On Climbing Big Mountains

by Joe Marasco
Senior Vice President
Rational Software

I have been thinking that building a large software system is very much like climbing a big mountain. Surprisingly, I find that many of the lessons I learned from climbing reasonably big mountains -- of the Alpine, not Himalayan, variety -- apply. Both activities require the coordinated efforts of a team of highly qualified people under circumstances they can foresee and plan for, but not control. Success in both cases is a probabilistic calculation. This article explores this comparison in more detail.

Planning

For both activities, good planning increases the chances of success. The first step in good planning is to understand the scope of the task. For a mountain climbing expedition, this consists of understanding the height of the mountain, the relative difficulty of the terrain, special weather conditions, and so on. One can hardly imagine a climbing team, when asked about the height of the mountain they are about to scale, responding, "We'll know when we get to the top!" In a similar vein, we should expect a software team to understand the scope of the job they are about to undertake: How big is this mountain? How many lines of code, how many special device drivers, what fancy user interfaces, what performance characteristics, etc., will be required? In neither case can we foresee every possible "gotcha," but the salient features of the problem must be identified and written down so that they are addressed in subsequent planning.

A second step in planning is to review the scope of the project and the major obstacles to success and to identify the size and skill set of the team necessary to accomplish the objective. My experience in both domains tells me that the smallest possible team is best. All participants need to have a minimum skill level so they don't drag the team down. For

example, if the expedition warrants having a medical doctor as part of the team, the doctor should be a good climber and not a burden as the team makes its way to the summit. In both domains, participants who can play multiple roles are more valuable than specialists; when things get difficult, flexibility is an incredible asset in a small-team environment.

Selecting the Team

When identifying potential team members, it is important to evaluate candidates along several dimensions. Clearly, you are going to recruit specialists as required; if you must traverse a difficult, crevasse-riddled glacier, then you are going to need a good ice climber. In a similar vein, if you are producing a product with real-time requirements, you are going to need a runtime expert. In addition, as mentioned above, there should be a strong effort to choose overall competent climbers and engineers; in drafting for professional sports, this is the philosophy of picking the best athletes as opposed to drafting for position players. Finally, you need to gauge character; in addition to assessing skills, you need to evaluate how each individual will work as a team member, and how each person will bear up to stress and adversity. It is important that the prospective team member be a good climber and partner in foul weather as well as in fair. Since foul weather is certain at some time or other on most climbs, one can even make the case that performance under those conditions is the most important criterion.

One factor above all else helps to factor risk: experience. The more people you can add to the team who've been up this kind of mountain before, the better. Climbers who have done many, many mountains are sure to have better judgment, based solely on Darwinian considerations! This is known as the "Alaskan Bush Pilot Algorithm": When choosing amongst alternatives, pick the pilot who has been in business the longest. Experienced sailors have a saying that "Excellent sailors use their excellent judgment to keep them out of situations requiring their excellent skills." The same applies to mountaineers and software managers. It is better to avoid problems through savvy than to solve them through heroic efforts. The single easiest discriminator for judgment is experience. Look for people who have done it before and who have been successful. All other things being equal, take those people who have survived a tough mountain over those who haven't climbed at all.

Can you imagine attempting Everest without Sherpas?

Organizing the Team

Having put together a prospective pool of climbers/engineers, the team leader now needs to think about how to assemble the sub-teams. In climbing, this means allocating two, three, or four people to each rope.

It is a given that no one climbs unroped on a mountain of any significance. In addition to the great danger this would present to any climber foolish enough to desire it, it also represents risk to the rest of the party. They would have to take extraordinary measures to rescue their colleague from even the smallest of missteps. Likewise, in large programming projects,

lone-wolf behavior is very risky. At least two on a rope should be the rule.

For teams on which three or more engineers need to collaborate, pay attention to composing the right mix. You always need a strong lead climber, and the mix of skills and personalities needs to be distributed so that you don't wind up with one rope that is much weaker than the others. The guiding principle is balance: No one rope should be unbalanced, and the ensemble of teams should also be well-balanced. If you can't construct reasonable sub-teams from the pool, you need to analyze why, and either add to or prune the pool until you can.

The goal should always be the smallest number of small parties. Four teams of three or four players each can accomplish great things. Remember, in software engineering as in climbing, there is an insidious logistical pyramid silently at work -- the more people you attempt to put on the summit, the more people you need to support the effort organizationally. The growth tends to be exponential with the height of the mountain.

