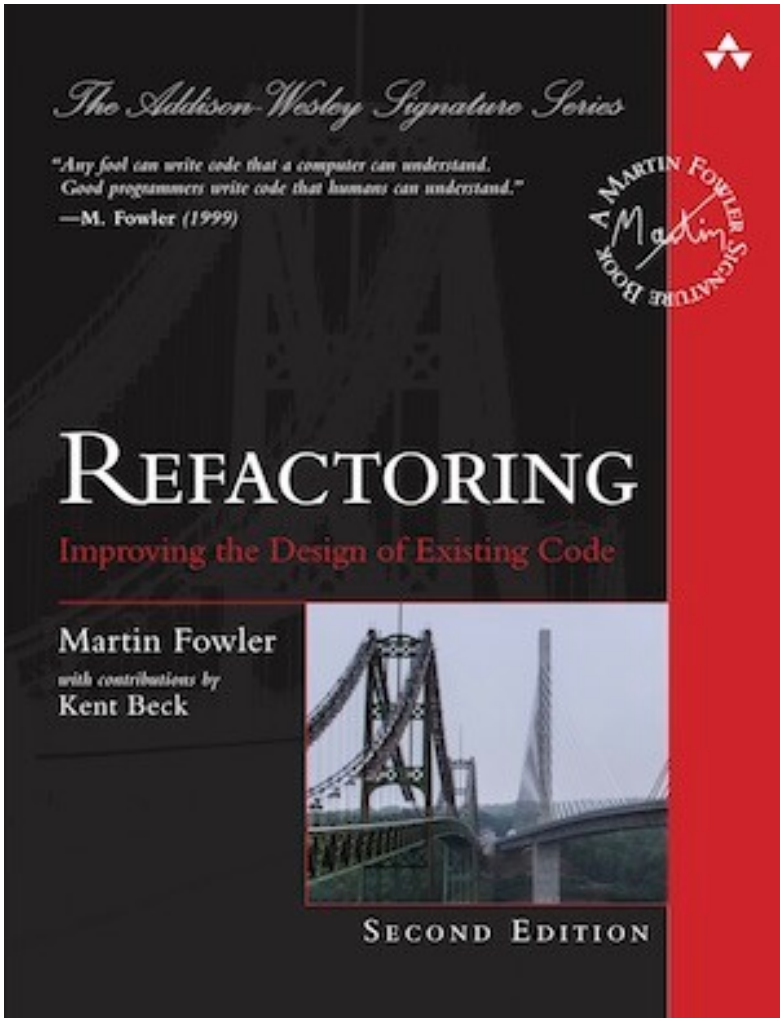
integrata
cegos

Refactoring von Java-Programmen

Literatur und Quellen

integrata cegos



Einige Hinweise

integrata cegos

- Dies ist ein praktisches Seminar
  - Übungen vertiefen die vermittelten Inhalte
  - Ergänzend Diskussion, Demonstrationen
- Die in diesem Seminar verwendete Werkzeuge und Frameworks sind Open Source
  - LPGL Lizenzmodell
- Dokumentation und Ressourcen stehen auch im Internet zur Verfügung
  - Insbesondere die API-Dokumentation
- Konventionen
  - Befehle werden in `Courier-Schriftart` dargestellt
  - Dateinamen werden in *`kursiver Courier-Schriftart`* dargestellt
  - Links werden in `unterstrichener Courier-Schriftart` dargestellt

Copyright und Impressum

Javacream

Dr. Rainer Sawitzki

Alois-Gilg-Weg 6
81373 München

eMail: training@rainer-sawitzki.de

Inhalt

integrata cegos

integrata cegos

1

EINFÜHRUNG

integrata cegos

1.1

WAS IST REFACTORING?

Was ist Refactoring?

- Refactoring ist ein Verfahren zur Verbesserung der internen Struktur und des Designs einer Anwendung
  - Dabei darf natürlich die eigentliche Funktionalität nicht beeinflusst werden.
- Refactoring scheint für viele Software-Projekte ein überflüssiger Schritt im Entwicklungsprozess zu sein
  - "Wir haben die Anwendung sauber designed!""
  - Dies ist aber nur in seltenen Fällen dauerhaft so
    - Ausnahmen von der Regel, die für kurzfristige Bug-Fixes oder schnell bereitzustellender neuer Features toleriert werden, verbleiben permanent im Quellcode
    - Neue oder auch externe Mitarbeiter verinnerlichen das Design erst einmal nicht, da grundlegende Informationen oder Gründe für bestimmte Entscheidungen nicht ausreichend dokumentiert wurden
    - Auch Design-Änderungen sind immer möglich, so dass dann der bisher entwickelte Quellcode nicht mehr passt
    - Neue Technologien oder Versionen ändern bestimmte Best Practices.

Begriffsdefinitionen: Design Pattern

integrata cegos

- Ein Design Pattern besteht aus mehreren Teilen:
  - Eine allgemeine, wieder erkennbare und damit wohldefinierte Problemstellung.
  - Einer allgemein gültigen Lösungsstrategie.
  - Klaren definierten Auswirkungen bei ihrem Einsatz.
  - Umsetzungen in konkrete Programmiersprachen.
  - Design Patterns sind in verschiedenen Katalogen organisiert. Sicherlich der bekannteste und etablierteste ist der Katalog der GoF-Pattern von Erich Gamma und seinen drei Mitautoren, der „Gang of Four“. In den letzten Jahren ist hierfür eine Reihe von Pattern für verteilte Anwendungen hinzugekommen. Als Beispiel hierfür dient der Katalog für Verteilte Anwendungen von Martin Fowler bzw. die JEE Patterns von Sun.

