Entwurfsmuster in ABAP

Webinar, 31.5.2019, 10:00 Uhr

Domi Bigl

Senior Consultant at Cadaxo

eMail: dominik.bigl@cadaxo.com
Twitter: @DomiBiglSAP

Beratungsschwerpunkte

- Konzeption & Management von Entwicklungsprojekten
- Qualitymanagement & Performanceoptimierung
- ABAP Units
- SAP Fiori, SAP UI5

Johann Fößleitner

Senior Consultant at Cadaxo

eMail: johann.foessleitner@cadaxo.com
Twitter: @foessleitnerj

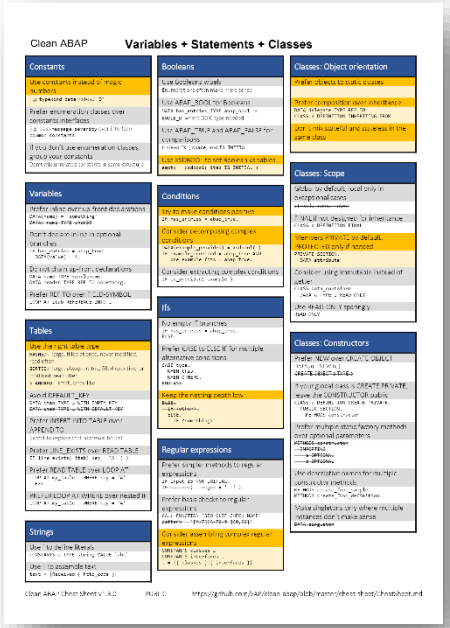
Beratungsschwerpunkte

- Konzeption & Management von Entwicklungsprojekten
- Qualitymanagement & Performanceoptimierung
- Clean Code
- S/4HANA Custom Development

- **Webinare – Rückblick / Ausblick**
  - **#ABAPsNotDead, Clean ABAP**
- **Kopplung & Kohäsion & ein paar Clean Code Principles**
- **Einführung Entwurfsmuster**
- **Entwurfsmuster Kategorien**
- **Elemente von Entwurfsmustern**
- **Beispiele**
- **Fazit / Empfehlung**

- **Rückblick / Ausblick**
  - Jänner 2016 erstes ABAP Webinar
  - Im Schnitt ca. 30 angemeldete User
    - Spitzenwert 50
  - Bisher 12 Webinare (674 Slides) mit Focus ABAP
    - ADT, Clean Code, SAP BOPF, CDS Views, ABAP Units, Modern ABAP
  - Ausblick – wir bleiben am ABAP Ball
    - ABAP RESTful Programming Model, S/4Extensions, ABAP Units, ABAP Daemons, …

https://shop.spreadshirt.de/se38



Clean ABAP Cheat Sheet

Clean ABAP The Golden Rules

https://blogs.sap.com/2019/05/03/clean-abap

cadaxo

Bevor wir mit den Mustern loslegen …

Nie mehr Spaghetti Code

Entwurfsmuster in ABAP

- **Kopplung & Kohäsion**
  - Ziel einer guten Softwarearchitektur
    - Lose Kopplung zwischen den Komponenten
    - Starke Bindung (Kohäsion) innerhalb der Komponenten
  - Die einzelnen Komponenten sind auf eine Aufgabe fokussiert

cadaxo

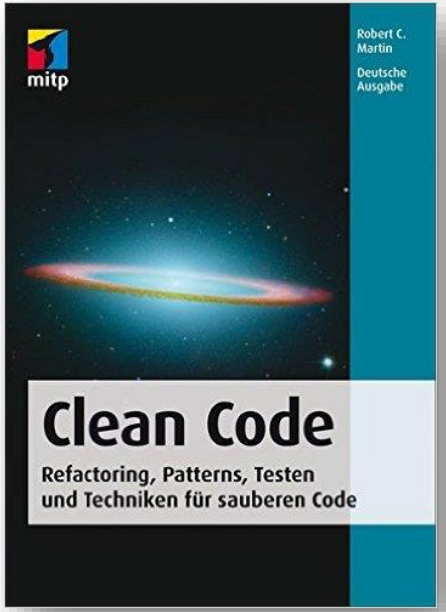
Clean Code Prinzipien

Separation of Concerns (SoC)
Single Responsibility Principle (SPR)
Open Close Principle (OCP)

