Borland Software Craftsmanship: A New Look at Process, Quality and Productivity

James O. Coplien
Software Production Research Department

AT&T Bell Laboratories

Abstract

The Borland Quattro Pro® for Windows (QPW) development is one of the most remarkable organizations, processes, and development cultures we have encountered in the AT&T Bell Laboratories Pasteur process research project. The project assimilated requirements, completed design and implementation of 1 million lines of code, and completed testing in 31 months. Coding was done by no more than eight people at a time, which means that individual coding productivity was higher than 1000 lines of code per staff-week. The project capitalized on its small size by centering development activities around daily meetings where architecture, design, and interface issues were socialized. Quality assurance and project management roles were central to the development sociology, in contrast to the developer-centric software production most often observed in our studies of AT&T telecommunications software. Analyses of the development process are ''off the charts'' relative to most other processes we have studied.

> Jim — Thanks again for speaking at BIC '93. I'm also glad you could stop by Borland and experience what we call Borland Software Craftsmanship. We are a young company, started by a Frenchman, with young bright and excited developers. In my 8 years at Borland I have been in the center of it all and can't imagine another place to be. Let us know if there is anything else we can do for you.
>
> — David Intersimone, Director of Developer Relations, Borland

1: Introduction

Last year, Borland invited me to speak at the Fourth Annual Borland International conference in San Diego, California, and to visit their location in Scotts Valley, California. I made arrangements with David Intersimone, Director of Borland Developer Relations, to speak at the conference in exchange for access to one of their development organizations. Interviews with such development organizations have helped the process community better understand the high-level characteristics of software development organizations. We can use this understanding to help projects assess their development methods against those used in other development cultures. I was enthusiastically received and graciously hosted, a harbinger of other positive signs of the Borland culture that I would observe that day. I was treated to insights into one the most stunning development efforts I have had the pleasure to study.

In this paper, I relate what I learned while meeting with the development team for Borland's Quattro Pro for Windows 1.0 (QPW) on May 20, 1993, in Scotts Valley. I feel there is much to be learned about their process, technology, and organization that we can apply to projects across the industry, including large projects and perhaps even embedded and real-time system developments such as we have in AT&T.

It is important to understand that this is a retrospective on the development of the software for the initial offering of QPW. There was little or no embedded base, and the project didn't face the constraints one finds in the legacy code projects common in large, traditional telecommunication projects. Even so, the phenomenal productivity of this group and the factors contributing to that productivity are thought-provoking. Most organizations should be able to take a page from Borland's book as a basis for their own process improvements.

This paper starts with a high-level description of the project and describes the personalities in the development effort. The next section gives a brief review of the data acquisition technique used. Analyses of the data from our process analysis technique follow in the next section. Subsequent sections of the paper describe aspects of the QPW development that stood out as contributing to its success.

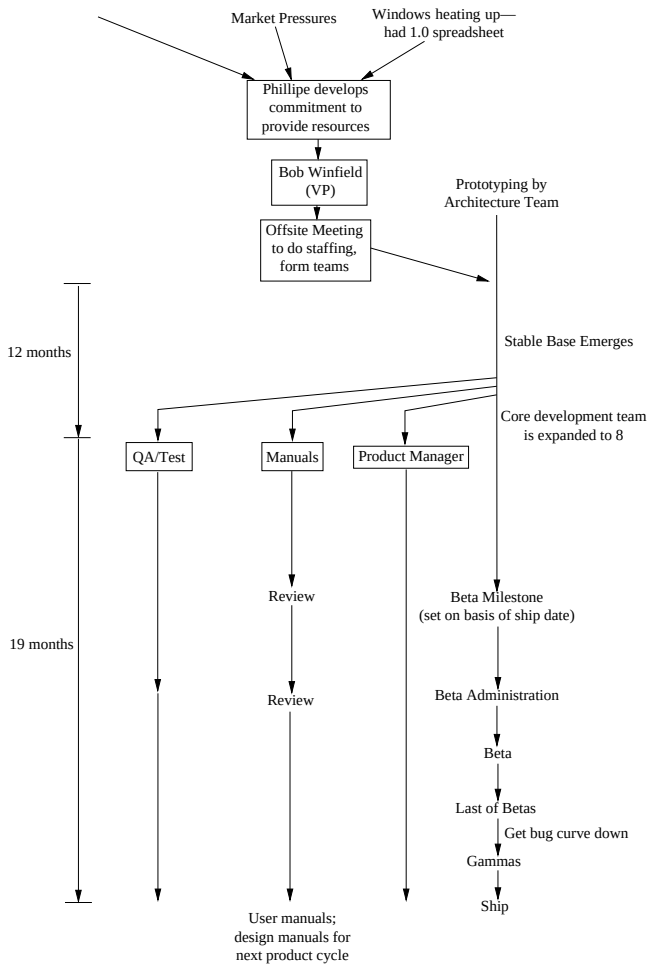

Figure 1. High-Level Business Flow of the Borland QPW Development

2: What Quattro Pro for Windows Is and How it got Started

Borland launched development for QuattroPro for Windows as a natural follow-up to their DOS spreadsheet offering. QPW offers spreadsheet and database functionality in the spirit of most spreadsheet products on the market today. The team I interviewed created QPW 1.0, the so-called base generic development for the product.

