Software Ascents

by Jim Highsmith[1]

High on the north ridge of Mt. Jefferson in the Oregon Cascades, near the top of a 25-foot ice wall -a left cramponed foot with two small front-points in the ice, an ice ax with its pick buried at least a quarter inch deep in the ice over my head, right foot scrabbling for purchase on a rock nubbin, and my rear extremity hanging 800 feet above the Jefferson glacier -- I asked myself, "Did I pick the wrong mountain to climb?"

Being an avid (although amateur) mountaineer, I have often thought of the parallels between climbing and developing software -- both somewhat dangerous undertakings. Software development in the 90's needs both a revised procedural paradigm and a re-emphasis on the human dimension -- congruent action and personal effectiveness. By drawing some parallels between successful mountain ascents and successful software "ascents," maybe these issues can be surfaced more successfully.

In climbing, one continually evaluates the "margin of safety," matching the objective and current conditions with the climbing team's capability. A reasonable, rational match leads to safe, exhilarating, successful climbs. Cutting the margin of safety too closely can lead to disaster. Software development has its own "margin of safety," although it is often unrecognized or ignored. If we approached software projects with a little more of the humility that goes into climbing, utilizing some of the same senses of wonder, awe, care, and fear -- we might improve our software success ratio.

Software development methods are gradually entering a new era. In the beginning there were no methods and the world was good. Then the great religious leaders were anointed by the multitude -- DeMarco, Codd, Yourdon, Orr, Parnas, Martin, Gane & Sarson. So began the age of methods. We went from a state of Adhocracy (see figure 1) to a state of Bureaucracy, from everyone doing their own thing, to everyone doing one thing -- not at all what the guru's intended, by the way. While many projects benefited from these methods, many more floundered in a sea of paperwork, forms, diagrams, and repository entries. As my friend Lynne Nix says, "An ad hoc approach gives the analysts and programmers an excuse not to think, while the bureaucratic approach give managers an excuse not to think."

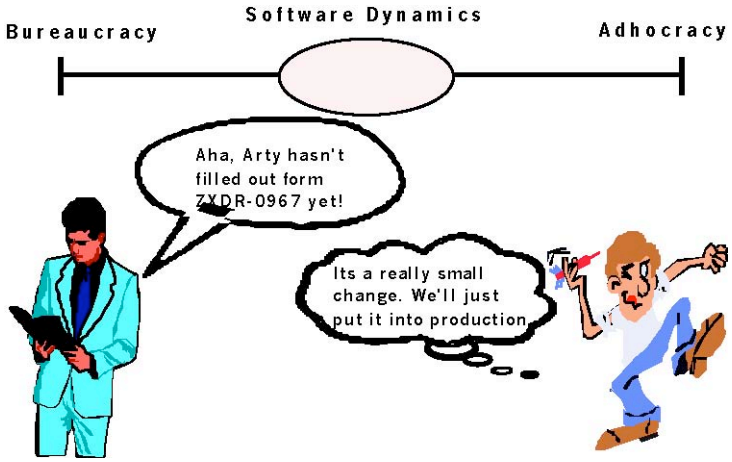

Figure 1 - Bureaucracy vs. Adhocracy

[1] "American Programmer Magazine," June 1992.

A better paradigm is emerging. We need not remain trapped, oscillating back and forth between no methods and burdensome methods. Many of the aforementioned pioneers have advocated a "toolbox" approach to development, but practitioners have often interpreted that as an excuse to return to adhocracy. The emerging pattern is neither abandoning methods nor force feeding methods, but one, to paraphrase Jerry Weinberg's terminology, of steering methods -- adapting methods to fit the situation based on careful analysis and congruent action. I contrast climbing and software development in the belief that climbing mountains is also a "steered" activity. Venturing into the mountains without the proper skill set is a sure recipe for disaster. Taking a mental mind set that allows no variation for changing environmental conditions into the mountains is not much better.

