Software Engineering II
Fachvortrag

spring

Sarah Kaschper, Nicole Netzbandt, Nessi Durkaya, Christoph Kozielski, Marno Schmigotzki

Das Spring Framework - Gliederung

- Frameworks
- Java-Frameworks
- Aspektorientierte Programmierung (AOP)
- Das Spring Framework
- Demo zum Spring Framework

Was ist ein Framework?

- „Rahmengerüst", Programmiergerüst in Entwicklung von Software
- Rahmen für Ordnung in Programmierarbeiten
- vor allem in objektorientierter Programmierung

*„Ein Framework ist eine **semi-vollständige Applikation**. Es stellt für Applikationen eine **wiederverwendbare, gemeinsame Struktur** zur Verfügung. Die Entwickler bauen das Framework in ihre **eigene Applikation** ein, und **erweitern** es derart, dass es ihren **spezifischen Anforderungen** entspricht."*

Artikel „Designing Reusable Classes" im Journal of Object-Oriented Programming
Ralph E. Johnson, Brian Foote

Inhalte eines Frameworks

- Grundbausteine eines Programms
- grundsätzliche Aspekte des Designs
- kein fertiges Programm, sondern Muster
- nicht für einmalige Benutzung, Grundlage für viele Umsetzungen von Programmen
- allgemein zur Wiederverwendung entwickelt und benutzt

Unterschiedliche Frameworks

Frameworks können nicht ohne Berücksichtigung darauf, dass es verschiedene Anwendungs-bereiche gibt, bestehen. Meist sind Frameworks auf einen bestimmten Anwendungsbereich beschränkt.

