14 Goto AspectJ

Wer nicht über die Zukunft nachdenkt, wird keine haben.
(John Galsworthy, englischer Literaturnobelpreisträger)

Geschmack auf AspectJ bekommen, aber Ihr Chef macht nicht mit? Nun, da gibt es ein paar Möglichkeiten, damit umzugehen:

- Überzeugungsarbeit leisten
- Don't tell the manager

Im ersten Fall benötigen Sie gute Argumente für Ihr Vorhaben, im zweiten Fall benötigen Sie gute Argumente, falls Sie sich rechtfertigen müssen.

14.1 Gründe für AspectJ

Im Einführungsbeispiel in Kapitel 1.2 auf Seite 7 haben wir die Grenzen der Objektorientierung gesehen. Nicht alle Probleme lassen sich mit AspectJ lösen, aber doch einige. Vor allem winken bei einem behutsamen Einsatz von AOP folgende *Vorteile*:

Vorteile

- weniger Code (und damit weniger Wartungsaufwand)
- höhere Produktivität (nach einer Lernphase)
- komplexere Aufgaben können bewältigt werden
- höhere Modularität durch Herausziehen der Querschnittsbelange möglich (»Separation of Concerns«)
- Anpassung von Fremd-Software möglich

Allerdings gibt es diese Vorteile nicht zum Nulltarif, und AOP hat auch einige *Nachteile*, deren Sie sich bewusst sein sollten:

Nachteile

- Lernaufwand
- noch keine grafische UML-Notation verfügbar
- fehlende Erfahrung
- fehlende Tool-Unterstützung (z. B. für Refactoring)

14.1.1 Erwarteter Nutzen

Weniger Wartung

Mit AspectJ haben Sie das notwendige Werkzeug in der Hand, um übergreifende Dinge (»Crosscutting Concerns«) wie Logging, Persistenz, Fehlerbehandlung, ... auch übergreifend behandeln zu können. Dadurch können Sie sich in Ihrem Java-Code auf die wichtigen Bereiche wie die Business-Logik konzentrieren. Bei richtigem Einsatz wird Ihr Code übersichtlicher und wartungsfreundlicher.

Wiederverwendung

Ein großes Versprechen der Objektorientierung, das sich im Nachhinein leider als zu optimistisch herausgestellt hat, war die Wiederverwendung von Code. Trotz aller Ernüchterung, die sich inzwischen eingestellt hat, darf nicht vergessen werden, dass im Vergleich zur prozeduralen (oder auch funktionalen) Programmierung doch ein Fortschritt stattgefunden hat. Mit AOP und AspectJ wird sich dieser Trend fortsetzen. Denkbar sind hier Plug n' Play-Aspekte, die in Form von Bibliotheken bereitgestellt werden.

Neue Patterns

Auch im Bereich Entwurfsmuster (engl.: Patterns) werden sich bestehende Muster effektiver implementieren lassen und neue herausbilden, die durch den Einsatz von Aspekten erst möglich sind.

Erfahrungen[1] zeigen, dass sich durch AspectJ der Code-Umfang gegenüber konventioneller OO-Programmierung reduzieren lässt. Auch die Entwicklungszeit kann vermindert werden, nachdem die Lernphase überwunden ist.

Klare Trennung

Der Hauptvorteil aspektorientierter Programme liegt aber hauptsächlich in der klaren Trennung der einzelnen »Concerns«. Auf der Webseite zum Buch finden Sie als Beispiel ein einfaches Schiffe-Versenken-Programm – einmal konventionell in Java und einmal aspektorientiert mit AspectJ kodiert. Vorteil der zweiten Variante ist weniger der geringere Umfang (ca. 15%), als vielmehr das Herausziehen des (moderaten) Logging- und Persistenz-Anteils in zwei Aspekte. Damit werden Änderungen überschaubarer, und auch der Einsatz in anderen Projekten wird durch die lose Kopplung mit den Querschnittsbelangen nicht verbaut.

