OO Anthropology: Crossing the Chasm

Panel Session

Steven Fraser, Alistair Cockburn, Leo Brajkovich, Jim Coplien,
Larry Constantine, Dave West

Abstract: Anthropology is the study of civilization, particularly its societies, customs, structure, and evolution. Our premise is that there are cultural "chasms" to be crossed to ensure the success of the technological beach-head established by innovators and early adopters of the OO paradigm. Our panelists will address the following questions:

- What anthropological cultural factors have to be "matured" (and how?) to foster the success of the OO paradigm?
- What mechanisms are required to facilitate communication between cultures of differing maturity?
- What cultural chasms must be crossed to develop a successful organization/culture within the framework supported by the object-oriented paradigm (or more perversely, is there anything "special" about the OO paradigm)?

This panel will interest practitioners as a forum to share and debate experiences related to our current-day software culture metamorphosis.

Leo Brajkovich

Similar to the pan-human study of kinship, OO Anthropology has to decipher the roles and relationships that are important to the development and success of OO projects. Once deciphered, these roles must be staffed, developed, and managed similar to any other position in an organization. Past research suggests that some group structures (and management styles) are better suited to certain types of work. For instance, hierarchical group structures are better suited to stable, somewhat routine tasks, while egalitarian (decentralized) structures are better suited to highly variable, complex tasks. Since OO development can involve problems defined all along the continuum of stable/routine to variable/complex, its study will require mapping the relationships between work group members and analyzing the roles that are key to group success. Recent research on emergent roles in high technology start-up organizations suggests that "liaisons," or individuals that possess knowledge and skills spanning various disciplines and are "linked" in the social network to various emerging groups, are a valuable catalyst to team functioning and superior performance. Research also suggests that liaisons are typically unacknowledged/unperceived and systematically under- appreciated by managers. As more and more work group situations are understood in this manner, and our systematic knowledge is increased and capitalized on, we shall cross the chasm.

Leo Brajkovich is a Consulting Sociologist and Project Director with International Survey Research Corporation (ISR). He has directed many employee an management research projects for Fortune 500 companies. His research focuses on employee and organizational cognition, organization culture(s), work group dynamics and functioning, and overall organizational effectiveness. His published work has appeared in academic and practitioner journals. In 1992, he received an Award for Excellence from the Pacific Division of AAAS. In addition to a Ph.D. in Social Science from the University of California, Irvine, he holds Master's degrees in Mathematical Behavioral Science and Sociology, and Bachelor's degrees in Physics and Sociology. Prior to ISR, he worked at the RAND Corporation, Ultrasystems Incorporated, and the Hughes Aircraft Company.

Jim Coplien

What anthropological cultural factors have to be "matured" (and how?) to enable the OO paradigm?

Little of software culture relates to paradigm per se. At some level, we have confusion in language and terminology and divergence in ceremony (methodology). I see these as issues of healthy individuation, and don't raise them to the level of culture as much as others do. And I don't think culture aligns as closely with paradigm or method as anyone would have us believe.

What mechanisms are required to facilitate communication between cultures of differing maturity?

Trust is a great communication enabler. For two cultures to communicate with each other, each must be convinced the other has something to offer. In the worldview of software paradigms, that means letting go of part of my worldview and making room for others. Gosh, maybe there *is* room for databases in my OO Weltanschaung! Next, I must go beyond the acceptance to understanding: exactly where can databases help me? Those who can let go of their OO identity and broaden to other cultural models can help lead individual cultures to broader perspectives. My own work in multi-paradigm design might be one example of such an effort.

Language can be a powerful tool to bridge cultures, and programming language is no exception. C++ evidences carefully considered compromises designed to bridge the C culture with an emerging object culture. The compromise is a turn-off to those who wish to remain culturally isolated, but its wide acceptance speaks clearly to the success of the technique. Should our future languages do this?

The issue of disparate maturity is a hallmark of technological trades. We must learn to acknowledge the value of technologies whose heyday is past, displaced by successive sources of hopes for panaceas. That's a tough row to hoe, but patterns have made interesting inroads in this area. I'm encouraged by the progress we've made.

What cultural chasms must be crossed to develop a successful organization/culture within the framework supported by the object-oriented paradigm (or more perversely, is there anything "special" about OO)?

I think that there is neither anything particularly special about OO, nor that this is the only goal towards which we should strive. We should continue to work on convergence in vocabulary, share understanding of recurring solutions (folklore), but not in method, process, or other ceremonial attributes of the object culture. These haven't proven themselves terribly useful. Furthermore, we should reach out to other disciplines and cultures to integrate with them, including cultures outside of computer science and altogether outside of technology. We can learn much from crafts- people and artists. I think we'll have fully appropriated the cultural aspects of the object paradigm and other facets of software culture when we cease to think of them as sciences, and start thinking of them as art. In fact, that step may be a necessary precondition to the beginnings of meaningful cross-cultural dialogue.

