Erfahrungen mit Refactorings in LabVIEW

Kay Schützler, Sebastian Huß
Humboldt-Universität zu Berlin
{schuetzl, huss}@informatik.hu-berlin.de

24. März 2005

1 Einleitung

Das Konzept hinter Refactoring, mit verhaltenserhaltenden Transformationen auf einer Abstraktionsebene strukturelle Verbesserungen zu erzielen, ist universell und nicht auf ein bestimmtes Programmierparadigma oder eine bestimmte Programmiersprache beschränkt. In einer Übersicht in [3] findet sich eine Reihe von Referenzen für Refactoring-Untersuchungen an verschiedensten Paradigmen und Sprachen. Darin nicht vertreten sind grafische Programmiersprachen. Es wird jedoch der Trend festgestellt, Refactoring auch auf das Design-Level - also auf grafische Elemente wie beispielsweise Klassen- oder Zustandsdiagramme - auszuweiten. In diesem Artikel werden die Erfahrungen mit Refactoringaktivitäten in einer grafischen Programmiersprache beschrieben.

2 Das Ausgangsprojekt

Im Rahmen eines Reengineering-Projekts wurde ein hardwaresteuerndes Softwaresystem aus einer prozeduralen Umgebung (Turbo Pascal) in eine grafische Programmiersprache (die Sprache G der Entwicklungsumgebung LabVIEW [4]) überführt. Effektiv handelte es sich um eine Re-Implementation unter Beibehaltung der wichtigsten bisherigen Funktionalität. Mithin wurden also Regressionstestfälle definiert, die das Verhalten des Systems beschrieben.

Da mit der neuen Programmierumgebung seitens der Entwickler nur geringe Erfahrungen vorlagen, musste davon ausgegangen werden, dass die mit kurzfristiger Terminsetzung erfolgte erste Umsetzung in LabVIEW über ein verbesserungswürdiges Design verfügen würde. Ein weiterer Faktor, der die erste Version der neuen Software ungünstig beeinflusste, war die Vorgehensweise bei der Übernahme der erhaltenswerten Teile der Funktionalität. Im Sinne eines schnellen und stabilen Ergebnisses wurde der entsprechende Teil der Originalsoftware Stück für Stück nach LabVIEW übertragen. Es wurden dabei auch Design-Entscheidungen des Originals übernommen, die bereits für das Original ungünstig waren oder nicht vollständig der datenflussorientierten Strukturierung der neuen Entwicklungsumgebung entsprachen.

Anlass zu kontinuierlichen Restrukturierungen gaben auch die Änderungen an der Funktionalität der bearbeiteten Software. Dabei war klar, dass nur ein systematisches Vorgehen erfolgversprechend sein würde. Das Konzept des Refactoring bot sich an, da es zum einen methodische Vorgehensweisen beinhaltet, zum anderen aber je nach Bedarf kurzfristig einsetzbar ist.

3 Die Refactorings

Vorgehensweise

Als Ausgangspunkt für die Refactorings im betrachteten Projekt dienten die von Fowler in [1] beschriebenen Bad Smells. Es wurde untersucht, inwiefern die ihnen zugrundeliegenden Eigenschaften auf eine datenflussorientierte grafische Programmiersprache übertragbar sind.

Erwartungsgemäß konnten dabei einige typische Bad Smells wie *Duplicated Code* und *Long Method* direkt oder mit einfacher Anpassung an die Sprachmittel der Zielumgebung übernommen werden; bei anderen waren größere Anpassungen nötig.

Eine Stärke von Fowlers Werk ist die Zuordnung empfehlenswerter Refactorings zu den einzelnen Bad Smells. So konnte anhand der übertragenen Bad Smells eine Auswahl der zu übertragenden Refactorings motiviert werden.

Dokumentation der Refactorings

Die Bad Smells und speziell die Refactorings wurden analog zu der Vorgehensweise in [1] dokumentiert. Die Bad Smells wurden verbal charakterisiert und um Empfehlungen ergänzt, mit welchen Refactorings ihnen zu begegnen ist.

Die Beschreibung einzelner Refactorings setzt sich aus jeweils zwei bis drei Abschnitten zusammen. In einem Motivationsabschnitt werden typische Situationen diskutiert, die das beschriebene Refactoring notwendig machen. Im anschließenden Abschnitt werden Mechanismen zur Umsetzung des Refactorings präsentiert. Dies wird nach Bedarf direkt durch ein Beispiel unterstützt. Der optionale dritte Abschnitt erläutert mögliche Schwierigkeiten, Wechselwirkungen und Besonderheiten der einzusetzenden Mechanismen.

Refactoring-Beispiel

In diesem Abschnitt wird eines der umgesetzten Refactorings vorgestellt. Es handelt sich dabei um die LabVIEW-Entsprechung des wohl häufigsten Refactorings: *Extract Method*. Da in LabVIEW das einer

Methode entsprechende Konzept als SubVI[1] bezeichnet wird, wurde auch das Refactoring entsprechend in *Extract SubVI* umbenannt. Die nachfolgenden Abbildungen veranschaulichen das notwendige Vorgehen.

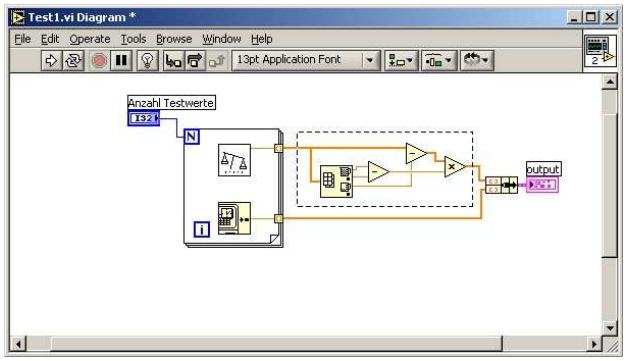

Abbildung 1: Den zu extrahierenden Code markieren...

