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 [5] 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* [7], like printing, persistence, and display capabilities, are also common concerns, as are *aspects* [3], like concurrency control and distribution, *roles* [1], *viewpoints* [4], 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" [6]: 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, impossi-

ble 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" [2].

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 represents a set of very ambitious goals. They apply to any software development language or paradigm. Recent approaches [8] go some way towards satisfying these goals in various ways in various contexts. Considerable research is still required, however, before any approach fully achieves the goals. 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. Material and links related to the workshop, including position papers and contact information for the organizers, are available at the workshop Web site [8].

REFERENCES

1. 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.

2. 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.

3. 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.

4. 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.

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

6. 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.

7. 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.

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