OOPSLA, Liminality, and Deep Theory

Dave West
University of New Mexico

The OOPSLA 2005 San Diego poster hangs on my door just as other posters have hung the past 20 years. I have missed only three of these annual gatherings and feel I have some perspective on the flow and ebb of this conference.

There has been a fair amount of discussion the past couple of years about OOPSLA's decline. Not in quality but in terms of interest and even, perhaps, relevance.

Eyes focused on poster – murmurings of OOPSLA criticism as mental background hum – perhaps it is inevitable to ask, "Why do I plan to attend this year?" Or more generally, why would anyone want to attend? Or more generally yet, why does or should OOPSLA exist?

A Big Idea Conference

Academic and professional conferences are generally viewed as dissemination vehicles – places to present new work and generally update a community of interest on some area of common interest. Networking, job hunting, and sales are important adjuncts to most conferences but are not the primary reason for their existence.

Every once in a while a different kind of conference comes into existence – a big idea conference. A small group of people believe they have gained a novel insight or synthesized a big new idea and they create a conference for the purpose of exploring that idea and inviting others to share in the excitement.

Big idea conferences have a higher than normal level of energy and an atmosphere that is unique. Metaphors saturate the conversational airspace. Grand visions and freshman-dorm-room profound conversations fill every waking hour. The camaraderie is palpable. A clear distinction between "us" (those that "get" the big idea and are excited by it) and "them" (the prosaic masses that haven't a clue what we are talking about) is clearly evident.

Almost everyone is at the juncture of Alan Kay's "pink and blue planes" and completely engaged in limning new vistas and marking trails into virgin territories.

OOPSLA was such a conference. Others in recent history include the first few years of AAAI/IJCAI (artificial intelligence), Sigraph (the virtual reality years), IJCNN (neural networks), XP/Agile Universe and Agile Development.

Anyone lucky enough to attend the big idea years of a conference like OOPSLA are bound to be disappointed in later years because, it seems, that big idea conferences have a very short half-life. They rapidly devolve into pretty staid and traditional academic conferences. The butterfly morphs to a wooly cocoon.

A desire to become respectable – meaning the boss will pay for the trip and I will get tenure for presenting a paper – is the primary force prompting this devolution. The number of attendees is also a measure of respectability and big idea conferences are forced into becoming "big tent" conferences – allowing participation of a lot of people whose understanding of and commitment to the big idea is peripheral at best.

Is devolution inevitable? What other evolutionary path might a big idea conference follow? There is a possible alternative but to see it requires exploration of a different problem.

Background

Peter Naur, in 1985, challenged the prevailing view of software development as an activity of "production." Then, and now, the dominating metaphors employed when discussing software development were mechanistic. Methods (a set of work rules stating sequences of actions to be taken and deliverables produced) and processes (methods with metrics) were, and are, touted as the means to make software development more reliable and manageable.

Instead of production, Naur proposed "theory building:"

> *"Programming properly should be regarded as an activity by which the programmers form or achieve a certain kind of insight, a theory, of the matters at hand ...*
>
> *... theory is understood as the knowledge a person must have in order not only to do certain things intelligently but also to explain them, to answer queries about them, to argue about them, and so forth.*
>
> *What has to be built by the programmer is a theory of how certain affairs of the world will be handled by, or supported by, a computer program."*

A theory is not a model – a model is but an abstraction of one aspect of a theory. A theory is a rich gestalt understanding of the real world (problem space), an intimate understanding of the program (solution space) and the relationships between them. A theory integrates problem and solution as a single whole – reminiscent of Alexander's view of design as an integration of problem and solution. A theory is dynamic and so the things built according to theory are equally dynamic.

A theory is ineffable – it exists inside of human minds and can be shared but it cannot be expressed and therefore cannot be communicated.

> *"A main claim of the Theory Building View of programming is that an essential part of any program, the theory of it, is something that could not conceivable be expressed, but is inextricably bound to human beings. ... the building of a program is the same as the building of the theory of it by and in the team of programmers. ... The death of a program happens when programmer team possessing its theory is dissolved."*

Echoes of Naur's idea can be found in the work of Nygaard, Ehn, Alexander, and contemporary postmodern theorists like Floyd and Coyne. The theory building viewpoint leads to some interesting consequences:

- The only people that can possible possess a theory are those that worked together and simultaneously created theory and artifact (program).