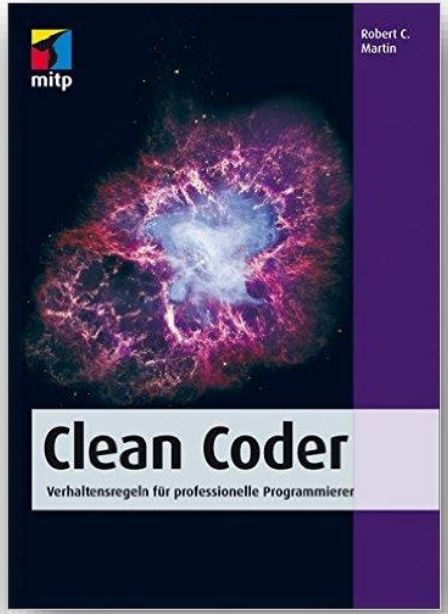
- **Separation of Concerns (SoC)**
  - Eine Codeeinheit sollte immer eine klare Aufgabe habe
  - Mehrere „Belange“ sollten nicht in einer Einheit zusammengefasst sein
    - Belange: Tracing, Logging, Caching, Transaction, …

- **Single Responsibility Principle (SRP)**
  - Eine Komponente sollte nur eine Verantwortlichkeit haben, sie sollte sich nur um eine Sache kümmern
  - Dadurch entstehen kleiner Klassen, Methoden
  - Robert C. Martin: „*A class should have one responsibility, one reason to change*“

- **Open Close Principle (OCP)**
  - Ein Softwaremodul soll offen für Erweiterungen, jedoch geschlossen für Modifikationen sein



ISBN 978-3-8266-5548-7



ISBN 978-3-8266-9695-4

https://twitter.com/unclebobmartin

Einführung Entwurfsmuster

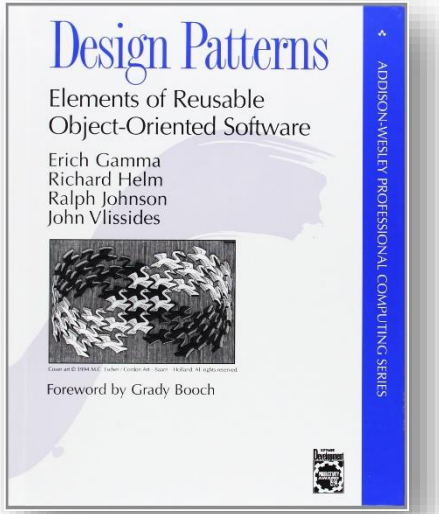
cadaxo

- **Wie alles anfing**
  - Christopher Alexander hat 1977 eine erste Sammlung von Entwurfsmustern zusammengestellt
  - Kent Beck & Ward Cunningham griffen 1987 die Ideen von Alexander auf und entwarfen neue Entwurfsmuster
  - Der Schweizer Erich Gamma verfasste 1987 seine Dissertation zum diesem Thema
  - Auf einer Konferenz[1] 1991 hat Erich Gamma einen ersten Katalog von Entwurfsmuster vorgestellt

https://twitter.com/WardCunningham
https://twitter.com/KentBeck
https://twitter.com/ErichGamma

1) OOPSLA, seit 2010 heißt die Konferenz SPLASH

- **Gang of Four, GoF**
  - 1994 von Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
  - Best Practice of Design Patterns
    - Beschreibung der 23 Pattern



https://twitter.com/ErichGamma
https://twitter.com/Grady_Booch

- **Was sind Entwurfsmuster**
  - Wiederkehrende Probleme immer auf die gleiche Art lösen
  - Lösungsbeschreibung des Problems = Muster
  - Lösung ist möglichst flexibel und wiederverwendbar
  - Es wird auf Expertenwissen zurückgegriffen, das Rad soll nicht neu erfunden werden

cadaxo

- **Eigenschaften von Entwurfsmustern**
  - Entwurfsmuster sind sprachunabhängig
  - Entwurfsmuster verfolgen einen objektorientierten Designansatz

- **Was sind Entwurfsmuster nicht**
  - Algorithmen
  - Frameworks
  - Wunderwaffen

- **Was sind eigentlich Antimuster**
  - Lösungsansätze die ungünstig oder schädlich sind
  - In der Regel entstehen Antimuster durch mangelhafte Erfahrung oder fehlender Qualifikation
    - Aufgeblähte Software
    - Gottklassen
    - Gasfabrik (unnötige Komplexität)
    - Magic Numbers
    - …

- **Warum Entwurfsmuster?**
  - Code Qualität und Code Struktur wird verbessert
  - Flexibilität bei Neuanforderungen
  - Vermeiden die Neuerfindung des Rades
  - Zeitersparnis bei der Entwicklung
  - Bessere Wartbarkeit
  - Kommunikation im Team wird erleichtert

- **Nachteile von Entwurfsmustern**
  - Entwurfsmuster sind keine Wunderwaffe, kein Garant für gutes Design
  - Höherer Entwicklungsaufwand möglich
    - Mangelndes Know-how
  - Testen wird komplizierter