Begriffsdefinitionen: Idiom, Best Practice, Anti Pattern

integrata cegos

- Ein Idiom ist ein Design Pattern, das nur für eine bestimmte Programmiersprache oder Plattform sinnvoll eingesetzt werden kann.
  - Die Anwendbarkeit von Idiomen ist damit deutlich eingeschränkt. Die Abgrenzung „Idiom“ zu „Design Pattern“ ist natürlich etwas willkürlich und damit fließend.
- Eine Best Practice ist eine bestimmte etablierte Sequenz, die jedoch keine wirklich allgemeine Problemstellung aufweist.
  - Eine Best Practice ist damit auch nicht allgemein Wieder verwendbar, eventuell sogar Projektabhängig.
- Ein Anti-Pattern ist ein Beispiel für eine Umsetzung, die so gerade nicht erfolgen sollte.

Tests als Grundvoraussetzung für Refactoring

integrata cegos

- Falls die Entscheidung für einen Refactoring-Zyklus getroffen wurde ist es absolut essenziell garantieren zu können, dass die durchgeführten Änderungen keinesfalls die vorhandenen Funktionen der Anwendung negativ beeinflussen.
- Um dies zu erreichen müssen möglichst viele Teile der Anwendung durch Testfälle abgedeckt sein.
- Jeder Schritt im Refactoring wird durch einen Lauf der Testfälle kontrolliert, die natürlich erfolgreich sein müssen.

1.2

KRITERIEN FÜR SCHLECHTEN CODE

Einige Kriterien für schlechten Code

integrata cegos

- Die folgende Liste enthält einige Kriterien für schlecht codierte Anwendungen, also Anti Patterns. Die Begriffe entstammen dem Refactoring-Buch von Martin Fowler:
  - Duplicate Code:
    - Redundante Codestrecken
  - Long Method:
    - Lange Methodenrümpfe
  - Large Class:
    - Große Klassen
  - Long Parameter List:
    - Lange Parameterlisten
  - Divergent Change:
    - Eine Klasse, die unterschiedliche Rollen übernimmt. Werden zwei unterschiedliche Rollen-Vorgaben geändert muss diese Änderung in derselben Klasse umgesetzt werden.

Einige Kriterien für schlechten Code

integrata cegos

- Shotgun Surgery:
  - Verteilung von Rollen über verschiedene Klassen hinweg. Dann muss bei der Änderung einer Vorgabe eine ganze Reihe von Klassen geändert werden.
- Feature Envy:
  - Eine Methode benötigt sehr detaillierten, feingranularen Zugriff auf den Zustand einer anderen Klasse
- Data Clumps:
  - Logisch zusammengehörende Daten, die nicht in Objekten gesammelt werden.
- Primitive Obsession:
  - Insbesondere Neulinge in der Objektorientierung bevorzugen Gruppen von primitiven Datentypen statt Klassen
- Switch-Statements
- Parallel Inheritance Hierarchies:
  - Parallele Vererbungshierarchien: Das Einführen einer neuen Subklasse in einem Zweig der Hierarchie verlangt das Hinzufügen einer anderen Subklasse in einem anderen Zweig.

Einige Kriterien für schlechten Code

integrata cegos

- Lazy Class:
  - Überflüssige Klassen
- Speculative Generality:
  - Codestrecken für nicht benutzte Features. Diese Features werden vom Entwickler oder Designer als „werden irgendwann mal benötigt“ klassifiziert und deshalb vorausschauend vorbereitet und implementiert.
- Temporary Field:
  - Temporäre Attribute werden innerhalb von Klassen nur teilweise gesetzt und stehen sonst auf null.
- Message Chains:
  - Der Zugriff auf Informationen verlangt eine lange Delegation über verschiedene Objekte
- Middle Man:
  - Die Kommunikation von zwei Objekten wird unnötig verkompliziert durch die Einführung eines Vermittlers.

Einige Kriterien für schlechten Code

integrata cegos

- Inappropriate Intimacy:
  - Klassen besitzen zu viele Detailkenntnisse über den internen Zustand einer anderen Klasse.
- Alternative Classes with Different Interfaces:
  - Klassen, die nicht dieselbe Signatur aufweisen, enthalten Methoden mit gleicher Funktionalität.
- Incomplete Library Class:
  - Eine benutzbare Bibliothek ist unvollständig in ihrer Funktionalität.
- Data Class:
  - „Dumme“ Datencontainer als Anwendungsklassen sind unüblich. Dies ist nicht zu verwechseln mit dem Design Pattern Transfer Object: Diese Objekte dienen ausschließlich der Übertragung über das Netzwerk und sind deshalb keine „richtigen“ Objekte.
- Refused Bequest:
  - Subklassen erben von den Superklassen überflüssiges Verhalten.
- Comments:
  - Ausschweifende Kommentare sind notwendig, um schlechte Kodierungen zu erläutern

