Die Rolle von Features und Aspekten in der Softwareentwicklung

Sven Apel

Lehrstuhl für Programmierung
Universität Passau
Innstraße 33, 94032 Passau
apel@uni-passau.de

Abstract: Feature-orientierte Programmierung (FOP) und Aspekt-orientierte Programmierung (AOP) sind komplementäre Technologien. Obwohl beide auf die Modularität von sogenannten querschneidenden Belangen abzielen, tun sie dies auf unterschiedliche Art und Weise. Im Rahmen der Arbeit wurde beobachtet, das FOP und AOP kombiniert werden können, um ihre individuellen Schwächen zu überwinden. Die Arbeit schlägt mit Aspekt-basierten Feature-Modulen (Aspectual Feature Modules) und Aspektverfeinerung (Aspect Refinement) zwei Techniken zur Symbiose von FOP und AOP vor. Beide Techniken werden in einer Fallstudie evaluiert, und entsprechende Programmierrichtlinien werden zum Einsatz von FOP und AOP abgeleitet. Schlussendlich wird mittels der Analyse von acht AspectJ-Programmen unterschiedlicher Größe die Frage beantwortet, wie Implementierungsmechanismen der FOP und der AOP heutzutage Verwendung finden.

1 Einleitung

Der Begriff *Softwaretechnik* und die damit verbundene Offensive erwuchs in den 60ern und 70ern aus den anhaltenden Problemen bei der Entwicklung von Software, welche unter dem Begriff *Softwarekrise* zusammengefasst werden [NR69]. Obwohl sich seitdem einiges bewegt hat, ist die derzeitige Situation in der Softwareentwicklung alles andere als zufrieden stellend. Laut dem aktuellen Bericht der Standish Group werden nur 34% aller Softwareprojekte erfolgreich zum Abschluss gebracht [Gro03].

Seitdem werden zwei Prinzipien eng mit der Überwindung der Softwarekrise in Verbindung gebracht: *Trennung von Belangen (Separation of Concerns)* und *Modularität (Modularity)* [Par72, Dij76]. Finden diese Prinzipien in der Entwicklung von Software Beachtung, läßt sich die Verständlichkeit, Wartbarkeit, Wiederverwendbarkeit und Anpassbarkeit von Software signifikant verbessern. Allerdings stellte sich schnell heraus, dass es weit komplizierter ist, adäquate Konzepte, Methoden, Formalismen und Werkzeuge zu entwickeln, als zunächst angenommen.

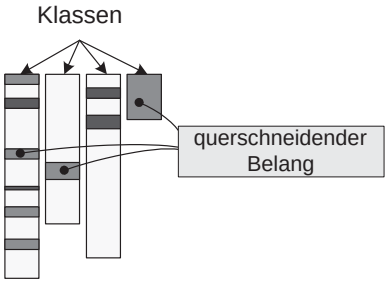

Abbildung 1: Die Implementierung eines querschneidenden Belangs resultiert in verstreutem und vermischtem Quelltext.

2 Zielstellung

Die Dissertation hat zum Ziel, zu diesem Bereich der Forschung beizutragen [Ape07]. Im Speziellen beschäftigt sich die Arbeit mit zwei derzeitig diskutierten Programmierparadigmen, der *Feature-orientierten Programmierung (FOP)* [Pre97, BSR04] und der *Aspekt-orientierten Programmierung (AOP)* [KLM+97, EFB01]. Beide Paradigmen konzentrieren sich auf eine bestimmte Klasse von Entwurfs- und Implementierungsproblemen, die so genannten *querschneidenden Belange (Crosscutting Concerns)*. Ein querschneidender Belang entspricht einer einzelnen Entwurfs- oder Implementierungsentscheidung bzw. einer Fragestellung oder eines Ansinnens, dessen Implementierung typischerweise über weite Teile eines Softwaresystems verstreut ist (siehe Abbildung 1). Etablierte Programmierparadigmen wie z. B. Objekt-orientierte Programmierung verfügen weder über genug Ausdruckskraft noch über geeignete Modularisierungsmechanismen, um querschneidendene Belange modular zu implementieren [KLM+97, EFB01]. Tarr et al. [TOHS99] attestieren Programmierparadigmen, die auf hierarchischer Dekomposition beruhen (z. B. Objektorientierung), eine prinzipielle Unfähigkeit querschneidendene Belange modular zu implementieren. D. h., unter Zuhilfenahme derzeitig etablierter Paradigmen widersprechen querschneidende Belange den Prinzipien der Trennung von Belangen und der Modularität.

