# Modularity in the New Millenium: A Panel Summary

Premkumar Devanbu, (*Panel Chair*)
Bob Balzer, Don Batory, Gregor Kiczales, John Launchbury, David Parnas, Peri Tarr (*Panelists*)

## 1 Introduction (Chair's Statement)

Parnas' seminal work [2] on *separation of concerns* in design has led to diverse innovations in programming language design, to support modularity. However, there has been a growing sentiment in many quarters that there are some concerns that stubbornly resist tidy confinement, when using established modularization mechanisms in programming languages. A diverse set of new approaches have emerged in response: aspects [1], monads [5], mixin layers [3], and multi-dimensional separation of concerns [4]. These approaches arose more or less independently of each other, and have (to varying degrees) developed technical maturity, real-world credibility and strong user bases. We are also now beginning to see strong scholarly comparisions of the intellectual foundations and practical utility of these different aproaches. This panel aims to support this trend.

In this panel, we bring together leading experts (Profs. Batory, Kiczales, and Launchbury, and Dr. Tarr) in these different areas. Each represents a particular perspective on how to evolve and adapt the old idea of modularization to deal with new challenges such as security, fault-tolerance, distribution, and auditing. In addition, we also have two pioneering researchers (Profs. Balzer and Parnas) to provide us with a historical perspective on the evolution (sic) of program modularization and evolution techniques.

Position statements of some of the panelists follow, presented in alphabetical order of their names:

## 2 Panel Statement: Don Batory

The future of software engineering lies in automation. Perhaps the most successful example of automated software engineering is relational query optimization (RQO). A query is specified in a declarative domain-specific language (SQL), mapped to an inefficient relational algebra expression, optimized by rewrite rules, and then an efficient query evaluation program is generated. RQO is also a great example of the holy grail problem of automatic programming transforming a declarative specification to an efficient program. Feature Oriented Programming (FOP) aims to generalize this powerful paradigm to other software domains. An FOP domain model is a set of operators that define an algebra. Each operator implements a feature that is characteristic of programs in that domain; compositions of these operators define a specific program. FOP is ideally suited for product-lines and program generators. AHEAD is both a model and tool suite for FOP that is based on step-wise refinement, the paradigm that builds complex applications from simple applications by progressively adding features. The AHEAD tool suite has been bootstrapped, where its 150+K LOC are being generated from simple, declarative specifications. Optimizations in our tool building process are now being introduced to make it like RQO.

## 3 "It's the cross-cutting": Gregor Kiczales

The idea of separation of concerns is not new; it is a basic element of modern thought.

We are however, constantly renewing our tools for separating, composing and otherwise operating on concerns throughout the lifecycle. We have seen several such evolutions, including the most recent major success, object-orientation.

The critical idea in aspect-oriented programming (AOP) is that no single decomposition can capture all the concerns in a complex system in a modular way. This incredibly simple idea is, in retrospect, nearly obvious. But until AOP we spent countless hours (days, weeks...) refactoring a system over-and-over trying to get a structure in which all concerns are modular. The frustration of working with single decompositions of complex systems led to the realization that choosing a nice structure for modularizing many concerns inherently means that some others cannot be modular, if they have to live within that primary structure. We call such concerns crosscutting, because they cut across the natural lines of the rest of the system.

The rapid uptake of AOP in industry is because developers find that adopting AOP tools like AspectJ is simple and natural their own experience helps them understand cross-cutting concerns at a deep intuitive level.

## 4 Panel Statement: David Parnas

To a man with a hammer, everything looks like a nail. To a Computer Scientist, everything looks like a language design problem. Languages and compilers are, in their opinion, the only way to drive an idea into practice.

Proceedings of the 25th International Conference on Software Engineering (ICSE'03)
0270-5257/03 $17.00 © 2003 IEEE

My early work clearly treated modularisation as a design issue, not a language issue. A module was a work assignment, not a subroutine or other language element. Although some tools could make the job easier, no special tools were needed to use the principal, just discipline and skill.

When language designers caught on to the idea, they assumed that modules had to be subroutines, or collections of subroutines, and introduced unreasonable restrictions on the design. They also spread the false impression that the important thing was to learn the language; in truth, the important thing is to learn how to design and document.

We are still trying to undo the damage caused by the early treatment of modularity as a language issue and, sadly, we still try to do it by inventing languages and tools.

## 5 Panel Statement: Peri Tarr

Multi-dimensional separation of concerns (MDSOC) allows developers to encapsulate overlapping, interacting and crosscutting concerns, including features, aspects, variants, roles, business rules, components, frameworks, etc., simultaneously. One does not have to choose between, say, a data decomposition and a feature decomposition, since both can coexist, and each can be used when appropriate. All concerns are first-class components that can be integrated flexibly. MDSOC further supports developers by allowing concerns to be identified throughout the software lifecycle and relationships among concerns to be managed. MDSOC grew out of earlier work on subject-oriented programming. It retains the same style of composition, while adding the ability to extract concerns from existing software and maintain multiple decompositions simultaneously. The approach has been successful at feature-based development and integration, inserting probes, retrofitting and reusing design patterns, and unanticipated extension and adaptation, and has shown promise with creating and evolving product lines, extracting concerns from legacy software and reconciling different views and perspectives. Our first-generation tool, Hyper/J, was less successful at "uniform crosscutting," where common functionality is applied uniformly in multiple contexts, and did no semantic checking. AspectJ pointcuts, with their powerful patterns, address uniform crosscutting more effectively, and mixin layers provide semantic checking and predictability. In our current work on a new Concern Manipulation Environment (CME), we are introducing componentry to address these and other limitations. We have used Hyper/J to build a portion of the CME, and have identified many other places where the more powerful capabilities of the CME itself could be brought to bear.

## 6 Conclusion

This paper presents a brief summary of some of the panelist's positions. We hope that the contents will serve to encourage colleagues to attend the panel at the conference, and to stimulate further discussion.

## References


[1] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. In *European Conference on Object Oriented Programming*, 2001.

[2] D. Parnas. The criteria to be used in decomposing systems into modules. *Communications of the ACM*, 14(1):221–227, 1972.

[3] Y. Smaragdakis and D. Batory. Implementing layered designs with mixin layers. In *European Conference on Object Oriented Programming*, 1998.

[4] P. L. Tarr, H. Ossher, W. H. Harrison, and S. M. S. Jr. N degrees of separation: Multi-dimensional separation of concerns. In *International Conference on Software Engineering*, 1999.

[5] P. Wadler. The essence of functional programming. In *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 1–14, Albequerque, New Mexico, 1992.