The initial development was to be heavily loaded with features. The project goal was to produce a product with the maturity and feature richness of a third- or fourth-release product. The team felt they had achieved that goal when the product shipped.

Like most Borland products, QPW is designed to be a self-contained deliverable that is compatible with other members of a product family. Its human interface is consistent with other Borland products. Its database interfaces allow it to interwork with other Borland products. Borland views itself as a vendor of individual business solution components, from which a customer can select combinations to meet their needs. The total code volume of all Borland products, expressed as original source lines, is huge: tens, if not hundreds, of millions of lines of code (my estimate). Products are largely independent of each other, yet share common infrastructure and look-and-feel (and, conjecturally, the code providing this functionality).

QPW had a small core team—four people—who interacted intensely over two years to produce the bulk of the product. Prototyping was heavily used: Two major prototypes were built and discarded (the first in C; the second, called ‘‘pre-Crystal,’’ in C++). Four core developers defined an architecture, built early prototypes and the foundation code for the product, and participated in implementation through its delivery. Additional programmers were added after about six months of intense effort by the core of four.

The methodology was iterative. Modulo the architectural dialogue, the core developers worked independently. Early code can be viewed as a series of prototypes that led to architectural decisions, and drove the overall structure of the final system.

The programming language was C++. The final implementation stages of QPW stressed their C++ compiler—which was being developed in parallel with QPW—to its limits. There was uncharacteristically tight coupling between the QPW group and the language group. QPW was one of the largest and earliest projects to stress their C++ compiler release. Cooperation between the two allowed each to contribute to the quality of the other.

After the product took shape (after about a year), additional roles were engaged in development activities. *QA* (quality assurance), testers, and others were at last allowed to see and exercise copies of the code that had been kept under wraps during early development. These roles had been staffed earlier, but engaged in development only when the developers felt they had something worthy of testing.

While the *QA* organization conducts its own testing, there is an active beta program to uncover bugs as only real users can. This is a luxury that tool purveyors enjoy to a greater extent than most telecommunications companies can (and that we enjoy to a greater extent than some contractors in, say, the aerospace industry).

The QPW product entered the market to high acclaim. PC Sources said, ‘‘Borland International Inc’s Quattro Pro for Windows spreadsheet software package makes better use of the Windows graphical user interface (GUI) than any other spreadsheet package to date.’’ [1] PC User says that ‘‘Borland International’s Quattro Pro for Windows (QPW) is the world’s best spreadsheet software.’’ [2] Computer Shopper quips, ‘‘Borland International Inc’s Quattro Pro for Windows spreadsheet software outperforms the standing champion of Windows spreadsheet management, Microsoft Corp’s Excel 4.0.’’ [3] InfoWorld, [4] PC Magazine, [5] and many others also offer positive reviews, which dominate the press perspective on the product. I found other reviews that are more balanced, but uncovered no reviews that found the product lacking in key areas.

The team members I interviewed included:

- Charlie Anderson, the Director of Applications for Borland, who was one of the QPW architects. He is experienced and thoughtful, the apparent ‘‘spiritual leader’’ of the group, but only in a subtle sense.
- Weikuo Liaw, a renowned expert on spreadsheet engines and one of the QPW architects. Wei is a highly revered developer, almost to the point of inspiring awe, but rather shy and among the most introverted of the group.
- Murray Low, an energetic, darting, bright and witty engineer who worked on the QPW/UI side (user interface stuff) and who was a QPW architect.
- David Intersimone, Borland Developer Relations, who facilitated my appearance at Borland but who was not part of the QPW development.
- Dan Horn, also from Developer Relations. He helped put me in touch with the Borland people while I was at the conference to make final arrangements.

3: The CRC Card/Pasteur Process Evaluation Technique

Process research carried out in the Bell Laboratories Software Production Research Department has used, among other techniques, an organizational analysis tool borrowed from object-oriented analysis. The technique is called *CRC cards*. CRC is an acronym for classes, responsibilities and collaborators, three of the most important dimensions of abstraction in object-oriented analysis. The technique provides an ‘‘object-oriented analysis’’ of the structure of an organization by dividing it into objects that are cohesive locales of related responsibilities. A more common term for these abstractions is *role*. Each role’s responsibilities to the organization are written on the left side of a 3x5 index card. The right side of the card lists the helpers, or collaborators, employed by a role to carry out its responsibilities. Responsibilities and collaborators are discovered in a role-playing exercise where development scenarios are simulated. The interests of a role are represented by someone who commonly fills the role, by a recognized domain expert in the appropriate area, or by someone who is otherwise familiar with the work.

