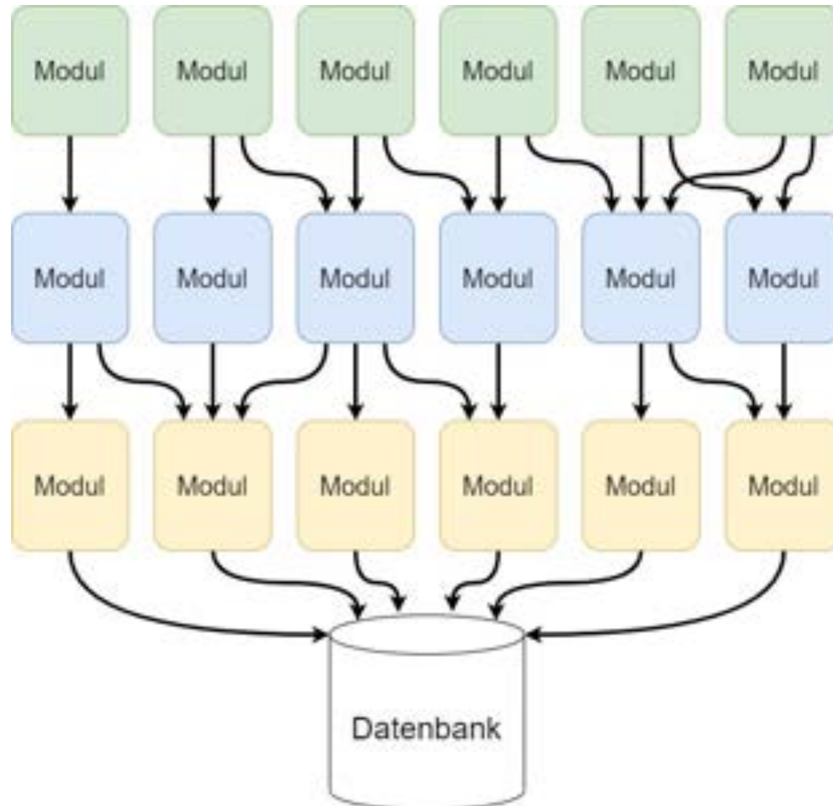
© SEQIS GmbH, 2022

Schichtenarchitektur



© SEQIS GmbH, 2022

Schichtenarchitektur



© SEQIS GmbH, 2022

Dependency Injection

```
class Foo {
  SomeService _someService;
  PageNavigation _pageNavigation;

  Foo(PageNavigation pageNavigation) {
    this._someService = new SomeService();
    this._pageNavigation = pageNavigation;
  }
}
```

© SEQIS GmbH, 2022

9. Pflegen Sie die Architektur Ihrer Software

- Machen Sie die Architekturplanung zu einem evolutionären Prozess
- Nutzen Sie Modularität

© SEQIS GmbH, 2022

Was eine gute Dokumentation beinhaltet

- Dokumentation außerhalb des Sourcecodes
  - Anforderungen
  - Coding Conventions
  - High Level Architektur

- Granularer wenn notwendig
  - Einstiegshilfe
  - Nachschlagwerk

© SEQIS GmbH, 2022

Was eine gute Dokumentation beinhaltet

- Dokumentation im Sourcecode
  - Code als Dokumentation
  - Kommentare nur wenn notwendig

© SEQIS GmbH, 2022

Kommentare als Pflaster für die Lesbarkeit

```
double updateV(double vOld, double a, double f, double max) {
    double v = vOld + a;
    v = v * f;

    if (v > max) {
        return max;
    }
    return v;
}
```

© SEQIS GmbH, 2022

Kommentare als Pflaster für die Lesbarkeit

```
// calculates the updated velocity after applying the increase in velocity caused by accelerating
// and the decrease in velocity through friction
double updateV(double vOld, double a, double f, double max) {
    double v = vOld + a; //add acceleration from engine to previous velocity
    v = v * f; //reduce velocity by passed percentage to simulate friction

    if (v > max) { //limit velocity to passed maximum
        return max;
    }
    return v;
}
```

© SEQIS GmbH, 2022

10. Sorgen Sie für gute Dokumentation

- Schreiben Sie nur Dokumentation die auch gebraucht wird
- Schreiben Sie Code Kommentare nur wenn notwendig

© SEQIS GmbH, 2022

1. Streben Sie ein qualitätsorientiertes Mindset an
2. Integrieren Sie Wartbarkeit in den Entwicklungsprozess
3. Nutzen Sie automatisierte Tests
4. Vermeiden Sie Wiederholungen
5. Bleiben Sie konsistent
6. Priorisieren Sie Lesbarkeit
7. Sorgen Sie für Übersichtlichkeit
8. Strukturieren Sie Ihren Code übersichtlich
9. Pflegen Sie die Architektur Ihrer Software
10. Sorgen Sie für gute Dokumentation

© SEQIS GmbH, 2022