Jim Coplien is well-known as an author and speaker in the areas of advanced C++, object-oriented design, and software patterns. He is a principal investigator at Bell Labs, where his research agendas include empirical studies of software development organizations, multi-paradigm design, and legacy software pattern mining. He is one of the editors of the PLoP book series (two volumes), and is author of Advanced C++ Programming Styles and Idioms, *of the Software Patterns White Paper, and of a forthcoming book on multi-paradigm design. He writes a regular column on patterns, is on the program committee of ECOOP '96 and COOTS, and the program chair for OOPSLA '96. His extracurricular interests include genealogy and Aikido.*

Larry Constantine

Cultural anthropology since the 19th century has been systematically shedding the vestiges of a worldview that saw some cultures (usually Western European in origins) as more advanced or mature than others. Despite their appealing conceptual elegance, linear models of progressive maturation within the human sciences, whether in child development, moral reasoning, group development, language acquisition, or family development, have steadily declined in influence and favor among scientists if not practitioners. Under the light cast by more sophisticated research and emerging empirical evidence, the inherent, multi-threaded complexity of human developmental patterns has had to be recognized.

The field of computer software engineering, however, remains enthralled by simplified, stagewise notions of progress and maturation, most notably the group-oriented Capability Maturity Model, and the individualized Personal Capability Maturity Model developed by the Software Engineering Institute at Carnegie Mellon. This notion of advancement is also implicit in much of the writing on the emergence and evolution of the "object-oriented paradigm," which is seen to be a later and more highly developed view of software and applications development than the traditional structured or procedural methods and models it purports to replace.

Having both actively advocated the adoption of object technology and publicly predicted its eventual triumph as a conceptual and practical tool within the discipline, I nevertheless remain a skeptic as to its paradigmatic status. As such, of course, I am regarded by some as an irredentist of questionable repute.

If it is a paradigm in the strictest Kuhnian sense, then object orientation is grounded in implicit and unexamined assumptions that can only be fully understood from without. These tacit foundations not only color the cultural interpretations of its adherents, but can historically and dialectically be anticipated to contribute to its own eventual replacement, not by something necessarily superior or more "mature" but most certainly by something different that will likely incorporate its most tenable tenets as a subset. Only to the extent that the evolution of practice and theory in software engineering begins to be more informed by empirical investigation and systematic research can we anticipate that stepwise progress will begin to play a greater role than that now played by the charismatic personality, market dominance, and conceptual rhetoric that have thus far dominated the debate. If we are to sort out questions of cultural maturity we must look both backwards to the historical roots of competing concepts and forward to an invigorated program of investigation that supplements if not supplants philosophical speculation with generous doses of observation, measurement, and accurate reflection on the historical record.

Larry L. Constantine is one of the pioneers of software and applications development methodology who has also made recognized contributions in the human sciences, including original ethnographic research and contributions in human systems theory. A graduate of the Sloan School of Management at the Massachusetts Institute of Technology, Constantine has published more than a hundred papers and eight books in several fields, including, most recently, Constantine on Peopleware *(Prentice Hall, 1995), based on his popular column in Software Development, and the software engineering classic, Structured Design, written with Ed Yourdon. His current research and consulting interests include software usability and the role of work culture in high-performance technical teamwork and organizational change. He is a Professor of Computing Sciences at the University of Technology, Sydney, and Principal Consultant with the international consulting firm of Constantine & Lockwood, Ltd.*

Alistair Cockburn

Value conflicts exist legitimately on software projects. Therefore, before intervening to change a group's values, we should create a map of the value systems. Ethnographic methods are necessary, but insufficient, since experience is needed to interpret the situations. I should like to call on experienced developers to apply ethnographic techniques to capture the value maps of the different parties, and then open discussion on intervention strategies. What does that all mean?

In software development, different cultures work side by side. They include the executive sponsors, the marketing group, the managers, the users, analysts, UI designers, programmers, database designers, testers, etc. There are groups along other dimensions: the novices vs. the experts, the mathematical vs. intuitive designers, etc. Each group encourages its own values. Their values are sometimes in conflict. The cultures have forms of repulsion and protection for their own survival. Their values help them contribute to the project...usually, but occasionally not.

The usual approach on discovering value dissonance is to try to change the cultures. This means trying to get the project manager to let the programmers develop iteratively "until they are done". Or trying to write a one-pass schedule containing no time for learning and revision. It means asking the users or the manager to become expert on object technology, or asking informalist designers to use formal notations. The groups try to change each other.

