

Sanierung objektorientierter Systeme

Holger Bär, Markus Bauer, Oliver Ciupke,
Thomas Genßler, Radu Marinescu, Benedikt
Schulz, and Joachim Weisbrod

Forschungszentrum Informatik (FZI)
Haid-und-Neu-Straße 10-14
76131 Karlsruhe
Germany

{baer|bauer|ciupke|genssler|
marinesc|bschulz|weisbrod}@fzi.de

1 Einleitung

Viele Unternehmen versprechen sich von der Einführung objektorientierter Technologien eine Verbesserung der Qualität ihrer Software sowie eine Reduzierung der Entwicklungszeiten. Insbesondere erhoffte man sich durch den Einsatz objektorientierter Mechanismen wie Vererbung, Kapselung oder Polymorphie zu flexibler, erweiterbarer, verständlicher und somit einfacher wartbarer Software zu kommen. Heute sind in der industriellen Praxis bereits Systeme mit mehreren Millionen Codezeilen im Einsatz, die aufgrund ihrer inhärenten Größe, Komplexität und Entwicklungszeit die Grenzen der Objektorientierung erreicht haben und die gewünschten Eigenschaften nicht aufweisen: Es handelt sich um monolithische, unflexible und kaum noch zu erweiternde Systeme.

Um diese Systeme wieder einer technisch und wirtschaftlich sinnvollen Nutzung zu erschließen, müssen sie saniert¹ werden. Leider ist das Gebiet der Software-Sanierung, insbesondere für objektorientierte Systeme, noch unzureichend erschlossen und so ist weder klar, welche Verfahren bei der Software Sanierung zum Einsatz kommen sollen, noch in welcher Reihenfolge die Probleme gelöst werden müssen, um den Erfolg eines Sanierungs-Projektes garantieren zu können.

In dem vorliegenden Papier beschreiben wir in Anlehnung an [Cas98] einen Lebenszyklus für die Sanierung objektorientierter Systeme, der im Rahmen des europäischen Verbundprojektes

¹ engl.: reengineered

FAMOOS² entwickelt wurde und sich in großen industriellen Fallstudien bewährt hat. Innerhalb dieses Lebenszyklus identifizieren wir die wichtigsten Phasen und tragen zu deren Weiterentwicklung durch Methoden und Werkzeuge bei.

2 Der Lebenszyklus der Software Sanierung

Während es für die Konstruktion von Software bereits verschiedene Lebenszyklus-Modelle gibt, ist bislang kein solches Modell für die Software Sanierung bekannt. Ein solches Modell ist jedoch wichtig, um bestehende Verfahren vergleichen und bewerten zu können, um Lücken bei der methodischen Unterstützung aufzudecken, und um ein Rahmenwerk³ für die Entwicklung von notwendigen Werkzeugen bereitzustellen.

Der Lebenszyklus der Software Sanierung umfaßt die folgenden Phasen:

1. Anforderungsanalyse⁴: Es gibt verschiedene Ursachen für die Notwendigkeit einer Sanierungsmaßnahme (z.B. zur Flexibilisierung vor der Einführung neuer Funktionalität, zur Modularisierung vor der Portierung auf ein anderes System[Bau99] oder zur Verringerung der Übersetzungs- und Bindezeiten des Systems). Um in den folgenden Phasen nur die relevanten Teile der oftmals sehr großen Systeme untersuchen zu müssen, werden in dieser Phase die Anforderungen an das neue System aufgestellt.
2. Modellbildung⁵: Das zu sanierende System ist oft unzureichend dokumentiert und wird, da die ursprünglichen Entwickler oft nicht mehr verfügbar sind, nicht mehr verstanden. In dieser Phase wird ein grundlegendes Verständnis des Systems aufgebaut und in Form eines Modell des Systems dokumentiert.
3. Problemidentifikation⁶: In dieser Phase werden die Teile des Systems ermittelt, die die in der Anforderungsanalyse ermittelten Kriterien verletzen.

² <http://dis.sema.es/projects/FAMOOS>

³ engl.: Framework

⁴ engl.: Requirements analysis

⁵ engl.: Model capture, reverse engineering

⁶ engl.: Problem detection

4. Problemanalyse⁷: Nachdem die Problemstellen identifiziert sind, werden sie in dieser Phase untersucht und Zielstrukturen entworfen, die den Kriterien aus der Anforderungsanalyse genügen. Oft handelt es sich hierbei um Architektur- und Entwurfsmuster.
5. Transformation⁸: In dieser Phase wird schließlich die eigentliche Transformation der Systeme vorgenommen, indem die Problemstellen durch die entsprechenden flexiblen Zielstrukturen ersetzt werden.
6. Konsolidierung⁹: Handelt es sich bei dem zu sanierenden System um ein Rahmenwerk oder eine Bibliothek, so werden in dieser Phase die Systeme angepaßt, die das sanierte System nutzen. Greift das zu sanierende System auf persistente Daten zu, so muß die Datenbank einer Schemaevolution unterzogen werden.

