

SOLID / GRASP / **STRATEGY PATTERN**

WEITER WISSEN →



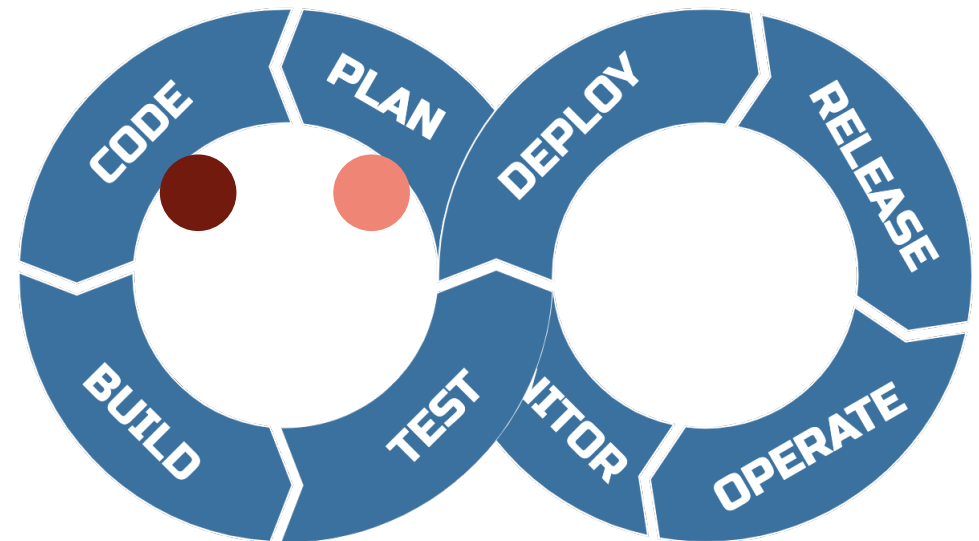
Ziel

Nach der Lektion können die Studierenden die SOLID und GRASP Prinzipien gegenüberstellen und diese kombinieren.

Zudem haben Sie ein Strategy Pattern in ihr **spring-starter** implementiert.

Strategy Pattern

- Bereits benutzt! ExMan Packer (Pack) nutzt eine Strategy unter der Haube
 - Hoffen wir zumindest
- Automatisierte Tests um alles zu testen
- Wir implementieren Test-Driven (Test zuerst) ein Strategy Pattern
 - Inkl. Action
 - Repetition Git
 - Repetition Controller/Spring



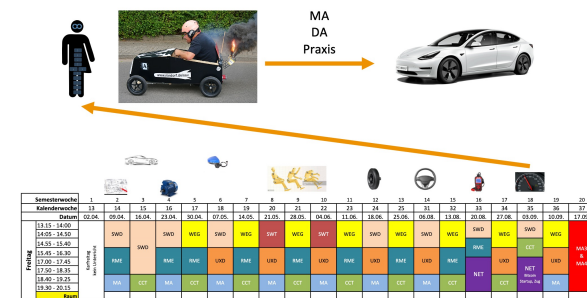
Agenda

- Ziel
- SOLID / GRASP Takeaways
- Strategy Pattern
 - Relation zu GRASP/SOLID
- Zielkontrolle

Design Patterns fördern auch Clean Code!

Agenda

- Ziel
 - SOLID / GRASP Takeaways
 - Strategy Pattern
 - Relation zu GRASP/SOLID
 - Zielkontrolle
-
- Es gibt zu viele Entwurfsmuster um alle anzuschauen
 - Wir schauen drei an, andere:





Schlüssel für Software Engineering 🐣 Novizen


- Eine Klasse / Funktion tut nur etwas und das gut
- Mehr als 1 Zeile Code kopieren → Duplikate → Verletzt ein Prinzip (sobald kopiert werden muss → Refactoring)
- Es lohnt sich (fast) immer, konkrete Klassen (Implementationen) mit Interfaces zu abstrahieren
- Generell gibt es für jede «Kernfunktionalität) Entwurfsmuster
 - wer selber was wurstet hat 90% einen Designfehler
- Mit Interfaces erweiterbar gestalten aber Funktionalität abschliessen (Open/Closed)
- Kein Spaghetti Code (hier nach da nach dort nach hier nach rüber), wenig koppeln, viel «zusammenhalten»
- Keine Monsterinterfaces
- Ist eine Klasse nicht «testbar» liegt ein Designfehler vor


Das Pattern

- Beste Seite: <https://refactoring.guru/design-patterns/strategy>
- Also, geeignet um verschiedene Implementationen einer «Funktionalität» (die etwas spezifisches tut) zu abstrahieren

 Use the Strategy pattern when you want to use different variants of an algorithm within an object and be able to switch from one algorithm to another during runtime.

 Use the Strategy when you have a lot of similar classes that only differ in the way they execute some behavior.

 Use the pattern to isolate the business logic of a class from the implementation details of algorithms that may not be as important in the context of that logic.

 Use the pattern when your class has a massive conditional operator that switches between different variants of the same algorithm.

Wie SOLIDE ist das Strategy Pattern?

- S (Single Responsibility)
 - Jede Sub-Klasse hat eine einzige Verantwortlichkeit!
- O (Open-Closed)
 - Wenn TDD benutzt wird ist das Pattern «vertretbar» Open/Closed.
- L (Liskov Substitution)
 - Die Subklassen implementieren ein Interface, sind also ersetzbar.
- I (Interface Segregation)
 - Gefördert vom Strategy Pattern aber abhängig vom Entwickler.
- D (Dependency Inversion)
 - Involvierte Algorithmen werden absichtlich vor den höheren Ebenen abstrahiert! Höhere Ebenen sind unabhängig von den Implementationen!
- Zusammenfassung: Sehr schön SOLID designed.

Strategy Pattern

Aufgrund des Feedbacks lösen wir den Task alle zusammen!

```
git clone --branch 1.0.0 https://github.com/nds-swe/spring-starter.git strategy-pattern
cd strategy-pattern
rm -rf .git (Windows: rmdir /S /Q .git)
idea .
```

Verletzung eines Bedenken

Wir verletzen aktuell die Isolation, wo?



WEITER WISSEN.

Wir begleiten Sie!