14.1.2 Performance

Meistens werden dieselben Gründe gegen die Einführung einer neuen Technologie genannt. *»Das haben wir schon immer so gemacht!«* ist ein beliebter Satz zur Verteidigung bestehender Vorgehensweisen. Der zweite Grund, der gern vorgebracht wird, ist die schlechtere Performance. Während man gegen den ersten Einwand erfahrungsgemäß einen schweren Stand hat, trifft der zweite Punkt nur bedingt zu.

[1] Quelle: `http://www.jugs.de/protokolle2003.html`

Es stimmt zwar, dass durch das Einweben von Aspekten zusätzlicher Code und Methodenaufrufe die Class-Dateien aufblähen. Aber dieser Code müsste ohne AspectJ ohnehin selbst geschrieben werden. Im Gegenteil kann AspectJ die Performance dadurch verbessern, dass es viele Flexibilitätslayer (wie z. B. die Setter- und Getter-Methoden) überflüssig macht und den Einsatz von Reflexion (von James Gosling auf der OOP 2003[2] als »... *Bug No. 1 in Java*« bezeichnet) reduziert.

Performance-Probleme sind Architekturprobleme

Vielleicht noch ein abschließender Hinweis zu diesem Thema: Performance-Probleme sind Architekturprobleme (nicht immer, aber meistens). Da hilft es auch nichts, wenn Sie Ihr Programm in Assembler schreiben.

14.1.3 Gefahren

Die größte Gefahr bei neuen Technologien ist ein übertriebener Einsatz um jeden Preis. Wenn man nur einen Hammer in seinem Werkzeugkasten hat, sieht jedes Problem wie ein Nagel aus, auch wenn es sich um eine Schraube oder Wäscheklammer handelt. Und ähnlich ist es auch mit AspectJ. Probleme, die sich besser mit Java lösen lassen, sollten auch mit Java gelöst werden.

Ein übertriebener AspectJ-Einsatz birgt die Gefahr ausufernder Aspekte, deren Abhängigkeit nicht mehr überblickt werden kann. Änderungen an derart fragilen Systemen werden so zum Alptraum. Hüten Sie sich trotz aller Begeisterung vor übermäßigem Einsatz. Eine Überdosis Aspekte kann zum Tod jedes Projekts führen, während die richtige Dosierung die Entwicklung beschleunigen kann.

14.1.4 Erfahrungen

Erfahrungen von größeren Projekten gibt es noch kaum, aber die vorliegenden Untersuchungen und Berichte lassen darauf schließen, dass bei behutsamem Einsatz von AOP und AspectJ sich die versprochenen Vorteile tatsächlich einstellen. So wurde auf einem Vortrag bei der JUGS[3] berichtet, dass durch Modularisierung redundanter Code mit Hilfe von Aspekten (abhängig vom Einsatzszenario) um 30–95% reduziert werden konnte.[4]

Auch bei der Entwicklungszeit sind Einsparungen möglich. Alternativ kann die gesparte Zeit auch zur Erhöhung der Code-Qualität (und damit einer Reduzierung des späteren Wartungsaufwands) eingesetzt

[2] Die OOP-Veranstaltung findet jährlich in München statt, s. a. `http://www.sigs-datacom.de/sd/kongresse/oop_2003/index.htm`

[3] Java User Group Stuttgart, `http://www.jugs.de/`

[4] Quelle: `http://www.jugs.de/protokolle2003.html#300103`

werden. Nichtfunktionaler Code wie das lästige Exception-Handling oder Tracing lässt sich mit AspectJ mit deutlich weniger Aufwand erstellen.

14.2 Einführungsstrategien

Welche Möglichkeiten gibt es, eine Firma zu ruinieren?

- ❏ durch Aktienspekulation – das ist die schnellste;
- ❏ durch Frauen – das ist die schönste;
- ❏ durch Outsourcing – das ist die sicherste.

Genauso gibt es auch bei der Einführung von AspectJ verschiedene Möglichkeiten, hier vorzugehen. Fangen wir mit der schwierigsten an, die Ihnen aber die Aufmerksamkeit Ihrer Vorgesetzten sichert.

14.2.1 Top-down (Unterstützung des Managements)

Je mehr Top, desto Flop.