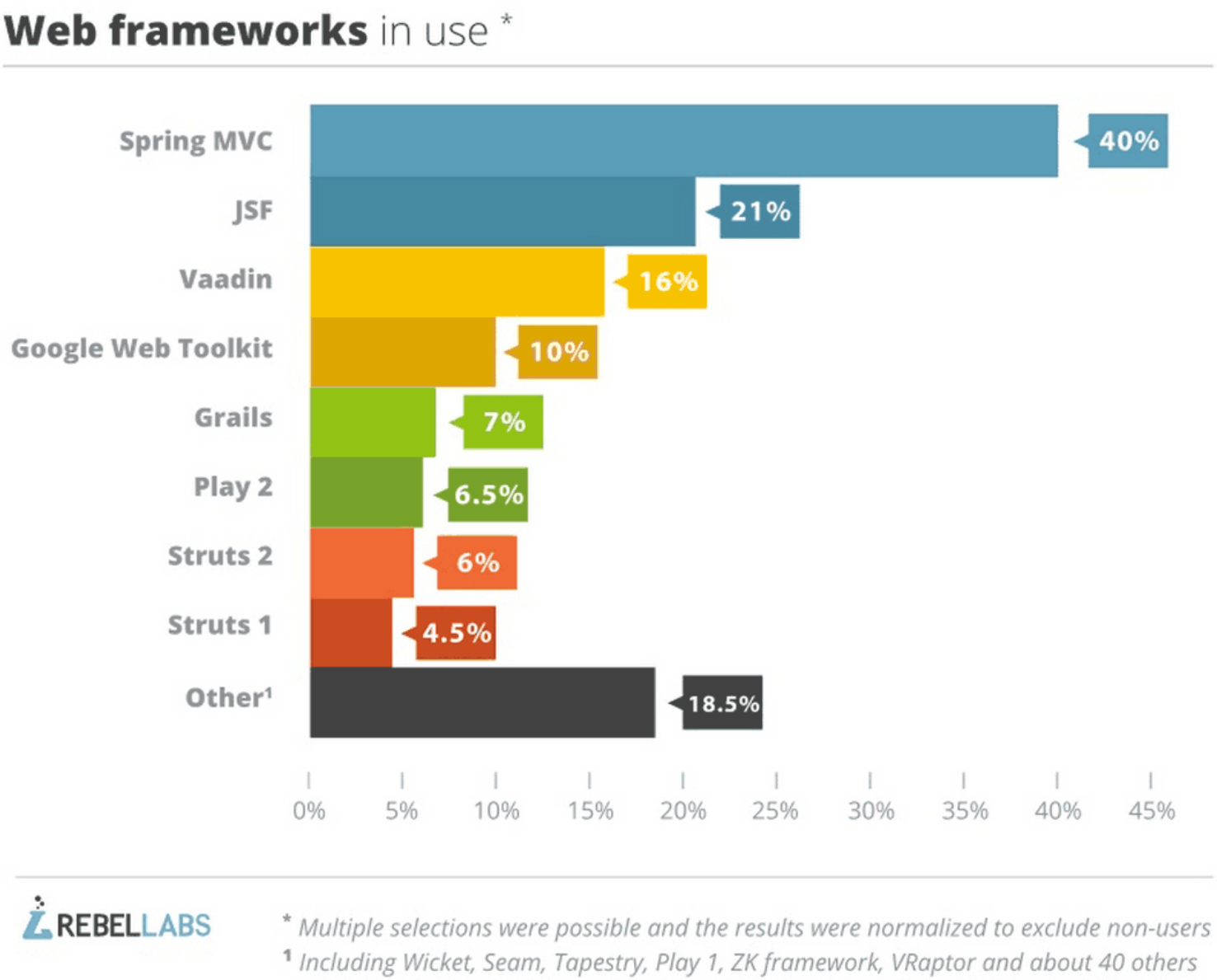
Whitebox/Blackbox

Whitebox: Die Klienten können die Implementierung einsehen und eigenen studieren
Blackbox: Im Idealfall kann der Klient keine Details hinter der Schnittstelle einsehen

Typen

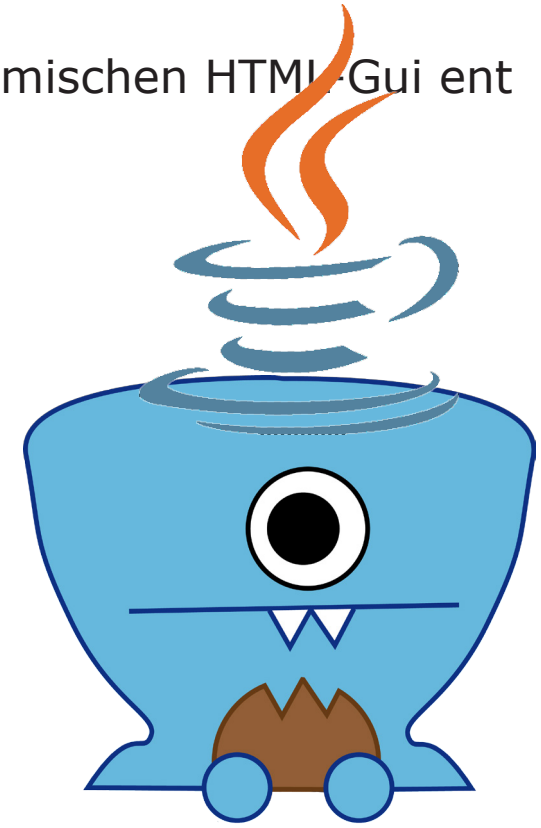
- Application Frameworks
- Domain Frameworks
- Class Frameworks
- Komponenten Frameworks
- Coordination Frameworks
- Test Frameworks
- Webframeworks

Frameworks


Web frameworks in use *
Spring MVC
JSF
Vaadin
Google Web Toolkit
Grails
Play 2
Struts 2
Struts 1
Other[1]
40%
21%
16%
10%
7%
6.5%
6%
4.5%
18.5%
0%
5%
10%
15%
20%
25%
30%
35%
40%
45%
REBELLABS
* Multiple selections were possible and the results were normalized to exclude non-users
[1] Including Wicket, Seam, Tapestry, Play 1, ZK framework, VRaptor and about 40 others