This emerging paradigm reflects Watts Humphrey's software engineering maturity model, particularly moving organizations into level 3 -- the Defined Process. Jerry Weinberg's recent book, Quality Software Management, offers a similar progression he calls software patterns, where level 3 is the Steering Level. These patterns, or models, integrate the more traditional software or information engineering techniques with project management methods, and quality innovation methods -- an approach I've referred to as Software Dynamics. Successful mountaineering is a blend of careful objective evaluation, adequate preparation, continuous observation and assessment of the environment and progress, adjusting tactics, and reviewing final results (see figure 2). Software Dynamics is a similar process applied to software projects.

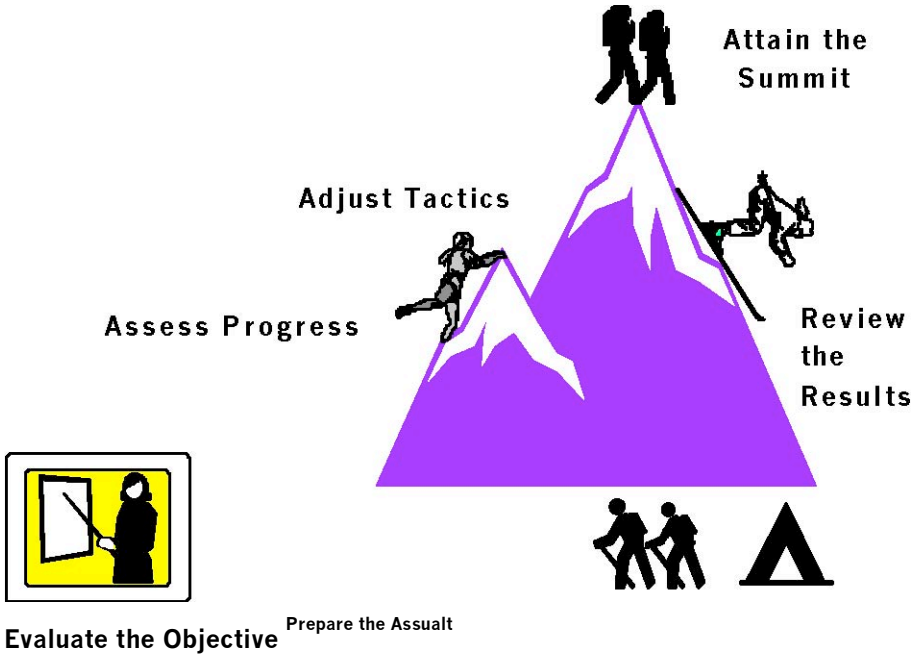

Figure 2 - Software Ascents

Evaluating the Objective

When the average non-climber thinks of climbing Mt. Everest, they might recall National Geographic or TV coverage of "siege" assaults, large teams of climbers and Sherpas. Over a period of many weeks they set up a succession of camps on the mountain to prepare for the final summit attempt. Most would barely believe Reinhold Messers' solo ascent without oxygen in a few days. There are only a few people (maybe one) with a chance of success in such a climb. Climbing begins with a careful evaluation of both the objective and the climbing team. Where is the mountain? How do you get there? What will we consider success? What season should we climb? Which route do we take? For example, the "tourist" route on Mt. Rainier is a difficult (and potentially dangerous) hike with minimal skill requirements

while the Willis Wall on the north side was considered suicidal for many years because of unstable rock and ice. Do we climb alpine style or not? What are the teams' strengths and weaknesses? How much time is available, and how much money do we want to spend? There are many factors that need evaluation to ensure a safe, successful climb, and there are many times the team chooses a more limited objective based on this evaluation. When lives are at stake, there is little incentive for the attitude expressed in many software organizations of "well, I do not think we have the skill set or experience to do this project, but the client wants it done, so let's give it a try anyway."