In Abbildung 1 ist dieser Lebenszyklus dargestellt. Insbesondere zeigt die Abbildung, daß die Phasen der Problemidentifizierung und der Problemanalyse nicht direkt auf dem Quellcode sondern auf einem Modell des Quellcodes durchgeführt werden.

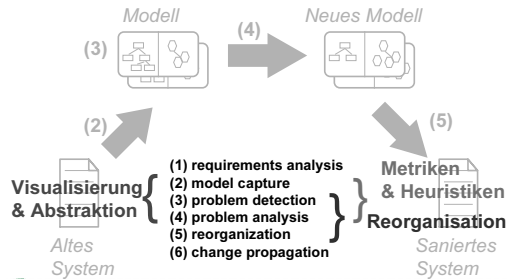


Abbildung1. Lebenszyklus der Software Sanierung

Bei der Evaluierung unseres Lebenszyklus an Systemen aus der industriellen Praxis (z.B. mehr als 5 Millionen Codezeilen umfassende Telekommunikationssoftware [Rit98]) stellte sich heraus, daß Sanierungsprojekte von einer methodischen Weiterentwicklung in den folgenden Gebieten stark profitieren würde:

- Metriken und Heuristiken

⁷ engl.: Problem analysis

⁸ engl.: Reorganisation

⁹ engl.: Change propagation

- Visualisierung und Abstraktion
- Reorganisation

Diese Gebiete wurden daher zum Schwerpunkt unserer Arbeiten. Da eine Methode in der industriellen Praxis nur dann einsetzbar ist, wenn sie durch Werkzeuge unterstützt wird, widmeten wir uns nicht nur methodischen Fragestellungen zu, sondern entwickelten auch eine Reihe von Werkzeugprototypen. Die folgenden Abschnitte sind den einzelnen Schwerpunkten gewidmet.

2.1 Metriken und Heuristiken

“Man kann nicht kontrollieren, was man nicht messen kann.”¹⁰ Diese Aussage macht klar, warum man messen muß, wenn man Prozesse, Produkte oder Ressourcen verbessern will. Wir setzen Metriken ein, um Problemstellen in Systemen zu identifizieren und den Erfolg der Sanierungsmaßnahme zu kontrollieren. Da die gängigen Metriken und Metrik-Werkzeuge sich auf prozedurale Metriken konzentrieren, haben wir nicht nur bestehende objektorientierte Metriken implementiert und evaluiert [Mar97], sondern entwickeln ein Rahmenwerk neuer, objektorientierter Metriken.

Entwurfsheuristiken (z.B. “Eine Oberklasse soll unabhängig von ihren Unterklassen implementiert sein.”) sind Regeln für einen guten Entwurf, die sich in einer Vielzahl von Fällen bewährt haben. Verstöße gegen diese Heuristiken sind damit potentielle Problemstellen des Systems. Wir haben Methoden zur Formalisierung von Designheuristiken entwickelt, die es uns ermöglicht haben, Verstöße gegen diese Regeln werkzeugunterstützt zu lokalisieren [BC98].

2.2 Visualisierung und Abstraktion

“Ein Bild sagt mehr als tausend Worte!”, diese Weisheit gilt auch für die Analyse objektorientierter Systeme. Oft genügt schon ein Blick, um Stellen zu identifizieren, die die Modularität eines Systems zerstören oder Funktionalität einer Bibliothek verwenden, die ausgetauscht werden soll. Wir haben daher sowohl ein mathematisches, auf Graphen basierendes Modell objektorientierter Systeme entwickelt, als auch Werk-

¹⁰ “You cannot control what you cannot measure.” [DeM82]

zeuge implementiert, die diese Modelle darstellen können[Ciu97]. Die zu sanierenden Systeme

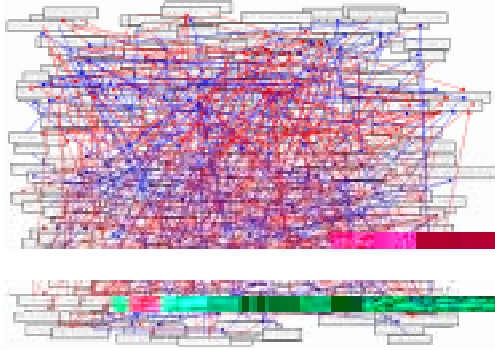


Abbildung 2. Visualisierung vor der Abstraktion

sind häufig so groß, daß eine Darstellung aller Entitäten (Klassen, Methoden, Attribute) nicht zum Verständnis beiträgt (vgl. Abbildung 2). Daher haben wir Mechanismen entwickelt, die durch Filterung und Abstraktion die Graphen auf die zum Verständnis wesentlichen Elemente reduzieren (vgl. Abbildung 3).

