3.– 6. September 2012
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Schatzsuche

In-House-Bibliotheken identifizieren und fördern

Dr. Malte Clasen

adesso

Über uns

- Malte Clasen
  Softwareentwickler
  http://malteclasen.de

- adesso AG
  IT-Dienstleister
  http://adesso.de

Problem: Code-Schnipsel-Sammlungen

- wiederholt auftretende Probleme, die nicht zur Kernfunktionalität gehören
  - Emails versenden, HTML filtern, Passwörter generieren, …
- betrifft insbesondere Installationsroutinen, die einmal pro System ausgeführt werden müssen
  - Datenbank konfigurieren, Benutzerkonten einrichten, …
- Copy-Paste-Übernahme in neue Projekte/Module
- hauptsächlich Skripte, aber auch einzelne Klassen

Copy-Paste-Problematik

- unübersichtlich, schlecht wartbar, etc.

- Copy & Paste & Bug
  Vom Umgang mit Redundanz in Software-Artefakten

  Elmar Jürgens
  CQSE GmbH

  A23 Mittwoch, 5.9., 14:00 – 15:10 Uhr

Grund: Projektmanagement

- keine Zeit
  - knappe Deadlines
- nicht kundenverrechenbar
  - in T&M-Projekten (nach Aufwand bezahlt), wenn der Kunde nur wenig Ahnung von Softwareentwicklung hat
- trivial
  - wenn Entwickler sich und/oder ihre Kollegen überschätzen
  - häufig zusammen mit fehlenden automatisierten Tests
- nicht im Projektplan vorgesehen
  - häufig zusammen mit fehlendem Refactoring

Grund: Restriktive Richtlinien

- zu hoher Aufwand, Änderungen an Core-Modulen einzureichen
  - nur wenige „Gurus“ dürfen am „Core“ arbeiten
  - langsame Integration-Tests
- Basis-Funktionalität über weitere Module verstreut
  - Entwickler lösen die Probleme dann eben lokal
- Copy-Paste
  - Umgehung des „Core“-Moduls

Grund: Undurchsichtiger Code

- viele Entwickler implementieren dieselbe Funktionalität mehrfach
  - häufig bei zu komplexen Interfaces
  - fehlende Beispiele/Tests
- jeder Abschnitt für sich erscheint trivial
- jede Implementierung mit minimal unterschiedlichem Verhalten

Ansatz: Bibliotheken

- Grundidee: Bibliothek
  - Sammlung von an mehreren Stellen genutztem Code
  - allen Entwicklern zugänglich (nutzen und erweitern)
- Problem: Das Ziel ist jedem Softwareentwickler klar, der weg dahin oft undefiniert

Beispiel: MyBatis

- Data-Access-Layer für Kryptographie-Software
- Weiterverwendung in JPetStore-Beispiel
- Aufmerksamkeit in der Open-Source-Community
- Fokus auf Data-Access-Layer
- Code-Spende an Apache Software Foundation
- http://www.mybatis.org/about.html

Beispiel: Fluom

- Datenstruktur-Updates in SharePoint über Ad-Hoc-Code
- Identifikation der wiederkehrenden Muster
  - Spalten hinzufügen, umbenennen, löschen, …
- Interface angelehnt an vergleichbare Technologien
  - SQL: LiquiBase, dbDeploy
- Implementierung ermöglicht neue Muster
  - automatisierte Tests, LINQ, …
- Verwendung als Hilfsmittel bei adesso
- Verwendung für neue adesso-Produktlinie
  - Intranet-Templates: Identifikation wiederkehrender Kundenfunktionalität, „High-Level-Bibliothek“

Copy-Paste-Wiederverwendung

- man kopiert Code

- Indiz: keine funktionale Veränderung notwendig
- Indiz: man kann die Funktionalität benennen
  - beispielsweise „MIME-Type aus Dateiendung ableiten“
  - über den aktuellen Projektkontext hinaus wertvoll