Java Web-Frameworks

- Beschleunigen die Entwicklung von Web-Anwendungen.
- Java-Applets sorgen für eine hohe Geschwindigkeit, da sie direkt in HTML eingebet tet werden können.
- Dadurch lassen sich Java-Webanwendungen mit einer dynamischen HTML-Gui ent wickeln.
- Trennt Präsentation und Logik sauber voneinander
- Hier spielt der Model-View-Controller eine Rolle
- Servlet und JPS teilen sich die Arbeit:
  - Servlet -> Controller-Eigenschaften
  - JPS -> View



Java GUI-Frameworks

- Ein gutes Beispiel sind Swing und AWT Klassen
- Unterstützen und erweitern Code und das Interface
- Sammlerklassen sind sehr groß und sehr komplex, so das sie auch schon als Frame works bezeichnet werden
- Mehr mit einer Bibliothek vergleichbar, da Methoden nur in gebrauchten Klassen aufgerufen werden.
- Nachteil: Es werden mehrere Programme benötigt um die einfachtse Gui zum laufen zu bringen.

Java Collection-Frameworks

- Stellt eine Bibliothek, mit häufig genutzten Datenstrukturen zur Verfügung.
- Collection-Framework ist an die Java Generic Library von Objectspace angelegt.

Architektur besteht aus drei wichtigen Elementen:

- Im Paket “java.util” befinden sich alle Methoden, welche man mit dem Collection-Framework benutzt. Unterschiedliche Datenstrukturen, besitzen unter schiedliche Interfaces.

- Das Interface ist die Schnittstelle für eine Datenstruktur. Speicherung der Daten, wird durch die Implementierung des Interface entschieden.

- Jedes Interface wird durch solch eine Implementierung unterstützt.
  Die Verwaltung der Daten unterscheidet die Implementierungen:
  Eine Liste kann mit einem Array realisiert werden, als auch durch eine Verkettung von Elementen.

Aspektorientierte Programmierung

Allgemeines

“Aspektorientierte Programmierung (AOP) ist ein Programmierparadigma für die objektorientierte Programmierung, um generische Funktionalitäten über mehrere Klassen hinweg zu verwenden (Cross-Cutting Concern).”
2011 kam das erste PHP-Framework auf den Markt, welcher dieses Programmierparadigma unterstütze. Es trug den Namen TYPO3 Flow.

Wozu braucht man das ganze?

Es gibt zum Beispiel bei der Programmierung einer Anwendung immer wieder Stellen an denen man bestimmte, nicht zur eigentlichen Programmlogik gehörende, wiederkehrende Funktionalitäten braucht.
Logging bietet hierfür ein gutes Beispiel, welches zum Protokollieren von Fehlern und anderen Daten benötigt wird. Dadurch, dass dieser Codesnippet immer wieder neu eingebunden wird, verschlechtert die Lesbarkeit enorm. Deshalb werden derartige Anforderungen als cross-cutting concerns bezeichnet, weil sie die Kernfunktionalität durchschneiden.

Cross-Cutting Concern

Die modulare Implementierung solcher cross-cutting concerns macht die aspektorientierte Programmierung aus und die Module werden Aspekte (aspects) genannt, welche zur Laufzeit hinzugefügt werden.

Typische Beispiele für Crosscutting Concerns sind:

- Die Erfassung von Daten.
- Funktionen zum Reporting und zur Protokollierung.
- Die Prüfung der Integrität von Transaktionen.
- Die Prüfung von Fehlern.
- Sicherheit und Authentifizierung
- Persistenz von Daten
- Funktionen zur Steigerung der Performance

Grundlage zur AOP

Gregor Kiczales entwickelte das Konzept der AOP Programmierung in der Firma Xerox.
Auch AspektJ gilt als eine der ersten Erweiterungen für Java, bis sie dann schliesslich von einer weiteren Entwicklung von Eclipse weitergeleitet wurde.
Heutzutage haben andere Programmiersprachen wie Python oder C++, auch ihre eigene AOP Programmiersprache.

