Software-Wiederverwendung (Stichworte)

Udo Kelter

09.06.2010

Zusammenfassung dieses Lehrmoduls

Software-Wiederverwendung wird als ein entscheidendes Mittel angesehen, die Entwicklungszeit und -Kosten von Software zu senken und die Qualität zu erhöhen. Dieses Lehrmodul analysiert zunächst verschiedene Arten von Wiederverwendung (ungeplante, geplante, in Systemfamilien bzw. Produktlinien), die möglichen Kosteneinsparungen und an der Wiederverwendung orientierte Vorgehensmodelle.

Vorausgesetzte Lehrmodule:

empfohlen: – Softwarearchitekturstile

Stoffumfang in Vorlesungsdoppelstunden: 1.0

Inhaltsverzeichnis

1 Wiederverwendung

Grundidee: Produkte möglichst weitgehend aus vorhandenen / vorgefertigten Standard-Komponenten konstruieren
Vorteile:
- Kostensenkung, weniger Neuentwicklung
- Erhöhung der Qualität

– eigentlich keine neue Idee, passiert tausendfach:

- Benutzung von Bibliotheken
- Muster, ...

1.1 "Wiederverwendung" vs. Entwurfs- / Struktur-Erfahrung

nach Größe der Artefakte sortiert:

1. Programmstrukturen, Konzepte der Programmiersprache
2. Entwurfsmuster, Analysemuster
3. Architekturmuster
4. Standardarchitekturen

sind nur Gestaltungsregeln, die erst "instanziiert" werden müssen, keine Komponenten, die man i.w. unverändert einbaut

überlappt thematisch mit Wiederverwendung

1.2 Wiederverwendung vs. technologische Arbeitsteilung

Fremdherstellung von Komponenten ist in klassischen Industrien üblich:

- ggf. eigenes, umfangreiches Know-How erforderlich, das nicht zu den Kernkompetenzen des Anbieters gehört

- Ausnutzung von Skaleneffekten bei kleinen Produktionsmengen

allgemeines Prinzip der industriellen Arbeitsteilung:

- spezialisierte Marktteilnehmer,
- geringe Fertigungstiefe,
- Konzentration auf Kernkompetenzen

wichtig: nicht zu viele Modelle, Normen / offene Spezifikationen ("DIN-Schrauben")

Besonderheiten in der Informatik:

- Größe der Produktionsmengen irrelevant für SW-Produkte
- "versteckte" technologische Arbeitsteilung durch Benutzung von Netzwerkprotokollen, Graphikpaketen, DBMS, mathematische Bibliotheken, Transaktionsmonitore usw.

  wenig sichtbar, weil keine expliziten Kopien bzw. Lizenzgebühren

→ technologische Arbeitsteilung auch in der Informatik selbstverständlich

ist auch ohne Wiederverwendung sinnvoll (also wenn zugekaufte Komponenten werden nur 1* verwendet)

1.3 Wiederverwendung in engeren Sinne

Fälle, wo *mehrere einander ähnliche Systeme* in der gleichen Technologie entwickelt werden

- in verschiedenen Zeiträumen
- insb. bei Systemen, die nicht Varianten voneinander sind, sondern unabhängig voneinander entwickelt werden
- gleichzeitig: Entwicklung einer **Systemfamilie**

  Besonderheit: gemeinsame, wiederverwendbare Komponenten können gezielt bestimmt werden

1.4 Typen wiederverwendbarer Dokumente

im Prinzip beliebige Dokumenttypen, in beliebigen Entwicklungsstufen:

- Anforderungen, Testfälle
- Quelltexte von Programmen
- Modul-/API-Spezifikationen
- Architekturen bzw. Architekturfragmente
- Datenbankschemata
- Gestaltungselemente von GUIs
- Dokumentation: Bedien-, Installationshandbücher, Glossare etc.

ggf. zusammenhängende Gruppen, z.B. GUI-Komponente bestehend aus Code, Gestaltung, Hilfesystem, Manual

Schwerpunkt in der Praxis: Quelltexte, Architekturen, API-Spezifikationen, Datenbankschemata

1.5 Geplante vs. ungeplante Wiederverwendung

Wiederverwendung hier im Sinne von Wiederverwendbarkeit

ungeplante Wiederverwendung:

- Komponente wird nach ihrer Entwicklung für Wiederverwendung entdeckt
- muß i.d.R. abgeändert / verbessert / nachdokumentiert werden

geplante Wiederverwendung / Wiederverwendbarkeit:

- Komponente von vorneherein zwecks Wiederverwendung entwickelt und gestaltet
- (a) Bibliotheksfunktionen: nur zur Wiederverwendung gedacht
- (b) Teile eines konkreten Systems, die vermutlich später noch einmal verwendet werden können