3 FOP und AOP

FOP beschäftigt sich mit der Modularität von Features im Softwareentwicklungsprozess. Ein Feature entspricht einer Anforderung an ein Softwaresystem und stellt ein Inkrement in dessen Funktionalität dar. Features werden benutzt, um Gemeinsamkeiten und Unterschiede von Softwaresystemen zu kommunizieren [KCH+90, CE00]. Üblicherweise lassen sich Features nicht durch einzelne Prozeduren oder Objekte implementieren. Oftmals betrifft die Implementierung eines Features viele Stellen im Code eines Softwaresystems [Pre97, SB02, BSR04, MO04]. Features sind somit typische Beispiele für quer-

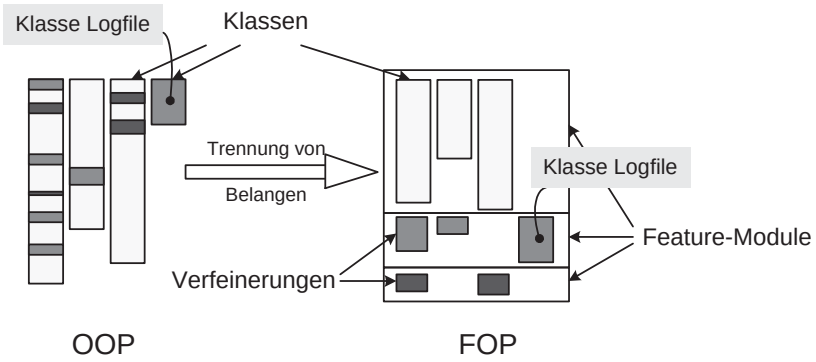

Abbildung 2: Zwei Feature-Module implementieren jeweils einen querschneidenden Belang als modulare Erweitertung zu einem Basisprogramm.

schneidende Belange. FOP stellt Techniken zur modularen Implementierung von Features bereit. Feature werden als Kollaborationen von Klassen modelliert [RAB+92, VN96]. Eine jede Kollaboration wird durch ein Feature-Modul gekapselt. Eine Besonderheit ist, dass die Klassen eines Features Erweiterungen/Verfeinerungen zu bereits bestehenden Klassen darstellen können. Die Kombination aus Kollaborationen und Klassenverfeinerungen erlaubt die modulare Implementierung von querschneidenden Belangen (siehe Abbildung 2).

AOP beschäftigt sich mit ähnlichen Themen. Aspekte sind Mechanismen, die per Definition querschneidende Belange so implementieren, dass der entsprechende Code wohlmodularisiert vorliegt. Aspekte sind nicht an Anforderungen bzw. Gemeinsamkeiten oder Unterschiede von Softwaresystemen gebunden, können aber zu deren Repräsentation verwendet werden. Ein Unterschied zu FOP ist die Verwendung der Programmiermechanismen. Während FOP hauptsächlich auf die Erweitertung von Klassen und Methoden mittels Objekt-orientierter Techniken setzt, verwendet AOP auch Techniken aus dem Bereich der Metaprogrammierung, z. B. zur Identifikation der Punkte in einem Programm, die erweitert bzw. modifiziert werden sollen [MK03, FF05]. Abbildung 3 zeigt die Implementierung zweier querschneidender Belange durch zwei Aspekte. Aspekte und Basisprogramm werden mittels eines Aspektwebers zu einem fertigen Programm zusammengesetzt.