Ein Katalog für Refactorings

integrata cegos

- Die folgenden Kapitel beschreiben den etablierten Katalog von Refactorings von Martin Fowler aus seinem Buch „Refactoring: Improving the Design of Existing Code“. Das Buch ist Online verfügbar unter http://safari.oreilly.com.
- Die formale Beschreibung folgt dabei dem Aufbau eines Design Patterns:
  - Jedes Refactoring hat einen Namen.
  - Es folgt eine Beschreibung der Situation, in der das Refactoring eingesetzt werden wird. Die Problemstellung ist fett gesetzt. Der Lösungsansatz ist *kursiv*.
  - Es folgt ein Java-Beispiel. Die Beispiele sind allesamt in elektronischer Form jeweils vor und nach dem Refactoring verfügbar und können so effizient nachvollzogen werden.
  - Erschließt sich der Sinn des Refactorings nicht direkt folgt eine Reihe von Hinweisen.
- Hinweis:
  - Manche der beschriebenen Refactorings scheinen auf den ersten Blick trivial bzw. konstruiert nach dem Motto „wer programmiert denn so was?“.
  - Es ist hier jedoch zu beachten, dass die Beispiele natürlich aus didaktischen Gründen einfach gehalten und die Problemstellung deutlich sichtbar ist.
  - Für die Analyse eigener Anwendung sind die gezeigten Prinzipien entsprechen abstrahiert zu übertragen.

integrata cegos

2

DER KATALOG DER REFACTORING PATTERNS

integrata cegos

2.1

COMPOSING METHODS

Extract Method

- Gruppen von Code Sequenzen gehören logisch zusammen.
- *Diese Gruppen werden jeweils in eigene Methoden ausgelagert.*
- Beispiel: Printer.java
- Hinweise:
  - Dieses Refactoring ist nicht immer möglich: Falls in den ursprünglichen Sequenzen die Werte lokaler Variabler geändert werden, so ist dies bei einer einzigen Änderung noch durch einen Rückgabewert realisierbar. Bei komplexeren Änderungen ist Extract Method nicht möglich.
  - Ist kein sinnvoller, sprechender Methodenname verfügbar so sollte dieses Refactoring ebenfalls unterbleiben

Inline Method

- Die Implementierung einer Methode ist praktisch deren Name
- *Die Methode wird gelöscht und die Implementierung an die Stelle des Aufrufs gesetzt.*
  - Dies ist effektiv die Umkehrung von Extract Method
- Hinweise:
  - Ein Inlining-Zyklus kann direkt gefolgt werden von einem anschließenden Extract Method-Prozess. Dies ist sinnvoll bei einer schlecht sortierten Aufrufhierarchie, die dadurch verbessert werden wird.
  - Inlining ist bei polymorphen Methodenaufrufen selbstverständlich verboten!

Inline Temp

- Eine temporäre Variable ist überflüssig und verhindert weitere Refactorings
- *Die temporäre Variable wird entfernt.*
- Hinweise:
  - Dieses Refactoring wird meistens im Zusammenhang mit weiteren Aktionen durchgeführt.
  - Es muss garantiert sein, dass die temporäre Variable keine weiteren Zuweisungen bekommt. So kann vor dem Löschen der Variable diese als final deklariert werden.

Replace Temp with Query

integrata cegos

- Eine temporäre Variable hält das Ergebnis eines Ausdrucks.
- *Der Ausdruck wird durch eine Methode ersetzt. Bei Bedarf wird dann noch die temporäre Variable durch den Methodenaufruf ersetzt. Durch die Einführung der Methode wird die Wiederverwendbarkeit des Ausdrucks garantiert.*
- Hinweise:
  - Der Sinn dieser Aktion besteht darin, die Verwendung lokaler Variabler zu minimieren. Diese stehen ja ausschließlich innerhalb einer Methode zur Verfügung und verführen deshalb zu langen Code-Sequenzen. Nach dem Refactoring steht die Information innerhalb der gesamten Klasse zur Verfügung.
  - Wird die temporär Variable innerhalb einer Schleife benutzt so wird die gesamte Schleife in eine Methode gesteckt.
  - Die Performance der Anwendung wird durch dieses Refactoring verschlechtert, falls die lokale Variable mehrfach benutzt wird! Wahrscheinlich ist die Auswirkung auf die Performance eh gering und falls dennoch ein Problem auftritt kann ein sauber überarbeiteter Code wesentlich besser optimiert werden als vorher. Die Argumentation dieses Refactoring trotzdem durchzuführen ist sicherlich diskussionswürdig. Die Praxis beweist jedoch den unbezweifelbaren Nutzen der Maßnahme.

Introduce Explaining Variable

integrata cegos

- Ein komplexer Ausdruck ist nur schwer nachvollziehbar.
- *Der komplexe Ausdruck wird durch die Einführung sprechender temporärer Variabler vereinfacht.*
- Hinweise:
  - Durch die Einführung der Variablen ist die Notwendigkeit des Kommentars verschwunden.
  - Eine konsequente Weiterführung des Refactorings führt dann ein Extract Method durch. Die Einführung der temporären Variablen ist damit nur ein Zwischenschritt.
  - Eine Ausnahme dieser Regel ist dann gegeben, wenn der Aufwand der Methodenextraktion zu groß ist.

Split Temporary Variable

integrata cegos

- Einer temporären Variable wird mehrfach ein Wert zugewiesen.
- *Jeder Zuweisung wird eine eigene temporäre Variable zugewiesen.*
- Hinweise:
  - Eine mehrfache Zuweisung an eine Variable ist Hinweis darauf, dass die Variable nacheinander mehrere Rollen ausführt.
  - Dieses Splitting der Variablen ist eventuell Voraussetzung für ein Extract Method.

Remove Assignments to Parameters

integrata cegos

- Innerhalb einer Methode wird einem Parameter ein Wert zugewiesen.
- *Statt des Parameters ist eine temporäre Variable zu benutzen.*
- Hinweise:
  - Dieses Refactoring beseitigt die auftretende Verwirrung, wenn die Parameter-Variable im Laufe der Methodenausführung plötzlich nicht mehr den Parameter sondern etwas anderes referenziert.
  - Dieses Reactoring bedeutet nicht, dass eine Änderung im Zustand des Parameters unzulässig ist. Die Call-by-Reference-Semantik der Programmiersprache Java lässt dies als Konstruktion ja explizit zu.