Am schwierigsten ist es, die Unterstützung des Managements für die Einführung von AOP und AspectJ zu erlangen. Während Sie vermutlich Ihren Chef noch davon überzeugen können bzw. abschätzen können, ob ein Vorstoß in diese Richtung erfolgversprechend ist, wird die Aussicht auf Erfolg immer ungewisser (und leider auch unwahrscheinlicher), je höher Sie in der Hierarchie aufsteigen müssen, um eine Bewilligung zu erhalten.

Rechnen Sie mit dem Schlimmsten.

Sinnvoll ist es, sich dazu ein kleineres und unkritisches Projekt auszusuchen. Erfahrungen bei der Einführung von OO-Techniken haben gezeigt, dass die ersten OO-Projekte Fehlschläge waren. Hauptgründe waren meist:

- ❏ Lernkurve wurde unterschätzt
- ❏ fehlende Begleitung von erfahrenen Experten
- ❏ überzogene Erwartungen

Das klingt nicht gerade ermutigend, zumal der Umstieg von OO auf AOP vergleichbar ist mit dem Umstieg vom prozeduralen Ansatz auf die Objektorientierung. Aber:

- ❏ Der Aufwand wird nicht kleiner!
- ❏ AOP wird kommen!
- ❏ Wenn nicht jetzt umsteigen, wann dann?

Rechnen Sie damit, dass das erste Projekt zum Fehlschlag werden könnte (deswegen auch der Rat, mit einem unkritischen Projekt anzufangen).

Aber es wird nicht vergebens sein – die gemachten Fehler und gewonnenen Erfahrungen werden die Erfolgschancen künftiger AOP-Projekte beträchtlich erhöhen, wenn es Ihnen gelingt, weitere Mitstreiter zu gewinnen.

Müssen Sie Wochen- oder Monatsberichte schreiben? Nutzen Sie diese Gelegenheit, um sich indirekt die Unterstützung des Managements zu sichern. Erwähnen Sie beiläufig, dass Sie zur Lösung des Problems X den AspectJ-Compiler verwendet haben. Erfahrungsgemäß dienen diese Berichte nur zur Archivierung und werden seltenst gelesen, höchstens überflogen. Falls es später wegen des Einsatzes von AspectJ Ärger geben sollte, können Sie sich darauf berufen, dass Sie es in Ihren Berichten erwähnt haben.[5] Sollten Sie wider Erwarten zu Ihrem Chef gerufen werden, haben Sie die Gelegenheit, ihn vom Einsatz von AOP und AspectJ zu überzeugen.

»*Ceterum censeo Carthaginem esse delendam!*«[6] – mit diesem Satz beendete der Senator Marcus Porcius Cato jede seiner Reden im römischen Senat. Und er hatte Erfolg damit – Karthago wurde auf sein Betreiben hin von Rom zerstört. Falls also Ihr erster Anlauf nicht zum Erfolg führt, geben Sie nicht gleich auf, sondern wiederholen Sie Ihr Anliegen bei jeder sich bietenden Gelegenheit.

14.2.2 Bottom-up (Don't tell)

Ich gebe zu, die Unterstützung des Managements ist nicht leicht zu bekommen oder erweist sich aus politischen Gründen als illusorisch. Dann kommt Plan B zur Anwendung: »*Don't tell the manager*«. Dieser Ratschlag von Martin Fowler in seinem Buch »Refactoring« [4] kann auch hier mit derselben Begründung gebracht werden:[7] Wer hat mehr Ahnung von der Software-Entwicklung – Sie oder das Management?

Viel interessanter für das Management als die Frage der Technologie und der richtigen Programmiersprache ist die Frage des Erfolges. Wenn Sie erfolgreich sind, ist die Wahl der Mittel zweitrangig. Wenn Sie also der Meinung sind, dass Sie Ihre Aufgabe mit AspectJ schaffen können – gehen Sie das Risiko ein. Sie brauchen dabei nicht auf Netz und doppelten Boden zu verzichten. Schreiben Sie Ihre Anwendung so,

[5] Ich stand am Anfang der OO-Welle vor einem ähnlichen Problem, als ich meine ersten Gehversuche mit einer Testversion von Rational Rose machte. Erst gegen Ende des Projekts habe ich eine neugierige Nachfrage vom Entwicklungsleiter bekommen, was sich denn hinter OOA und OOD verberge – und das auch nur, weil wir uns zufällig begegnet sind.

