Entwicklung von Softwarevarianten mit FeatureIDE

One Size fits all?

One size fits all? In der Softwareentwicklung trifft dieses Motto nur selten zu. Die Entwicklung von Programmen, die alle erdenklichen Funktionen beinhalten, ist oft nicht die beste Lösung. Solche Programme beinhalten viele unnötige Fehlerquellen, sind übermäßig aufgebläht und tragen regelmäßig zur Verwirrung des Benutzers bei. Während den meisten Anwendern wenigstens ein paar Vertreter dieser Gattung von eierlegenden Wollmilchsäuen bekannt sein dürften, sind in vielen Anwendungsbereichen solche Programme nicht einmal vorstellbar. Eingebettete Systeme, Smartphones oder der Automobilbereich sind bekannte Beispiele solcher Domänen. Die meisten Softwareprodukte werden zumindest in einer Reihe von Varianten entwickelt. Der Kunde kann sich dann diejenige Variante aussuchen, die ihm am meisten zusagt oder die er sich leisten kann.

von Thomas Thüm und Fabian Benduhn

Eine gängige Technik, um Softwarevarianten zu entwickeln, ist die Verwendung von Versionsverwaltungstools wie SVN, Git oder Mercurial. Dabei wird jede Variante in einem eigenen Branch entwickelt. Allerdings haben viele Varianten einen nicht unerheblichen Anteil an gleichem Quelltext und unterscheiden sich nur in bestimmten Aspekten. Beim Erstellen einer neuen Variante wird oft der gesamte Quelltext einer existierenden Variante in einen neuen Branch kopiert. Danach kann der Quelltext verändert oder erweitert werden. Durch die Verwendung von Versionsverwaltungstools für die Entwicklung von Softwarevarianten ergeben sich einige Nachteile:

- Das Finden und Beheben von Fehlern in einer Variante zieht oft identische Änderungen in anderen Varianten nach sich, das hat einen erhöhten Arbeitsaufwand zur Folge.
- Das Erstellen einer Variante ist komplex und teuer. Varianten können deshalb nicht auf Vorrat produziert werden, was zu langen Entwicklungszeiten führt.

Das Eclipse-Plugin FeatureIDE bietet eine neuartige Möglichkeit für die Entwicklung von Softwarevarianten, die diese Probleme vermeiden. FeatureIDE erlaubt eine komfortable und benutzerfreundliche Entwicklung von Softwarevarianten und erlaubt es dabei, Quelltext wiederzuverwenden, der in mehreren Varianten benötigt wird.

FeatureIDE am Beispiel

Die wesentliche Idee von FeatureIDE besteht darin, jede Funktionalität (Feature) separat zu entwickeln. Hinterher können durch Kombination dieser Features unterschiedliche Varianten automatisch erzeugt werden. Wir zeigen an einem kleinen Beispiel, einem Taschenrechner, wie das funktioniert.

Zuordnung von Quelltext zu Features

In FeatureIDE wird jede Anwendung in unterschiedliche Quelltextfragmente aufgeteilt, so genannte Features. Das mag erst einmal ähnlich zu dem bekannten Konzept von Klassen klingen, aber es gibt einen wichtigen Unterschied. Ein Feature, in unserem Sinne, beschreibt einen Aspekt der Software, die für den Benutzer eine Bedeutung hat, z. B. sind die Operationen *Addition* und *Multiplication* Features, da es für den Benutzer einen Unterschied macht, ob der Taschenrechner diese Funktionen unterstützt oder nicht.

