Tough Medicine for the Dot-Com Culture

Conquering the anarchy that plagued so many failed start-ups

David Ruble
Vice President & Chief Methodologist
November, 2000

P.O. Box 4008 – Federal Way, WA 98063
(253) 946-2690
www.ocgworld.com

Your experiences and opinions are important to us at OCG.
After reading this paper, please feel free to email the author at:
david.ruble@ocgworld.com

Tough Medicine for the Dot-Com Culture

Conquering the anarchy that plagued so many failed start-ups

David Ruble
Olympic Consulting Group
November, 2000

A colleague once described his first day consulting at a dot-com company like walking into a frat party. Expensively-coiffed generation-X ladies (must be from marketing) mingled in the crowded hallways with a gang of post-grunge T-shirts, tattoos and slept-in-them gym shorts guys (must be from development). The Foosball table clattered above the din, as an errant Nerf dart whizzed past his head. A voice shouted that the network was down, and an impromptu meeting that was clogging the kitchen dispersed as everyone's pagers and cell phones seemed to go off at once.

Meanwhile, a minor herd of program managers argued with a gaggle of engineers over who which group had properly booked the only meeting room. Neither prevailed, as someone from management commandeered the room and skillfully guided a knot of potential investors into the glass chamber and closed the door. The door flew open again, as a harried VP quickly borrowed a chair from the nearest unoccupied cubicle – leaving the cubicle's programmer/occupant to stand when he returned from the bathroom. He scooted his CPU out from under the table, straddled it like a true PC cowboy, and resumed coding.

Sound like chaos? Certainly! This was the cultural antithesis of the software development shops of large industrial manufacturing firms to which many of us had become accustomed. There's a certain amount of excitement that can be generated working in a fast-paced and fluid environment. There comes a point, however, where the fluidity, ambiguity and continuous multi-tasking begins to mask a pathological lack of direction, purpose and process. People are very, very busy – there's a heck of a lot *activity*, but is anything really getting done? And how can anyone *think* in all of this noise?

Now that the bubble has burst on lavish funding for dot-com enterprises, it's time to take stock of the situation and examine what went right and what went wrong in some of these endeavors. The next wave of Internet ventures will need to be more nimble, efficient, economical and effective than their wild pioneering predecessors.

What follows are some of the major problems that my colleagues and I have observed amongst the various dot-com companies we have encountered. Associated with each problem, I suggest a form of remediation.

Problem 1: Viewing the site as a collection of web pages, rather than a complex suite of applications.

A dot-com company is soft. There's no brick and mortar to visit and no tires to kick. The success of the operation hinges on the software's ability to function properly, to entice buyers, fulfill orders and bill customers. One of the most dangerous tar pits that dot-com management can fall into is the belief that their site is simply a bunch of web pages. This HTML-centric view of the business focuses business leadership on branding, messaging and content management. They come to view their software as a collection of flat pages, which exist to enumerate a set of ever expanding features – rather than see their software as a complex and inter-dependent suite of business applications that just happen to have a public-facing browser.

What is the root cause of this bias? In many of the dot-com firms we have encountered, the initial focus has been on customer acquisition, or market share. Market share in the dot-com space is often measured in terms of "number of registered users" or subscribers, and not necessarily on sales transaction volume or value. In these types of cases, the management focus is on increasing the number of visitors – which can be done with enticing content, marketing and promotions. Thus, dot-com management teams have been laden with idea people – original entrepreneurs and people with an eye on marketing, finance and funding. What's often missing from the team in this "new economy" is a role that's far less glamorous, but one that should not be overlooked. Someone in the top ranks needs to have a firm grasp of the technological

architecture and operational underpinnings that it takes to operate serious large scale supply chains, and enterprise-wide integrated systems. The application software behind the glitzy site is often very traditional – not too exciting, but enormously important to delivering on the value proposition that attracted customers in the first place.

Time and again, we've seen people promoted to positions within start-ups that are far beyond their realm of expertise, often as a reward for being part of the original "gang" that started the firm. The problem is that no member of the original "gang" may have had technical skills beyond setting up a basic web site. It is no wonder that Venture Capital Groups and e-commerce firms are starting to raid the CIO ranks of very traditional brick-and-mortar firms – who have been dealing with ERP systems (Enterprise Resource Planning), Supply Chain Management and large-scale integration issues for years. It is precisely this type of background that a dot-com company needs on its technology management team. The issues of large-scale application architecture and integration take years to master. The web environment adds a layer of complexity, and an immature technical environment, over the top of formidable, existing inter-enterprise communication challenges. To compound the issue, most dot-com firms cannot assume that they have the luxury of time on their side to risk allowing someone to "grow into the job."