Collaborations between roles form a network, or graph. The edges of the graph are subjectively weighted by the participants (high, medium, or low) according to the strength of the coupling between two roles. The graph can be visualized in many different ways: as a force-based network; as a topologically sorted hierarchy or directed graph; as an interaction grid; and others. We use the Pasteur process analysis environment to create and interact with such visualizations.

These visualizations offer insights into subtle (and sometimes not-so-subtle) organizational dynamics. For example, cliques can be readily identified from the natural force-based networks. Interaction grids offer insight into the cohesiveness of an organization. Highly specialized patterns have been noted in visualizations using each of these techniques, including a tendency for roles to cluster

according to their degree of extrovertedness or introvertedness with respect to the process community.

For more information on the use of CRC cards to debrief organizations on their structure, see the reference by Cain and Coplien. [6] For more information on the use of CRC cards for object-oriented analysis, see the reference list of that same paper.

4: Analysis of the Pasteur Data for QPW

We most frequently use a *natural force-based* network analysis to analyze organization data collected in the Pasteur data base. This analysis produced an *adjacency diagram*. In these diagrams, a default repelling force is established between each pair of roles. There is also an attracting force between pairs of roles that are coupled to each other by collaboration or mutual interest; a stable placement occurs when these forces balance. Figure 2 shows the picture that results by applying this analysis to QPW. There are several things worth noting in these pictures that set them apart from most other organizational process models we've made. Here is a summary of those properties:

- *The QPW process has a higher communication saturation than 89% of the processes we've looked at.* The adjacency diagram shows that all roles have at least two strong connections to the organization as a whole. The project's interaction grid is dense. The coupling per role is in the highest 7% of all processes we have looked at. This is a small, intensely interactive organization.

- *There is a more even distribution of effort across roles than in most other processes we've looked at.* The roles in the adjacency diagram are shaded according to their intensity of interaction with the rest of the organization. In the QPW process, *Project Manger* and *QA* glow brightly; *Coders* a little less so; *Architect*, *Product Manager*, and *Beta Sites* are ''third magnitude stars''; and *Tech Support*, *Documentation*, and *VP* still show some illumination. Most ''traditional'' processes we've studied show a much higher concentration of interaction near the center of the process. That is, most other processes comprise more roles that are loosely coupled to the process than we find in QPW. That may be because QPW is self-contained, or because it is small. It may also be because the process was ''intense'': a high-energy development racing to hit an acceptable point on the market share curve.

- *Project Manager and Product Manager are tightly coupled, central roles in the process.* These managerial roles were filled by individuals who were also key technical contributors to the project (they wrote real code), which contributed to their acceptance and success as process hubs. *Product Manager* was a role that was employed only after a year of development.

- *Quality Assurance is a tightly coupled and central role.* Many organizations consider QA to be an external function, outside their organization and process. At Borland, QA becomes a way of life after development has converged on a good design and a stable user interface. For QPW, this was about 12 months into development.

- *The CEO (Philippe) figures strongly in the organization.* In a company of thousands of employees, it is unusual to find the CEO as tightly coupled to development as we find in QPW. It is instructive to examine the responsibilities associated with Philippe Kahn's role:
  - Ensure product is commensurate with current market environment;
  - Ensure product market coordination is done in a timely and cost-effective manner;
  - Determine pricing, product positioning;
  - Shape public perceptions and handle PR for the product prior to and after ship;
  - Determine cosmetic changes to keep consistency among all Borland products and to call out certain features (in other words, usability testing);
  - Playing jazz to avoid press questioning on ship dates.[1]

- *The overall interaction grid pattern (Figure 3) is uncharacteristic of what is found in other processes.* Interaction grids show patterns of interactions in an organizations, and are particularly useful when the organization is large or when its interactions are dense. We most often use an interaction grid where roles are ordered on both axes by their degree of coupling to the organization as a whole. The most integral roles are placed near the origin. Most other processes exhibit a characteristic pattern of points along the axes, with lower point density and lower