4 Symbiose von FOP und AOP

FOP und AOP stellen beide methodische und programmiersprachliche Mittel und Werkzeuge bereit, gehen das Problem der querschneidenden Belange aber auf sehr unterschiedliche Weise an. In der Dissertation wird jedoch festgestellt, dass FOP und AOP keine konkurrierenden Ansätze sind, sondern dass ihre Kombination die individuellen Schwächen überwinden kann. Diese Einsicht wird untermauert durch eine Klassifikation von querschneidenden Belangen und eine Evaluierung von FOP und AOP hinsichtlich der

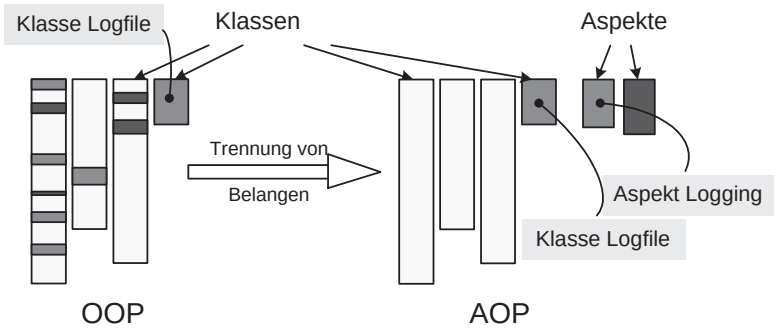

Abbildung 3: Zwei Aspekte implementieren jeweils einen querschneidenden Belang als modulare Erweitertung zu einem Basisprogramm.

verschiedenen Klassen querschneidender Belange. So ist AOP beispielsweise vorteilhaft für die Implementierung von querschneidenden Belangen, die ein Basisprogramm an vielen Stellen auf dieselbe Art und Weise erweitern (*homogene querschneidende Belange*). FOP erlaubt die Implementierung einer Programmerweiterung in Form einer Kollaboration aus unterschiedlichen Klassen und Klassenverfeinerungen, die die originale Struktur des Basisprogramms widerspiegeln (*heterogene querschneidende Belange*). Dies erleicht das Verständnis des Programms. Ergebnis der Gegenüberstellung von FOP und AOP ist ein Satz von Programmierrichtlinien in Form eines Katalogs, der ihre Stärken und Schwächen gegenüberstellt. Die Symbiose von FOP und AOP ist daher sinnvoll.

5 Aspekt-basierte Feature-Module

Um von den individuellen Stärken beider Paradigmen zu profitieren, wird in der Dissertation die Symbiose von FOP und AOP angeregt. Insbesondere wird der Ansatz der *Aspekt-basierten Feature-Module (Aspectual Feature Modules – AFMs)* vorgeschlagen. AFMs setzen die Symbiose von FOP und AOP um, indem sie ihre Entwurfsphilosophien, Sprachmechanismen und Werkzeuge geeignet kombinieren. Abbildung 4 zeigt die Integration von Aspekten in den Kollaborationen-orientierten Stil von FOP. Aspekte sind somit einfach eine weitere Art von Softwareartefakten, die innerhalb einer Kooperation ein Feature implementieren. Die Kombination von FOP- und AOP-Techniken verbessert die Modularität eines Programms. Eine Evaluierung und eine Gegenüberstellung mit traditioneller FOP und AOP demonstrieren die Überlegenheit von AFMs.

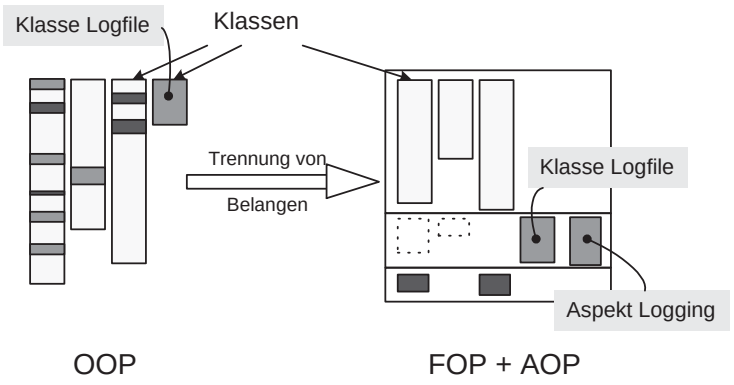

Abbildung 4: Aspekt-basierte Feature-Module integrieren Features und Aspekte.

