Tracing Software Systemevolution

Harry M. Sneed
SoRing Kft. H-1221 Budapest
Birgit Demuth
Technische Universität Dresden, Institut für Softwaretechnik
Harry.Sneed@T-Online.de

Abstrakt: Dieser Beitrag behandelt das Thema Konsistenz in der Software Evolution. Softwaresysteme bestehen aus mehreren Schichten von Modellen, Dokumenten, Codekomponenten und Testfällen. Sie sollten im Gleichschritt fortgeschrieben werden, sonst driften sie auseinander und werden nicht mehr brauchbar. Ihre statische Konsistenz, bzw. ihre Traceability, zu prüfen ist eine Aufgabe der Qualitätssicherung, denn um effektiv zu bleiben müssen sie konsistent bleiben. Jede Änderung zum Code muss in der Anforderungsdokumentation, sowie in den Testdokumenten reflektiert werden. Dafür ist ein automatisiertes Prüfverfahren erforderlich bei dem die Software-Artifakte miteinander abgeglichen und Inkonsistenzen ausgeweisen. In diesem Beitrag wird ein derartiges tool-gestütztes Verfahren vorgestellt.

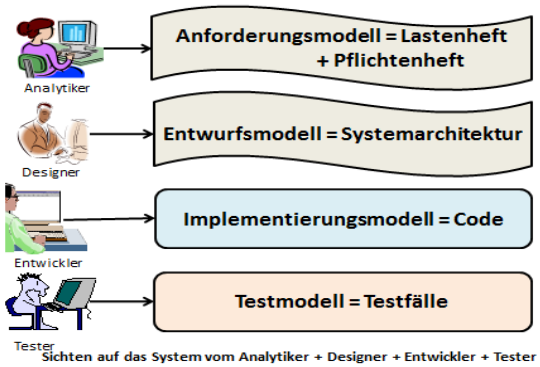
Schlüsselwörter: Software-Evolution, Software Modellierung, Anforderungsmodelle, Entwurfsmodelle, Testdokumente, Traceability

1 Zusammensetzung eines komplexen Softwaresystems

Komplexe Softwaresysteme bestehen aus verschiedenen Artifaktentypen. Neben den Codekomponenten, gibt es Entwurfsmodelle, Anforderungen, Testskripten und Bedienungsanleitungen. Diese Artifakte sind in Teilprodukte zusammengefasst. Nach dem V-Modell-XT kommen neben dem Source-Code mindestens vier weitere Teilprodukte vor:

- Lastenheft
- Gesamtsystemspezifikation bzw. Pflichtenheft
- Entwurf der Systemarchitektur
- Prüffallspezifikation [Höh08].

Diese Teilprodukte haben wenig Wert wenn sie nicht aufeinander abgestimmt sind. Wenn der Code geändert wird muss auch der Test und das Entwurfsmodell angepasst werden. Der Verfall eines Softwaresystems beginnt nach Lehman, der Vater der Software Evolution mit der Inkonsistenz der Systemteile [Lehm98]. Deshalb muss die Konsistenz ständig überwacht und Abweichungen sofort aufgefangen werden.



2 Prüfung der Systemkonsistenz

Am besten wäre es wenn die Konsistenzprüfung gleich bei der Änderung eines bestehenden Artifakts, bzw. beim Einfügen eines neuen Artifakts, online stattfindet. Der Entwickler soll darauf aufmerksam gemacht werden, dass er auch die assoziierten Artifakte ändern muss. Da die meisten Entwicklerbetriebe keine derartig integrierte Entwicklungsumgebung haben, bleibt es nur übrig die Konsistenzprüfung nach Vollendung der Änderung offline im Batchbetrieb zu prüfen. Dies hat aber den Vorteil, dass sämtliche Abhängigkeiten gleich geprüft werden können und der Entwickler bekommt einen Bericht über alle erkennbaren Inkonsistenzen.