- Gegenanzeige: Komplexität durch Wahl der Programmiersprache
  - for (std::vector<int>::iterator it=v.begin(), itEnd=v.end(); it!=itEnd; ++it)

    ist nicht hübsch, aber idiomatisch in C++03

Vorgehen erklären lassen

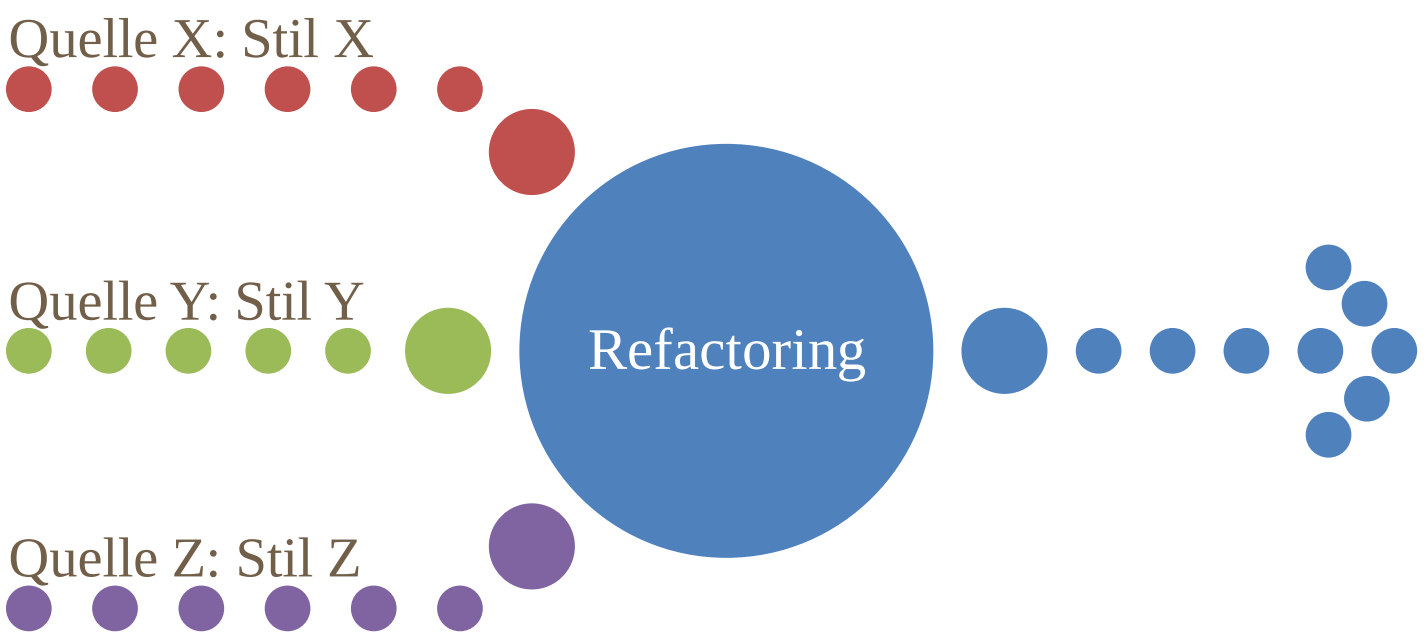
- neue Aufgabe, die von einem Kollegen in ähnlicher Form bereits gelöst wurde, von diesem erklären lassen

- Indiz: Code, der von irgendwo hergezaubert wird
  - Transfer zwischen Projekten via Email, USB-Stick, …

Wiederverwendung

- Nur Code bei der ersten oder zweiten Wiederverwendung extrahieren
- nicht im Voraus umfangreiche Bibliotheken erstellen
  - Prototypen/Spikes sind natürlich ok, zählen aber bei der Wiederverwendung nicht mit

Refactoring



Gültigkeitsbereiche

- manche Probleme treten nur bei manchen Projekten/Kunden auf
  - mögliche Ursachen: Eigenwillige Workflows, Richtlinien, …
- Code nicht über die Grenzen des Problems hinweg verallgemeinern
  - ein Spezialfall darf gern eine spezielle Lösung bekommen, wenn die allgemeinere Fassung bei Berücksichtigung unnötig kompliziert wird

Abgrenzung: Gemeinsame Infrastruktur

- Unterschiedliche Projekte haben unterschiedliche Anforderungen
- Beispiel: Hintergrund-Tasks
  - Wiederholungen im Fehlerfall?
  - Transaktionen möglich/notwendig?
  - Abhängigkeiten zwischen Tasks?
- Nichts zwangsweise verallgemeinern bzw. vereinheitlichen
  - erhöht die Komplexität von Interface und Implementierung
- http://ayende.com/blog/154753/beware-the-common-infrastructure

Toolkit vs. Framework

- Framework
  - bestimmt die Architektur
  - projektspezifischer Code wird vom Framework aufgerufen
- Toolkit
  - eigene Architektur
  - projektspezifischer Code ruft Toolkit-Funktionen auf
  - ➔ meist die bessere Wahl