2 Grundformen wiederverwendungsorientierter Entwicklungsprozesse

- primär in den mittleren bis spätern Phasen
- ohne genaue Kenntnis der Wiederverwendungen

2.1 Grundform: Architektur- und Komponentenwiederverwendung

1. Bilden der (Grob-) Architektur des Systems
2. Suche nach geeigneten Komponenten in einem Vorrat wiederverwendbarer Komponenten;
   ggf. anpassen der Architektur (d.h. zurück zu Schritt 1)
3. gefundene Komponenten in die Entwicklungsversion übertragen
4. Anpassen (abändern) bzw. ggf. Konfigurieren der übernommenen Komponenten

Anmerkung zu Schritt 1:

i.d.R. bestimmte architektonische Strukturen und Konventionen bei Komponenten vorausgesetzt, z.B. Fehlerbehandlung

bei Frameworks ist das Hauptprogramm komplett vorgegeben
→ individuelle Teile des Systems anpassen

Anmerkung zu Schritt 4:

- **black-box-Wiederverwendung**: Komponente wird völlig unverändert wiederverwendet
- **white-box-Wiederverwendung**: Komponente wird verändert

2.2 Kostenanalyse

Vergleich mit klassischer Neuentwicklung:

Schritt 1: wahrscheinlich kein großer Unterschied

Schritt 2: Suche nach Komponenten ist ohne besondere Vorbereitungen problematisch!

Hindernisse:
- finden der entsprechenden Dateien (ggf. auf Archivierungsmedien)
- schlechte Trefferqualität bei Stichwortsuche
- hoher Aufwand zur Analyse der einzelnen Fundstellen, Einschätzung des erforderlichen Anpassungsaufwands unsicher

Schritt 3: Übernahme der Komponente in die Entwicklungsversion arbeitsaufwendig, falls es sich z.B. um verstreute Codefragmente handelt

Schritt 4: ggf. umfangreiche Anpassungsarbeiten;

anschließend kompletter Test der Komponente erforderlich (Aufwand!)

→ reduziert Kosteneinsparung durch die Wiederverwendung

Rechenbeispiele für eine Komponente, deren Neuentwicklung 15 Stunden dauert:

- black-box-Wiederverwendung 0.5 Stunden, kein Suchaufwand
  → Aufwand um den Faktor 30 reduziert!
- langwierige Suche: 2 Stunden
  Änderungen an der Komponente: 6 Stunden incl. Test
  im Durchschnitt 2 vergebliche Suchvorgänge auf einen erfolgreichen
  → Aufwand um 20 % reduziert,
  ungünstige Risikostruktur: möglicher Gewinn 8 Stunden, möglicher Verlust 2 Stunden

Schlußfolgerungen:

- sehr kleine Komponenten (Aufwand ca. eine Stunde): Wiederverwendung lohnt i.a. nicht

- umfangreiche Komponenten: lohnt im Prinzip, aber reduzierte Wahrscheinlichkeit, daß die Komponente unverändert zum aktuellen Bedarf paßt
  → erhöhte Wahrscheinlichkeit von Änderungen

2.3 Geplantes Wiederverwenden

Voraussetzungen / positive Einflußfaktoren für Wiederverwendung:

- gute Suchfunktionen
- leichte Beurteilung gefundener Komponenten
- möglichst unveränderte Übernahme der Komponenten

... können gezielt verbessert werden → geplante Wiederverwendung

Denkbare vorbereitende Maßnahmen

- Sammlung aller potentiell wiederverwendbaren Komponenten in einem (logisch) zentralen Repository
- zusätzliche Beschreibung der Komponenten anhand eines Klassifikationsschemas und durch Schlagworte
  Möglichkeit, Klassifikation / Schlagworte bei der Suche auzunutzen
- erhöhte Qualität der technischen Dokumentation der Komponenten, der Strukturierung und Lesbarkeit des Programmcodes usw.
- andere (allgemeinere) Gestaltung der Funktionalität und/oder Schnittstellen der Komponenten
- besonders sorgfältiger Test der Komponenten (Vertrauen der Entwickler und Akzeptanz erhöhen)

Mehraufwand im Bereich von 30 - 60 % des normalen Entwicklungsaufwands

Faustregel: amortisiert sich erst nach nach 3 Wiederverwendungen

2.4 Wiederverwendungs-Repositories

2.4.1 Funktionsbereiche von Wiederverwendungs-Repositories

(Komponenten- bzw. Wiederverwendungs-) **Repository**: System, das wiederverwendbare Komponenten verwalten kann und das die Suche nach Komponenten unterstützt; Funktionen eines Repositorys:

1. *Verwaltung deskriptiver Daten zu den Komponenten:* abhängig vom Typ der Komponenten

   Bsp: Programmiersprache, Compilerversion, Autor, Erstellungsdatum, Stichworte zum Inhalt, Klassifizierung, usw. → **Metadaten**

2. *Pflege der Klassifikationsschemata:* Funktionen für Aufbau und Weiterentwicklung; zugehörige Statistikfunktionen

3. *Suche nach Komponenten:* i.d.R. auf Basis der deskriptiven Daten, nicht die Komponenten selbst

   vage Suche (Information Retrieval), iterative Verfeinerung / Anpassung der Suchkriterien

4. *Import von Komponenten* aus Dateien oder Projektdatenbanken

5. *Export von Komponenten* (sofern die Komponenten überhaupt im Repository verwaltet werden)

6. *Nachweis von Wiederverwendungen:* Erfassung von erfolglosen Analysen und erfolgreichen Wiederverwendungen. Relvante Informationen:
   - Wie oft ist diese Komponente schon bei einer Suche gefunden worden?
   - Wer hat sich diese Komponente schon einmal angesehen und analysiert (wovon man profitieren könnte)?

     → Entwickler sollten Komponenten bewerten können

- In welchen Systemen ist diese Komponente wiederverwendet worden? Welche anderen Komponenten wurden dort ebenfalls wiederverwendet?

7. *Nutzer- und Rechteverwaltung,* d.h. Kontrolle, wer das Repository wie nutzen darf

8. *Abrechnungsfunktionen* abhängig von den rechtlichen Rahmenbedingungen

Rolle **Administrator**: kontrolliert Import, Wartung der Klassifikationsschemata, Rechteverwaltung

Auffassung: Sammlung wiederverwendbarer Komponenten ist ein wertvolles Gut, das dem Unternehmen gehört und betreut werden muß

alternative Auffassung: Wiederverwendungsrepository als eine Austauschplattform

2.4.2 Autarke vs. integrierte Repositories

autarkes Repository:

Beispiel: Sammlung von Quellprogrammen, über eine WWW-Schnittstelle nutzbar

klare Trennung zwischen Anbieter und Käufer / Nutzer von Komponenten

Repository technisch getrennt von der Entwicklungsumgebung der Nutzer der Komponenten; nur gelegentlicher Zugriff

in eine SEU integriertes Repository : wird von Entwicklern intensiv bei täglichen Arbeit genutzt

verfolgt über die Wiederverwendung hinausgehende Ziele:
- Anwendungen zu dokumentieren
- Anwendungen inhaltlich zu integrieren
- Doppelentwicklungen zu vermeiden
- Auswirkungen von Änderungen einzuschätzen

2.4.3 Abgrenzung zu anderen Systemen

Versionsarchive von VM/KM-Systemen: enthalten überwiegend nicht wiederverwendbare Komponenten
Suche nach Komponenten wird nicht unterstützt
keine geeignete Dokumentation von Komponenten

Data-Dictionary-Systeme: dokumentieren primär Datenelementtypen in Datenbanken,
Bereitstellung dieser Metadaten zur Laufzeit der Programme auf dem Produktionsrechner.

3 Systemfamilien bzw. Produktlinien

Systemfamilie / **Produktfamilie** / **Produktlinie**: eine Gruppe ähnlicher Software-Systeme, denen eine gemeinsame Architektur sowie gemeinsam genutzte Komponenten zugrundeliegen

hoher Grad an Wiederverwendung der

- Modelle
- Implementierungen
- Dokumentation

3.1 Hauptmerkmale

1. Systemfamilie als solche ist vorgegeben und bekannt!!
   typische Systemfamilie: Produkte innerhalb eines Marktsements
   → gemeinsame Betrachtung der Menge von Systemen,
   → es ist klar erkennbar, wo Potential für Wiederverwendung besteht
   → mehrfach verwendbare Komponenten zielgerichtet auf ein Produktportfolio unter vorhersehbaren technischen Rahmenbedingungen entwickeln!

