How Do Teams Shape Objects? – How Do Objects Shape Teams?

Panel Session

Steven Fraser, Kent Beck, Grady Booch, Derek Coleman, Jim Coplien, Richard Helm, and Kenny Rubin

Background

Each panelist was asked to provide his definition of team and a position statement reflecting on:

Resolved: *That Teams Affect Objects, But Objects Do Not Affect Teams.*

For the purpose of initial positions, panelists were grouped by the chair as:

For the proposition:
Derek Coleman, Jim Coplien, and Kenny Rubin

Against the proposition:
Kent Beck, Grady Booch, and Richard Helm

Steven Fraser
Panel Chair

Bell-Northern Research Ltd.,
Ottawa, Ontario, Canada.
sdfraser@bnr.ca

Teamwork is the most important key to success in almost all software projects. What makes a good team (roles, organization, education, communication bandwidth, management, process, etc.) remains an open question in many situations. To quote Gerald Weinberg from *Understanding the Professional Programmer* (Dorset House, 1988): *The simple fact is, that we don't know how to assess the "difficulty" inherent in any meaningful problem. In every useful experiment on programming so far performed, where several programmers or teams worked on the "same" problem, there have been variations of 10 to 1 or more in performance. Usually 30 to 1 or 50 to 1 is a more typical figure. My own conclusion [Weinberg's] from this is that in many cases the concept of "problem difficulty" makes no sense as a measurement. The difficulty seems to be a relationship between the problem and the approach to solving it.*

The goal of this panel is to examine how teams affect objects and how objects affect teams. Why should a chicken-and-egg challenge be worthy of debate? Two related questions at the heart of the matter arise: Firstly, is there a team organization that can facilitate object system development? Secondly, is there an object organization that makes teams more effective? The assumption is that insight into these questions will improve the state of the practice.

According to Peter Senge *et. al.* (*The Fifth Discipline Fieldbook,* Doubleday 1994) *the word team can be traced back to the Indo-European word "deuk" (to pull); it has always included a meaning of "pulling together." ... We* [Senge, *et. al.*] *define "teams" as any group of people who need each other to accomplish a result.*

Tasks performed by a team can be characterized broadly as generation (planning), selection (decisions), negotiation (trade-offs), and execution (implementation). Effective team performance is dependent on a collective strategy, group motivation, and focused capability that matches skills, roles, and responsibilities to the set of tasks.

Biography

Steven Fraser is on staff at Bell-Northern Research's Computing Research Laboratory in Ottawa. He is currently on assignment in Pittsburgh at the Software Engineering Institute (SEI) collaborating with the Application of Software Models project on the development of domain engineering techniques.

Since joining BNR in 1987, Fraser has contributed to the ObjecTime project, an OO-based CASE-Design Tool and to the BNR BCS software development process. Fraser completed his doctoral studies at McGill University in Electrical Engineering. He holds a Master's degree from Queen's University at Kingston in applied Physics and a Bachelor's degree from McGill University in Physics and Computer Science. Fraser is an avid photographer and opera buff.

Kent Beck
FirstClass Software
70761.1216@compuserve.com

The most important remaining barrier to programmer productivity is the bandwidth of human communication. I am exploring an approach to improving communication called patterns, which encode common usage in simple phrases. Objects are an ideal target for patterns because they encourage deferring some design decisions until much later than other programming methods.

Biography

Kent Beck has 10 years of experience in objects. He is best known for his contributions to the CRC object finding technique and for applying pattern designing and building objects.

Grady Booch
Rational Software Corporation
egb@rational.com

For the purpose of this debate, let me broadly define "team" to include those members of an organization that are either directly responsible for writing software or for supporting those who do. Thus, my definition would include programmers, analysts, toolsmiths, program managers, quality assurance personnel, testers, writers, designers, and architects. Quite intentionally, this definition includes anyone who has an impact upon the success or failure of a software project, particularly as it relates to creating tangible products and to mitigating software risks.

I first concede the point that teams affect objects. Software folklore, backed up by a few legitimate studies involving some hairy mathematics, clearly shows that the structure of a software product - either good or bad - is directly impacted by the structure of the team that created it. Consider, for example, one Really Large object-oriented project whose analysts were housed in one building, and whose developers were housed in another. I for one was not surprised to see this project turn into a classic case of software meltdown. Similarly, the hyper productive projects I've encountered are all characterized by having an open, elegant, and high-energy team structure - and their software products reflect these same characteristics.