Proper evaluation of software objectives -- prior to initiating a full blown project -- is rarely done adequately. But software projects are just as variable as mountains, from ones most teams could handle with relative ease, to those only highly skilled, specialized teams should even attempt. There are some models around for classification of software projects, but most organizations have not yet realized the need. One of the better starts at a classification scheme is defined in Capers Jones' recent book, Applied Software Measurement, where he provides a model with four project classification parameters -- Nature (new development, enhancements, ...), Scope (prototype, stand-alone program, major system, ...), Class (private use, internal program for use at multiple locations, ...), and Type (batch, scientific, robotics, ...). He goes on to say, "Each of the 210 possible class-type combinations will have notably different productivity, quality, and skill 'signatures' associated with them."

Classification is really the second step. The first is a realistic and rational evaluation of the enterprise objectives and the driving need to solve a problem or exploit an opportunity. The lack of good, well thought out, un-ambiguous, agreed-on objectives doom many projects before they start. Many climbs have ended in disaster because the team members did not agree to measures of success before the start -- from reaching the summit, to climbing style (style is the "key" issue to many climbers), to just having a good time. If one team member's objective is to get to the summit at any cost and another's is to enjoy the climb, being in the mountains, and the comradeship -- a potential disaster is brewing.

Another aspect of evaluating the objective is assessment of the team's skill and experience in tackling this kind of project. Again, we usually have such an activity in software projects, but it is often cursory and half-hearted. Technical climbing is partially defined by the connection of team members by a rope. For safety, in ice and rock climbing, one team member is "anchored" while the other climbs. In glacier travel, the rope protects against one member falling into a hidden crevasse. In either case, rope team members protect each other against injury or worse. You do not "rope-up" with someone unless you are confident in their skills and their "team" attitude. Yet we often "rope-up" and commit our projects, our careers, and our organizations with little real thought about people's skills and how individuals might coalesce as a team.

My final comments about objective evaluation have to do with framework selection. Most organizations have some framework, a "Methodology," for managing the software process. Again, we often spend little time asking whether or not the framework is applicable to the project at hand, or whether it needs "tailoring." Relevant questions at this point are asked within the context of the project's objectives, classification, and team capabilities. We also often fail to consider or differentiate between the software development, quality management (including metrics), and project management aspects of the framework. For example, many methodologies "integrate" the software processes with the project management processes. What is often not evident are the underlying assumptions built into this integration and that it is inappropriate for many projects.

Software managers must ask more questions like -- "Is this the right framework? Is this a crash project or a package project or a RAD project or an extremely defects sensitive project or ...? What are the characteristics indicating one framework over another?" A client recently wanted to know about RAD since they had a project under significant time pressure. However, after analysis, the project turned out to have all the characteristics antithetical to rapid development. Many methodology vendors are now providing more guidelines for utilizing "paths" through various phases and steps. Some even have automated support

for generating tailored work breakdown structures. Ultimately good project leaders must think through the process rather than relying on "packaged" tailoring.

RAD (Rapid Application Development), for example, has become in some circles a way to get around methodology, and in others the next "silver bullet." Somehow, they think RAD can be implemented without the pain of other approaches. As with other processes, RAD is a progression -- from BAD to JAD to RAD. From an understanding of Basic Application Development techniques, to an inclusion of group consensus techniques (Joint Application Development), to executing those basic skill sets faster -- RAD. Again, the process is not a set of un-wavering tasks, but a steered, thoughtful progression.