Bottom line: Hire or acquire a pro. Get somebody that has actually integrated large-scale applications within and between organizations. You may not be able to find a CIO who has implemented a complete supply chain via XML technology – but at the management level, a CIO who has successfully integrated vendors and customers using EDI technology has conquered most of the technical and organizational hurdles they will face.

Problem 2: Too many people, too little process

The rash of recent dot-com layoffs is beginning to mitigate the rampant over-hiring that occurred in 1999 – 2000. However, I think it important to look at the factors that led to hiring too many people in the first place. It's not just a case of, "We had the money so why not?" It's far more complex than that.

Inexperience. The tendency to hire young people with little or no prior experience was overwhelming. There was a sense of euphoria that this new generation of technologist was somehow smarter and faster than the last. Plus – people that are unsaddled with the demands of home and family are more opt to work long hours and work for less money than those with a mortgage and mouths to feed. On the surface this makes perfect sense. "We need to find people who will work 7x24 until our site clobbers the competition. When we do, we'll reward them with fabulous stock options and they'll be rich before age 30." Some start-ups danced around the issue of age-discrimination. Others were blatant about it. A forty-something marketing executive, eminently qualified for a position, was turned down by a 20-something recruiter who said, "Lady, you're too old."

A serious ossification of this trend took root in the HR department of many dot-com companies when they established fixed pay grades for potential new recruits that were often 20% below market rate – and often targeted at entry-level pay scales to begin with. The candidate would have to be willing to trade cash on the barrel today for a shot at riches tomorrow. Youth and exuberance showed up in droves. In many cases, seasoned software professionals stayed away.

Prior to the meltdown of the NASDAQ in the spring of 2000, the generational euphoria of having created a "new economy" was palpable inside the dot-coms. When our firm was brought in to teach object-oriented analysis and design to developers and program managers, we were told to compress the duration of the already-aggressive curriculum by half. "Our people are smarter and faster." The lofty height of the stock price seemed to convince them that the normal human bell curve didn't apply to the intelligence quotient of this select population. To be sure, we encountered some very smart people – most of whom had a shocking lack of prior exposure to the basic underpinnings of the body of knowledge that has been amassed in the software engineering industry over the last thirty years.

On the first day of class, I always ask my students to tell me about their prior software engineering exposure or experience so I know how to pace the course. I was accustomed to having a smattering of

entry-level programmers in my classes at more traditional firms. In the dot-com world, however, I was struck by the number of people with little or no experience being placed in positions of authority. Take these responses, for example:

> "I'm a development lead in the interface group. I've been coding for fourteen months. Prior to that I was in a rock band."
>
> "I'm a senior programmer/analyst. I've been here for six months. Prior to that I was selling jeans at the mall."
>
> "I'm a program manager. I'm in charge of defining requirements for the live site. I have no prior online retailing or software experience. My background is in clinical pathology, but a friend talked me into coming here, and it sounded like fun."

Nothing against rock bands – I was even in one once. However, the experience did not prepare me for engineering quality software. (I got good at dodging flying beer cans, but couldn't create a class model to save my life). To be fair, there *were* plenty of experienced people in the dot-com organizations we encountered. However, you can probably guess what their days consisted of – putting out the fires. "We've got a lot of kids running around here with lit matches," said one very talented software architect. Another one of the more experienced developers spent his time recoding areas of the application that were discovered to be botched-up *after* the code went live. Rather than devote this kind of talent to up-front design, they were forced to use him post-production just to keep the site running.

Lack of process. In many dot-coms, there has been little or no defined process for producing software. Process includes writing things down and communicating effectively between the diverse groups (program managers, analysts, designers, developers, testers) who must collaborate to produce the product. If nothing is written down, then the process becomes verbal. Like the prehistoric societies that subsided without benefit of the written word, many dot-coms cultures, by design or by accident, flourished simply by cramming all members of the society into the same cave. Proximity takes the place of process whereby immediate and constant verbal communication supplants written communication – hence, the cramped quarters and no-cubicles environment actually serves a purpose. It partially makes up for the absence of process. The problem is that it won't scale up. The cave may pass as a functional society with 50 occupants, but not with 500. To make matters worse, the propensity to add headcount in this environment escalates. Here's why:

In one dot-com we observed, management demanded new features be added to the site every two weeks. Any project manager knows that the three sides of the resource triangle: Time, resources and functionality (or quality). If you affix the time and functionality, you can only vary one thing: resources, and since nobody measured the actual amount of human effort it took to produce software, there was no penalty for simply adding more people. After all – a start-up wants to "look big" to Wall Street.