Replace Method with Method Object

integrata cegos

- Eine lange Methoden-Implementierung benutzt temporäre Variablen in einer Form, die Extract Method nicht anwendbar macht.
- *Die komplette Methode wird in ein Objekt umgewandelt. Die lokalen Variablen werden zu Attributen.*
- Das Refactoring erzeugt heraus eine eigene Klasse. Diese besitzt alle Methodenparameter sowie die ursprünglichen temporären Variablen als Attribute. Weiterhin hält sie eine Referenz auf die ursprüngliche Klasse, die für die Aufrufe von Methoden benutzt werden kann.
- Hinweise:
  - Die CheckCondition-Klasse kann nun natürlich weiteren Refactorings unterzogen werden. Diese sind durch die Method Class einfach möglich.
  - Dieses Refactoring verlangt, dass die Method Class Zugriff auf die Methoden der ursprünglichen Klasse erhält. Dies kann die Verwendung einer inneren Klasse erfordern.

Substitute Algorithm

integrata cegos

- Ein umständlicher Algorithmus soll durch einen anderen ersetzt werden.
- *Die Implementierung einer Methode wird ausgetauscht.*
- Hinweis:
  - Hier ist die Einführung eines kompletten Unit-Tests natürlich besonders wichtig!

integrata cegos

2.2

MOVING FEATURES BETWEEN OBJECTS

Move Method

- Eine Methode wird häufiger von einer anderen Klasse benutzt als die deklarierende Klasse.
- *Die Methode wird in derselben oder einer ähnlichen Signatur in der anderen Klasse implementiert. Die ursprüngliche Methode delegiert an diese neue Methode weiter oder wird komplett gelöscht.*
- Hinweise:
  - Das Refactoring koppelt die beiden Klassen relativ stark miteinander, da nun auch der AccountType über den Parameter eine Abhängigkeit zum Account aufweist.
  - Die Account-Klasse benötigt Zugriffsmethoden auf ihren internen Zustand.

Move Field

- Ein Attribut wird von einer anderen Klasse häufiger benutzt als von der deklarierenden Klasse.
- *Das Attribut wird der anderen Klasse zugeordnet.*

Extract Class

- Eine Klasse übernimmt zwei Aufgaben gleichzeitig.
- *Eine neu zu erzeugende Klasse enthält die Attribute und Methoden, die der zweiten Aufgabe zuzuordnen sind.*
- Hinweise:
  - In diesem Beispiel erkennt der Verwender der Person-Klasse das Refactoring. Alternativ hätte Person die ursprünglichen Methoden behalten können. Diese delegieren dann an die neu geschriebene Klasse weiter.
  - Dieses Refactoring kann bei der Verwendung von Persistenz-Frameworks wie Hibernate oder bei geforderten Sperrmechanismen erheblichen weiteren Aufwand nach sich ziehen.

Inline Class

- Eine Klasse enthält sehr wenig Funktionalität.
- *Die Funktionalität wird in eine andere Klasse verschoben. Die ursprüngliche Klasse wird gelöscht.*
- Hinweis:
  - Dieses Refactoring ist die direkte Umkehrung von Extract Class. Das Beispiel des vorhergehenden Kapitels kann einfach in umgekehrter Reihenfolge benutzt werden.

Hide Delegate

integrata cegos

- Ein Client benutzt zwei Klassen, wobei die erste auch an die zweite delegiert.
- *Die erste Klasse wird um eine Methode erweitert, die vom Client angesprochen wird und die Delegation enthält.*

Remove Middle Man

integrata cegos

- Eine Klasse steht zwischen einem Client und einer anderen Klasse und implementiert ausschließlich Delegationslogik.
- *Der Client kann das Delegationsobjekt direkt benutzen.*
- Dieses Refactoring ist die Umkehrung des Hide Delegate-Verfahrens. Als Beispiel kann deshalb das des vorhergehenden Abschnitts umgekehrt benutzt werden.

Introduce Foreign Method

integrata cegos

- Eine Klasse benötigt eine weitere Methode. Die Klasse selber darf jedoch nicht verändert werden.
- *Die aufrufende Klasse bekommt eine weitere Methode, die als ersten Parameter eine Instanz der nicht-veränderbaren Klasse bekommt.*
- Hinweise:
  - Dieses Verfahren ist selbstverständlich ein Workaround. nach Möglichkeit sollte diese Funktionalität der Klasse zugeordnet werden, die diese auch wirklich bereits stellen soll.
  - Foreign Methods können selbstverständlich auch in allgemein zugänglichen Klassen gesammelt werden. Das sind dann typische Utility-Bibliotheken. Ein schönes Beispiel für Foreign Methods sind die Klassen der Apache Commons Lang-Bibliotheken, die größtenteils nichts anderes sind als Foreign Methods für die Klassen aus java.lang.

integrata cegos

2.3

ORGANIZING DATA

Organizing Data

integrata cegos

- Self Encapsulate Field
  - Auf ein Attribut wird direkt zugegriffen. Dies kann auch innerhalb der deklarierenden Klasse zu Nebeneffekten führen.
  - *Der Zugriff auf die Attribute erfolgt auch innerhalb der Klasse ausschließlich über getter- und setter-Methoden.*
- Replace Data Value with Object
  - Daten benötigen zusätzliche Informationen oder Verhalten.
  - *Der Typ der Daten wird in einer eigenen Klasse definiert.*
- Change Value to Reference
  - Von einer Klasse sind viele Instanzen vorhanden. Diese sollen durch eine einzige Instanz ersetzt werden.
  - *Das Objekt wird in ein Referenz-Objekt umgewandelt.*
- Change Reference to Value
  - Ein Referenzobjekt ist unveränderlich und schwierig zu verwenden.
  - *Das Referenzobjekt wird in ein Wertobjekt umgewandelt.*