Als Voraussetzung für diese Prüfung muss zunächst eine Entity/Relationship Repository aufgebaut werden. Dies geschieht über die statische Analyse aller Teilprodukte. Die Anforderungsdokumente werden mit einem Textanalysator für die natürliche Sprache analysiert um die Hauptwörter und Prädikate zu entnehmen. Die UML Schemen werden mit einem XML Analysator gelesen um die Elemente und Beziehungstypen zu identifizieren. Der Code wird mit einem Code-Analysator geparst um die Code-Abschnittsnamen, die Objektnamen, die Operanden und die Operatoren auszuschneiden. Die Testfälle werden mit einem Tabellenparser verarbeitet um die Namen der Testdaten und der Testprozeduren zu gewinnen. Die Beziehungen vom Basisentität, z.B. der Anwendungsfall, zur Zielentität, z.B. dem Datenobjekt, werden erkannt und in die relationale Tabelle für diesen Entitätentyp eingefügt. Zum Schluss steht eine Repository, bzw. eine Datenbanktabelle, mit mehreren tausend E/R Beziehungen. Der Syntax der Tupel ist:

BasisTyp; BasisName;Bezieungstyp; ZielTyp;ZielName

Anforderungstabelle:

```
PROC;Auftragsbearbeit ;OWNS;CASE;UseCase Rechnungsstellung
CASE;UseCase-05       ;HAS ;ACT ;  Buchhalter
CASE;UseCase-05       ;HAS ;TRIG;  Menuauswahl
CASE;UseCase-05       ;FILL; REQU; stelle monatliche Rechnung.
CASE;UseCase-05       ;USES;OBJT;  GO-01-Kunde
CASE;UseCase-05       ;USES;OBJT;  GO-06-Rechnungsposten
CASE;UseCase-05       ;INPT; MASK;GUI-Rechnungsmaske
CASE;UseCase-05       ;OUTP;REPO; LIST-05-Rechnung
CASE;UseCase-05       ;OUTP;REPO; LIST-06-Rechnungsprotokoll
CASE;UseCase-05       ;IMPL; RULE; GR-09-Rechnungsbetrag
CASE;UseCase-05       ;IMPL; RULE; GR-10-Mehrwert
CASE;UseCase-05       ;HAS ; PRE ;  Rechnungsposten sind da
CASE;UseCase-05       ;HAS ; POST;  Rechnungen sind gedruckt
CASE;UseCase-05       ;PERF; STEP; Buchhalter startetAbrechnung.
CASE;UseCase-05       ;USES; DATA; Kundennr
CASE;UseCase-05       ;USES; DATA; Gesamtbetrag
```

Anschließend werden, ausgehend von den geänderten Einträgen, alle abhängigen Entitäten und Beziehungen gesucht und gesammelt. Danach erfolgt ein Bericht über die betroffenen Artifakte.

3 Analyse der Anforderungsdokumente

Anforderungsspezifikationen haben bestimmte wohldefinierte Eigenschaften wie Geschäftsziele, Anforderungen, Geschäftsregeln, Geschäftsobjekte, Systemschnittstellen und Anwendungsfälle. Diese Eigenschaften sind über Schlüsselwörter im Text zu erkennen. In der Regel werden als Erstes die Ziele definiert, gefolgt von Regeln, Objekten, Akteuren, Anforderungen, Schnittstellen und zuletzt den Anwendungsfällen. Möglicherweise werden auch die fachlichen Testfälle angehängt. Falls es noch nicht gemacht wurde, werden vor der maschinellen Verarbeitung die Schlüsselwörter per Hand in den Anforderungstext eingefügt. Bei einem großen Dokument kann das mehrere Tage dauern.