1. There's a story in here, but I didn't hear it. Philippe is an accomplished jazz musician.

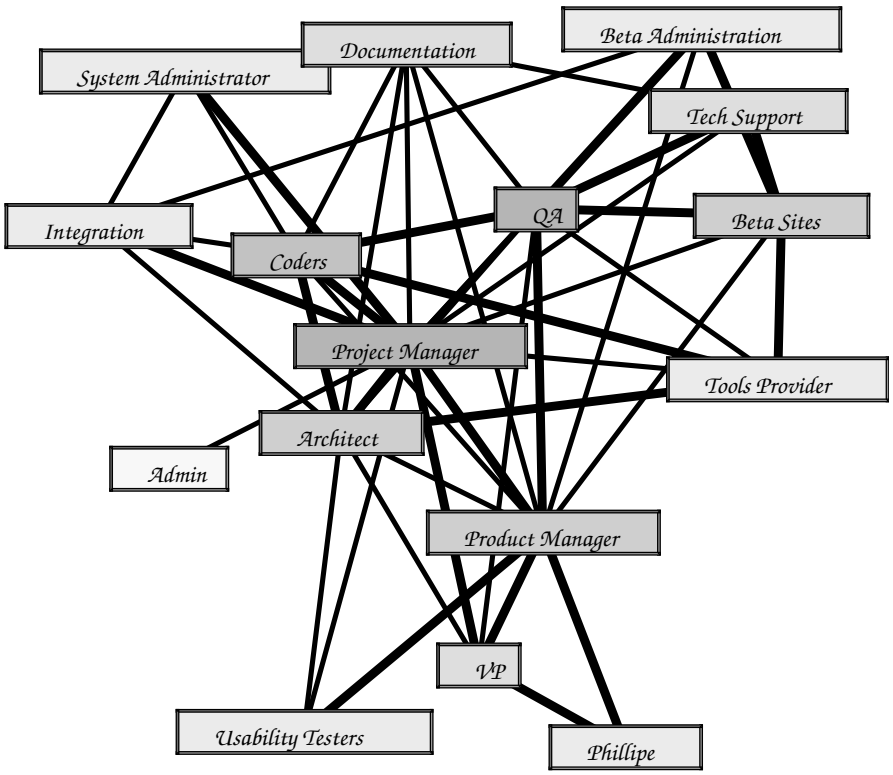

Figure 2. Natural Force-Based Analysis of QPW

intensity for increasing distances from either axis. In QPW, there is a general lessening of density and intensity as one moves toward the northeast quadrant of the interaction grid. The northwest and southeast quadrants of the Borland grid remain more dense than we've seen in other processes.

Between 30% and half of the processes we've studied exhibit a pattern called *schismogenesis*. Schismogenesis is a term from classic anthropological literature that describes a tendency for societies to stratify into sociological ''comfort zones.'' This phenomenon appears in interaction grids as a clustering of points around the diagonal. For organizations where this phenomenon is present, the effect is particularly pronounced in the northeast quadrant of the interaction grid. It indicates that organizations contain splinter groups.

The QPW process is characteristically ''anti-schismogenetic.'' That is, there is *blank space* around the diagonal of the interaction grid, particularly in the northeast quadrant. While we have seen graphs with random scatterings of points, the QPW graph is the first where the points seem to abhor the diagonal, yet fill out the rest of the graph.

We have not yet explained the schismogenesis phenomenon to our satisfaction, nor correlated it to properties of an organization. It is worth mentioning, however, that the Borland process is unique in having this anti-schismogenetic quality.

5: Personal Excellence and Integrity

The initial QPW development team comprised highly productive professionals who viewed each other with the highest respect. These perhaps sound like hollow words that most managers would apply to their organizations, until one looks more deeply into what ''highly productive'' and ''respect'' mean.

The QPW development team has chronologically mature membership by industry standards. ''We have professionals, not hired guns'' noted one member of the development team. People are brought into the team for their recognized expertise in domains of central importance to the project: spreadsheet engines, graphics, databases, and so forth. No one is viewed as a warm body, or general engineer, or interchangeable employee: Each brings special talents to the effort.

QPW had a small core team—four people—who interacted intensely over two years to produce the bulk of the product. Prototyping was heavily used: Two major prototypes were built and discarded (the first in C; the second, called ''pre-Crystal,'' in C++). Additional