2. Variantenmanagement: Fokus auf den Unterschieden zwischen den Einzelsystemen, keine isolierte Behandlung von Einzelsystemen

   Variationspunkte: Merkmale, in denen sich die Einzelsysteme unterscheiden und bei denen eine von mehreren Optionen gewählt werden muß

3. Architekturzentrierte Entwicklung: einheitliche Architektur und Plattform für alle Einzelsysteme (betrifft oft die Geschäftsstrategie des Unternehmens)

4. dedizierter Entwicklungsprozeß: Product Line Engineering

   beinhaltet zwei verschiedene Entwicklungsprozesse:

   1. **Domänenentwicklung**: Analyse und Entwicklung der wiederverwendbaren Artefakte
   2. **Applikationsentwicklung**: Erstellung der eigentlichen Einzelsysteme

3.2 Domänenentwicklung

Product Line Engineering:

- Gemeinsamkeiten und Unterschiede der Mitglieder der Systemfamilie bestimmen, z.B. mit Feature-Modellen
- bedingt sehr gutes Domänenwissen
- Haupttätigkeiten:
  1. Domänenanalyse – Verstehen der Anforderungen
  2. Domänendesign – Entwerfen der Architektur
  3. Domänenimplementierung – Umsetzen der Architektur

Domänenanalyse:

- mögliche Produkte und Produktanforderungen; Abgrenzung, Ausgrenzung, was nicht mehr enthalten

- Bestimmung der Variabilitäten von Software-Systemfamilien (technische, fachliche)
- Entwicklungsprognosen
- Entscheidung über prinzipielle Herangehensweise: Domänenspezifische Sprachen / MDD oder bedingte Compilierung oder Framework oder ....
- "Konfigurationsraum" für Mitglieder der Systemfamilie bestimmen; Abhängigkeiten zwischen Variationspunkten bestimmen

Scoping: Planung, welche Bestandteile wiederverwendbar zu implementieren sind und welche nicht

Ziele:

- nur tatsächlich mehrfach verwendete Funktionen wiederverwendbar machen (Aufwand minimieren)
- vorhandenes Wiederverwendungspotential ausnutzen (Nutzen maximieren)

Anforderungen an die Applikationsentwicklung bzw. die Domänenentwicklung trennen!
(auch bei späterer Weiterentwicklung der Produktlinie)

konkret: aufteilen aller Anforderungen, Codeteile, sonstige Entwicklungsartefakte in drei Gruppen:

- solche, die *für alle* Einzelsysteme gelten (Gemeinsamkeiten; feste, nicht austauschbare Komponenten),
- solche, die nur *für einen Teil* der Einzelsysteme, aber nicht für alle relevant sind (Variabilitäten; austauschbare Komponenten) und
- solche, die nur *für bestimmte* Einzelsysteme relevant sind (produktspezifische Anforderungen / Komponenten)

Domänendesign:

- Implementierung der Plattform
- Wahl von Bindungspunkten
  (→ Gestaltung des Applikationsentwicklungsprozesses)

3.3 Bindungszeitpunkte

"Zeitpunkt" (Stadium im Entwicklungsprozeß der Applikationsentwicklung), in dem für einen Variationspunkt eine Entscheidung getroffen wird; Beispiele:

- bei der Auswahl wiederverwendbarer Komponenten
- bei Entwurf oder Kodierung
- beim Übersetzen (z.B. durch bedingte Kompilierung)
- Package time (beim Zusammenstellen von jar-Archiven, statischem Binden, ...)
- durch installationsseitige Anpassungen
- bei der Produktinstallation
- beim Systemstart (angebene Optionen / Parameter)
- während der Systemlaufzeit

es können *mehrere* Bindungszeitpunkte zum Einsatz kommen, z.B.

- Product Manager trifft an frühen Bindungszeitpunkten einheitliche unternehmensweite Entscheidungen
- Benutzer entscheidet zur Laufzeit anhand aktuellen Bedarfs

mehrere Bindungszeitpunkte → Prozeß der Applikationsentwicklung ist eine Art Pipeline, in der eine Folge von Artefakten entsteht, die immer weniger offene Details enthalten
(partiell oder ganz "instantiierte Produkte")

3.4 Applikationsentwicklung

Formen des Produktionsprozesses:

- vollautomatische Generierung durch Produktgeneratoren bzw. -Konfiguratoren; Eingabe u.a.: Spezifikation der Auswahlentscheidungen
- manuelle Entwicklung: anhand eines als Text vorliegenden Produktionsplans, ggf. mit manueller Entwicklung von "glue code"
- Mischformen

3.5 Vertiefende Lektüre

1. Variability in Software Product Lines CMU/SEI-2005-TR-012; `http://www.sei.cmu.edu/reports/05tr012.pdf`
   dort insb. lesen: Kap. 6 Variation Mechanisms
2. A Framework for Software Product Line Practice, Version 5.0; SEI; `http://www.sei.cmu.edu/productlines`

Aufgaben:

1. Vertiefende Lektüre durchsehen
2. Wo haben Sie selber schon einmal ein Produkt mit Varianten entwickeln müssen oder wollen?
   Welche Variationsmechanismen haben Sie dabei benutzt bzw. in Betracht gezogen?