Mechanismus der AOPu

Der Mechanismus, wie das die AOP ihn benutzt, ist im weiteren Sinne auch schon in Verbindung mit der subjektiven Programmierung bekannt.
In der Literatur wird im Zusammenhang mit den Komponenten-übergreifenden Aspekten auch häufig von sogenannten Crosscutting Concerns gesprochen.
In Aspektorientierten Systemen werden diese nicht separat in jede betreffende Klasse eingearbeitet sondern zunächst isoliert implementiert.
Die Verbindung zum eigentlichen, anwendungsbezogenen Programmcode erfolgt automatisiert über die programmierten Aspekte.
Man bezeichnet diesen Prozess der Manipulation von Klassen auch als Einweben (engl. weaving) der Aspekte in den Programmcode.

Ziele der AOP

Die Zielstellung der aspektorientierten Programmierung ist die klare Trennung von Komponenten und Aspekten.
Man spricht in diesem Zusammenhang auch davon, dass Aspekte Komponenten quer schneiden.

Dabei ist eine Komponente im Sinne der AOP eine Systemeigenschaft, die in einer verallgemeinerten Prozedur gekapselt werden kann.
Im Gegensatz dazu ist ein Aspekt eine Systemeigenschaft, die nicht in einer verallgemeinerten Prozedur gekapselt werden kann.

Einsatzgebiete der AOP

Einsatzgebiete:

Fehlerbehandlung
Transaktionssteuerung
Verifikation
Profiling

Logging / Tracing
Persistenz
Policy Enforcement

Sicherheitsprüfungen
Caching und Replikation
Nebenläufigkeitskontrolle

Aspektorientierte Programmierung

Allgemeines

Motivation für die AOP: Untersuchung der Grenzen der objektorientierten Entwicklung und Entwicklung von Ansätzen zur Steigerung der Produktivität durch vereinfachte Modularisierbarkeit von Code.

Zentrale Ideen:
Komponenten: Modularisierung von Common Concerns
Aspekte: Modularisierung von Crosscutting Concerns

Die Bereitstellung von Regeln durch AOP, um Aspekte und Komponenten miteinander kombinieren zu können.

Modularisierung Begriffserklärung:
Ein Prinzip, nach dem viele Systeme entwickelt werden. Ein komplexeres System wird nach dem Baukastenprinzip aus Einzelbausteinen zusammengesetzt.

Aspektorientierte Programmierung

Codebeispiele

Das Aspekt- Grundgerüst:

```
public aspect MannersAspect{

public void greet(){
    System.out.println(“Good Day”);
    }

public coid say Goodbye(){
    System.out.println(“Thank you. Goodbye!”);
    }
}
```

Die Anwendung

```
public class Talker{

public static void say(String message){
    System.out.println(message);
    }
}
```

Aspektorientierte Programmierung

Codebeispiele

```
public aspect MannersAspect{

    pointcut callsay():call(public static void Talker.say(String));

    befor():callSay(){
        greet();
    }
    after(); callSay(){
        sayGoodbye();
    }
}
```

Aspektorientierte Programmierung

Begriffserklärung

- **Target:** Zielklasse, in der die Verwendung von Aspekten benötigt wird.
- **Join Point:** Ein Punkt bei der Programmausführung an dem man eine bestimmte Methode eines Aspekts ausführen möchte. Join Points sind implizit vor jeder Methode vorhanden.
- **Advice:** Methode in einem Aspekt, die an einem oder mehreren Join Points mit in den Programmablauf eingebunden wird.
- **Pointcut:** Ein Ausdruck zur Auswahl von Join Points
- **Pointcut expression:** Schreibweise eines Pointcuts, mit Hilfe der Pointcut Expression Language (PEL).
- **Introduction:** Durch eine Introduction kann man eine Zielklasse um zusätzliche Interfaces und deren Methoden als auch Eigenschaften erweitern, ohne den Code der Zielklasse selbst anpassen zu müssen.