[6] »Im Übrigen fordere ich, dass Karthago zerstört wird!«

[7] s. a. `http://books.slashdot.org/comments.pl?sid=1403&cid=1664693`

dass sie notfalls ohne Aspekte und ohne AspectJ übersetzt werden kann und trotzdem die Mindestfunktionalität abdeckt.

Hüten Sie sich aber davor, gleich zu viel über Aspekte lösen zu wollen. Sonst könnte Ihr Sicherungsnetz Risse bekommen und die Erfolgsaussichten für das Projekt schrumpfen. Sammeln Sie erst Erfahrungen, um künftige Aufgaben besser einschätzen zu können.

14.2.3 Über die QM-Schiene

Ein viel versprechender Ansatz für die Einführung läuft über die QM-Schiene[8]. Qualität ist ein Begriff, dem sich niemand verschließen kann. Die Bereitschaft zur Verbesserung der Qualität ist beim Management meist ausgeprägter als beim Entwickler.

Definieren Sie eigene Fehlermeldungen.

In Kapitel 6.10.2 auf Seite 193 haben wir gesehen, wie eigene Compiler-Meldungen deklariert werden können. Damit haben Sie die Möglichkeit, den Einsatz unerwünschter Klassen oder Methoden als Warnung oder gar als Fehler zu definieren. Wenn z. B. kein JDBC-Code in der Anwendung vorkommen soll, weil Sie ein Persistenz-Framework verwenden, können Sie folgende Deklaration einsetzen:

```
declare error : call(public * java.sql..*(..)) :
    "bitte Persistenz-Framework benutzen!";
```

Für die eigentliche Übersetzung des Projekts können Sie nach wie vor den Java-Compiler verwenden, falls dies gefordert sein sollte.

Eine weitere Chance für AspectJ bietet der Einsatz von Test-Aspekten, wie bereits in Kapitel 13.5 auf Seite 352 angesprochen. Aber auch Logging-Aspekte sind ein Thema, das spätestens dann zur Diskussion ansteht, wenn ein Problem beim Kunden auftritt, für das sich das implementierte Logging nicht als ausreichend erweist. Hier können Sie vorbeugen, indem Sie einen Logging-Aspekt bereitstellen, wie z. B. in Kapitel 11.1 auf Seite 309 vorgeschlagen.

14.3 Zukunftsaspekte

So langsam kommt aspektorientierte Programmierung aus ihren Startlöchern heraus und dringt ins Bewusstsein von Entwicklern und Management vor. Die Situation ist vergleichbar mit dem Beginn der OO-Ära, und es gibt durchaus Parallelen dazu. Sollte nicht bereits die Einführung der OO-Technologie viele Probleme lösen, die heute immer noch aktuell sind?

[8] QM = Qualitäts-Management

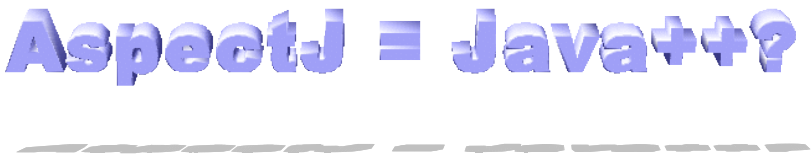

Abbildung 14.1
AspectJ = Java++?

14.3.1 Hype or Hope?

Die Probleme ähneln sich: Die Software, die geschrieben werden muss, wird immer komplexer und der Durchblick immer schwieriger. So wie damals die Objektorientierung einen Ausweg aus diesem Dilemma versprach, bietet sich heute die Aspektorientierung an, um die Probleme von morgen zu lösen. Das gibt AspectJ die Chance, die verschiedenen Anforderungen an eine Anwendung mit weniger Code zu meistern und eine effektivere Entwicklung anzustreben. Dabei bedeutet AspectJ keine Abkehr von Java und der Objektorientierung, sondern baut darauf auf.