And yet, nobody has yet described the map of the values of the groups. What is the successful part of a group's strategy? A classic example is programmers trying to keep the manager away and trying to stay out of meetings. This "antisocial" behavior is part of gaining think time and reducing distractions. It became part of the programmers' culture, a protection and a success strategy. What is a successful-unsuccessful value? It is the strong aversion to learning and using other groups' results, the "invent here and now" mandate. That value propels careers, but operates at the expense of the industry.

What else contributes to each group's success? Which of the values and cultural characteristics can safely be changed? Where is there harmony, where is there conflict between cultures? Where is the map of these different value systems?

Rather than walking into a project asserting that group A or B needs to use method / technique / tool / work style X, or even, walking into a project asserting that they all should be tolerant of each other, I should like to stake the position that we should spend some time finding out what the groups need, and how they already function.

What do software developers really need in the way of support for their work? I have not seen a decent answer to that question, nor even a decent proposal to research it since the brief efforts of MCC in the 1980's. Yet tool vendors, methodologists, and even researchers constantly suggest new solutions to the purported problems of the software practitioner.

Here I offer a mea culpa. I developed advanced software environments for several years, thinking I was really paying attention to the way my target group worked. It slowly became clear that although I was an experienced software designer, my knowledge of what the target group really needed was neither accurate nor based on facts. It was based on my guesses and wishes. I dropped environments for methodology to better learn developers needs. After several years studying projects and writing methodology, I again discovered I had made a guess on needs and values. This is a difficult topic. I have made three attempts of several years each to address the question. Each time, I discovered that I did not know what the real problem was, that my suggestions were based on guesses and wishes, not facts. On a recent project, I finally began to notice the astonishingly complex and fluid set of working relationships across the project staff. It was too difficult to document, but it began to show some useful information. It also revealed the need for proper ethnographic studies of development groups.

However, ethnographers have their own values, one of which seems to be excitement about studying cultures they do not understand at all, as opposed to ones they already understand somewhat. The flaw with this value is that they miss important cues. I am staking the additional position that experienced developers are needed to learn and apply ethnographic techniques. They will be able to spot important clues that others would miss, and will be able to represent the culture's values. What does it take to address the question?

First, recognize that there are cultures on a project. They have cultural values, the values differ, the value systems of each have utility. We can get a sense of the value structures even without detailed value maps.

Second, find people who can work on both sides of pairs of cultures. They are needed to communicate, protect the essential cultural values and translate across major differences.

Third, inform each of the groups on the purposes, rationales, and values of the others. The groups devalue each other too often. As Jim Coplien said, for two cultures to communicate with each other, each must first become convinced the other has something to offer.

Finally, apply ethnographic techniques to study each group, discovering their strengths and weaknesses. Software groups have been producing software successfully for decades. Find out what works before naming changes. Use experienced members of each group to create their value maps

In short, lay some facts on the tables, instead of the usual wishes and guesses.

Alistair Cockburn is methodologist and Consulting Fellow at Humans and Technology. He designed the OO methodology for the IBM Consulting Group using some of the techniques described in this position statement. These techniques resulted in a methodology, a book on surviving OO projects, and the start of a risk management catalog. Cockburn is a practitioner of double-loop learning, and seems to particularly enjoy sitting underwater, traveling, and forgetting the languages he took a lot of trouble to learn.

Dave West

Dr. Constantine pointed out the connection between a paradigm (in the sense advanced by Thomas Kuhn) and a culture. Both are examples of a group of people with shared "technology, ideas, values, and world views."

In contrast to Dr. Constantine, I am not at all skeptical about the status of OO as a paradigm. I would suggest, however, that it is not necessarily a "new" paradigm. Rather it is merely the most current (and perhaps most advanced) incarnation of a long-standing philosophical paradigm. I believe that the questions posed for this panel (and their answers) will be better understood if the larger philosophical and cultural context is exposed.

Western culture (especially since the "Age of Enlightenment") has been dominated by a "Formalist" paradigm or philosophy. Members of this culture (paradigm) have shared values and a world view that includes: notions of control, of centralization, and of the ultimate good of mathematics, logic, and deterministic prediction.

The shared technology - the instrument that has contributed most to the ascendancy of this culture - is the digital computer. The central ideas can be traced from Descarte, Leibniz, and Hobbes, through Babbage, Turing, and von Neumann, reaching their "pinnacle" in the early work in Artificial Intelligence by Newell, Simon, and Minsky. A practical side effect of this tradition was the rise of "structured" methods and software engineering.