Einbindung von Advices

Ein Advice kann auf verschiedenste Art aufgerufen werden:

- **before:** vor der Zielmethode .
- **after:** nach der Zielmethode. Dieser Aufruf erfolgt auf jeden Fall, egal ob eine Exception geworfen wurde oder die Methode erfolgreich war.
- **after-returning:** nach der Zielmethode, wenn diese erfolgreich und ohne Exception beendet wurde. Der zurückgegebene Wert kann vom Advice gelesen, jedoch nicht verändert werden.
- **after-throwing:** nach der Zielmethode, wenn diese eine Exception geworfen hat.
- **around:** vor und nach der Zielmethode, wenn Logik vor und nach der Zielmethode ausgeführt werden muss. Hierbei kann auch der Rückgabewert verändert werden!
- **statt:** der Zielmethode, indem diese unterdrückt wird.

Es gibt also viele einfache Möglichkeiten Advices einzubauen. Es sollte jedoch darauf geachtet werden, dass immer die einfachst mögliche Einbindung erfolgt, um die Komplexität nicht unnötig zu erhöhen.

Aspektorientierte Programmierung

Codebeispiele

```
let aspectjs = require(„aspectjs“);

function logging(){
    console.log(„function start with „ + JSON.stringify(arguments));
}

let calculator = new Calculator();

aspectjs.addAdvice(logging).before(calculator, „add“);
console.log(calculator.add(15,15));
```

Aspektorientierte Programmierung

Codebeispiele

```
function logging(invocation){
    console.log(„function start with „ + JSON.stringify(arguments));
let result = invocation.proceed();
console.log(„result: „ + result);
return result;
}

let calculator = new Calculator();
aspectjs.addAdvice(logging).around(calculator, „add“);
console.log(calculator.add(15,15));
```

AOP Vorteile

- Aspekte ein- und auszubauen geht sehr schnell, solange man die entsprechende Syntax beibehält.
- Besonders bemerkbar macht es sich beim Profiling, wo Ausführungszeiten einzelner Methoden zum De bugging gemessen werden können. Stellen können über Pointcuts ganz schnell angepasst, vergröbert, oder verfeinert werden.
- Der Einsatz kann simpel auf ganze Packages ausge weitet oder bis auf einzelne Methoden in bestimm ten Klassen heruntergebrochen werden, eben so schnell, wie die Funktionalität wieder ausgebaut werden kann.

AOP Vorteile

Verantwortung: Jedes Modul ist für seinen Bereich zu ständig

Modularisierung: Bessere Modularisierung durch Vermeidung von Scattering und Tangling. (Weniger Redundanz, erhöhte Verständlichkeit, gepflegtere Software)

Einfachheit: Architektur und Entwurf konzentrieren sich mehr auf die Anwendungen und Anforderungen

Wiederverwendbarkeit: Aspekte sind von ihrem Ansatz weniger eng an eine anwendung gekoppelt

Stabilität: Crosscutting Concerns werden bei der Implementierung nicht mehr übersehen

Kosten: Einfachheit, Wiederverwendbarkeit und Stabilität, wirken sich positiv auf die Kosten aus

AOP Nachteile

- Wiederverwendbarkeit und Stabilität der Aspekte hängt stark von deren Implementierung ab (richtige Wahl der Abstraktion).
- Ein Crosscutting Concern kann Seiteneffekte auf Common Concerns haben. Die Kapselung von Modu len kann gebrochen werden. Die Semantik eines Mo duls ist nicht mehr allein dem Quellcode des Moduls zu entnehmen.
- Das Testen und Verifizieren eines Systems mit As pekten ist deutlich komplizierter. Verminderte stati sche Analysierbarkeit des Kontrollflusses

AOP Nachteile

Debugging: Kontrollfluss schwer nachvollziehbar, durch Trennung von Code- und Prozessstruktur

Kapselung: Durch die AOP wird das objektorientierte Prinzip der Kapselung gebrochen, da eine Klasse nicht mehr ihr gesamtes Verhalten unter eigener Kon trolle hat.