However, it is painfully evident that objects affect teams (and, not to reduce this debate to personal attacks, my esteemed opponents in this debate are clearly pond scum if they consider otherwise). Consider the structure of an organization that develops software for a mainframe COBOL application, versus one that develops software in C++ or Smalltalk for a client/server application. In the successful project, there exist very different roles and responsibilities among the members of each of these teams. Indeed, there are certain roles in the object-oriented project that have no analog in the non-object-oriented one. For example, a typical object-oriented project will include architects (who invent clusters of classes and mechanisms), abstractionists (who are skilled at discovering classes and objects), and application engineers (who take the components developed by the architect and the abstractionists and assemble them into applications).

I'll state my point even more directly: attempting to develop a complex object-oriented software system with a non-object-oriented team will add significant risk to the success of any project.

Biography

Grady Booch is Chief Scientist at Rational. He has been with the company since its foundation in 1980. Booch has pioneered the development of object-oriented analysis and design methods. His work centers primarily around complex software systems. Booch is the author of three books published by Benjamin - Cummings, including *Software Engineering with Ada* and *Software Components with Ada.* As a derivative work to his second book, Booch developed foundation class libraries written in Ada and C++. His third book, titled *Object-Oriented Analysis and Design,* describes the theory, notation, process, and pragmatics of object- oriented technology. He is currently working on a fourth book, dealing with the management of object-oriented projects. He has also published more than 75 technical articles on object-oriented technology and software engineering. Booch has lectured on these topics at numerous conferences and workshops in the United States, Europe, and the Pacific Rim.

Booch is a Distinguished Graduate of the United States Air Force Academy, where he received his B.S. in Computer Science in 1977. He received an M.S.E.E. in Computer Engineering from the University of California at Santa Barbara in 1979. Booch is a member of the American Association for the Advancement of Science, the Association for Computing Machinery, the Institute of Electrical and Electronic Engineers, and Computer Professionals for Social Responsibility.

Derek Coleman
Hewlett-Packard Laboratories
Palo Alto California
dc@hplsrd.hpl.hp.com

From experience, I believe teams are more important than objects for getting successful software development. Establishing a shared vision and putting the organization in place to make it happen is worth an awful lot of class frameworks - whoever wrote them.

Most of us are in the business of delivering products, thus products, too, count more than objects. Products might be built from objects, but most customers do not care much about objects. Because object teams should focus on product delivery, the old roles are still applicable - the analyst who defines the required product; the designer who builds the architecture and the implementer who codes the architecture.

Prototyping is a good reason for using objects, but prototyping was not invented by objects. So again, object teams adopt the old roles: the analyst prototyper who helps resolve what the customer wants and the design prototyper who makes "proof of concept" models for difficult technical issues. We should not forget the other key role: the person who makes sure that prototypes are not delivered as products!

Software reuse is a much vaunted reason for adopting object technology - but one can get reuse without objects. Plans which call for asset producing teams and asset consuming teams are very ambitious. Unless the domain is very well understood it is hard to produce a stable set of really reusable object classes. Even then a large investment is required to ensure that the objects are truly generic and of high enough quality. Then there is business problem of how to get the return on that investment. Finally, even if all these issues are under control, when push comes to shove, product schedules have a tendency to override asset production.

Probably the most practical form of reuse is the incremental development and delivery of product families, i.e. reuse constrained by the schedule of delivering products. Incremental delivery relies on having a clear definition of each product release and then using the product release schedule to drive the scheduling of the architecture design and the code production.

To plan a product-family planning the features of each release need to be precisely specified early on during OOA. (Note: A feature is a stimulus-response pair such that if the user provides a stimulus then the product responds by changing its state and outputting a response.) Precise specifications of features allows them to be prioritized and grouped together to constitute the releases.

Features have dynamic behavior which has to be mapped on to object interactions (the "key mechanisms", contracts, etc.) during OOD. Thus an incremental delivery team must work with a notion of architecture that includes object interactions as well as the usual class hierarchy or relationship model. Of course, if the software is to be multiprocess then each process must also be specified by the architecture.

To keep the development on track the architecture must be always visible to the whole team. Its development needs to be the responsibility of the system architect role whose job it is to maintain the architecture and ensure that it remains consistent and that all changes are propagated to other team members.

So whichever way I look at it - it seems that it is the team that affects objects and not the other way around. In fact, an even stronger form of the proposition might hold: "teams are in business to deliver products, product plans drive architectural development and the architecture affects the objects".

Biography

Derek Coleman holds a B.Sc (Physics) and an M.Sc (Computer Science) from the University of London. He is manager of the Application Engineering Department of Hewlett Packard Laboratories, Palo Alto.