Organizing Data

integrata cegos

- Replace Array with Object
  - Ein Array enthält Daten unterschiedlicher Bedeutung.
  - Das Array wird durch eine Klasse ersetzt. Die Elemente des Arrays werden zu Attributen.
- Duplicate Observed Data
  - Eine GUI-Element enthält Daten, die auch in der Domäne benötigt werden.
  - *Die Daten werden kopiert und über einen Observer- bzw. Benachrichtigungsmechanismus synchronisiert.*
  - Hinweise:
    - Diese Benachrichtigung muss prinzipiell in beiden Richtungen durchgeführt werden.
    - Moderne Java UI-Frameworks wie Swing, JFace oder JavaServer Faces enthalten diese Mechanismen bereits integriert.
- Change Unidirectional Association to Bidirectional
  - Von zwei Klassen, die sich gegenseitig benötigen, referenziert nur eine die andere.
  - *Es werden Referenzen auch in die Gegenrichtung eingeführt. Collections werden auf beiden Seiten aktualisiert.*
- Change Bidirectional Association to Unidirectional
  - Eine Seite einer bidirektionalen Verbindung wird nicht benötigt.
  - *Die unnötige Assoziation wird entfernt.*

Organizing Data

- Replace Magic Number with Symbolic Constant
  - Eine Zahl bekommt eine symbolische Bedeutung.
  - *Erzeuge daraus eine Konstante mit sprechender Benennung.*
- Encapsulate Field
  - Es existiert ein public Attribut.
  - *Die Sichtbarkeit wird auf private eingeschränkt und es werden getter/setter eingeführt.*
- Encapsulate Collection
  - Eine Methode hat als Rückgabetyp eine Collection. Eine Modifikation der Collection außerhalb Kontrolle der Klasse ist nicht zulässig.
  - *Die Methode liefert eine read-only-Collection. Die Klasse selber bekommt add/remove-Methoden für die Elemente der Collection.*
- Replace Record with Data Class
  - Ein Java-Programm benötigt Zugriff auf eine Datenstruktur.
  - *Erzeuge ein triviales Datenobjekt.*
  - Hinweise:
    - Ein triviales Datenobjekt besteht nur aus privaten Attributen mit setter-/getter-Methoden.
    - Durch weiteres Refactoring kann für diese Klasse auch Logik identifiziert werden.

Organizing Data

integrata cegos

- Replace Type Code with Class/Enumeration
  - Eine Klasse benutzt Konstanten für die Definition eines Zustands. Diese werden jedoch für keine Berechnungen benutzt.
  - *Diese Zahlen werden durch eine Enumeration ersetzt.*
- Replace Type Code with Subclasses
  - Ein unveränderlicher Attributwert beeinflusst das Verhalten einer Klasse.
  - *Der unveränderliche Attributwert wird durch eine Klassenhierarchie ersetzt.*
- Replace Type Code with State/Strategy
  - Das Verhalten einer Klasse hängt von einem internen Attribut ab. Die Bildung einer Subklassen-Hierarchie ist nicht möglich bzw. unerwünscht.
  - *Die Klasse führt das Design Pattern „State“ ein.*
- Replace Subclass with Fields
  - Subklassen unterscheiden sich nur dadurch voneinander, dass Methoden Subklassen-typische Konstanten zurückgeben.
  - *Die Klassenhierarchie wird zu Gunsten eines Attributs aufgegeben.*

integrata cegos

2.4

SIMPLIFYING CONDITIONAL EXPRESSIONS

Simplifying Conditional Expressions

integrata cegos

- Decompose Conditional
  - Es ist eine komplizierte Abfrage vorhanden.
  - *Die Abfrage wird durch die Einführung von Methoden lesbarer gemacht.*
- Consolidate Conditional Expression
  - Pfade einer Abfrage liefern dasselbe Ergebnis.
  - *Die Pfade mit gleich lautenden Ergebnissen werden logisch zusammengefasst und der Abfrageausdruck in eine Methode ausgelagert.*
- Consolidate Duplicate Conditional Fragments
  - Redundante Codestrecken sind Bestandteil verschiedener Abfrage-Zweige.
  - *Diese Codestrecken werden aus der Abfrage entfernt.*
- Remove Control Flag
  - Eine Variable wird als steuerndes Flag innerhalb einer Abfrage benutzt.
  - *Die Variable wird durch ein break oder return ersetzt.*
- Replace Nested Conditional with Guard Clauses
  - Innerhalb einer Methode steht eine Abfrage, die den Pfad der Ausführung nicht einfach wiedergibt.
  - *Alle Spezialfälle werden durch simple Abfragen mit return ersetzt.*

