Workshop on Multi-Dimensional Separation of Concerns in Software Engineering

Peri Tarr, William Harrison, Harold Ossher (IBM T. J. Watson Research Center, USA)
Anthony Finkelstein (University College London, UK)
Bashar Nuseibeh (Imperial College, UK)
Dewayne Perry (University of Texas at Austin, USA)

Workshop Web site: http://www.research.ibm.com/hyperspace/workshops/icse2000

ABSTRACT
Separation of concerns has been central to software engineering for decades, yet its many advantages are still not fully realized. A key reason is that traditional modularization mechanisms do not allow simultaneous decomposition according to multiple kinds of (overlapping and interacting) concerns. This workshop was intended to bring together researchers working on more advanced modularization mechanisms, and practitioners who have experienced the need for them, as a step towards a common understanding of the issues, problems and research challenges.

Keywords
Separation of concerns, decomposition, composition

1 SEPARATION OF CONCERNS

Separation of concerns [17] is at the core of software engineering, and has been for decades. In its most general form, it refers to the ability to identify, encapsulate, and manipulate only those parts of software that are relevant to a particular concept, goal, or purpose. Concerns are the primary motivation for organizing and decomposing software into manageable and comprehensible parts.

Many different kinds, or *dimensions*, of concerns may be relevant to different developers in different roles, or at different stages of the software lifecycle. For example, the prevalent kind of concern in object-oriented programming is *data* or *class;* each concern in this dimension is a data type defined and encapsulated by a class. *Features* [19], like printing, persistence, and display capabilities, are also common concerns, as are non-functional concerns, like concurrency control and distribution, *roles* [1], *viewpoints* [13], variants, and configurations. Separation of concerns involves decomposition of software according to one or more dimensions of concern.

"Clean" separation of concerns has been hypothesized to reduce software complexity and improve comprehensibility; promote traceability within and across artifacts and throughout the lifecycle; limit the impact of change, facilitating evolution and non-invasive adaptation and customization; facilitate reuse; and simplify component integration.

2 THE TYRANNY OF THE DOMINANT DECOMPOSITION

These goals, while laudable and important, have not yet been achieved in practice. This is because the set of relevant concerns varies over time and is context-sensitive—different development activities, stages of the software lifecycle, developers, and roles often involve concerns of dramatically different kinds. One concern may promote some goals and activities, while impeding others; thus, any criterion for decomposition will be appropriate for some contexts, but not for all. Further, multiple kinds of concerns may be relevant simultaneously, and they may overlap and interact, as features and classes do. Thus, different concerns and modularizations are needed for different purposes: sometimes by class, sometimes by feature, sometimes by viewpoint, or aspect, role, variant, or other criterion.

These considerations imply that developers must be able to identify, encapsulate, modularize, and manipulate multiple dimensions of concern simultaneously, and to introduce new concerns and dimensions at any point during the software lifecycle, without suffering the effects of invasive modification and rearchitecture. Even modern languages and methodologies, however, suffer from a problem we have termed the "tyranny of the dominant decomposition" [18]: they permit the separation and encapsulation of only one kind of concern at a time.

Software started out being represented on linear media, and despite advances in many fields, such as graphics and visualization, hypertext and other linked structures, and databases, it is still mostly treated as such. Programs are typically linear sequences of characters, and modules are collections of contiguous characters. This linear structure implies that a body of software can be decomposed in only one way, just as a typical document is divided into sections and subsections in only one way. This one decomposition is dominant, and often excludes any other form of decomposition.

Examples of tyrant decompositions are classes (in object-oriented languages), functions (in functional languages), and rules (in rule-based systems). It is, therefore, impossible to encapsulate and manipulate, for example, features in the object-oriented paradigm, or objects in rule-based systems. Thus, it is impossible to obtain the benefits of different decomposition dimensions throughout the software lifecycle. Developers of an artifact are forced to commit to one, dominant dimension early in the development of that artifact, and changing this decision can have catastrophic consequences for the existing artifact. What is more, artifact languages often constrain the choice of dominant dimension (e.g., it must be *class* in object-oriented software), and different artifacts, such as requirements and design documents, might therefore be forced to use different decompositions, obscuring the relationships between them.

We believe that the tyranny of the dominant decomposition is the single most significant cause of the failure, to date, to achieve many of the expected benefits of separation of concerns.