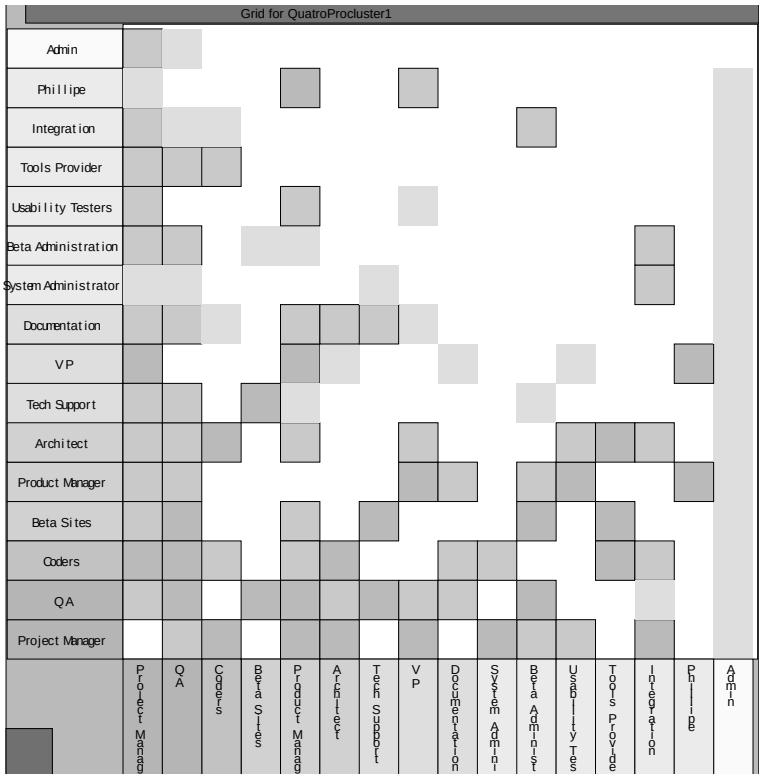

Figure 3. Interaction Grid for QPW

programmers were added after six months or so of intense effort by the core of four. These prototypes drove architectural decisions that were discussed in frequent (almost daily) project meetings. A million lines of code were written over 31 months by about eight people: that's about 1000 lines per person per week.[2] And that doesn't include the code in the prototypes.

Trust extends to eschewing code reviews. But while reviews are rare, group buy-in and trust are important. Each project member must *personally* sign off on a set of project floppy disks before they can be released to the next stage (e.g., beta test or to the ''street''). Personal evaluation of the software, as well as informal dialogue, build the confidence for such a sign-off.

There is a complex and highly non-linear relationship between project productivity, programmer skill, and project organization. There will always be debate about how much of the phenomenal productivity of QPW owes to its culture, how much to its choice of staff, and how much to other factors.

6: Do One Thing and Do It Well

This is Brian Kernighan's admonition of how to view C functions. Analogous advice is starting to appear for classes in object-oriented systems. And the same might apply to the people who write those classes.

QPW is organized along lines of domain specialization. Domains important to QPW are dependency registration software, human interfaces, databases, and a few others. An individual was identified for each of those domains.

In their domain, each individual does what they are good at. They excel at bringing their domain expertise to the table in architecture meetings. They know what the right abstractions are. They know how to implement those abstractions. They bring C++ or DOS or Windows® proficiency to the project, or quickly develop it (through analogy to related domain experience).

Equally important is what these individuals are *not* good at, and that they are not expected to take responsibility for domains not related to their specialization. Instead of working *in* these domains, they work *with* these domains. One good example is

2. I acknowledge that lines of code is an imperfect measure of productivity at best. However, a disparity of one or two orders of magnitude between the Borland experience, and more typical numbers from the rest of the industry, cannot be explained away with the usual attacks on source line measurements.

documentation. Developers are supported by a documentation organization that develops internal and external documentation. The time spent by developers in conveying this information to the documentation organization is far less than it would take for them to commit it to writing, put it into an acceptable format, and have the work edited for linguistic elegance.

By contrast, most of our AT&T developers write most of their own memos. It's not clear whether this owes to our history, our organizational boundaries, the nature of our business, or to reward mechanisms. Nevertheless, a deeply rooted cultural behavior at AT&T is that engineers draft their own memos. Developers spend much time (roughly 13% of total development time [7]) creating and refining memos in AT&T; in Borland, that job is deferred to people who are expert at it.

7: Architecture and Meetings; Coding and Iteration

QPW development was highly iterative. To understand the nature of the iteration, one must understand its ramifications for architecture and implementation. One must also understand the culture by which changes were approved and how decisions were made. This takes us into the realm of project meetings, always a topic of interest in a large development organization.

The core architecture team met daily to hammer out C++ class interfaces, to discuss overall algorithms and approaches, and to develop the basic underlying mechanisms on which the system would be built. These daily meetings were several hours in duration; from what I heard, *the project was made more of meetings than anything else.* Everyone's external interfaces were globally visible, and were globally discussed. The product *structure* was built on the domain expertise brought to the table by domain experts, but it was socialized and tempered by the needs of the product as a whole.

In spite of the intense meeting-oriented development culture the project built around its architectural development, class implementations were fleshed out in private. Individuals were trusted with doing a good job of implementation: after all, project members were acknowledged experts in their domains. Code reviews were rare.[3] The trust and respect engendered by this domain expertise made it possible to focus meetings on system-level issues.

There are three project principles worth noting about the QPW organization's communication architecture:

1. *Meetings are not a bad thing.* While we all cringe at the thought of a project centered on a meeting that carries over from one day to the next throughout early development. But our fear of meetings likely comes more from our memories of the *ineffectiveness* of our meetings, not from their *frequency.*

   At the First International Workshop on Software Process, I polled several process luminaries with the following question: Suppose I am among the most mature software organizations in the world (a CMM Level 5). [9] How much of my time do I spend in meetings? Responses from Vic Basilli, Watts Humphrey, and Bary Boehm ranged from 30% to 50%. Project communication, a shared vision, and meetings are important and productive if meetings are properly conducted.