Simplifying Conditional Expressions

integrata cegos

- Replace Conditional with Polymorphism
  - Eine Abfrage bezieht sich auf den Typ eines Objektes und veranlasst unterschiedliche Sequenzen.
  - *Die Sequenzen werden als polymorphe Methoden den einzelnen Objekten zugeordnet.*
  - Beispiel: Employee.java
- Introduce Null Object
  - Eine Abfragen fragt die null-Referenz ab um Aktionen auszuführen, die eigentlich dem Objekt zugeordnet sein sollten.
  - *Der null-Wert wird durch ein Null-Objekt ersetzt.*
  - Beispiel: Customer.java und BillingPlan.java
- Introduce Assertion
  - Ein Teil des Programms macht bestimmte Annahmen über den Zustand der Anwendung.
  - *Die Annahmen werden über Assertions geprüft.*

integrata cegos

2.5

MAKING METHOD CALLS SIMPLER

Making Method Calls Simpler

integrata cegos

- Rename Method
  - Eine Methode hat keinen sprechenden Namen.
  - *Die Methode wird sinnvoll umbenannt.*
- Add Parameter
  - Eine Methode braucht zusätzliche vom Aufrufer weitere Informationen.
  - *Der Methode wird ein Parameter hinzugefügt.*
- Remove Parameter
  - Eine Methode bekommt überflüssige Informationen.
  - *Der überflüssige Methodenparameter wird entfernt.*
- Separate Query from Modifier
  - Eine Methode hat einen Rückgabewert, ändert aber gleichzeitig noch den Zustand der Anwendung.
  - *Die Methode wird in zwei Methoden aufgespaltet, eine ändert den Zustand, die andere liefert die Rückgabe.*

Making Method Calls Simpler

integrata cegos

- Parameterize Method
  - Verschiedene Methoden enthalten gleiche Sequenzen mit unterschiedlichen Werten.
  - *Eine neue Methode bekommt die verschiedenen Werte als Parameter.*
  - Beispiel: Employee.java
- Replace Parameter with Explicit Methods
  - Eine Methode führt in Abhängigkeit von einem enumerierten Parameter verschiedene Sequenzen aus.
  - *Für jeden enumerierten Wert wird eine eigene Methode eingeführt.*
- Preserve Whole Object
  - Mehrere Attribute eines Objektes dienen als Parameter für einen Methodenaufruf.
  - *Das Objekt selbst wird als Parameter gesendet.*
  - Beispiel: Point.java
- Replace Parameter with Method
  - Ein Objekt benutzt den Rückgabewert einer Methode nur als Parameter für einen weiteren Methodenaufruf.
  - *Die zweite Methode ruft die erste Methode selbst auf.*

Making Method Calls Simpler

integrata cegos

- Introduce Parameter Object
  - Eine Gruppe von Parametern gehört logisch zusammen.
  - *Eine neue Klasse fasst diese Parameter als Attribute zusammen und wird als neuer Parametertyp benutzt.*
- Remove Setting Method
  - Ein Attribut ist unveränderlich.
  - *Alle setter-Methoden werden entfernt.*
- Hide Method
  - Eine Methode wird nicht von anderen Klassen benutzt.
  - *Die Methode wird als private deklariert.*
- Replace Constructor with Factory Method
  - Die Objekterzeugung verlangt mehr Aktionen als sinnvoll im Konstruktor untergebracht werden kann.
  - *Der öffentliche Konstruktor wird durch eine Factory-Methode ersetzt.*
- Encapsulate Downcast
  - Ein Objekt liefert ein Ergebnis, das vom Aufrufer spezialisiert werden muss.
  - *Der notwendige Cast ist Bestandteil der Methode.*

Making Method Calls Simpler

integrata cegos

- Replace Error Code with Exception
  - Eine Methode signalisiert einen Fehler durch einen speziellen Rückgabewert.
  - *Der Fehler wird durch eine Exception signalisiert.*
- Replace Exception with Test
  - Eine Exception wird auf Grund einer Situation geworfen, die der Aufrufer hätte prüfen können.
  - *Der Aufrufer prüft selbst.*

integrata cegos

2.6

DEALING WITH GENERALIZATION

Dealing With Generalization

- Pull Up Field
  - Zwei Subklassen deklarieren das selbe Attribut
  - *Das Attribut wird in die Superklasse aufgenommen.*
- Pull Up Method
  - Zwei Subklassen deklarieren die selbe Methode in gleicher Funktionalität
  - *Die Methode wird in die Superklasse aufgenommen.*
- Pull Up Constructor Body
  - Konstruktoren von Subklassen enthalten identische Sequenzen
  - *Diese gemeinsame Logik wird in einen Konstruktor der Superklasse verlagert, der von den Subklassen-Konstruktoren aufgerufen wird.*

Dealing With Generalization

integrata cegos

- Push Down Method
  - Eine Superklasse enthält eine Methode, die nur für eine Subklasse benötigt wird.
  - *Diese Methode wird in die Subklasse verlagert.*
- Push Down Field
  - Eine Superklasse enthält ein Attribut, das nur für eine Subklasse benötigt wird.
  - *Das Attribut wird in die Subklasse verlagert.*

Dealing With Generalization

integrata cegos

- Extract Subclass
  - Eine Klasse bietet Funktionalitäten, die nur in manchen Instanzen benötigt werden.
  - *Für die Submenge dieser Eigenschaften wird eine eigene Subklasse eingeführt.*
  - Beispiel: Person.java und Worker.java
- Extract Superclass
  - Zwei Klassen mit ähnlichem Verhalten sind vorhanden.
  - *Das gemeinsame Verhalten wird in einer Superklasse gebündelt.*
  - Beispiel: Employee.java und Department.java
- Extract Interface
  - Verschiedene Clients nutzen nur ein Subset von Methoden einer Klasse bzw. zwei verschiedene Klassen besitzen eine gemeinsame Subsignatur.
  - *Das Subset wird in einer Schnittstelle definiert, die von diesen Klassen implementiert wird.*
  - Beispiel: Hotel.java und Person.java
- Collapse Hierarchy
  - Eine Subklasse unterscheidet sich kaum von seiner Superklasse.
  - *Die beiden Klassen werden vereint.*