Zu jedem Feature können beliebig viele Quelltextartefakte gehören, Artefakte sind Klassen, Methoden, Felder oder auch einzelne Anweisungen. Auch zusätzliche Dokumente, wie z. B. Grafiken oder Benutzerdokumentation, können einem Feature zugeordnet werden. Besonders die Möglichkeit, Methoden und Klassen nur teilweise zu verändern, bietet eine große Flexibilität beim

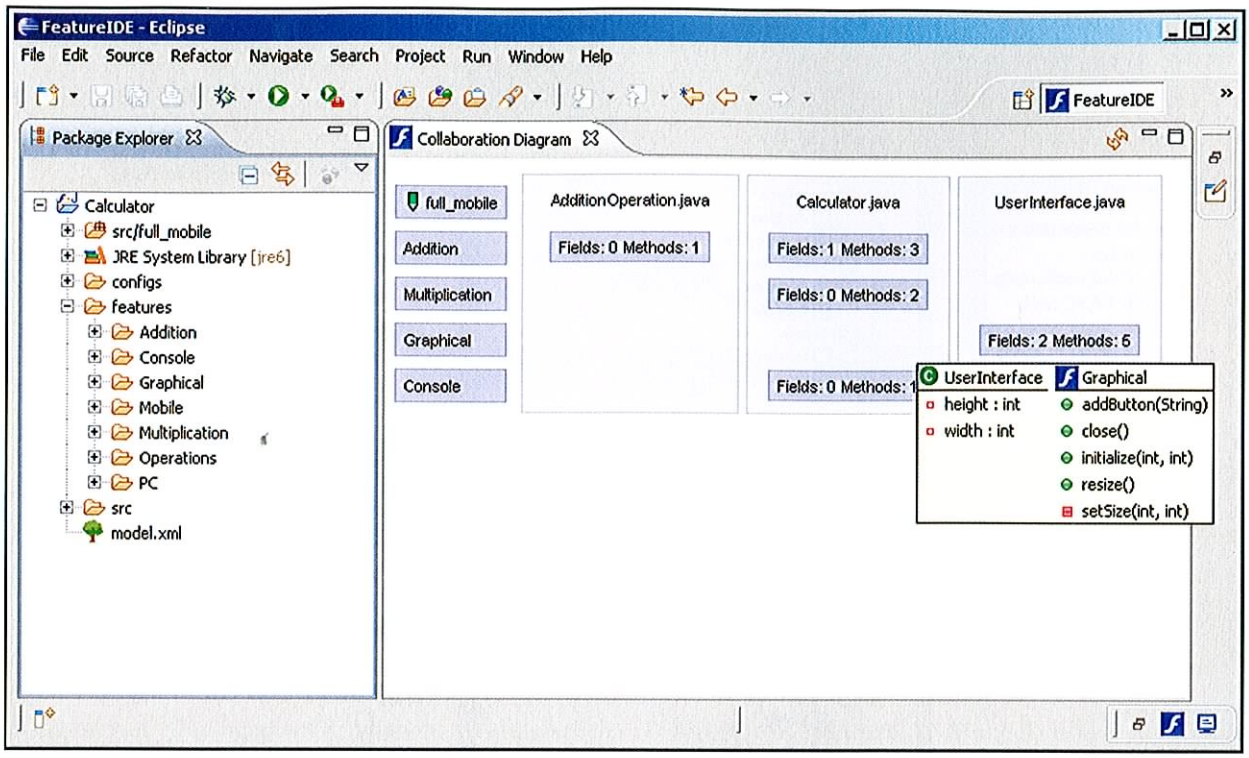

Abb. 1: Features der Beispielanwendung

Entwurf der Features. **Abbildung 1** zeigt die Features unserer Beispielanwendung mit den dazugehörigen Quelltextartefakten. Die Spalten beziehen sich auf Features und die Zeilen auf Klassen bzw. andere Dokumente. In den Zellen kann man genauer erkennen, welche Quelltextartefakte zu den jeweiligen Features gehören. Man erkennt dort etwa, dass das Feature *Addition* mehrere Methoden und ein Feld der Klasse *Calculator* enthält, *Multiplication* beinhaltet zwei Methoden und *Graphical* führt zusätzlich die Klasse *UserInterface* ein.