2. *Development takes place on two levels: architecture and implementation.* There is an architectural thread, and a development thread; both are ongoing, and they interact with each other strongly. New implementations suggest architectural changes, and these are discussed at the daily meetings. Architectural changes usually require radical changes to the implementation. The implementors' ability to quickly reflect those changes in their implementation is key to turning around architectural changes quickly. This is where the outstanding productivity of the project members comes into play: Their incredible productivity supports iterative development.

   Their may be a third development thread—product management and marketing—that goes beyond the scope of this inquiry.

3. *The development interaction style is a good match for the implementation technology the group had selected.* Object-oriented development leads to abstractions whose identity and structure are largely consistent across analysis, design and implementation. Classes hide implementations and localized design decisions, though their external interfaces are globally visible. Mapping C++ classes and people close together made it possible for developers to reason about the implementation off-line, away from the meetings that dealt with interface issues.

3. This can be viewed as an extreme of the position developed by Votta in his recommendations for less formal code review meetings; see the paper by Votta. [8]

Notice this is contrary to the commonly presumed model that the object paradigm makes it possible for an individual to own a class, interface and all, with a minimum of interaction with other class owners in the organization. It should be emphasized that classes are good at hiding implementation and detailed structure (e.g., in derived classes) but that they are *not* good at reducing the ripple effect of interface changes. In fact, because interactions in object-oriented systems form an intricate graph, and interactions in structured procedural systems usually form a tree, the ripple effect of interface changes in an OO system can be worse than in a block-structured procedural design.

A question frequently posed to organizations using iterative techniques is: ‘‘How do you mark progress or do scheduling?’’ For QPW, there are two parts to the answer. First, they relied on experience sizing similar jobs, and found the overall estimates to be satisfactory. Second, they kept multiple sets of books internal to Borland to achieve different goals. The hardest of the dates was owned by (and not divulged by) the parts of Borland that own the financial books. A ‘‘real’’ street date was needed so the company could provide planning and resource support to the development. But *internal* scheduling provided incentive, focus, and pressure for development to move ahead. Project management and corporate executives presented deadlines to the development teams that failed to telegraph the business view of the schedule, presenting a more compressed schedule for development than the business case allowed for.

8: Personality and Development: Do Nerds supplant Allen’s Gatekeepers?

Thomas Allen at MIT has noted the correlation between effective communication skills and prospects for advancement and success in technical organizations.[10] Individuals exhibiting extraordinary communications skills, and exercising those skills outside their line organization, he refers to as *gatekeepers.* They ‘‘control’’—or, more accurately, facilitate—the flow of information between the development organization and scholastic and competitive sources.

One might expect a team of developers of a highly successful product such as QPW to follow this model. My observations of the QPW team were brief, but I was left with the impression that their personalities run contrary to this stereotype. ‘‘Nerds’’ would be a more apt characterization. However, individuals were able to communicate intensely with each other as a group, with intense stereotypical male-style communication dynamics. Only David Intersimone—an outsider—took the role of posing pointed questions to the group (probably to make sure certain points were clear to me).

While it is unclear exactly what their communication behavior would portend for success in a more structured setting, their technical prowess has earned them the highest positions of esteem at Borland. Perhaps one needs to bring Allen’s models into question, at least as they apply to small, inbred developments (most of Allen’s organizations were large, government or military contract projects).

One might consider evolutions of the AT&T development culture where such technical expertise could be a better harbinger of advancement. Different AT&T organizations have emphasized different professional qualities at different times as criteria for supervisory promotion: technical ability, coordination and interworking skills, administrative skills, and so forth. There is a common perception that in our current business environment, technical skills don’t dominate considerations for reward or advancement to the same extent that they did in the heyday of academia in the 1960s and 1970s. They are clearly key to success in the Borland value system.

9: No Wine Before its Time

QPW used iteration from early in its development cycle through the latest stages of development, increasing the stability of their software and decreasing iteration over time. This iteration took place in what might be described as a traditional corporate context. From its outset, QPW was a strategic, visible product in the company. That meant that all development areas were primed for its deployment, including quality assurance, documentation, and product management.

Though these areas were staffed from the outset, they were denied access to the details of the product until about a year into its development. That gave the architect/developers room to change the functionality, interface, and methodology of the project before interfacing it with the corporate culture and ultimately with the ‘‘street.’’ Quality Assurance, Product Management, and documentation were allowed access to the project only after it had ‘‘conceptually congealed,’’ about a year into the development schedule.

10: Create, not Conform

Even though Microsoft’s Excel may have been a significant market motivator to start the QPW program, QPW developers paid it little heed during the design of their code and human interfaces. Functionality and interface were derived from first principles (project members were strongly conversant in spreadsheet issues)

and from consideration for compatibility with other Borland interfaces.

One major distinction between QPW, and most of the work done in large telecommunications projects at AT&T, is that QPW wasn't working to a customer requirements document. They simply knew what needed to be done.[4]

11: California Gold Rush?