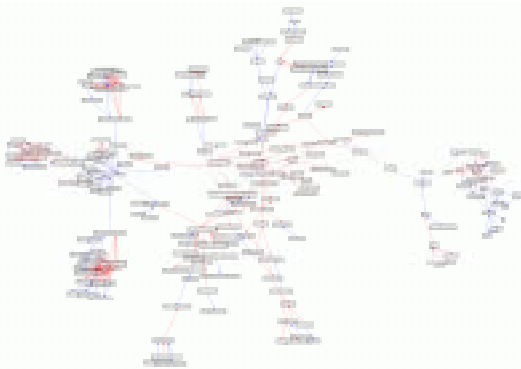


Abbildung 3. Visualisierung nach der Abstraktion

2.3 Reorganisation

Die Transformation einer Software unter Erhaltung syntaktischer semantischer Korrektheit ist eine schwierige, fehleranfällige und langwierige Aufgabe, sofern sie "von Hand" durchzuführen ist. Hinzu kommt oft noch die Anforderung, daß die Transformation auf der einen Seite zwar nicht-funktionale Eigenschaften des

Systems verbessern soll, auf der anderen Seite die eigentliche Funktionalität jedoch unberührt bleiben soll. Diese Aufgabe ist ohne Unterstützung durch Werkzeuge nicht mehr zu bewältigen[Moh98].

Wir haben daher eine Methode entwickelt, die es gestattet auf einem hohen Abstraktionsniveau Transformationen (bspw. Anwendung eines Entwurfsmusters auf ein bestehendes System[SGMZ98]) zu spezifizieren und durch ein Werkzeug durchführen zu lassen. Wir stützen uns dabei auf die Technik der statischen und dynamischen Metaprogrammierung.

3 Zusammenfassung

Die Sanierung von Systemen, die nicht mehr wirtschaftlich wartbar und erweiterbar sind, ist eine Disziplin der Softwaretechnik, deren Beherrschung eine immer zentralere Rolle in der Industrie einnimmt. Es besteht also die Notwendigkeit, die durch die Sanierungsproblematik aufgeworfenen wissenschaftlichen und technischen Probleme zu lösen und diese Lösung für die Industrie in Form von Methoden und Werkzeugen aufzubereiten.

Dieses Papier beschreibt Methoden und Werkzeuge für die zentralen Probleme der Sanierung objektorientierter Systeme und ordnet diese verschiedenen Phasen eines Lebenszyklus zur Software Sanierung, den erfolgreiche Sanierungsprojekte aufweisen müssen, zu. Der Lebenszyklus sowie die beschriebenen Methoden und Werkzeuge konnten sich in der industriellen Praxis schon bewähren.

Literatur

- [Bau99] Markus Bauer. Reengineering von smalltalk nach java. Master's thesis, Forschungszentrum Informatik (FZI) an der Universität Karlsruhe (TH), 1999. Supervised by T. Genßler and B. Schulz.
- [BC98] H. Bär and O. Ciupke. Exploiting design heuristics for automatic problem detection. In Stéphane Ducasse and Joachim Weisbrod, editors, *Proceedings of the ECOOP Workshop on Experiences in Object-Oriented Re-Engineering*, number 6/7/98 in FZI Report, June 1998.
- [Cas98] Eduardo Casais. Re-engineering object-oriented legacy systems. *Journal of*

- Object-Oriented Programming*, pages 45–52, January 1998.
- [Ciu97] Oliver Ciupke. Analysis of object-oriented programs using graphs. In Jan Bosch and Stuart Mitchell, editors, *Object-Oriented Technology - Ecoop'97 Workshop Reader*, volume 1357 of *Lecture Notes in Computer Science*, pages 270–271, Jyväskylä, Finland, March 1997. Springer.
- [DeM82] T. DeMarco. *Controlling Software Projects: Management, Measurement and Estimation*. Yourdan Press New Jersey, 1982.
- [Mar97] Radu Marinescu. The use of software metrics in the design of object oriented systems. Master's thesis, Universitatea Politehnica din Timișoara, 1997.
- [Moh98] B. Mohr. Reorganisation objektorientierter Systeme. Masters thesis, Forschungszentrum Informatik (FZI) an der Universität Karlsruhe (TH), March 1998. Supervised by B. Schulz.
- [Rit98] Fabian Ritzmann. Reverse Engineering of Large Scale Software Systems. Diplomarbeit, Universität Karlsruhe, June 1998. Supervised by O. Ciupke.
- [SGMZ98] B. Schulz, T. Genßler, B. Mohr, and W. Zimmer. On the computer aided introduction of design patterns into object-oriented systems. In *27th Conference on Technology of Object-Oriented Languages and Systems (TOOLS)*. IEEE, 1998.