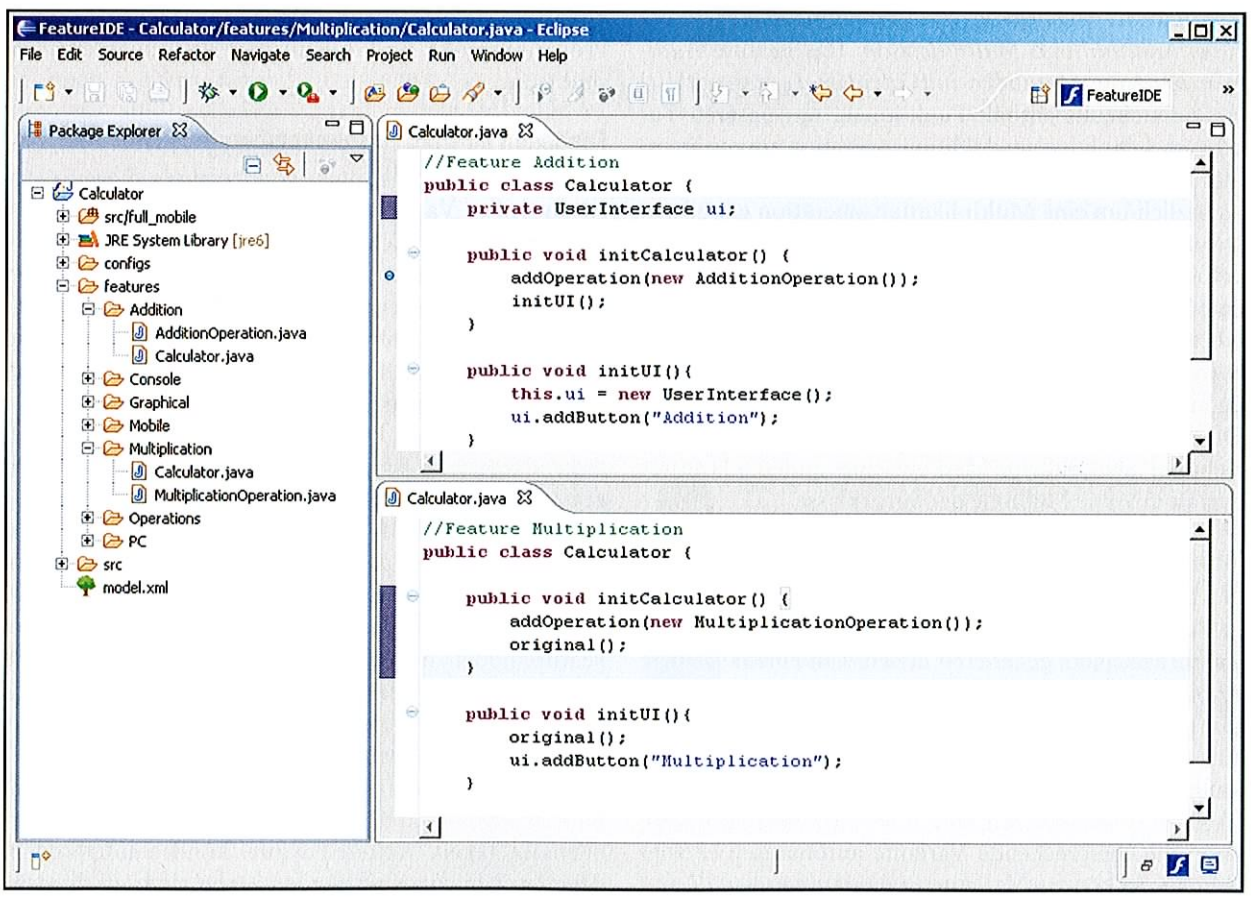

Abb. 2: Auszüge aus dem Quelltext

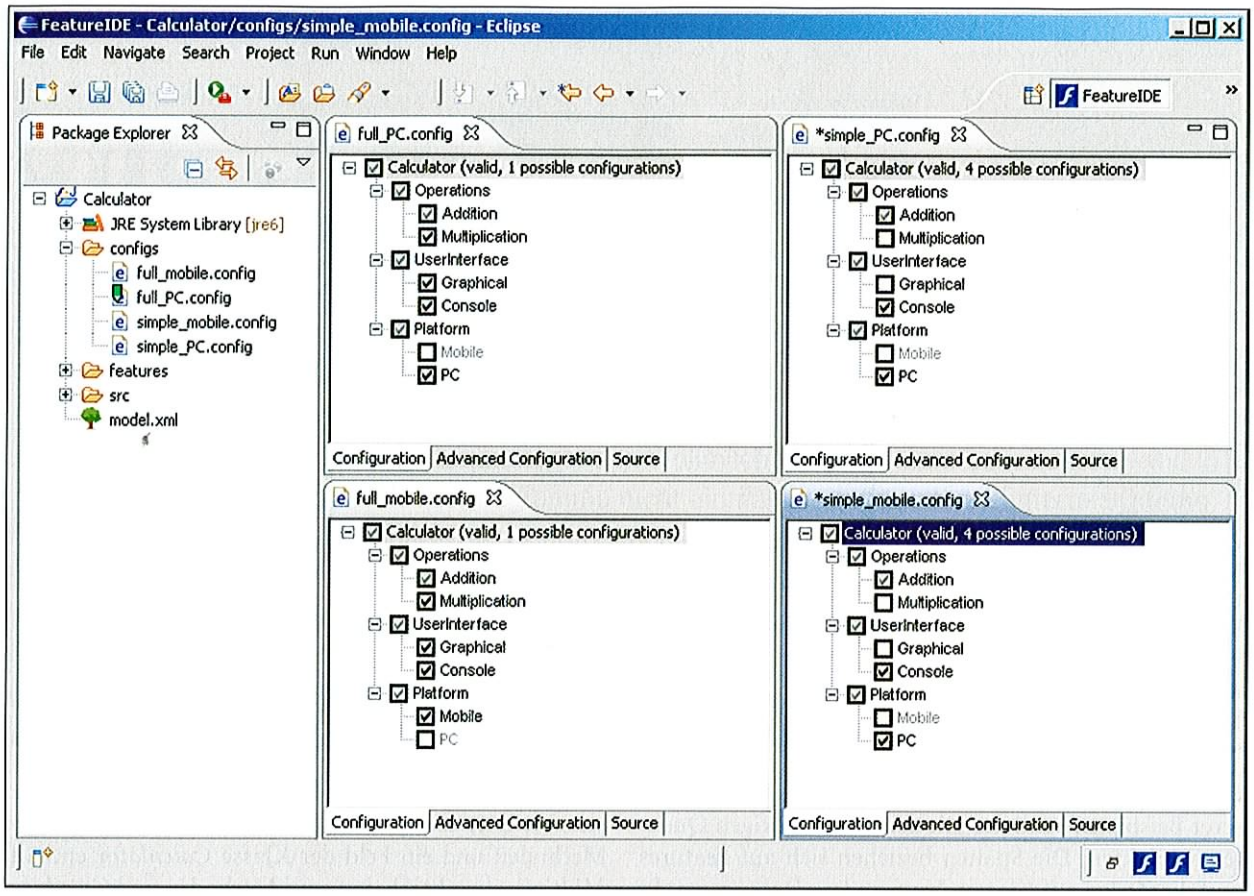

Abb. 3: Grafischer Editor

Implementierung der Features

Abbildung 2 zeigt Auszüge aus dem Quelltext der Features *Addition* und *Multiplication*. Das Feature *Addition* führt die Methoden *initCalculator()* und *initUI()* erstmals ein. Es enthält Funktionen, um unseren Taschenrechner mit einer Additionsoperation auszustatten. Das Feature *Multiplication* soll nun den Taschenrechner zusätzlich um eine Multiplikationsoperation erweitern. Dazu werden die Methoden *initCalculator()* und *initUI()* verfeinert. Damit die bestehende Funktionalität der beiden Methoden nicht verloren geht, wird das Schlüsselwort *original()* verwendet. Durch Aufruf von *original()* wird jeweils die im Feature *Addition* eingeführte Methode aufgerufen. Es funktioniert ähnlich wie das aus der Objektorientierung bekannte Schlüsselwort *super()*, bezieht sich aber auf eine Methode eines anderen Features anstatt auf eine Methode der Superklasse.