3 MULTI-DIMENSIONAL SEPARATION OF CONCERNS

We use the term *multi-dimensional separation of concerns* to denote separation of concerns involving:

- Multiple, arbitrary dimensions of concern.
- Separation along these dimensions *simultaneously*, i.e., a developer is not forced to choose a small number (usually one) of dominant dimensions of concern according to which to decompose a system at the expense of others.
- The ability to handle new concerns, and new dimensions of concern, *dynamically*, as they arise throughout the software lifecycle. Concerns that span artifacts and stages of the software lifecycle are especially interesting, and challenging.
- Overlapping and interacting concerns; it is appealing to think of many concerns as independent or "orthogonal," but they rarely are in practice. It is essential to be able to support interacting concerns, while still achieving useful separation.
- Concern-based integration. Separation of concerns is clearly of limited use if the concerns that have been separated cannot be integrated; as Jackson notes, "having divided to conquer, we must reunite to rule" [3].

Full support for multi-dimensional separation of concerns opens the door to *on-demand remodularization*, allowing a developer to choose at any time the best modularization, based on any or all of the concerns, for the development task at hand. Multi-dimensional separation of concerns thus represents a set of very ambitious goals, applying to any software development language or paradigm.

A good deal of research has been done within the last decade or so on "advanced" approaches to separation of concerns [1,2,3,4,5,6,7,8,9,10,12,13,14,15,16,18,20,21]. Considerable research is still required, however, before any approach fully achieves the goals stated above. We believe that it is necessary to achieve them in order to overcome the problems associated with the tyranny of the dominant decomposition and to realize the full potential of separation of concerns.

4 THE WORKSHOP

This workshop was intended to bring together researchers interested in pushing the frontier in this important and burgeoning area, and practitioners who have experienced problems related to inadequate separation of concerns that can help to guide their research. Twenty-five position papers were accepted to the workshop, all available at the workshop Web site [20]. The workshop consisted of five sessions, organized around some of the key themes that emerged from the position papers. Most sessions were introduced by brief presentations, and continued with general discussion.

The rest of this section outlines the sessions of the workshop and, where appropriate, the presentations that introduced them. The abstracts have been extracted verbatim from the position papers.

Introduction: Setting the Stage

A brief overview of common concepts and terminology, and motivation for multi-dimensional separation of concerns was presented by Peri Tarr. The foils are available at the workshop Web site [20].

Models of Decomposition and Composition

Fundamental to multi-dimensional separation of concerns are approaches to decomposing software that go beyond the standard modularization mechanisms provided by modern languages, and corresponding approaches to composition.

Don Batory, "Refinements and Separation of Concerns" (invited presentation).

Today's notions of encapsulation are very restricted — a module or component contains only source code. What we really need is for modules or component to encapsulate not only source code that will be installed when the component is used, but also encapsulate corresponding changes to documentation, formal properties, and performance properties — i.e., changes to the central concerns of software development. The general abstraction that encompasses this broad notion of encapsulation is called a "refinement".

Franz Achermann, "Language Support for Feature Mixing"

Object oriented languages cannot express certain composition abstractions due to restricted abstraction power. A number of approaches, like SOP or AOP overcome this restriction, thus giving the programmer more possibilities to get a higher degree of separation of concern. We propose forms, extensible mappings from labels to values, as vehicle to implement and reason about composition abstractions. Forms unify a variety of concepts such as interfaces, environments, and contexts. We are prototyping a composition language where forms are the only and ubiquitous first class value. Using forms, it is possible compose software artifacts focusing on a single concern and thus achieve a high degree of separation of concern. We believe that using forms it also possible to compare and reason about the different composition mechanisms proposed.

Lodewijk Bergmans, "Composing Software from Multiple Concerns: A Model and Composition Anomalies"

Constructing software from components is considered to be a key requirement for managing the complexity of software. Separation of concerns makes only sense if the realizations of these concerns can be composed together effectively into a working program. Various publications have shown that composability of software is far from trivial and fails when components express complex behavior such as constraints, synchronization and history-sensitiveness. We believe that to ad-dress the composability problems, we need to understand and define the situations where composition fails. To this aim, in this paper we (a) introduce a general model of multi-dimensional concern composition, and (b) define so-called *composition anomalies*.