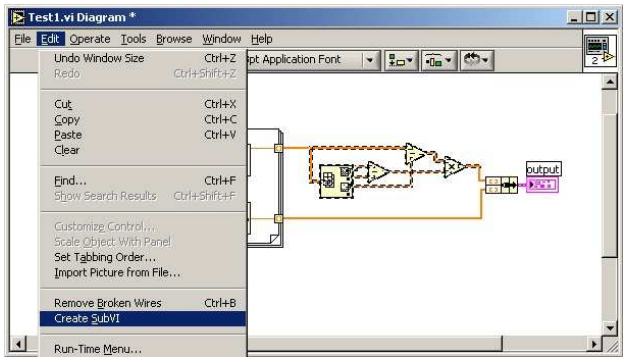

Abbildung 2: ... und SubVI erzeugen

Die eingesetzte Entwicklungsumgebung besitzt bereits einen Mechanismus zur Erzeugung von methodenähnlichen Strukturen aus markierten Code-Abschnitten. Dieser Mechanismus stellt unter Anderem sicher, dass das Programm ausführbar bleibt.

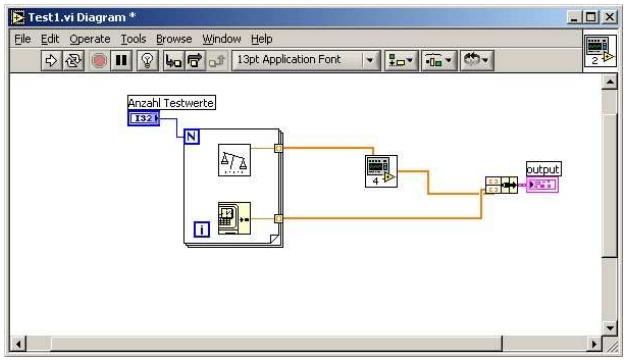

Abbildung 3: Zustand nach Erstellung des SubVI

[1] In LabVIEW werden Komponenten als Virtual Instruments (VIs) mit Ein- und Ausgängen (Connectoren) betrachtet. Ihr Verhalten wird in datenflussorientierten Blockdiagrammen beschrieben. Ein VI kann andere VIs als SubVIs enthalten.

In Abbildung 3 zeigen sich jedoch die Grenzen des eingebauten Automatismus: Ein nicht sehr aussagekräftiges Standard-Icon wurde erzeugt, das es zu überarbeiten gilt. Zu dieser Überarbeitung gehört die Vergabe aussagekräftiger Namen für das SubVI und seine Ein- und Ausgänge sowie die Erstellung einer Beschreibung und eines geeigneten Icons.

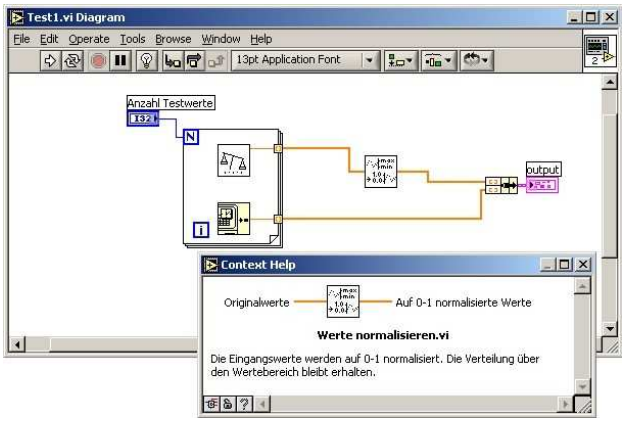

Abbildung 4: Das SubVI mit aussagekräftigem Icon, Beschreibung und Connectoren

4 Zusammenfassung

In [2] werden zehn Bad Smells und 17 Refactorings in LabVIEW dokumentiert. Die beschriebenen Bad Smells in LabVIEW haben ihren Ursprung in zehn analog von Fowler beschriebenen Bad Smells. 14 der Refactorings entstanden durch die Übertragung von elf objektorientierten Refactorings aus [1]. Die drei übrigen Refactorings wurden aus den Restrukturierungserfahrungen im erwähnten Projekt abgeleitet und sind spezifisch für die Software-Entwicklung mit datenflussorientierten grafischen Programmiersprachen.

Dass viele Refactorings aus [1] nicht übertragen wurden, liegt zum einen an der zeitlich motivierten Beschränkung auf tatsächlich angewendete Refactorings und zum anderen daran, dass mehrere dieser Refactorings speziell objektorientierte Bedürfnisse adressieren und ihre Übertragung daher nicht sinnvoll ist.

Literatur

[1] Fowler, M.: *Refactoring - Improving the Design of Existing Code.* Addison-Wesley, 1999.

[2] Huß, S.: *Refactoring in LabVIEW.* Diplomarbeit an der Humboldt-Universität zu Berlin, 2005.

[3] Mens, T., Tourwé, T.: *A Survey of Software Refactoring.* In: IEEE Transactions on Software Engineering, Vol. 30, No. 2, Februar 2004, Seiten 126-139.

[4] National Instruments: *LabVIEW-Homepage.* http://www.ni.com/labview/.