Kategorien

cadaxo

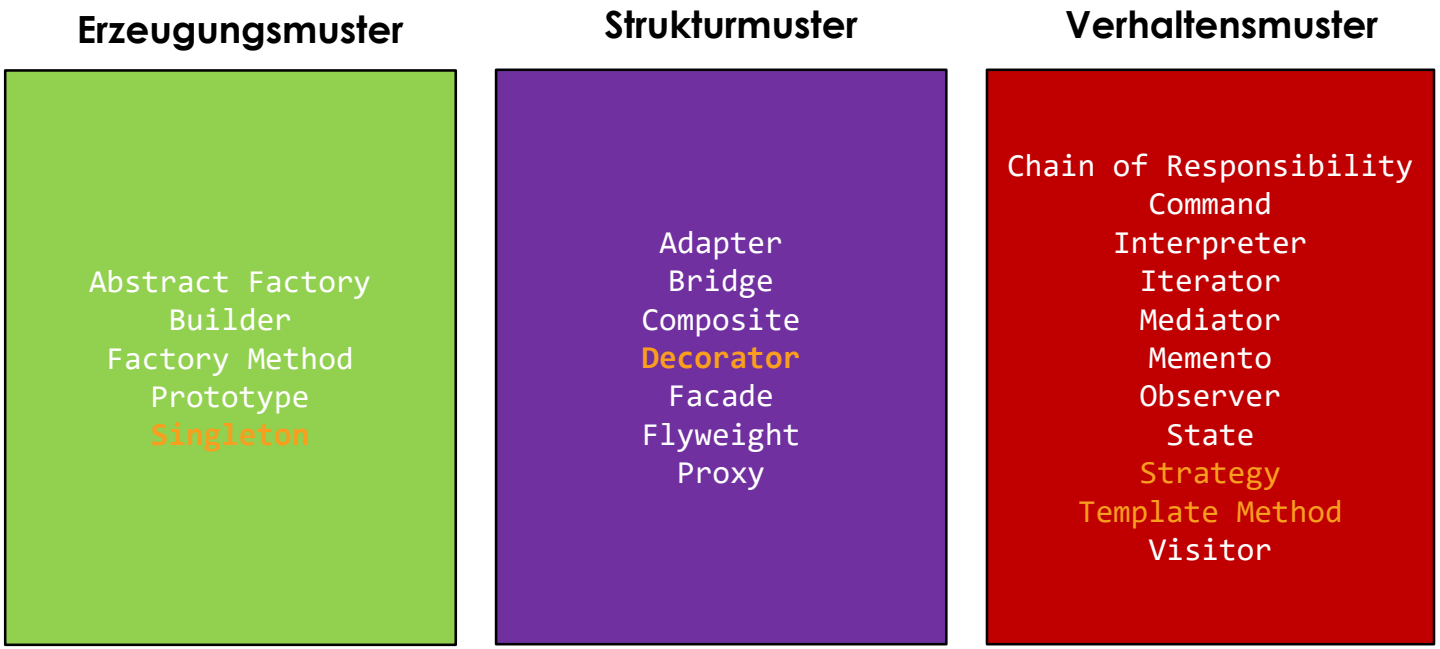
cadaxo

Elemente eines Entwurfsmusters

cadaxo

- **Elemente eines Entwurfsmusters**
  - Muster Name und Klassifizierung
  - Zweck
  - Motivation
  - Anwendbarkeit
  - Konsequenzen
  - …

cadaxo


Erzeugungsmuster
Abstract Factory
Builder
Factory Method
Prototype
Singleton
Strukturmuster
Adapter
Bridge
Composite
Decorator
Facade
Flyweight
Proxy
Verhaltensmuster
Chain of Responsibility
Command
Interpreter
Iterator
Mediator
Memento
Observer
State
Strategy
Template Method
Visitor

cadaxo

Beispiele

Singleton, Decorator, Template Method, Strategy

Singleton	Erzeugungsmuster
Zweck	Sicherstellung der Existenz einer einzigen Klasseninstanz und Bereitstellung eines globalen Zugriffspunkts für diese Instanz
Motivation	Die Klasse ist selbst für die Verwaltung der einzigen Instanz zuständig. Es wird sichergestellt, dass keine Erzeugung einer Instanz außerhalb der Klasse möglich ist. Da sich nur die Klasse um die Erzeugung und Bereitstellung der Instanz kümmert, wird eine Null-Referenz verhindert. Theoretisch ist das verhalten auch über statische Attribute realisierbar. Singleton verhindert jedoch, dass mehrere Objekte erzeugt werden.
Anwendbarkeit	• Es soll nur eine einzige Instanz einer Klasse geben • Die einzige Instanz soll per Unterklassenbildung erweiterbar sein
Konsequenzen	• Kontrollierter Zugriff auf die einzige Instanz • Singleton-Klasse kann durch Unterklassen erweitert und spezialisiert werden • Variable Anzahl von Instanzen zu einem späteren Zeitpunkt bei Bedarf möglich