Mark Chu-Carroll, "Software Configuration Management as a Mechanism for MDSOC"

Real software rarely conforms to one single view of the program structure; instead, software is sufficiently complex that the structure of the program is best understood as a collection of orthogonal divisions of the program into components. However, most software tools only recognize the decomposition of the program into source files, forcing the programmer to adopt one primary program de-composition which is well-suited to some tasks and poorly suited to others. Software tools can overcome this weakness by allowing programmers to transform their view of the program to a structure which is more appropriate for the task they need to perform. We propose that a *software configuration management (SCM)* system, which stores the source code for the project, can perform this task. By providing the SCM system with the capability to generate orthogonal program organizations through compositions of pro-gram fragments, the SCM system can support or-

thogonal decompositions of the program without performing any automatic alteration of the source code.

Real-Life Dimensions of Concern

The primary motivation behind multi-dimensional separation of concerns is that there are many different kinds of concerns that come up during the software lifecycle, all of which should be recognized and at least some of which should be separated. This session began with small discussion groups, each focusing on a particular phase of the lifecycle, followed by general discussion. The purpose of the session was to raise important dimensions of concern that turn up during each phase, and to discuss their relationships, and the extent to which they span lifecycle phases and interact with concerns arising in other phases.

Dimensions of Concern in Product Lines and Software Architecture

Product lines have particularly strong separation of concerns requirements. In addition to separating components, they must also separate *variants*, often involving multiple components, from one another and from the base. Separation of concerns is also one of the themes of software architecture. Care is taken to separate components from interactions, and a key distinguishing feature of different architectural styles is what kinds of concerns they separate, and how. This session explored the implications of multi-dimensional separation of concerns for these important areas.

Joachim Bayer, "Towards Engineering Product Lines using Concerns"

Separation of concerns is accepted as introducing numerous benefits into software development and maintenance. In this position paper, we argue for a method that introduces separation of concerns into product line software engineering. The method covers the complete product line life cycle and integrates the different concerns expressed at the different product line life cycle stages.

Juha Savolainen, "Improving Product-Line Development with SOP"

It has been demonstrated the product lines have introduced large improvements to quality, time to market and overall productivity. However, creating a successful product line is a highly complex and difficult task. There are still many technological barriers to overcome in effective product line development. The current industrial practice employs patterns, idioms and components to handle complexity, but shortcomings in current object-oriented languages limit the effectiveness of product line development. Subject-oriented programming and more recently multi-dimensional separation of concerns promise improved support for product line development. Ideally, a product line can be composed of slices of an overall system that provide low coupling among components, good separation of unrelated concerns and improved understandability of the system structure. In this paper we describe our experiences on applying subject-oriented programming to product line development.

Tools and Visualization

Identification of and encapsulation according to multiple dimensions of concern simultaneously introduces the need for tools that perform a variety of functions, including to find, display, identify, extract, analyze, separate and compose concerns. It also opens up rich possibilities for visualizing software in flexible ways based on different dimensions of concern at different times, not constrained by any dominant decomposition.

Bill Griswold, "Aspect Browser: Tool Support for Managing Dispersed Aspects."

Although modularization, if used properly, separates the concerns of primary design decisions, it often fails to cost-effectively separate lower-order design decisions. These lower-order decisions may cross-cut the primary module structure. Changes to these cross-cutting design decisions tend to be more costly since they are dispersed throughout the system and tangled with the primary design decisions and each other. When the code relating to a particular change is not localized to a module, an *information-transparent* software design allows a programmer to use available software tools to economically identify and quickly view the related code, easing the change. That is, the "signature" of the changing design decision can be used to approximate the benefits of locality, in particular providing a way to quickly view and compare the elements of the cross-cutting aspect without distraction from inessential details. This signature is one or more shared characteristics of the code to be changed, such as the use of particular variables, data structures, language features, or system resources. Since the intrinsic characteristics of a design decision can be incidentally shared by unrelated code, it is helpful if the programmer has adopted distinguishing conventions such as stylized naming of identifiers.

Anthony Finkelstein: "Consistency Management of Distributed Documents using XML and Related Technologies"

In this talk I will describe an approach to managing consistency of distributed documents. I will give an account of a toolkit which demonstrates the approach. The toolkit supports the management of consistency of documents with Internet-scale distribution. It takes advantage of XML (eXtensible Markup Language) and related technologies. The talk will include a brief discussion of the base technologies, a discussion of related work and a demonstration. The approach and the toolkit will be described in the context of a typical application in the area of software engineering.