Kopplung: Pointcuts werden über Namen von Klassen, Me thoden, Variablen, oder Paketen definiert. Dies koppelt Komponenten und Aspekte

Disziplin: Setzt die Einhaltung von Software-Standarts vo raus, was in größeren Entwicklungsteams mit Schwierigkeiten verbunden ist.

Fazit

Um AOP anwenden zu können, muss man gut verstanden haben wie es funktioniert und was man damit erreichen möchte, da man Gefahr läuft den Überblick zu verlieren. Zudem sind Abhängigkeiten schwer zu erkennen, was das Debugging sehr erschwert. Seiteneffekte und Wechselwirkungen sind die Folge!


spring

Die Schlüsselstrategien von Spring

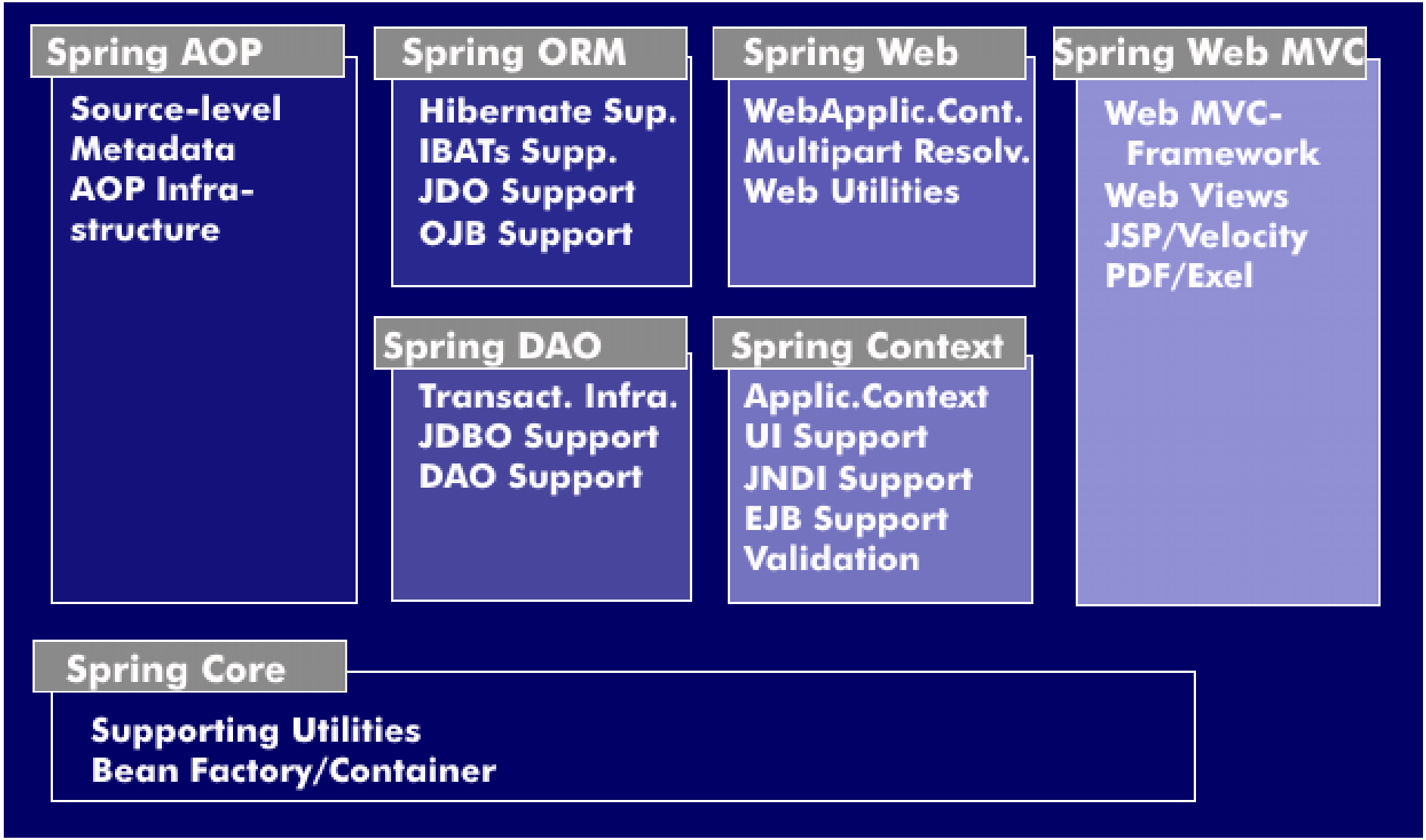
- leichtgewichtige Entwicklung mit POJOs (Plain Old Java Objects)
- lose Kopplung durch Dependency Injection (DI)
- Deklarative Programmierung durch Aspektorientierte Programmierung (AOP)
- Vermeidung von Code durch Templates