Dealing With Generalization

integrata cegos

- Form Template Method
  - Zwei Subklassen implementieren unterschiedliche Sequenzen, die jedoch in gemeinsamer Reihenfolge ausgeführt werden.
  - *Die gleichlautenden Sequenzen werden in eine gemeinsame Methode innerhalb der Superklasse zusammengeführt.*
- Replace Inheritance with Delegation
  - Eine Klasse benötigt nur einen Teil der Superklassen-Funktionalität oder kann/will nicht von der Superklasse erben.
  - *Die Klasse hält eine Referenz auf die andere Klasse und delegiert an die gewünschten Methoden weiter.*
  - Beispiel Stack.java
- Replace Delegation with Inheritance
  - Eine Klasse enthält viele simple Delegationsaufrufe.
  - *Aus der delegierenden Klasse wird eine Subklasse.*
  - Beispiel: Employee.java

integrata cegos

3

TEST UND QUALITÄTSSICHERUNG

3.1

ÜBERSICHT

Überblick

integrata cegos

- Checkstyle (http://sourceforge.net/projects/checkstyle) ist ein Entwicklungswerkzeug, das die Einhaltung frei definierbarer Coding-Konventionen automatisiert.
  - Als ein Beispiel bzw. Ausgangspunkt für solche Code-Konventionen können die "Sun Code Conventions" (http://java.sun.com/docs/codeconv/) dienen.
- Die zu prüfenden Richtlinien werden dabei aus einem relativ einfach erweiterbaren Satz von mitgelieferten Check-Klassen komponiert.
  - Dies erfolgt in eine XML-Konfigurationsdatei, in der die einzelnen Prüfungen in Modulen zusammengefasst werden.
  - Die Ergebnisse werden in der Regel in einer Datei abgelegt, deren Format entweder eine einfache Liste oder eine XML-Dokument ist.

Mitgelieferte Standard-Checks

integrata cegos

- Die in der Standard-Distribution enthaltenen Checks decken ein weites Spektrum ab.
  - So gibt es eine ganze Reihe von Prüfungen, die die formale Korrektheit des Quellcodes (Einrückungen, Leerzeichen, Platzierung der öffnenden und schließenden geschweiften Klammern, Benennungen etc.) prüfen.
- Es sind aber auch Programmiertechnisch höherwertige Routinen enthalten:
  - Sollen Klassen, Methoden und Parameter als final deklariert werden?
  - Sind javadoc-Kommentare enthalten und passen diese zu den Methodensignaturen?
  - Sind komplexere Literale in Konstanten deklariert?
  - Erkennen von redundanten (duplizierten) Codestrecken.
- Eine vollständige Liste ist Bestandteil der mitgelieferten Dokumentation.

Aufruf

- Checkstyle benötigt für seine Arbeit notwendigerweise Optionen:
  - Lokation der Konfigurationsdatei, in der die Sammlung der Prüfroutinen steht.
  - Konfiguration der Module mit deren Properties. Jedes Modul hat einen Satz von Standard-Werten, die durch das Setzen einer Systemvariable oder Angabe einer Properties-Datei überschrieben werden können.
  - Eine zu prüfenden Quellcode-Datei oder ein ganzes Verzeichnis.
  - Angabe des Ausgabeformats.
  - Angabe der Ausgabedatei.
- Weiterhin ist möglich das so genannte "package name file".
  - Diese Datei enthält eine Liste von Paketnamen, die bei der Auswertung der Konfigurationsdatei benutzt wird.
  - Ziel der Aktion ist es, die Prüfklasse in der Konfigurationsdatei unqualifiziert ansprechen zu können, in dem beim Auflösen des Klassennamens die Pakete des "package name file" vorangestellt werden.

Weitere Informationen

- Aufruf
  - Standalone
  - ANT-Task
- Detaillierte Arbeitsweise:
  - Benutzt die ANTLR-Bibliothek
  - Näheres im Begleitbuch

3.2

WERKZEUGE

JDepend

- JDepend analysiert eine Klassenbibliothek und erstellt dabei Qualitäts-Metriken für jedes Java-Paket. JDepend wird in den normalen Build-Prozess integriert und misst die Qualität der Anwendung bezüglich Wartbarkeit, Wiederverwendbarkeit und Robustheit. Folgende Metriken werden pro Paket erstellt:
  - Anzahl der Schnittstellen und der Klassen. Die Menge an abstrakter Klassen und Schnittstellen ist ein Maß für die Erweiterbarkeit des Pakets.
  - Aufsteigende Kopplungen („Afferent coupling“, Ca): Wie viele andere Pakete benutzen Klassen dieses Pakets. Dies ist ein Indikator für das Maß an Verantwortung, die dieses Paket übernimmt.
  - Absteigende Kopplung („Efferent coupling“, Ce): Wie viele andere Pakete werden von diesem Paket benutzt? Ein Maß für die Unabhängigkeit dieses Pakets.
  - Abstraktheit (A): Das Verhältnis von abstrakten Klassen + Schnittstellen zu konkreten Klassen.
  - Instabilität (I): Das Verhältnis Ce/(Ce + Ca). Ein Maß für die Stabilität des Pakets bei Änderungen. I = 0 bedeutet komplett stabil, 1 komplett instabil.
  - „Entfernung“ (Distance D)von der Ideallinie, die sich aus der Idealisierung A + I = 1 ergibt. Wie ausgewogen ist das Paket bezüglich Abstraktheit und Stabilität? Ideal wäre entweder A = 0 und I = 1 oder A = 1 und I = 0.
  - Zyklische Abhängigkeiten

JCoverage

- JCoverage ist ein kommerzielles Produkt, das die Vollständigkeit von Tests dokumentiert.
  - JCoverage erkennt, wie viele Teile des Codes bei den Testläufen ausgeführt wurden.