One cannot ignore the motivating power of bonuses that are of the same order of magnitude as annual compensation. While much of corporate America is turning more and more to ''egalitarian'' compensation structures, other companies strive to tie personal financial rewards tangibly to the market success of the fruits of an individual's labor. The stereotype may actually be true that bonuses and rewards for jobs well done are higher west of the Rockies than elsewhere. While I did not explore this with the Borland crowd, one might imagine that the west coast bonus stereotype extended to the QPW culture. The prospects for such rewards may make it easier for individuals to justify the energy and commitment they must commit to a high-intensity development for it to succeed.

12: QPW's Introspection about its Own Process

Can an organization without an explicit, conscious process effort enjoy the same process benefits as an organization with full process certification? Though there may be a tendency for certified organizations to experience stronger process benefits than those lacking any formal concern for process, this Borland project had many of the hallmarks of a mature development organization.

Borland is not subject to the ISO 9000 series process standards, has no concept of its SEI CMM rating, and is not conversant with the software development process lingo being used increasingly in large software organizations. For someone interested in process to visit them was a rare event. Before going through the CRC card exercise, my presence as a process guru was viewed with a range of responses that ranged from intrigued interest, through curiosity, to suspicious doubt. By the time the exercise ended, those involved were able to identify some parts of their value system and culture with what we call process. (By the way, the doubters went away saying, ''You know, I think you've got something there.'')

So even though the organization has no codified system of process, it is keenly aware of what it does, how it does it, and what works. It views software development as something fundamentally driven by special cases (at least for initial generic development) and repeatability is not an important part of their value system. Members of the organization were nonetheless able to articulate in great detail aspects of their process that demonstrated to my satisfaction that they shared a single model, perhaps based on development rules, of how development should be done.

Many organizations we have interviewed have a weak or confused notion of what their roles are, what the responsibilities of the roles are, and how the roles interact. Most AT&T organizations with a weak notion of process are those who have not gone through an ISO audit, yet developers' notions of their roles even in some ISO-certified organizations are fuzzy at best. Other organizations that do not have any conscious process culture are nonetheless able to articulate their process in explicit terms, at a level of abstraction that transcends technology, tools, or methodology.

Borland's QPW development was one such organization. When I asked what their development roles were (with a short definition of what I meant by role) the answers were immediate, intuitive, and reflected a single model of the organization shared by its members. Team members required little thought to come up with roles. Few roles were added during the role-playing exercise, and only one role was substantially redefined. The organization knew itself well, and was conscious of how people interacted with each other at an abstract level.

In his book, Gerry Weinberg suggests that there is a paradigm shift between Level 2 and Level 3 of the SEI Capability Maturity Model (CMM).[11] He believes that organizations at Levels 1 and 2 need strong (managerial) direction, while organizations at level 3 and above are self-directing. Borland clearly appears to be in this latter category—though it may not register a Level 3 rating according to commonly accepted criteria.

Charlie Anderson entertained us with a thoughtful monologue on how the project felt about itself and its accomplishments. ''We are satisfied by doing real work,'' he noted as he thought about how the project dovetailed daily architectural meetings with implementation. They learned how to improve the structure of their product, and how to improve their process, as they went through development. ''Software is like a plant that grows,'' he mused. You can't predict its exact shape, or how big it will grow; you can control its

4. One AT&T developer I talked to quiped, ''That's how a good . . . developer [for a large telecommunications system] does it, too.''

growth only to a limited degree. In the same vein, ‘‘There are no rules for this kind of thing—it’s never been done before.’’ In retrospect, though, he notes that there are a few things that every project should have. At the top of his list was that every project should have a good documentation department. This sounded intriguing to me (as it wouldn’t have been first on my list) but I didn’t get a chance to follow it up with Charlie (but see Section 6 above).

13: Process and Quality

One widely-held stereotype of companies that build PC products (or of California-based companies) is that they hire ‘‘hackers’’ and that their software is magic, unreadable spaghetti. Meeting with this group broke that stereotype for me. Their constant attention to architectural issues, their efforts to build an evolvable structure, their care to document the system well (both externally and internally), are all hallmarks of the highest professionalism. Those attitudes, coupled with the phenomenal *general-purpose* programming talents of the staff, plus the high level of *domain-specific* expertise, defined the kind of quality value system necessary to an effective and productive process.[5] There were few gratuitous shortcuts and few novice errors. From what I saw, these people produce *very* high quality code.

If there was any disappointment on the project, it was in their inability to bring down the bug curve in the project end game as fast as they wanted to. They noted that the shapes of software development bug curves are well-known, so there is hope of predicting how long it will take to ferret out an acceptable fraction of the remaining errors. However, the boundary conditions for the curve aren’t known at the outset, so it is difficult to predict the exact shape of the curve until developing experience with bug discovery and resolution. Inability to predict the exact shape of this curve resulted in a modest schedule slip.