Geschichte von Spring

- erste Ideen im Jahr 2002
- erste Freigabe des Quellcodes im Februar 2003 bereitgestellt (**Version 0.9**)
- erste offizielle Version im März 2004 (**Version 1.0**)erschien
- noch im gleichen Jahr verbesserte **Version 1.1**
- 2005 **Version 1.2** mit einigen Java-5-Funktionalitäten
- Ende 2005 die Ankündigung zur **Version 2.0**, die erschien dann 2006
- **Version 3.0** Ende des Jahres 2009
- aktuelle Version ist die **Version 4.1**

Das Spring Framework

Module in Spring



Erweiterungen der Module von Spring

• Spring .NET • Spring Boot • Spring AMQP • Spring Batch • Spring BeanDoc • Spring Data • Spring Dynamic Modules • Spring Extensions • Spring IDE • Spring Integration • Spring BlazeDS Integration • Spring LDAP • Spring MVC • Spring Rich Client • Spring Roo • Spring Security • Spring Social • Spring Web Flow • Spring Web Services • ColdSpring ColdFusion • Spring for Android • Spring Cloud •

Vorteile von Spring

Modularität

- Logik wird in Form von POJOs angelegt, wodurch Kapselung und Austauschbarkeit von Implementierungen möglich sind

Produktivität

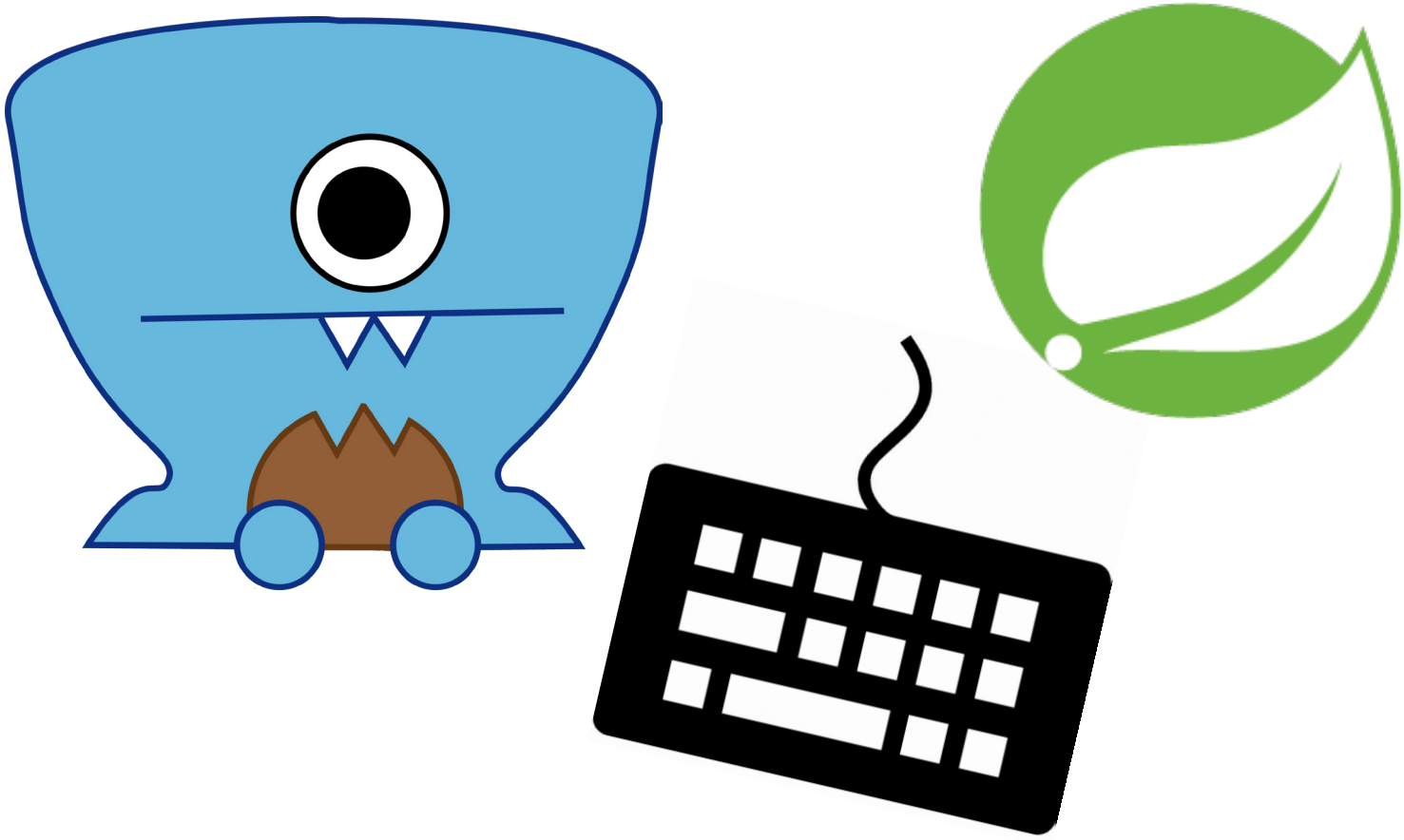
- Durch die Vereinfachung durch Spring kann der Programmierer sich auf das wesentliche konzentrieren und verringert damit den zu entwickelnden Code. Dadurch steigen Qualität und Produktivität.