REFERENCES

1. Mehmet Aksit, Lodewijk Bergmans, and S. Vural. "An Object-Oriented Language-Database Integration Model: The Composition Filters Approach." Proceedings of ECOOP'92, Lecture Notes in Computer Science #615, 1992.
2. E. P. Andersen and T. Reenskaug. "System Design by Composing Structures of Interacting Objects." Proceedings of the European Conference on Object-Oriented Programming (ECOOP), 1992.
3. Elisa L.A. Baniassad and Gail C. Murphy. "Conceptual Modules Querying for Software Reengineering." In Proceedings of the International Conference on Software Engineering (ICSE 20), April 1998.
4. Don Batory, Gang Chen, Eric Robertson, and Tao Wang, Design Wizards and Visual Programming Environments for GenVoca Generators, IEEE Transactions on Software Engineering, May 2000, 441-452.
5. Don Batory, Clay Johnson, Bob MacDonald, and Dale von Heeder, "Achieving Extensibility Through Product-Lines and Domain-Specific Languages: A Case Study", International Conference on Software Reuse, Vienna, Austria, 2000.
6. Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Reading, MA, June 2000.
7. D. D'Souza and A. C. Wills. *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison-Wesley, 1998.

8. Martin L. Griss. "Implementing Product-Line Features with Component Reuse," Proceedings 6th International Conference on Software Reuse, Vienna, Austria, June 2000.

9. W. Harrison and H. Ossher. Subject-oriented programming (a critique of pure objects). In *Proceedings of the Conference on Object-Oriented Programming: Systems, Languages, and Applications*, pages 411–428, September 1993. ACM.

10. I. M. Holland. Specifying reusable components using contracts. In O. L. Madsen, editor, *ECOOP '92: European Conference on Object-Oriented Programming*, pages 287–308, Utrecht, June/July 1992. Springer-Verlag. LNCS 615.

11. M. Jackson. Some complexities in computer-based systems and their implications for system development. In *Proceedings of the International Conference on Computer Systems and Software Engineering*, pages 344–351, 1990.

12. Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin. "Aspect-Oriented Programming." In proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241. June 1997.

13. Karl Lieberherr, *Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns.* PWS Publishing Company, Boston, 1996.

14. Mira Mezini, "PIROL: A Case Study for Multidemensional Separation of Concerns in Software Engineering Environments." In Proceedings of the Conference on Object-Oriented Programming: Systems, Languages, and Applications (OOPSLA), pages 188-207, October 2000.

15. Mira Mezini and Karl Lieberherr. "Adaptive Plug-and-Play Components for Evolutionary Software Development." In Proceedings of the Conference on Object-Oriented Programming: Systems, Languages, and Applications (OOPSLA), October 1998.

16. Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. "A Framework for Expressing the Relationships Between Multiple Views in Requirements Specifications." In *Transactions on Software Engineering*, vol. 20, no. 10, pages 260-773, October 1994.

17. David L. Parnas. "On the Criteria To Be Used in Decomposing Systems into Modules." *Communications of the ACM*, vol. 15, no. 12, December 1972.

18. Peri Tarr, Harold Ossher, William Harrison, and Stanley M. Sutton, Jr. "N Degrees of Separation: Multi-Dimensional Separation of Concerns." In *Proceedings of the 21st International Conference on Software Engineering*, pages 107–119, May 1999.

19. C. R. Turner, A. Fuggetta, L. Lavazza and A. L. Wolf. Feature Engineering. In *Proceedings of the 9th International Workshop on Software Specification and Design*, 162–164, April, 1998.

20. M. VanHilst and D. Notkin. Using roles components to implement collaboration-based designs. In *Proceedings of the Conference on Object-Oriented Programming: Systems, Languages, and Applications*, pages 359–369, October 1996. ACM.

21. Robert J. Walker, Elisa L.A. Baniassad, and Gail C. Murphy. "An Initial Assessment of Aspect-oriented Programming." In Proceedings of the International Conference on Software Engineering (ICSE 21), May 1999.

22. Workshop Web site: http://www.research.ibm.com/hyperspace/workshops/icse2000

Editor's Filler

Space to fill!

Not like the top of my desk :-)

.... or my hard drive.

I suppose I should say a few words about FSE-8 in San Diego

... the only ones that come to mind are

"very nice."

Of course there were the missing (I mean mis-placed) boxes of tutorials and proceedings, but John Knight and David Rosenblum should be congratulated for putting together a fine conference.

John and David couldn't have done it without the help of Debra Brodbeck and Peggy Reed.

And thanks to everyone who attended.

Next year FSE will be in Vienna with ESEC in September... a lovely city if you have not been there before, if I do say so myself!