Adequate Preparation

In software development, as in climbing, preparation is the easiest task "technically," but one of the more difficult to accomplish successfully because it is the "grunt" work that everyone lacks enthusiasm for. In climbing, preparation is primarily logistics, travel preparation, and conditioning -- what to bring, how to get it there, and how to endure the physical challenge. A weekend climb within driving distance requires one level of preparation, while a major expedition to Mt. Everest or K2 may require years. Many decisions are made which trade-off safety, weight, and comfort. Camping gear -- tents, sleeping bags, packs, bivy gear (basically a sack to climb into in un-planned overnights away from camp), etc. Clothes -- warm weather, cold weather, wet weather. Climbing gear -- ropes, ice gear, rock gear, helmets, harnesses, anchors, etc. Foot Gear -- leather, plastic, rock, camp, etc. Food -- how much, what kind, emergency, weight. (Climbing is a great weight reduction scheme. High altitude climbers need 6,000 - 8,000 calories per day -- and usually still have significant weight loss). Individuals spend long hours getting in top physical condition (My kids, my wife, and my cats used to get a big charge out of my hauling a backpack with 65 pounds of rocks in it up and down our long front stairs).

It takes a lot of thought and time to make these preparations. Good preparations do not overcome the technical difficulties in the mountains, but inadequate preparations can ruin a climb before the "fun" part even starts.

Software projects are no different. Good preparations do not ensure success, but poor preparations usually lead to failure. There are no "silver bullets" here. Good preparations mean having the discipline to do what we know how to do -- developing a plan including deliverables, milestones, schedules, etc., defining roles and responsibilities, making tool and methods' decisions, developing project and product documentation procedures, matching skills to responsibilities, choosing project measurements and change control procedures, training, and other seemingly mundane, but critical items.

On your next software project ask yourself -- "if this project was a mountain climb, how carefully would I think about food (development techniques), water (project management), and other essential items like climbing gear (quality assurance)?"

Continuous Assessment and Adjusting Tactics

The first paragraph of this article described part of a July 1991, climb of Mt. Jefferson by me and two friends -- Jerry Gordon and Jerry Strom (both employed in I/S by the way). There were two critical junctures in this climb. First, the ice wall mentioned earlier was supposed to be a simple rock chute, but we were attempting an early season ascent, and found ice. It called for serious re-assessment of tactics. Was there another route? Did we have the right tools and protective gear? Who felt comfortable (or at least not completely freaked out) leading the way? There was pressure not to stop to make this assessment. Time was becoming critical to reaching the summit. We were in the shade, partially surrounded by ice and cold. But the pressure to rush was tempered by the risks involved in making the wrong decision.

Past the ice wall, the second assessment came two hours later. High on the mountain, the summit only 800 feet higher, we stopped to consider abandoning the climb. We had started at 4:30 AM and it was now 2:00 PM with at least 2-3 hours of climbing to the summit. We could see more ice than we had anticipated on the remainder of the route that might slow us down even more. It was a 4-6 hour decent to

high camp, an hour to pack up tents, etc., and then a six mile walk out to the car with full packs. Whether or not to continue was not an easy decision? Climbers are extremely reluctant to abandon a summit.
What about software projects? There is a need to continually assess all aspects of the project and make hard decisions. A complete assessment needs to look not only at the product's progress, but also needs to assess the continuing viability of the processes themselves. Are the objectives still valid? Do we add resources or slip the schedule? Do we alter the deliverables? The tools? The techniques? The formality? The defect counting process? The project management measures? The team's training? Too many project leaders are reluctant (or constrained from) altering the framework (assuming they had an explicit framework in the beginning) established in the beginning. And so they plunge ahead.

To even consider abandoning, or significantly altering the original objectives is an anathema to most organizations. Most organizations have some type of project progress assessment, but the range of potential actions based on that assessment is narrowly limited to "politically acceptable" options. We have to face defeat in the end, because we never allowed ourselves to "seriously" consider defeat (and therefore altering course) in the middle.

Reviewing the Results

Just like fishing stories, climbing stories get "longer" with repeated telling. Talking about the adventure is sometimes more fun than actually being there -- just ask anyone who has spent hours slogging up a glacier field anticipating that each "ridge" is the top. Climbers recount each and every phase of the endeavor, from the food to the "bear" tracks to the long walk back to each and every crux move (a climb usually has one or more critical, difficult sections called a crux). Decisions made along the way are reviewed and analyzed -- on the Jefferson climb, we did turn back short of the summit. Non-participants (spouses in particular) can become extremely bored and difficult around such interesting and insightful "debriefings." Apart from the fun of reviewing such results, these sessions are important learning experiences. Learning experiences that may have significant impact on the next outing (for example, never let Jerry cook again).