Somebody should have slipped management a copy of "The Mythical Man Month." You can't take nine women and make a baby in one month. There comes a point in any project where you have too many people and their inability to effectively communicate creates absolute gridlock. And gridlock is what many of these firms got.

Conway's Law states: "The structure of a software application mirrors the structure of the organization that built it." The anarchy of the organization is passed directly into the software. A colleague of mine described one dot-com's software architecture as a "*fractal*" – "It's an absolute mess at any level of magnification!" The result was fragile and rickety application code, an undocumented and disorganized database with rampant redundancy and errors, and the inability to integrate with third party packages to deliver to customers the very value proposition the site infers.

Bottom Line: Look at each of the various groups that it takes to run a successful e-commerce business and ask yourself, "Which of these functions *really* benefits from being run like frat-house?" There may be areas of the business where wild and unbridled creativity should be encouraged. Then ask yourself,

"Which of these functions benefit from being run like a sober, traditional software shop?" You will find that many of the core integration tasks, database design and administration, network operations, and enterprise architecture are severely imperiled by anarchy – and should operated as serious engineering disciplines.

As for lack of experience – the HR department should throw out the pay scale sheet. Instead hiring ten inexpensive programmers with little or no large-application experience, hire several highly experienced developers and pay them what they're worth. Put promising young talent in entry-level positions where they are encouraged to explore new ideas, but are also exposed to the discipline and mentorship of people who have mastered solutions to large-scale problems. Where you can't hire qualified employees, bring in experienced consultants. In the end, you'll spend less money and have better software.

Problem 3: "Time to market" thwarts all other objectives

Throughout the first wave of Internet start-ups, "time to market" was everything. The conventional wisdom was, "There's no time to do it right, because we'll be out of business if we don't get the software done yesterday!" In a cutthroat competitive environment, the pressure to rush to code at break-neck speed has always been at odds with software quality.

Software projects, like any other projects, must have a set of underlying objectives – often multiple (and sometimes conflicting) objectives. To achieve one specific objective, the project team may vary the degree to which they meet the others.

Take two conflicting objectives: (1) Time to market, and (2) Scalability and Extensibility. It conceivably takes longer to create a software project that is well engineered, scalable and extensible than it does to simply meet the functional requirements and push it out the door. The overwhelming pressure placed by financial backers upon public-facing Internet sites to "go live" often sweeps aside all other considerations for the future architectural welfare of the site.

The risk to the architecture, of course, is that the venture actually becomes successful, and the technology is unable to support it. The examples of the after-effects of placing time-to-market above all else are numerous. Here's a sampling of just some of the major issues plaguing dot-coms who are struggling to overcome this legacy:

> Sites which cannot be effectively partitioned, therefore, changes to any sub-section of the site requires a complete redeployment of the entire site, which requires complete regression testing, and risks "build" errors being introduced into untouched sections of code.
>
> Redundant code, whereby more than one module or routine has been created that executes the same essential function. For example, the function that calculates an order's price and total is coded separately on the shopping cart, the pre-authorization of credit cards, which is again different than the code that does billing. The result is inconsistencies and errors, which become exposed to the customer.
>
> Code, which has not been "designed" per se, so there has been little thought given to reuse – which could speed subsequent development.
>
> Hastily designed data schemas, which ultimately fall victim to rampant redundancy, errors and make poor use of optimization and query tools that come with the database management system.
>
> Lack of documentation – which creates havoc in an environment of high turnover. In the absence of any documentation, each new developer has to rediscover what is there, or often simply creates new code, rather than take the time to reuse existing. Despite the obvious inefficiencies, this code tends to become very fragile and breaks easily every time it's touched.

The lack of analysis, design or "as built" documentation creates a bottleneck on the developer – who is burdened with not only creating the application, but also must design it, and re-analyze ill-defined requirements, and fill in large gaps in the requirements as well. This type activity cannot be partitioned effectively across multiple developers, and thus creates a severe dependency on the developer who must upload all of the requirements and understanding into his head in order to create the software.

Bottom Line: Now that the gold rush to IPO has slowed down, firms will take a second look at building robust software that works – especially now that the bricks-and-clicks firms are re-emerging as serious e-retailers – backed by the on-going cash flow and existing infrastructure that allows them to take their time.

Problem 4: Lack of direction, focus or vision

From a software engineering standpoint – this problem is the hardest to control. As dot-com companies were buffeted on the waves of the stock price storm, many of them continued to shift gears, change focus and direction – looking for ways to become profitable. Constantly looking over their shoulders at the competition, many "me too" decisions were being made that had little or no fit with their core business model.