Before relocating to Palo Alto in 1994, Derek was a project manager in the HP Labs Bristol Research Centre in UK, where he led a team researching into object-oriented analysis and design techniques. He is a co-author of *Object-oriented Development: the Fusion method* published by Prentice-Hall in 1993. Derek is an active member of the object-oriented research community and is the author of many papers on software engineering and formal methods.

Organization and Objects: Are They Separable?

James O. Coplien
AT&T Bell Laboratories
cope@research.att.com

A team is a group of people who work together toward a common goal, driven by a complementary set of instincts and talents. As we use the term in software, these people are closely coupled to each other.

I acknowledge a wide variety of design techniques and programming languages that support a spectrum of design approaches that we collectively call "the object paradigm," but they all build on encapsulation, instantiation, and polymorphism, and usually on inheritance. A class is a unit of encapsulation; an object is an instantiation of a class. The power of the object paradigm is in the abstraction it provides through inheritance, and the polymorphism that makes inheritance invisible at run time.

Objects appear to be neither empirically sufficient nor necessary for teams. In our organizational studies done by AT&T, we have seen highly productive teams that use object-oriented techniques (such as Borland's QPW effort), but have seen equally productive teams that use vanilla C. We have also seen object-oriented efforts that lack effective teamwork.

To echo Beck's position, communication is key to effective teams. With the advent of objects came the promise of well-documented interfaces that communicated behavioral intent while hiding implementation details. Experience has shown that design is more subtle than that. Abstractions beyond objects dominate the subtleties of design. These include the mechanisms of Booch, the idiomatic forms of handle/body classes, iterators, and the like, and the patterns of message flow in a multi-process or distributed systems. I believe many of these abstractions can be effectively communicated as patterns.

Software architecture supports organizational communication only to the extent it leverages Conway's law: that the software and organization structures are mirror images. But human communication and sociological grouping have many limitations and quirks for which there are no software drivers. We have found that people can maintain about 5 long-term relationships (collaborators) in a development organization. We find characteristically even patterns of coupling in productive organizations, with no centralized control and no bottlenecks. We find that good organizations have characteristic 'shapes' of role grouping. In short, we understood good organizations by their patterns, too.

If you step back and look at these organizational principles, they may look familiar as OOD rules-of-thumb, if you substitute objects for people. And they are! But the reason they make such good rules of thumb for software design is because of what they portend for the people writing the software; the objects don't care about such things. For this reason, I believe that suitable organizational patterns are key to effective teams. In a given project, these patterns might mirror Beck's software patterns, but the focus should be on the people issues. Such patterns comprise not only lines of communication, but aspects of the reward and value system as well. For example, can the organization learn and introspect? This is reminiscent not only of the reflection work in the OO community, but also of Frank Buschmann's use of reflection in his patterns.

Biography

Jim Coplien is a member of the Software Production Research Department at AT&T Bell Laboratories, where he does research in patterns of world-wide software development organizations. He is also known for his exposition of advanced design and programming techniques in *Advanced C++ Programming Styles and Idioms.* He writes a software pattern column for the *C++ Report.*

Richard Helm
DMR Group.
1200 McGill College Ave.,
Montreal. QC H3B 4G7. Canada
lmcrihe@LMC.ericsson.se

Approaches to organizing teams include organization by deliverable to be produced, or organization by specialization and specific activities. Deliverable-based teams encourage cross fertilization, accountability, responsibility for deliverables across development phases, creativity, and team stability.

Object-oriented technology impacts deliverable based team structure because of the deliverables unique to object-oriented development. Analysis and design object models, class hierarchies, tool kits, and frameworks for example are appropriate deliverables around which to organize a team.

Designs based on frameworks can have a large impact on team structure. Typically some team members will be responsible for the design and implementation of the key framework abstract classes, their interfaces and the patterns of behavior of contracts between them. Once the design and implementation of the framework proceeds then there is an opportunity to develop "stripes" of functionality in parallel by other team members. A deliverable becomes a set of sub-classes of framework classes with each adding some part of a larger functionality. For example, in a direct manipulation graphical editing framework such as HotDraw or Unidraw, a team member can be made responsible for implementing the stripe consisting of the Circle class and all Tools, Commands and Manipulators that operate on and are appropriate to Circles. New functionality will often appear as subclasses of the framework and can generally be developed in isolation from other team members developing parallel stripes.