Decorator (auch bekannt als Wrapper)	**Strukturmuster**
Zweck	Dekorierer bieten eine flexible Alternative zur Unterklassenbildung, um die Funktionalität einer Klasse zu erweitern.
Motivation	Mit dem Decorator Design Pattern können dynamisch Fähigkeiten zu einer Klasse hinzugefügt werden. Dazu wird die Klasse (=Component), dessen Verhalten erweitert werden soll, mit anderen Klassen (=Decorator) dekoriert. Unterstützt die Umsetzung des Clean Code Prinzip „Open-Closed-Principle“: Offen für Erweiterungen aber geschlossen für Änderungen!
Anwendbarkeit	• Objekte sollen dynamisch und transparent mit neuer Funktionalität erweitert werden • Funktionalität soll ergänzt werden, welche auch wieder einfach entfernt werden kann
Konsequenzen	• Vermeidung funktionsüberladener Klassen in den oberen Hierarchieebenen • Ein Decorator-Objekt ist nicht mit seiner Komponente identisch • Viele kleine Objekte

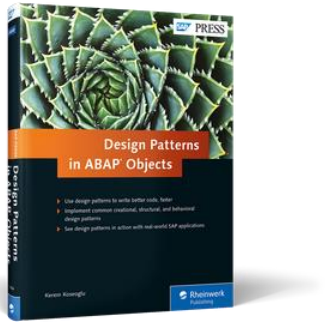
Template Method	Verhaltensmuster
Zweck	Definiert Teilschritte eines Algorithmus
Motivation	Unterklassen können bestimmte Schritte des Algorithmus überschreiben ohne die Struktur des Templates zu verändern. Unterstützt die Umsetzung des Clean Code Prinzip „Open-Closed-Princible“ und sorgt für schlanken Code.
Anwendbarkeit	• Unveränderlichen Teile werden einmalig implementiert und es wird Unterklassen überlassen, veränderliche Algorithmen zu implementieren • Erweiterungen von Unterklassen soll reguliert werden
Konsequenzen	• Template Methods stellen die Ablaufstruktur auf den Kopf: Basisklasse ruft Unterklassen auf – und nich umgekehrt. (Hollywood-Prinzip „Don’t call us, we call you. “

Strategy (auch bekannt als Policy)	**Verhaltensmuster**
Zweck	Definiert eine Familie von austauschbaren Algorithmen
Motivation	Diese Muster ermöglicht es, den Algorithmus unabhängig vom Verwender zu variieren. Folgende Fragestellungen helfen bei der Auswahl dieses Patterns: • Unterscheiden sich verwandte Klassen nur in ihrem Verhalten? • Werden unterschiedliche Varianten eines Algorithmus benötigt?
Anwendbarkeit	• Viele miteinander in Zusammenhang stehende unterscheiden sich nur in ihrem Verhalten • Verschiedene Variante eines Algorithmus sind erforderlich
Konsequenzen	• Es entstehen Familien verwandter Algorithmen • Strategy ist eine Alternative zum Erstellen von Unterklassen • Es können verschiedene Implementierungen desselben Verhaltens existieren

Empfehlung

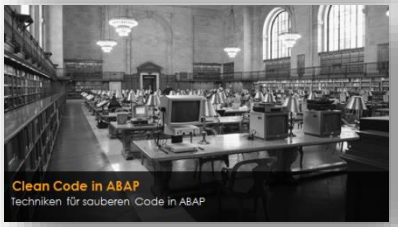
- **Wie sollte man loslegen?**
  - Verschafft Euch einen Überblick über die Muster
  - Beginnt erste Muster anzuwenden
    - Singleton, Decorator, Strategy, …
  - Erfasst und dokumentiert die verwendeten Muster

- **SAPPress Bücher**
  - Entwurfsmuster in ABAP (2015)
    - Nurgül Atilgan und Markus Straub
    - ISBN 978-3-8362-3810-6
  - Design Patterns in ABAP Objects (2017)
    - Kerem Koseoglu
    - ISBN 978-1-4932-1464-8



27. September 2019 10:00 – Thema: TBD

cadaxo



http://www.cadaxo.com/blog/

Unsere vergangenen Webinare

See you again!

Thank you for participating!

https://twitter.com/foessleitnerj
https://www.linkedin.com/in/johann-fößleitner-a9851b2a
https://www.xing.com/profile/johann_foessleitner
johann.foessleitner@cadaxo.com

https://twitter.com/domibiglsap
https://www.linkedin.com/in/dominik-bigl-9b98b68b
https://www.xing.com/profile/dominik_bigl
dominik.bigl@cadaxo.com