For every software project that made it to the "live site," there seemed to be scores of unfinished ideas or canceled initiatives that didn't. What accounts for this wastefulness? First and foremost – there is a disconnect between management as objective-setters, and software engineers as people who create software to achieve a specific set of objectives. When this partnership breaks down, either party could be to blame. A management team who can't set course and clearly articulate the business objectives is going to be continually frustrated by a software development group who, in an attempt to second-guess what the objectives are, codes a bunch of stuff that may or may not be what the business needs.

On the other hand, a software development culture based on the hope that order will arise out of anarchy will have a dubious track record of producing software that backs up management's vision. In the early days of a start up, the management visionary will literally sit next to the programmer and dictate requirements straight into his ear. Unfortunately, this direct path from idea to code cannot scale up.

As organizations get larger, and more people start throwing ideas into the mix, it is of paramount importance to develop a strong project chartering and evaluation process, whereby the management team – accompanied by the top technical leaders of the firm, prioritize new projects and feature sets according to their business value, cost and technical feasibility and time-sensitivity. Too many cooks in the kitchen yields the type of site-suicide failures whereby marketing conjures up new products and features for which the company has no ability to support either in the supply chain or in their customer service organization.

Bottom line: Institute a clear and consistent methodology for proposing new projects, evaluating them against the business plan, estimating their cost and technical feasibility, time-sensitivity – and launch only those projects that the company can handle, giving each initiative a running chance at success. Beware of knee-jerk "me-too" initiatives. Don't get caught up in the press and hype. If the latest "cool trend" on the web doesn't fit your business model – consider it carefully before jumping in.

Conclusion

During the rocketing rise of the NASDAQ in the winter of 1999/2000, I asked a colleague in the software industry, "When will it stop?"

"When all of the foolish money has been spent," he replied. Much like the California and Alaska gold rushes, the dot-com revolution has changed the face of American business forever. Some people got fabulously rich – many others got burned. Today, the average investor is in a far less speculative mood. Perhaps it was a mistake to throw too much money at any given start-up at once. Some of them managed it quite well. Others crashed and burned due to their own inexperience, lack of process, rush to market or faulty business plan.

Moving forward, we have a lot to learn from the recent past. We can look back through thirty years of software engineering and see the types of disciplines, skills sets and work environments that produced reliable, robust software. We also saw legendary and spectacular failures in the pre-dot-com world – whose projects share many of the same characteristics and causes that I just described. The remedies then, and the remedies now are strikingly similar.

A modern e-commerce web site is a collection of integrated, complex applications. The browser portion is only the piece visible to the public, but the background order fulfillment, inter-enterprise communication; inventory and supply-chain management and customer billing systems are the heart and soul of the operation. It takes discipline, experience, direction and focus to pull it off and do it right. The exuberance of the younger generation should not be discouraged, however, the business domain and technical architecture is far more complex than can be mastered in a few years of programming. It takes a mix of seasoned professionals, providing framework and direction to prevent an organization from re-discovering, re-inventing or simply ignoring the technical underpinnings of these complex software applications and architectures.

About the author

David Ruble is an analyst, designer, author and educator. He is widely regarded as an expert in the field of information modeling; object modeling and GUI (graphical user interface) design. He has been a principal analyst and designer of many mission-critical client/server corporate information systems, e-commerce systems, as well as applications in the public safety sector. As an educator, he has taught software engineering techniques to hundreds of students throughout the United States. He is the author of the popular book, *Practical Analysis & Design for Client/server & GUI Systems*, published by Prentice-Hall. David is a Principal at Olympic Consulting Group).

About Olympic Consulting Group

Olympic Consulting Group (OCG) is a full-service systems architecture and development firm. Located in the heart of the Pacific Northwest's high-tech corridor, OCG provides analysis, design, development and project management to a distinguished clientele, ranging from dot-coms to international manufacturing companies. www.ocgworld.com

---

Selected Bibliography

Brooks, Frederick P., Jr. *The Mythical Man Month, Essays on Software Engineering*. Reading, Mass: Addison-Wesley, 1975, 1995

Colkin, Eileen. "Venturing Outside of IT, Venture-capital firms are actively pursuing top IT executives to help them make more-intelligent funding decisions." *cmp.com*

Fishman, Charles. "They Write the Right Stuff." *Fast Company*, Issue 6, p. 95

Ryan, Michael. "Digital Debacle." *Smart Business*. November 2000, pp. 90-106

Scott, Alwyn. "Some Dot Coms Running on Empty." *The Seattle Times*, October 8, 2000