Qualität

- Code wird mindestens an zwei verschiedenen Stellen verwendet → erschwert manuelle Tests
- klare Schnittstellendefinition notwendig, sonst kann man sich nicht auf die Funktionalität verlassen
- daher maximal mögliche Code-Qualität notwendig
  - Unit-Tests, Integration-Tests, Regression-Tests, …
  - Continuous Integration
  - Refactoring

Erweiterung vs. Ableitung

- neue String-Funktion: Tabs zu Spaces
- Ableitung

```
class MyString : String {
    string TabsToSpaces() { … }
}
```

- Erweiterung

```
static class StringExtension {
    static string TabsToSpaces(this string source) { … }
}
```

- Erweiterung funktioniert mit allem bestehenden Code

Code vs. Design Pattern

- Code: Algorithmus
  - Beispiel: Sortierung
- Design Pattern: Prinzip
  - Beispiel: Factory

- Nicht alles sollte als Code in Bibliotheken implementiert werden
- Zentrale Stelle für (In-House) Design Pattern in Textform
- Helfer für Design Pattern erlaubt
  - z.B. Loki::SingletonHolder

Entwicklungsprozess

- Bibliothek von der Natur der Sache her agil
  - es existiert per Definition kein Pflichtenheft o.ä.
- leichtgewichtige agile Entwicklungsprozesse und Technologien können unterstützen
  - Anbindung an den Prozess des eigentlich zu lösenden Problems extrem wichtig
  - guter Kandidat: Kanban
- Aufpassen: geänderte Interfaces können sich auf viele Projekte auswirken
- Semantic Versioning: http://semver.org/

Richtlinien

- jeder Entwickler soll Code beisteuern
- zu Anfang gerne ein gemeinsames Repository mit Schreibrechten für alle
- Vandalismus-Risiko In-House minimal
- Fortbildung bei mangelhaften Commits
- restriktivere Regelungen nur bei Bedarf
- aufpassen, dass Commits nicht zu aufwändig werden

Projektleiter

- Bibliotheken werden idealerweise von mehreren Projekten verwendet
- Ressourcen mit anderen Projektleitern gemeinsam abstimmen
- möglichst niedrige formale Hürden für die Entwickler

Führungskräfte

- 20% Zeit für eigene Projekte
  - http://montyprogram.com/hacking-business-model/
- Entwickler werden selbst ihr Arbeitsumfeld optimieren
- Qualität und Wiederverwendbarkeit fördern
- Bonussystem kann falsche Anreize schaffen
  - wenn Arbeit an der Bibliothek den anderen Entwicklern hilft, höhere Boni einzuspielen, die eigenen aber reduziert

Variante 1: In-House

- Standard, keine weiteren Entscheidungen notwendig
- Problem: Einbringung in kundenspezifische Projekte
- mögliche Lösung: Projektbeistellung, z.B.
  - keine Kosten für den Kunden
  - Weiterentwicklung darf zurückfließen
  - Quellcode wird mit übergeben
  - keine Wartung/Gewährleistung
- vertragliche Regelung notwendig

Variante 2: Produkt

- Bibliothek selbst erzeugt Umsatz
- Wartung/Gewährleistung notwendig
- höhere Anforderungen an
  - Dokumentation (in druckbarer Form)
  - Vollständigkeit (Produkt wächst i.A. nicht so organisch)
  - Präsentation (Web-Auftritt, Messestände, …)
- Hemmschwelle bei Nutzern
  - Lock-In
  - eigene Implementierung ggf. vergleichsweise einfach
  - direkte Kosten

Variante 3: Open Source

- klare Lizenzbedingungen regeln kundenspezifischen Einsatz (z.B. LGPL)
  - kein Lock-In
- organisches Wachstum gängig
- restriktivere Commit-Richtlinien notwendig
- Community-Betreuung (Nutzer und Committer)
- ggf. Umsatz mit Support

Zusammenfassung

- wichtigstes Merkmal: Copy-Paste
- nur bei Wiederverwendung extrahieren
- hohe Qualität zwingend erforderlich
- passenden Entwicklungsprozess wählen
- formale Hürden zwischen Projekten minimieren
- Vorsicht bei Bonus-Systemen
- abwägen zwischen In-House, Produkt, Open Source

Firma

- mein Blog: http://malteclasen.de/blog
- mein Arbeitgeber: http://adesso.de
  - Stellenangebote: http://www.aaajobs.de/

Vielen Dank!

Dr. Malte Clasen
adesso AG