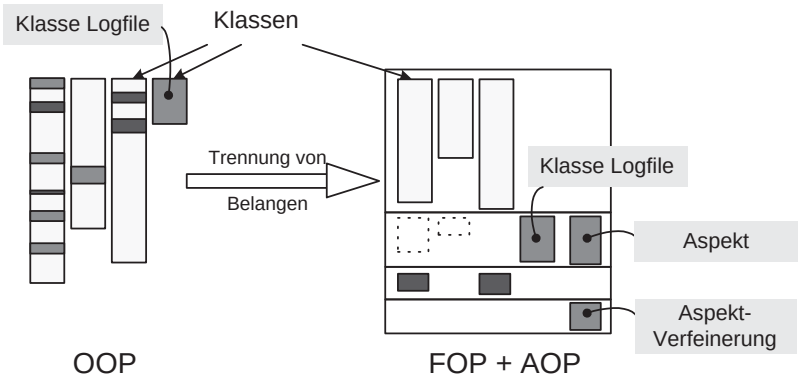

Abbildung 5: Aspekt-basierte Feature-Module können bestehende Aspekte verfeinern.

6 Verfeinerung von Aspekten

Desweiteren wird in der Dissertation herausgestellt, dass derzeitige AOP-Sprachen nicht uneingeschränkt geeignet sind, in die schrittweise Entwurfsphilosophie von FOP integriert zu werden. Konsequenterweise wird der Ansatz der *Aspektverfeinerung (Aspect Refinement – AR*) vorgeschlagen, welcher AOP und schrittweise Softwareentwicklung [Wir71] à la FOP vereinheitlicht. Weiterhin werden entsprechende Sprachkonstrukte und Werkzeuge zur Verfügung gestellt. Im Kontext von AFMs bedeutet dies, dass Aspekte innerhalb eines Feature-Moduls durch später hinzugefügte Aspekte verfeinert werden können, ganz so wie Klassen durch Klassenverfeinerungen verfeinert werden können. Abbildung 5 zeigt die Verfeinerung eines Aspektes. Aspektverfeinerung vereinheitlicht die Verfeinerung von Klassen und Aspekten im Rahmen von FOP.

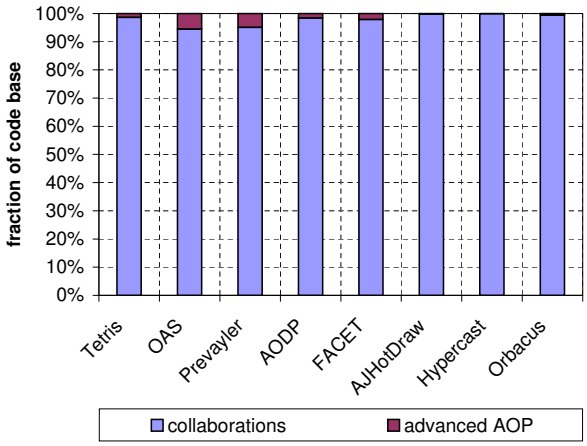

Abbildung 6: 98% der Codebasis 8 analysierter AspectJ-Programme ist mit Kollaborationen verknüpft; 2% nutzt die erweiterten Mittel von AOP.

7 Fallstudien und Analysen

Mittels einer nicht-trivialen Fallstudie wird die praktische Anwendbarkeit von AFMs und AR auf ein mittelgroßes Softwareprojekt demonstriert. Die Symbiose von FOP und AOP erweist sich als praktikabel. Die Mechanismen von FOP und AOP arbeiten problemlos zusammen und spielen ihre Stärken bei der Implementierung unterschiedlicher Typen von querschneidenden Belangen aus.

Die Studie wirft weiterhin eine fundamentale Frage auf: Wie werden Mechanismen von FOP und AOP heutzutage verwendet? Hintergrund ist, dass eine spezielle Klasse von querschneidenden Belangen eng mit FOP verknüpft ist, nämlich Kollaborationen [RAB+92]. Durch die fehlende Unterstützung von Kollaborationen in aktuellen Programmiersprachen wird dafür heute oft AOP benutzt.