Motivation

integrata cegos

- Traditionellen Programmentwicklung läuft nach folgendem Zyklus ab:
  - Definition der Fachvorgaben
  - Implementierung durch den Programmierer
  - Testen der Anwendung durch die Fachabteilung
  - Endabnahme
- Dieser Ablauf offenbart aber sehr schnell zwei wesentliche Mängel:
  - Nach fertig programmierten Programmteilen wird ein recht großes Modul getestet und die Tester selber sind meistens nicht in der Lage, den Quellcode zu lesen.
  - Bei Programm-Änderungen (Fehlerbereinigung, neue Möglichkeiten) muss der komplette große Testzyklus durchlaufen werden.
- Um diese Probleme gar nicht erst auftreten zu lassen, sind die Tests wesentlich fein-granularer zu halten und müssen Bestandteil der normalen, täglichen Anwendungsentwicklung werden.
  - Zu jeder fachlichen Implementierung gehört ein Satz von Testfällen, die automatisch ablaufen können. Dies ist die allgemeine Grundlage der Unit-Tests.
- Als weiteres Hilfsmittel sind auch die so genannten „Mock Objects“ bekannt:
  - Diese Attrappen (englisch: „mock“) werden insbesondere bei komplexeren Anwendungen mit mehreren Schichten eingesetzt, um Subsysteme, die Fehler enthalten können, durch Klassen zu ersetzen, die wohl definierte Ergebnisse produzieren. Damit können die einzelnen Schichten unabhängig voneinander getestet werden.

Grundprinzip

integrata cegos

- Ein wesentliches Element von JUnit ist dessen Einfachheit:
  - Unit-Tests müssen ja vom Entwickler neben der „eigentlichen“ Arbeit zusätzlich programmiert und gestartet werden.
  - Es ist deshalb darauf zu achten, dass der Programmierer motiviert bleibt, die Tests sowohl zu erstellen als auch anzupassen.
  - Und genau diese Vorgabe erfüllt JUnit hervorragend.
- Laufzeitumgebung
  - Standalone
  - Grafische Oberfläche
  - ANT-Task
  - IDE-Unterstützung

JUnit-API

integrata cegos

- Die Klasse junit.framework.TestCase definiert drei Gruppen von Methoden:
  - Lebenszyklus des Tests.
    - Die Methoden setUp() und tearDown() werden vor jedem Testlauf automatisch aufgerufen. Werden darin beispielsweise die Testobjekte initialisiert bzw. zurückgesetzt, werden alle im TestCase definierten Testmethoden unabhängig voneinander abgearbeitet. Dieses Verhalten, ein so genanntes Fixture, ist der Gegensatz zur der Variante, in der die Testobjekte im Konstruktor erzeugt werden. In diesem Falle beeinflussen sich Tests gegenseitig, was ja auch durchaus beabsichtigt sein kann.
  - Ein Satz von einfachen Hilfsmethoden
    - (setName(String), countTestCases())
  - Eine Menge von assert<Aktion>-Methoden mit verschiedenen Parameter-Kombinationen. Diese sind geerbt aus der Klasse Assert.

JUnit-API

integrata cegos

- Eine junit.framework.TestSuite ist selber wiederum ein Test, der aber eine Liste von TestCase-Objekten aufnimmt und somit als Composite dient, da sowohl TestCase als auch TestSuite die gemeinsame Schnittstelle Test implementieren.
  - Suiten gruppieren somit mehrere Tests zu einer Gesamtheit.
- Weitere Klassen
  - Es gibt im Framework noch einige wenige weitere Hilfsklassen, die im Folgenden kurz angesprochen werden sollen. Im Paket junit.framework sind dies:
    - TestFailure und TestResult sind Klassen, die die gesammelten Informationen für einen Testlauf enthalten.
    - TestListener ist eine Schnittstelle, in denen eigene Implementierungen auf Start, Ende, Fehler eines Testfalles durch einen Ereignis-Mechanismus benachrichtigt werden.
    - Protectable-Objekte müssen die Methode protect() implementieren. Die darin ausgeführten Anweisungen können ein beliebiges Throwable-Objekt werfen. Damit definiert Protectable einen geschützten Block.
  - Im Paket junit.extensions:
    - ActiveTestSuite startet für jeden Test einen eigenen Thread.
    - Mit ExceptionTestCase kann das Werfen einer bestimmten Exception (Übergabe der Exception-Klasse im Konstruktor) innerhalb des Tests geprüft werden.

Testen mit Mock-Objekten

integrata cegos

- Ein Mock-Objekt ist eine Implementierung einer Schnittstelle, die ein wohldefiniertes Verhalten zeigt. Im Gegensatz dazu liefert ein einfacher Dummy stets denselben Satz von Ergebnissen.
  - Nur mit Hilfe von Mock-Objekten sind echte Unit-Tests möglich.
- Als Problemlösung muss ein Mock als konfigurierbare Testklasse geschrieben werden.
  - Der Aufwand für dessen Erstellung ist jedoch nicht unbeträchtlich (Konfigurierbarkeit, Testen ob und welche Methoden in der richtigen Reihenfolge mit den richtigen Parametern aufgerufen werden etc.), kann jedoch durch Einsatz eines Werkzeugs deutlich vereinfacht werden.
  - Ein im Java Open Source-Umfeld gebräuchliches Tool ist EasyMock.