Contrarian subcultures exist, most notably the "Hermeneutic" tradition that asserted complexity, unpredictability, interpretation, and emergence as appropriate values and world view concepts. Contemporary evidence of this subculture can be seen in the area of Neural networks (emergence), Complexity Theory (self-organization, non-determinism), "postmodern" critiques of computer science, the new biology of Maturana and Varela and its importation into computer science and AI by Winograd and Flores. Perhaps the most obvious example of this subculture is object-orientation.

Before wrestling with the notion that object people are members of this subculture - something that may come as a surprise to many people in this audience - it should be noted that anthropological theory shows this same philosophical (paradigmatic) split.

Many anthropologists spend their time searching for the "rules of culture" or the "marxian infrastructural principles that determine culture." These individuals, until recently, comprised the majority of anthropological theorists. The minority view in anthropology is expressed by people like Clifford Geertz and more recent work on "cognition in the wild" by anthropologists like Lave and Shore.

Whether or not "objects" constitute a separate culture is an interesting question because the answer (a definitive yes) is obscured in two ways. First, sloppy use of language, almost any practice, method, or language can be labeled as "object oriented." Second, and most importantly, when it comes to software both object oriented and traditional systems must be implemented on the same physical hardware.

When it comes to a binary string or a set of hardware op codes it is impossible to distinguish the string or the op code that originated in an "object culture" from one that originated in a traditional "structured culture." A Smalltalk method carries not distinguishing characteristics that differentiate it from a COBOL subroutine.

Objects are different only in the gestalt and in concept. The differences in values and world view are most evident at the analysis and decomposition stage of the software development cycle.

The evidence, within the object community, of a separate and quite different subculture can be seen in the "language wars" and in methodological approaches. Contrast "behavioral" methods with "data" (e.g. Schlaer-Mellor) and "software engineering" (e.g. Booch-Rumbaugh-Jacobson) methods.

At the risk of inflaming passions, I would assert that there is an object culture. It is embodied in the minority (not necessarily a small minority) of the community of object practitioners that get excited at the "Zen" philosophy of Christopher Alexander, that use CRC cards and Smalltalk as their language of expression and who recognize encapsulating "data" and "procedures" create COBOL programs not objects. Members of this culture resonate to the new theories of complex systems, autonomous entities, and emergence. They see the autonomy of behavioral objects reflected in the theories of artificial life – and recognize the formalist folly implicit in traditional artificial intelligence.

Today the "object culture" is obscured within the broader "object community." To the extent that this is true then the answers to the questions put to this panel - in reverse order are:

- yes there is something "special" about objects and yes there is a separate culture - it is deeply and profoundly evident among the members of every class I have ever taught, at least by the end of the semester.
- if we are to enculturate others we need to spend more time dealing with the "philosophical" issues - getting people to "think like objects" – and less time dealing with the nuanced differences that might be found at implementation time.
- communication between the "formalists" and the "objectists" requires recognition and acknowledgment - on both sides - of the very real differences between them.
- and finally, members of the object culture need to articulate in a more robust fashion the "shared technology, ideas, values, and world views" that shape their culture.

Dave West's first OO project was the construction of an "Automated Cultural Informant" to be used to train budding cultural anthropologists in fieldwork while a graduate student in Anthropology. He is currently engaged in "enculturating" graduate students with a series of object courses he developed and teaches at the University of St. Thomas in St. Paul, MN. (Only in January does he wish it was the Virgin Islands.) He is Director of the Object Lab at that same University and has been President of the Object Technology User Group (over 500 members strong) for the past two years. As President of the User Group he led efforts to present two successful object conferences (COPE 94 and COPE 96) focused on Current Object Practice and Experience. He holds a BA in Oriental Philosophy from Macalester College, a MA in Cultural Anthropology and a MS in Computer Science (Artificial Intelligence) and a Ph.D. in Cognitive Anthropology (dissertation committee consisted of two anthropologists, two computer scientists, a psychologist, and a linguist), all from the University of Wisconsin at Madison. Prior to returning to school he worked as a software development professional for 14 years. He has promised (for two years now) a book on teaching and learning objects to be used as an enculturation tool.

Steven Fraser

Steven Fraser is an Advisor with the Design Process Engineering Team at the Nortel Lab in Santa Clara California. He is the General Chair of Nortel's Design Forum, a proprietary bi-annual conference held in more than 25 sites world-wide. In 1994 he was a Visiting Scientist at the Software Engineering Institute (SEI) collaborating with the Application of Software Models project on the development of team-based domain analysis techniques.

Since joining the Nortel organization in 1987, Fraser has contributed to the ObjecTime project, an OO-based CASE-Design Tool and to the 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. He is an avid photographer and "opera"tunist.