Durch das Aufkommen von Programmiersprachen, die Kollaborationen explizit unterstützen (z. B. Jiazzi [MFH01], Jak [BSR04], Scala [OZ05], Classbox/J [BDN05], FeatureC++ [ALRS05]), sowie durch die in der Dissertation präsentierten Klassifikation und Evaluierung, stellen sich jedoch folgende Fragen: Welcher Anteil von Aspektcode implementiert Kollaborationen? Welcher Anteil implementiert querschneidende Belange, die darüber hinaus AOP benötigen? Eine quantitative Analyse von 8 AspectJ-Programmen unterschiedlicher Größe ergibt, dass durchschnittlich 98% der Codebasis der analysierten Programme mit Kollaborationen verknüpft sind und nur 2% die erweiterten Mittel von AOP jenseits von Kollaborationen ausnutzen (siehe Abbildung 6). Weiterhin wird beobachtet, dass mit steigender Programmgröße der Einfluss von AOP sinkt. In der Dissertation wird die Frage beantwortet, warum dieses (Miss)Verhältnis zwischen AOP- und FOP-Code besteht, und warum dies nicht überrascht.

Trotz der sporadischen Nutzung von erweiterten AOP-Techniken ist ein positiver Einfluss auf den Anteil von repliziertem Codem zu verzeichnen. Durch den Einsatz von AOP-

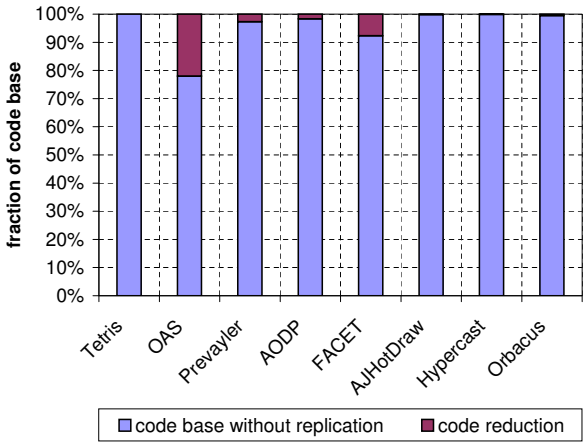

Abbildung 7: Im Durchschnitt konnte durch den Einsatz von AOP-Techniken der Anteil von repliziertem Code um 4% gesenkt werden.

Techniken konnte im Durchschnitt die Codegröße um 4% gesenkt werden (siehe Abbildung 7). Schlussendlich wird diskutiert, ob und wie der positive Einfluss von AOP gesteigert werden kann.

8 Schlusswort

Die Gegenüberstellung von FOP und AOP führte nicht nur zur Symbiose beider Programmierparadigmen, sondern lieferte auch tiefe Einsichten in ihre individuelle Natur, in die Einordnung in übergeordnete Paradigmen wie schrittweise Softwareentwicklung und in die aktuelle Programmierpraxis. Wenn Programmierparadigmen auch ständigem Wandel ausgesetzt sind, haben die Erkenntnisse der Arbeit das Potential die Entwicklung von FOP und AOP sowie ihrer Konvergenz zu beeinflussen.

Literatur

[ALRS05] S. Apel, T. Leich, M. Rosenmüller und G. Saake. FeatureC++: On the Symbiosis of Feature-Oriented and Aspect-Oriented Programming. In *Proceedings of the International Conference on Generative Programming and Component Engineering (GPCE)*, Jgg. 3676 of *Lecture Notes in Computer Science*, Seiten 125–140. Springer-Verlag, 2005.

[Ape07] S. Apel. *The Role of Features and Aspects in Software Development – Similarities, Differences, and Synergetic Potential*. VDM Verlag Dr. Müller, 2007.

[BDN05] A. Bergel, S. Ducasse und O. Nierstrasz. Classbox/J: Controlling the Scope of Change in Java. In *Proceedings of the International Conference on Object-Oriented Program-*

ming, Systems, Languages, and Applications (OOPSLA), Seiten 177–189. ACM Press, 2005.

[BSR04] D. Batory, J. N. Sarvela und A. Rauschmayer. Scaling Step-Wise Refinement. *IEEE Transactions on Software Engineering (TSE)*, 30(6):355–371, 2004.

[CE00] K. Czarnecki und U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[Dij76] E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.

[EFB01] T. Elrad, R. E. Filman und A. Bader. Aspect-Oriented Programming: Introduction. *Communications of the ACM (CACM)*, 44(10):29–32, 2001.

[FF05] R. Filman und D. Friedman. Aspect-Oriented Programming Is Quantification and Obliviousness. In *Aspect-Oriented Software Development*, Seiten 21–35. Addison-Wesley, 2005.

[Gro03] The Standish Group. Chaos Report. Bericht, Standish Group International, 2003.