Allerdings ist Vorsicht vor übertriebenen Erwartungen angebracht. Es lauern einige Fallstricke bei unsachgemäßer Anwendung von Aspekten, und ein maßloser Einsatz aller Sprachmittel kann genau das Gegenteil bewirken. Hier ist sehr viel stärker die Selbstdisziplin des Entwicklers gefordert als in der Objektorientierung.

Die Lernkurve von AspectJ ist vergleichbar mit dem Umstieg von C auf C++ und sollte nicht unterschätzt werden. Diese Lernkurve kann zwar durch Hinzuziehen von Experten verkürzt werden, aber diese werden gerade in der Anfangsphase Mangelware sein. Erschwerend kommt hinzu, dass auch eine andere Sichtweise gefordert sein wird, die anfangs gewöhnungsbedürftig ist. Neben übertriebenen Erwartungen waren dies in der OO-Pionierzeit einige der Gründe, warum die ersten OO-Projekte oft von Misserfolg geprägt waren. Eine ähnliche Entwicklung könnte sich bei den ersten AOP-Projekten anbahnen, wenn die Erfahrungen aus dieser Zeit ignoriert werden.

AspectJ wird nicht die Lösung aller Probleme bringen, aber es wird bei einigen von ihnen helfen, diese auf einfachere Weise lösen zu können.

14.3.2 Der Blick nach vorn

Wie wird es weitergehen mit AspectJ? Ein großer Schritt nach vorn war der Übergang von Version von 1.2 auf 5.0, nicht nur wegen der sprunghaften Zählweise. Wichtigstes Merkmal ist hier die Java-5-Unterstützung. Hier wurde vor allem im Bereich der Annotations Neuland beschritten, und es ist zu erwarten, dass sich hier in Zukunft auch

Abbildung 14.2
Was wird wohl die Zukunft bringen?

noch weitere Weiterentwicklungen ergeben werden. Sollten Sie die eine oder andere sinnvolle Idee dazu haben – das AspectJ-Entwicklerteam war in der Vergangenheit stets aufgeschlossen für neue Ideen.[9]

Runtime-Weaving

Für die Zukunft ist geplant, Aspekte dynamisch zur Laufzeit einweben zu können. Dazu passt auch die Ankündigung vom Januar 2005, mit AspectWerkz zusammenzugehen.[10] Daraus ergeben sich dann neue ungeahnte Möglichkeiten. So können Sie fremde Bibliotheken in das Logging mit einbeziehen, ohne dass Sie vorher die Jar-Datei instrumentieren müssen (und damit evtl. die Lizenzbedingungen verletzen). Oder Sie können am Wochenende einen Feierabend-Aspekt wirken lassen oder eine neue Zeitrechnung einführen. Machen Sie sich aber darauf gefasst, dass dieses Runtime-Weaving die Fehlersuche nicht vereinfachen wird!

Die ersten AOP-Projekte werden es schwer haben und aus den oben erwähnten Gründen nicht immer von Erfolg gekrönt sein. Ähnlich wie bei der Objektorientierung wird dies aber nicht den Erfolg von Aspekt-Orientierung und AspectJ aufhalten können, bedeutet es doch keine Abkehr von OO, sondern eine Weiterentwicklung dieser Technologie. Die ersten aspektorientierten Frameworks existieren bereits (z. B. von JBoss[11]), und es werden stetig mehr werden.

In wenigen Jahren wird der Umgang mit AspectJ genauso selbstverständlich sein wie der Umgang mit Java. Wie im Schwabenland Linsen mit Spätzle untrennbar miteinander verbunden sind, wird auch AspectJ nicht mehr von Java wegzudenken sein.

[9] Nutzen Sie die Möglichkeit der Mailingliste: `https://dev.eclipse.org/mailman/listinfo/aspectj-users`

[10] `http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/aspectj-home/aj5announce.html`

[11] `http://www.jboss.org/products/aop`

Abbildung 14.3
Bedeutung von AspectJ

14.4 Zusammenfassung

- Es gibt genug Gründe, um mit AspectJ anzufangen.
- Es gibt genug Gründe, um mit Unterstützung des Managements mit AspectJ anzufangen.
- Es gibt genug Gründe, um mit AspectJ ohne Unterstützung des Managements anzufangen.
- Es gibt genug Gründe, um mit AspectJ *jetzt* anzufangen.