Portabilität

- Spring Anwendungen brauchen keinen kompletten Application-Server, es wird nur ein Web-Container benötigt. Trotzdem läuft Spring auf alles JavaEE Servern und Cloud Plattformen
  Spring kann auch ganz ohne Container eigesetzt werden.

Testbarkeit

- Module können einzeln aber auch als Zusammenspiel getestet werden, da Abhängigkeiten sauber definiert werden können.



Spring - ein Beispiel

Quellen

http://www11.informatik.uni-erlangen.de/Lehre/WS0708/HS-AOSD/materialien/vortraege/adersberger-aop-tutorial.pdf
http://www.itwissen.info/Crosscutting-Concerns.html
https://www.heise.de/developer/artikel/Aspektorientierte-Programmierung-mit-JavaScript-3195349.html?artikelseite=2
http://www.inf-schule.de/programmierung/imperativeprogrammierung/konzepteimp/modularisierung
http://www.searchsecurity.de/definition/Framework
https://www.xovi.de/wiki/Framework
https://fastwp.de/4014/
http://uws-software-service.com/de/leistungen/java-kompetenz/softwareentwicklung-mit-dem-spring-framework.html
http://www.torsten-horn.de/techdocs/jee-spring.htm
https://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html
https://www.javatpoint.com/spring-tutorial
https://www.frank-rahn.de/spring-einem-einfachem-beispiel/
https://www.youtube.com/watch?v=OQIsKgfkYcE


Danke für die Aufmerksamkeit!
Noch Fragen?
??