[KCH+90] K. Kang, S. Cohen, J. Hess, W. Novak und A. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Bericht CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[KLM+97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier und J. Irwin. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Jgg. 1241 of *Lecture Notes in Computer Science*, Seiten 220–242. Springer-Verlag, 1997.

[MFH01] S. McDirmid, M. Flatt und W. C. Hsieh. Jiazzi: New-Age Components for Old-Fashioned Java. In *Proceedings of the International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Seiten 211–222. ACM Press, 2001.

[MK03] H. Masuhara und G. Kiczales. Modeling Crosscutting in Aspect-Oriented Mechanisms. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Jgg. 2743 of *Lecture Notes in Computer Science*, Seiten 2–28. Springer-Verlag, 2003.

[MO04] M. Mezini und K. Ostermann. Variability Management with Feature-Oriented Programming and Aspects. In *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*, Seiten 127–136. ACM Press, 2004.

[NR69] P. Naur und B. Randell, Hrsg. *Software Engineering: Report of the Working Conference on Software Engineering, Garmisch, Germany, October 1968*. NATO Science Committee, 1969.

[OZ05] M. Odersky und M. Zenger. Scalable Component Abstractions. In *Proceedings of the International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Seiten 41–57. ACM Press, 2005.

[Par72] D. L. Parnas. On the Criteria to be Used in Decomposing Systems into Modules. *Communications of the ACM (CACM)*, 15(12):1053–1058, 1972.

[Pre97] C. Prehofer. Feature-Oriented Programming: A Fresh Look at Objects. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Jgg. 1241 of *Lecture Notes in Computer Science*, Seiten 419–443. Springer-Verlag, 1997.

[RAB+92] T. Reenskaug, E. Andersen, A. Berre, A. Hurlen, A. Landmark, O. Lehne, E. Nordhagen, E. Ness-Ulseth, G. Oftedal, A. Skaar und P. Stenslet. OORASS: Seamless Support for the Creation and Maintenance of Object-Oriented Systems. *Journal of Object-Oriented Programming (JOOP)*, 5(6):27–41, 1992.

[SB02] Y. Smaragdakis und D. Batory. Mixin Layers: An Object-Oriented Implementation Technique for Refinements and Collaboration-Based Designs. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2):215–255, 2002.

[TOHS99] P. Tarr, H. Ossher, W. Harrison und S. M. Sutton, Jr. N Degrees of Separation: Multi-Dimensional Separation of Concerns. In *Proceedings of the International Conference on Software Engineering (ICSE)*, Seiten 107–119. IEEE Computer Society Press, 1999.

[VN96] M. VanHilst und D. Notkin. Using Role Components in Implement Collaboration-based Designs. In *Proceedings of the International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Seiten 359–369. ACM Press, 1996.

[Wir71] N. Wirth. Program Development by Stepwise Refinement. *Communications of the ACM (CACM)*, 14(4):221–227, 1971.

Dr.-Ing. **Sven Apel** ist akademischer Oberrat am Lehrstuhl für Programmierung der Fakultät für Informatik und Mathematik an der Universität Passau. Er studierte Informatik (1996-2002) an der Fakultät für Informatik der Otto-von-Guericke-Universität Magdeburg (Prädikat “sehr gut”). Dort folgte im März 2007 seine Promotion (Prädikat “summa cum laude”). Anfang 2006 war er für 6 Monate als Gastwissenschaftler an der University of Texas at Austin tätig. Sein Interesse in der Forschung gilt modernen Programmierparadigmen, Softwareproduktlinien sowie formalen Methoden für die Softwareentwicklung. Sven Apel erhielt im Januar 2007 den Forschungspreis der Fakultät für Informatik der Otto-von-Guericke-Universität Magdeburg für seine wissenschaftliche Gesamtleistung. Im September 2007 wurde ihm der Software-Engineering-Preis der Ernst Denert-Stiftung verliehen. Im November 2007 erhielt Sven Apel den Fakultätspreis der Fakultät für Informatik der Otto-von-Guericke-Universität Magdeburg für die beste Dissertation sowie den Dissertationspreis der Otto-von-Guericke-Universität Magdeburg. Sven Apel ist Autor von über 40 referrierten wissenschaftlichen Publikationen.