A disadvantage of deliverable-based organizations is that it can require management of deliverables that are larger than activity-based deliverables. Deliverables requiring months of effort are harder to monitor than the activities that produce the deliverable. Object-technology helps in that the object-oriented systems often are decomposable into relatively small pieces. Classes and methods are good examples. Indeed the benefits of encapsulation and polymorphism and well defined interfaces carry over to team structure. A well designed system which has well defined interfaces, the classes are encapsulated etc., encourage teams which are independent, and "encapsulated".

However, the distribution, atomization, and localization of control flow into classes in OO designs pose the danger that the "global flow" of the application is not anyone's deliverable. This is important, particularly when designing frameworks. The application's control flow, and the interfaces and relationships between, and functionality of, classes which participate in this flow of control are key to the ultimate flexibility, evolution and reuse of a design. Use cases are also distributed across multiple classes. Care must be taken to ensure there are team members who are assigned to these deliverables. If not, the "soul" of the application may be lost. Some deliverables span classes and team members; team members may be responsible for parts of many classes. This gives potential for conflicts and "hot-spots" as multiple team members have responsibility for a single class.

Biography

Richard Helm has been working with object-technology for the past five years. Currently, Richard is a Senior Technology Consultant specializing in object-technology with DMR Group, an international information technology and strategy consulting firm. Prior to DMR, Richard was a research staff member at IBM's T.J. Watson Research Center in New York, investigating object-oriented design and reuse, and setting research directions for object-technology. Richard has numerous international publications in object- technology and was a member of the ACM OOPLSA program committee in 1992. Richard has a Ph.D. and a B.Sc. in computer science from the University of Melbourne, Australia.

Kenneth S. Rubin
ParcPlace Systems
krubin@parcplace.com

I have worked with clients over the past six years to create and manage their project teams. In addition, as part of writing our book, *Succeeding with Objects: Decision Frameworks for Project Management,* we have conducted 39 case study interviews of projects and teams that have used object-oriented technology. These experiences are the background for my position.

A team is a group of people who work together in a coordinated way to meet a clear set of goals and objectives. A team has a purpose, to carry-out the activities of a project. Just as there are many different types of projects, so are there many different types of teams to carry out these projects. The purpose of the team guides the decisions about the team roles, management approach, communications, and structure. Within an organization the following decisions should be made regarding teams:

- decide which teams are needed
- identify the roles needed on each team
- decide on the style for managing each team
- determine inter/intra-team communication mechanisms
- decide on the structure for each team
- find team members.

I have seen four types of teams commonly employed on object-oriented projects:

Application Team. A team responsible for the analysis, design, implementation, delivery, and sometimes maintenance of an application that fulfills a contractual obligation with either an internal or external client.

Framework Team. A team dedicated to the construction of reusable frameworks and components to support one or more application projects.

Cross Project Team. A team whose purpose is to facilitate the sharing of project artifacts among a collection of simultaneous projects.

Reuse Team. A team that is responsible for executing the organization's reuse process model.

In addition, within and across these team we have seen the following new team member roles:

Analysis Prototyper. Develops the executable prototypes during the analysis phase.

Design Prototyper. Develops the executable prototypes during the design phase.

Object Technology Expert. Provides object-oriented technology expertise at several levels.

Object Coach. General resource, available to answer all team members' questions about the use of object-oriented technology in the project.

Framework Designer. Determines the architecture for a general description of parts and how they interact to form applications within a specific domain.

Reuse Evaluator. Determines whether or not software components have been designed for broad applicability.

Reuse Manager. Has overall responsibility for the reuse process model.

Reuse Administrator. Responsible for identifying and acquiring new reusable assets for the corporate or project library.

Reuse Librarian. Responsible for certifying, classifying and storing new reusable assets into the corporate or project library.

Reuse Expert. Creates, maintains and updates reusers of the reusable assets.

Biography

Kenny Rubin is Manager of Methodology Development at ParcPlace Systems where he manages the development of ParcPlace's Object Behavior Analysis and Design (OBA/D),and Project Management Methodologies and Tools. Previously, he was Manager of Professional Services at ParcPlace where he directed ParcPlace's consulting and training business.

Kenny has co-authored a book on managing object-oriented software projects and has numerous other publications on the topics of managing object-oriented projects, object-oriented analysis and design, artificial intelligence and human-computer interaction. He has publicly spoken on these topics over 100 times at major corporations and conferences around the world. In addition, he is a member of the Editorial Board of Object Magazine and the OMG OOA/OOD SIG, as well as a faculty member of the Stanford University Western Institute in Computer Science (WICS).

Kenny received his B.S. in Computer Science from the Georgia Institute of Technology and his M.S. in Computer Science from Stanford University.