Varianten generieren

Nachdem wir uns um die Implementierung der Features gekümmert haben, können wir nun Varianten unseres Taschenrechners generieren lassen. Um eine lauffähige Applikation zu erhalten, wählen wir diejenigen Features aus, die wir in der Variante benötigen. Eine solche Auswahl von Features wird Konfiguration genannt und kann mithilfe eines grafischen Editors erstellt werden (**Abb. 3**). Nachdem wir eine Auswahl getroffen haben, kann die entsprechende Variante automatisch erzeugt werden. So können Varianten generiert werden, die auf funktionale Anforderungen, wie Addition und Multiplikation, sowie nicht funktionale Anforderungen, wie Programmgröße und Ausführungszeit, zugeschnitten sind.

Festlegen gültiger Featurekombinationen

Bestimmte Kombinationen von Features führen nicht zu sinnvollen Varianten, z. B. dürfen *Mobile* und *PC* in unserem Beispiel nicht zusammen ausgewählt werden, weil die Software entweder auf einem PC oder einem mobilen Gerät laufen kann. FeatureIDE unterstützt uns bei der Auswahl von Features, indem Optionen, die zu solchen ungültigen Konfigurationen führen, deaktiviert werden. Zum Beispiel wird, nachdem *Mobile* selektiert wurde, das Feature *PC* im Konfigurationseditor ausgegraut dargestellt, da es sich um alternative, d. h. sich gegenseitig ausschließende, Features handelt. Genauer gesagt bedeutet es, dass unsere Anwendung nicht beide Features in ein und derselben Variante erlaubt.

Um festzulegen, wie Features miteinander in Beziehung stehen, wird ein Featuremodell erstellt. Bei einem Featuremodell handelt es sich um eine hierarchische Darstellung von Features, die in FeatureIDE grafisch oder textuell bearbeitet werden kann (**Abb. 4**).

Das Wurzelfeature, also das oberste in der Hierarchie, muss in jeder Konfiguration ausgewählt sein. Es kann Basisfunktionalitäten enthalten, die alle Varianten gemeinsam haben. Weitere Features können auf verschiedene Arten hinzugefügt werden. Dabei sind verschiedene