Geschäftsobjekte:
&GO-01: Kunde
&GO-11: Rechnung
Geschäftsregel:
&GR-09: Eine Rechnung ist innerhalb 30 Tage zu bezahlen.
&GR-10: Der Rechnungsbetrag ergibt aus der Menge der bestellten Artikel x Artikelpreis + Mehrwertsteuer.
Akteure:
&Akteur: Buchhalter

Funktionale Anforderungen:
&FUNC-REQ-04 Stelle eine monatliche Rechnung

4 Analyse des Entwurfsmodells

Die Analyse des Entwurfsmodells geht einfacher weil der Entwurf schon in einer formaler Sprache UML-XMI formuliert ist. Es geht nur darum die gewünschten Modelltypen zu erkennen und herauszuholen.

```
<packagedElement;xmi:type='uml:Class'
xmi:id='5558_4770' name='CustomerOrder'> →

COMP;OrderEntry       ;OWNS; CLAS; CustomerOrder;
```

5 Analyse des Codes

Der Code wird vom Tool SofRedoc analysiert um die wesentliche Codeelemente heraus zu picken. Codeelemente können andere Codeelemente besitzen, benutzen, aufrufen, usw. Codeelemente können auch Datenelemente besitzen, benutzen, abfragen, setzten, usw. Hier kommt es auf die Namen der Operanden. Sie werden mit den Namen der Attribute in UML und mit den Hauptwörtern in den Anforderungen verglichen. Falls die Namen ungefähr gleich sind, wird angenommen dass der Codebaustein zu dem der Datenname gehört die Anforderung implementiert zu der das Hauptwort gehört. Natürlich müssen mehrere Namen, mindestens zwei, übereinstimmen bis eine feste Zuordnung möglich ist. Falls keine Namen übereinstimmen, wird angenommen, das die Anforderung nicht implementiert ist, zumindest nicht nachweislich.

Codetabelle:

```
Base;  Entity          ;Relation; Target; Entity
SYST ;Auftragsbearbeitung;OWNS;  COMP;Geschaeftsobjekte
COMP;Geschaeftsobjekte  ;OWNS;   CLAS;Geschaeftsobjekt
CLAS;Geschaeftsobjekt   ;HAS ;   CLAS;Rechnungsposten
CLAS;Rechnungsposten    ;OWNS;   FUNC;getRechnung
FUNC;getRechnung        ;RTRN;   OBJT;Rechnung
FUNC;getRechnung        ;CALL;   FUNC;getInt
CLAS;Rechnungsposten    ;OWNS;   FUNC;getInt
```

6 Analyse der Testdokumentation

Systemtestfälle werden parallel zum Code entwickelt um das System sobald wie möglich zu testen. Sie werden in der Regel in Tabellen erfasst, aus denen ausführbare Testprozeduren generiert werden. Falls die Testfälle manuell erstellt und fortgeschrieben sind, werden sie sich unweigerlich von der ursprünglichen Anforderungsdokumentation entfernen. Um die Konsistenz des Produktes zu gewährleisten, gibt es zwei Alternativen. Erstens, die Anforderungsdokumentation wird weggeworfen, und die Testfälle werden als Produktspezifikation benutzt. Zweitens, die Anforderungsdokumentation wird fortgeschrieben, und die aktuellen Testfälle werden ständig gegen die Anforderungsdokumentation geprüft. Die Prüfung wird hier mit dem Tool *SofTrace* durchgeführt. Über die Objekt- und Datennamen, auf die sich der Testfall bezieht, wird der jeweilige Testfall einem Anwendungsfall zugewiesen und dieser den Anforderungen. Auf dieser Weise wird eine Kette von der Anforderung über das Entwurfsmodell und den Code bis zum Testfall gebildet. Der Kreis schließt sich.

7 Literaturhinweise

[Leh98] Lehman, M. (1998): „Software‘s Future – Managing Evolution, IEEE Software Magazine, Jan, 1998, S.40.

[HöH08] Höhn, R., Höppner, S. (2008): Das V-Modell XT, Springer Verlag, Berlin, S. 193