- Theories are “idiosyncratic” – apply only to the integrated problem and solution space defined by the scope of the program.
- A theory cannot be communicated as it is ineffable and is not captured in any form of documentation. At best documentation can provide a starting point for re-creation of a new theory in the heads of a new team.
- Method does not help. (Software methods should be “scientific” but this presupposes, “…that there is such a thing as scientific method and that it is helpful to scientists.” Naur agrees with Feyerabend and Medawar, “that the notion of a scientific method as a set of guidelines for the practicing scientist is mistaken.”)
- There are no shortcuts – a new team will take just as long to re-create a theory - create a new theory - as did the original team.
- Programmers are not interchangeable parts because replacement programmers will not and cannot possess the necessary theory. Corollary, programmers with theory should be respected and valued.
- If you want to keep a program around and adapt that program to changing circumstances you need to keep the original team intact and working on the program. (A rather depressing thought for programmers constantly looking for new challenges – and jobs.)

The mechanics of theory building are not discussed in Naur’s paper. Had they been it is likely that they would conform to what we currently call agile practices: e.g. “One Team,” pair programming, collective code ownership, metaphor, iterative-incremental development, open workspace, intense and constant communication, short cycle feedback, test driven design, and even emergent architecture. All of these practices accelerate the acquisition of theory by a development team.

Agile practices would seem to be a necessary prerequisite to successful theory building. (Such practices have long been recognized by cultural anthropologists as fundamental to the establishment of culture – a people’s attempt to make sense of the world around them.) But they are not sufficient. Even a cursory examination of agile teams show that some establish a theory (and a social organization) that promotes project success and others do not. Some teams “gel,” others do not.

What mysterious X-factor might account for this observed phenomenon? One place to look for the X-factor is the threshold that separates a mixed group of individuals assembled on one side and the gelled team on the other.

Field Trip I – Rites of Passage

A threshold, literally, is the sill of a doorway. Metaphorically it is the point separating one world (public/outside) from another (private/inside). Thresholds can be traversed but only if the individual crossing is willing to undergo a transformation, a redefinition of role and of social relationships. (A traversal without transformation is an invasion.)

Thresholds are implicit at the beginning of any new project. Individuals are assembled and charged with a task. To succeed they must cross a threshold – make a transition – and become a team with a theory.

These kinds of transitions are very common in human cultures. All human cultures acknowledge some, but not always the same, set of transitions. Anthropologists study and theorize about these transitions. The general public is most likely to recognize Arnold von Gennep, who coined the phrase “rites of passage,” although Victor Turner and Mary Douglas have made major contributions to our understanding.

A transition, according to von Gennep, has three phases: separation, liminal, and incorporation. During the separation phase one’s old identity, roles, and social relationships are stripped away. Incorporation does the exact opposite – bestowing and confirming a new identity, new roles, and new social relationships.

Liminality, to use Turner’s phrase, is the period of “betwixt and between.” A liminal being is neither this nor that. Liminality is pure potential (you are nothing and can become anything) and pure hazard (without proper guidance you may become the “wrong” thing).

Human cultures have developed rites, observed, recorded and analyzed by anthropologists: the ritual actions, individual and group activities, symbols, and expressed ideals and values that are required to effect a successful transition. Depending on the culture these are elaborate and extensive or barely noticeable. (Compare the elaborate, frequently excruciatingly painful, and prolonged rituals that mark a transition to adulthood in many “primitive” societies with the ad hoc, essentially non-existent, 21st birthday celebrations in the United States.)

Field Trip II – Deep Theory

Let us leave the specialized realm of software development for a moment to examine a more general, but similar, problem – patterned human behavior – “culture.”

Survival, simple day-to-day living, requires the constant matching of solution (behavior = run) to problem (tiger!). Sometimes choices must be made among available alternative behaviors (run, climb), sometimes a behavior must be modified slightly (run, faster than you), and other times entirely new behaviors must be conceived (hey, there’s a long pointed stick, how can I use it?).

Different groups of people exhibit statistically similar behaviors in response to encountered problems. Behind this common behavior is a shared “theory” of the world integrated with a shared theory of proper human behavior. Anthropologists call this shared theory “culture.”

Culture shares many properties with the idea of theory proposed by Naur. Both are idiosyncratic – different groups, different cultures. Both are ineffable – can you articulate every aspect of your culture? Neither can be documented or communicated, both must be generated via shared experience.

Clifford Geertz and Marvin Harris (based on very different motivations) have demonstrated the impossibility of “documenting” culture or reducing it to a set of rules. Harris makes the same argument as Naur (citing Ryle) that culture cannot be reduced to a set of rules, “if the exercise of intelligence (read culture) depended on the following of rules there would have to be rules about following rules, and about how to follow rules about following rules, etc. in an infinite regress which is absurd.”