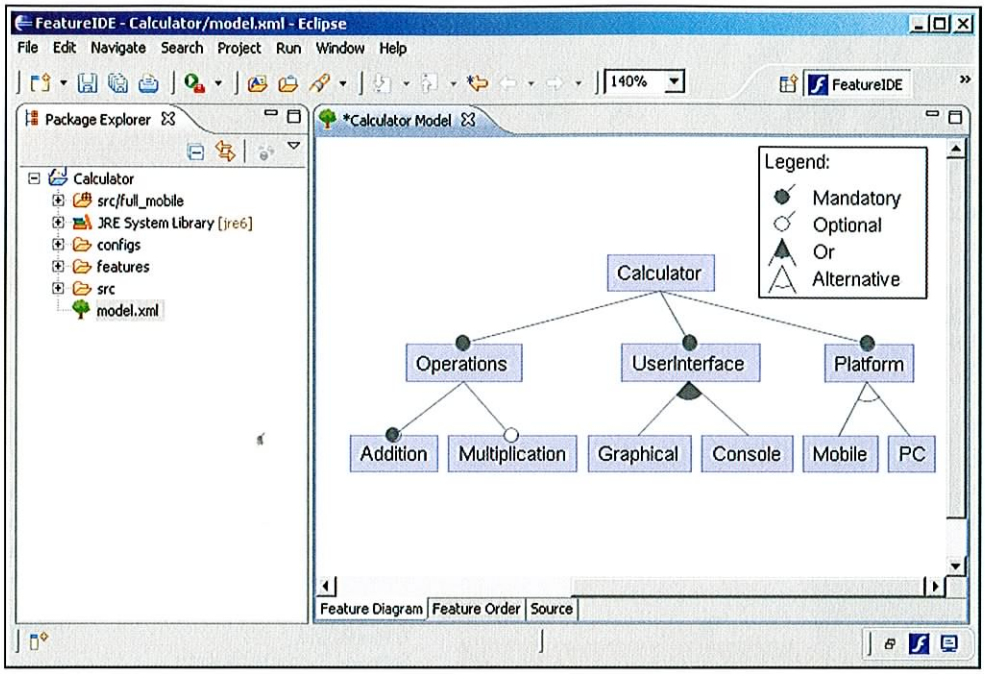

Abb. 4: Featuremodell

Formen von Abhängigkeiten möglich. Grundsätzlich wird eine Abhängigkeit zwischen zwei Features durch eine Verbindung im Featuremodell dargestellt. Das untergeordnete Feature kann nur selektiert werden, falls das darüber liegende Feature ebenfalls ausgewählt ist. So kann zum Beispiel *Multiplication* nur ausgewählt werden, falls *Operations* ausgewählt ist. Ein Feature kann mehrere unterliegende Features besitzen, aber umgekehrt höchstens ein darüber liegendes Feature.

Darüber hinaus gibt es noch weitere Beziehungen, die zwischen Features auftreten können. Zum Beispiel ist das Feature *Multiplication* optional und kann auch weggelassen werden. Die Features *Mobile* und *PC* sind Alternativen und können nicht zusammen ausgewählt werden. Die Features *Graphical* und *Console* befinden sich in einer Oder-Gruppe, d. h. mindestens eine der beiden muss in jeder Konfiguration ausgewählt werden.

Das Featuremodell hilft außerdem, die Features übersichtlich anzuordnen und zu strukturieren. So kann man z. B. leicht erkennen, dass es sich bei *Addition* und *Multiplication* um Operationen handelt. Besonders bei größeren Projekten kann das sehr nützlich sein.

Hintergrund

FeatureIDE wird als Open-Source-Werkzeug in einem akademischen Umfeld entwickelt und finanziell von der Metop GmbH unterstützt. Es unterstützt eine Vielzahl an Programmiersprachen, darunter Java, C, C++, C#, Haskell, XML und JavaCC. Die in diesem Artikel präsentierte Technik wird Feature-oriented-Programming genannt. Zusätzlich werden weitere Techniken unterstützt, die auf Präprozessoren (Antenna, Munge), aspektorientierter Programmierung (AspectJ) und Delta-orientierter Programmierung (DeltaJ) basieren. FeatureIDE wird ständig erweitert und verbessert, um die Entwicklung von Softwarevarianten weiter zu vereinfachen.

Zusammenfassung

FeatureIDE ermöglicht es, Softwarevarianten effizient zu entwickeln. Ob für die Entwicklung einiger Varianten einer Software oder für eine ausgewachsene Produktlinie – die nahtlose Einbindung in Eclipse macht den Einstieg leicht. Viele interessante Beispielprojekte und ein Tutorial helfen außerdem bei den ersten Schritten. FeatureIDE ist als Open-Source-Eclipse-Plug-in verfügbar und über den Eclipse Marketplace oder die FeatureIDE-Webseite erhältlich [1].

FeatureIDE wird als Open-Source-Werkzeug im akademischen Umfeld entwickelt.

Thomas Thüm ist seit 2010 Doktorand an der Otto-von-Guericke Universität in Magdeburg und Projektleiter von FeatureIDE. Er hat zusammen mit zwei anderen Studenten im Jahr 2007 einen Prototyp von FeatureIDE für die Lehre in Magdeburg entwickelt, der ständig erweitert wurde und inzwischen weltweit in Forschung und Lehre eingesetzt wird.

Fabian Benduhn (fabian.benduhn@st.ovgu.de) studiert Informatik an der Otto-von-Guericke-Universität Magdeburg und ist als studentische Hilfskraft in der Arbeitsgruppe Datenbanken am Institut für Technische und Betriebliche Informationssysteme tätig. Er ist seit 2010 Mitglied des FeatureIDE-Entwicklungsteams.

Links & Literatur

[1] http://www.fosd.net/featureide/