Other questions about the project can be answered only over time. The process described here was for initial product development. Can a similar process be used for ongoing maintenance? Probably not, though vestiges of the original process will certainly live on. How will maintenance affect productivity? Can the dual-line development continue to support architectural change with rapid alignment of the corresponding implementation? Initial experience is good; the first round of QPW changes earned it a PC Magazine Editor’s Choice award.[13] The editors were astounded by the amount of functionality that had been added so quickly. Maintenance questions will become increasingly important to Borland, as we already recognize them as crucial in telecommunications systems with long service lifetimes.

14: Conclusions

Can we capture the architecture of the Borland development organization and process, and expect phenomenal results if we apply it to large development projects such as we have at AT&T? Probably not. However, its staggering productivity offers a target to shoot for, and some aspects of its management policies and process guidelines may serve small- to medium-sized developments well. To the extent large jobs can be partitioned into small ones, the Borland approach may be suitable for individual parts of large developments.

Borland develops products for a domain and market which, today, has little overlap with the traditional telecommunications market. As large software development organizations move into new markets—such as software development environment platforms and soft human interfaces—the techniques used at Borland will become increasingly difficult to dismiss out-of-hand as irrelevant for large system development.

The software industry has long embraced rationales that dismiss the productivity of stereotypical ‘‘Silicon Valley’’ cultures. Their products are not UNIX®-based. We think of PC development efforts as small and simple. We say they have limited markets and don’t need to evolve. Borland defies these stereotypes. It will need to move into a market supported by Windows, Windows/NT, Pink®, and conjecturally others including Macintosh®—and maybe even UNIX. It will need to interface to a host of different windowing systems and hardware technologies. It is not small, even by AT&T standards (it was larger than the first release of the flagship AT&T local switching product). Borland was able to coax 1 million lines of production code from about eight people in 31 months. Perhaps a PC-based development environment and PC-based deployment platform make developers more effective, and perhaps QPW doesn’t have the same fault-tolerance requirements one finds in large telecommunications systems. But those considerations alone don’t seem to account for figures that are orders of magnitude above industry norms.

Software maintenance is of critical important to today’s large, complex software developments. One suspects that the same will be true for QPW as it offers new features, runs on new platforms, and adapts itself to

5. Al Barshefsky has pointed out that process work can move forward only when the organization broadly subscribes to a common quality standard (or what I prefer to call a value system); see [12].

new operating systems and windowing environments in the market place. One might guess that foreseeing such evolution is one reason for Borland to have chosen object-oriented development techniques and C++ as the basis for their development.

The Borland process operates at an extreme point in our continuum of development organizations. Having a set of extreme data points can be of use to us in our process research, as it helps bound the models we make. We hope the Borland model will provide data that will help us calibrate our process models, and help us better correlate properties of other models we study.

A great big thanks to Carol Johnson at Borland for taking care of most of the local arrangements. I'm indebted to Ruby Chu at AT&T for chasing down QPW product reviews. A special thanks to Doug McIlroy and Peter Weinberger for their critical comments, with hopes for more to come.

REFERENCES

1. O'Malley, Christopher. ''Borland turns the Windows page: Quattro Pro for Windows.'' *PC Sources*, vol. 4, no. 1, p. 281, January, 1993.
2. Whitehorn, Mark. ''Vorsprung durch spreadsheet.'' *PC User*, no. 195, p. 54, 7 Oct., 1992.
3. Bonner, Paul. ''Quattro Pro for Windows.'' *Computer Shopper*, vol. 12, no. 11, p. 605, Nov, 1992.
4. Walkenbach, John, et al. ''Quattro Pro for Windows Version 1.0.'' *InfoWorld*, vol. 14 no. 41, 12 Oct., 1992.
5. Stinson, Craig. ''Quattro Pro for Windows.'' *PC Magazine*, vol. 11 no. 19, p. 162, 10 Nov, 1992.
6. Cain, B. G., and J. O. Coplien. ''A Role-Based Empirical Process Modeling Environment.'' Proceedings of the Second International Conference on the Software Process, Berlin, February 25-6, 1993, 0-8186-3600-9/93 $3.00 © 1993 IEEE.
7. Personal discussion with Larry Votta (AT&T) and Nancy Staudenmayer (MIT).
8. Votta, Larry. ''Does Every Inspection Need a Meeting?'' Proceedings of the Symposium on the Foundations of Software.
9. Humphrey, W. ''Introduction to Software Process Improvement.'' Pittsburgh, Pa: Carnegie Mellon University, Software Engineering Institute, 1992.
10. Allen, Thomas J. *Managing the Flow of Technology*, Boston: MIT Press, ©1977, 141-182.
11. Weinberg, Gerry. ''Quality Software Management, Vol. 1.'' New York: Dorset House, ©1991.
12. Barshefsky, A. ''On the Road to Software Automation,'' Proceedings of ISS 92.
13. *PC Magazine*, Vol. 13 No. 1, 11 January 1994, page 191.