Geertz argues that the best one can do when faced with the problem of explicating culture is to provide a “thick description” – densely layered stories with lots of context and numerous revelatory asides that allow the reader to “experience” for herself the culture being described. A “good” ethnography has more in common with a good novel than it does with a “scientific” descriptive report with all kinds of charts, graphs, and statistical analyses.

A significant difference between culture and Naur's idea of theory – cultures can be and are nested. Almost all individuals simultaneously participate in many different cultures. For example: mystic, biker (only Harley riders allowed), object thinker, Xgilista (agile developer with a propensity for extreme programming), software developer, professor, New Mexican, Anglo-American.

As an object thinker, I share an idiosyncratic theory of programming that is quite different from that of procedural thinkers, or data thinkers. However, should I wish to share the theory of one of those other cultures, my efforts would be facilitated by the fact that we all share a common nesting culture – software developer.

The basically pessimistic consequences of Naur's theory building viewpoint could be alleviated (greatly alleviated in some cases) if idiosyncratic theories could be nested within more general "deep theory." But where might such a deep theory be found?

Off The Deep End

At least three foundations must be established if the profession is to establish a deep theory capable of providing a context for the various idiosyncratic theories that arise in each software development project.

- Shared values
- Richer metaphor
- Experience

Shared values will probably be the most difficult. We might start with the XP values: Communication, Simplicity, Feedback, Courage, and Respect. Or the values expressed by the Agile Alliance. It would not be long however, before someone suggested that "scientific" needed to be included, or that simplicity should be augmented with efficiency or performance. To keep discussion under any semblance of control some kind of ground rules (try it, if it helps keep it, if it doesn't toss it) need to be established.

Metaphor is, in some ways, an easier issue. A theory might be thought of as a way of thinking about the unknown. In general, (according to Lakoff, always), humans confront the unknown via metaphor.

> *"Along the philosophical fringes of science we may find reasons to question basic conceptual structures and to grope for ways to refashion them. Old idioms are bound to fail us here, and only metaphor can begin to limn the new order."* [Quine79]

A significant majority of software developers design and program under the influence of a set of "default" metaphors. Prominent among these default metaphors:

<u>Machine</u> – hardware "machines," abstractions of hardware systems and languages as "virtual machines"
<u>Organization Chart</u> – centralized hierarchical control.
<u>Entity</u> – a thing with characteristics the value of which must be remembered by the computer (or database).

Dualism – separation of software into two fundamentally different kinds of thing – e.g. (active) algorithms plus (passive) data structures."

In some cases these metaphors have proven to be constructive and helpful, but as applications move farther from the computer and seek to model, simulate, and/or interact with the real world, the less helpful these metaphors have proven to be. They must be supplemented with or replaced by other metaphors, e.g. the Lisp "list" and the "behavioral object."

A person skilled in the use of metaphor should be able to discover new metaphors and make decisions about which metaphor is most likely to yield solutions to specific problems. What techniques might improve our metaphorical skills?

Reflection – before all else make yourself aware of how you already use metaphors in almost every aspect of system development.
Experiment – just stop and challenge yourself to find two or more answers to the question, "my system is like a ________".
Expand your horizons – nothing will prove to be more fruitful for developing your metaphorical sense than exploring domains other than software development and computer science. Aspire to be a polymath.
Tell stories from different points of view – A story is a kind of metaphor. Imagine a brewery, a large room with vats of beer on one side a complex maze of conveyer belts, filling stations, capping stations, packaging stations, and shipping stations. Above the fray a master control panel that reflects the state of the brewery and which can be used to modify operations as necessary. Tell the story, "a day in the life of the brewery," first from the point of view of the control panel (typical top down functional decomposition) and then from the point of view of the beer - desperate to escape the confining vat and effervesce on the tongue of some thirsty drinker, the beer uses all the apparatus in the brewery, including the control panel, to provide services that enables the beer to achieve its goal.

Experience – "doing the relevant things under suitable supervision and guidance." Deliver software. Grow it in the company of others. Shape it in a manner consistent with common values and employ the richest set of metaphors available – do it! Subvert and invert the educational system in such a way that the traditional components of a curriculum are modularized and delivered just-in-time within a context of need and of expression. Do it again!

Conclusion

Naur identified a very real and still omnipresent problem confronting software development – a mischaracterization as an activity of production. Software development is theory building. Recognition of this fact presents its own set of problems – theory is ineffable, idiosyncratic, and not served by method. The pessimistic consequences of a theory building view of software development might be overcome or alleviated if nested within a common culture – a deep theory. Establishment of a shared deep theory will require a very different approach to enculturation: 1) the use of agile practices to accelerate the establishment of idiosyncratic theories; 2) acceptance of a set of core values; 3) an expanded practice of conscious selection of metaphor; and, 4) experience driven education and enculturation.