Our "mis-naming" of software project evaluations has contributed to their lack of effectiveness or even use in many organizations. By calling them project "audits" or "post-mortems," we relegate them to a "blaming," fault-finding status. Technical reviews and walk-thoughts, one of the easier and most effective development techniques, often suffer the same fate because of focusing on finding defects, rather than on the more positive learning aspect of the process. Finding defects in a code review and problems in a project review are important and should be one of the review's goals, but much of the literature and practice of reviews have lost the equally, if not more important, objective of learning -- learning from our successes, learning from our short-falls, learning from our failures.

Many organizations have launched quality improvement programs based on the concept of continuous quality (and, therefore, process) improvement. Until our review and evaluation processes change to be more like a group of climbers sitting around the fire; eating pizza, drinking beer, and evaluating the "project" as a learning experience, most quality improvement programs will continue to be more talk than action.

Congruent Action and Personal Effectiveness

In the press of daily competitive pressures -- higher quality, more features, time compression -- software development has increasingly become the realm of tools, techniques, methods, procedures, and processes. All good and necessary, but all doomed to partial success without more emphasis on the individual. In an earlier paragraph, I mentioned the terms "congruent action" and personal effectiveness then passed then by with no explanation. I now hope to remedy that omission (for insights into these concepts I am indebted to Jerry Weinberg's writings and his discussion of Virginia Satir's work).
Climbing is a very personal activity. Although often done in small groups, each climber makes personal decisions during an ascent. Usually the team designates a leader, someone with more experience, or more

knowledge of the particular mountain, or specific climbing skills. But in the end, each team member makes his/her own decisions about safety and style. It is in each person's self interest to act according to one's own convictions -- to act congruently. To paraphrase Virginia Satir, "the essentials of congruent action are: To communicate clearly, To cooperate rather that compete, To enhance individual uniqueness rather than categorize, To use authority to guide and accomplish 'what fits' rather than force compliance through the tyranny of power, To be personally responsible, To use problems as challenges and opportunities for creative solutions."

The question is often asked, "Why do people climb mountains?" The standard reply is "because they are there." Maybe a more powerful reason is that climbing is an intensely personal and focusing activity. It forces one to be congruent and personally effective, because to do otherwise would be folly. There is nothing that focuses your attention more than hanging off the side of a mountain.
Thinking about congruent action and personal effectiveness can be a positive force in software development. They help moderate organizational bureaucratic tendencies. Organizations tend to pressure us into acting in discordance with our thoughts and feelings. We know the right action to take, but act otherwise under this pressure. It's easy to focus responsibility on the ubiquitous "they" -- management, customers, users -- anyone but ourselves. If you were climbing the 3,600 foot Salathe Wall on El Capitan in Yosemite -- would you delegate responsibility for success, or failure, to "they?" If "I" take responsibility for the product of my effort and the responsibility for improving the process of developing that product, then little by little the "I's" will complete the ascent.

Finale

There is no software development "silver bullet," not even Object Technology. Recently on a Compuserve forum someone suggested Martin write a follow-up to his book, Application Development without Programmers, and call it, Software Engineering without Gurus. In some respects the issue is to become our own Guru. Hopefully, Software Dynamics -- objective evaluation, adequate preparation, continuous assessment and adjustment of tactics, and reviewing final results -- done within the context of personal effectiveness and congruent action, will help in this quest.

The cultural changes, both at a management and a technical staff level, to enable organizations to move to a dynamic software development environment are significant -- and personal. By contrasting software development with a more directly identifiably, personal, risky activity such as mountain climbing, some useful insights to this change process may be more fully recognized and understood and acted upon. Good software teams, like good climbing teams, do not happen by accident.