Scheduling

Another aspect of planning is scheduling. Once you sketch out the team, you can begin to figure out how much food and what kind of climbing gear you require. The software equivalent is figuring out how much development hardware and software you will need, and at what time. In order to make the resources come out right, you need some idea of how the team will progress up the mountain. Remember, there are three possible ways to go wrong: 1) not enough resources; 2) too many resources; 3) the wrong *kind* of resources, which translates to useless weight to carry. If the climb is going to take ten people four days, that is one set of resources; if it is fifteen people for two weeks, that is another matter entirely.

Now, the interesting thing is that in order to do this work, you need to know:

- What route you are going to take;
- What the intermediate stopping points are;
- About when you plan to get to each intermediate point with a given number of people.

That is all you need to draw up a schedule and calculate what you require to get the job done. You don't need to know every detail of how you are going to get to each stopping point. That's a good thing, because any plan that depends on that information is highly risky: most of those details are unknowable with any certainty before you actually get on the mountain. Even if you plan them down to the gnat's eyelash, they will all change as you make progress. Barring contingencies that force a route change, however, the overall stopping points, or milestones, should not change much.

Milestones are useful for two things: to get a gross idea of how you are

going to do the job and what resources you will need; and to monitor important progress. If you thought it was going to take you two days to get to your first base camp and it takes you six, then you had better sit down and think about the rest of the endeavor. That the team's morale is still wonderful and that the base-camp is the best-designed one you have ever seen are largely irrelevant, considering that the whole project is likely to take (at least) three times as long as planned.

Milestones or Inchpebbles?

So I advocate rough, bottoms-up planning with consensus from the team about how long it is going to take to achieve significant milestones. And I believe in taking very seriously the time it takes to actually get there. For this to be useful, the milestones cannot be too close together nor too far apart. On a climb of a day or two, the milestones are typically on the order of a few hours apart. For a climb in the Himalayas with a duration of weeks, my guess is that significant milestones are days apart. For a software project that has a duration of eighteen to twenty-four months, somewhere between three and six months feels right. All these translate to about six significant milestones for the project, give or take a few. If you are using an iterative development approach, the implication is that about six iterations will get you there. If it actually takes many more or many fewer iterations, that probably indicates that the granularity was wrong for the project-level planning.

There is nothing wrong with each sub-team having some finer granularity tasks if that helps. It is up to each team leader to organize his rope to make sure his party arrives in synch with the others.

Monitoring and Record Keeping

Most experienced team leaders I know keep some records in real time. For climbers, a small pocket notebook and stubby pencil usually come out at rest points, and some notes are written down. When we examine these notes after the climb, we find that, although we did not capture a lot of information, it is always very much to the point. Typical notations are about arrival times, discrepancies from planned arrival times, and unusual conditions. Sometimes they are informal notes on how the individuals and teams are performing. When you're planning a climb, notes like these about similar mountains can prove useful for adjusting initial estimates. For software projects, similar notations can be useful for gaining insight into actual project performance.

Handling Risk

Project plans also need to address contingencies. In climbing, the two variables that represent *forces majeures* are surprises in the selected route and the weather.

If the route chosen beforehand turns out to be too risky, an alternative route needs to be selected. This usually occurs because the ice or rock is not in the same condition it was in the last time this route was explored. A good plan uses natural stopping points in the climb to assess options and

choose among alternative routes. The software development analogy is to assess technical direction continuously, and then, based on the results achieved with each iteration, change the route slightly for the next iteration. Note that the goal, the mountain's peak, is a constant. However, changing conditions may affect our judgment as to the best way to get there. It is rare that there is only one path, and the superior mountaineer distinguishes himself by finding the right path in the face of new data.

The weather is a different thing entirely. When the weather turns on you unexpectedly, the whole nature of the enterprise changes radically. Now it is not an issue of getting to the top, but one of survival. Even if the party decides that further immediate progress is impossible, and that the correct strategy is to "hunker down" until the storm blows over, they may still perish if they run out of food before the weather abates. Because you have absolutely no control over the weather, you must view it with the utmost caution; the strength of your team can become irrelevant. More people die of ego on mountains than any other cause; failure to turn back at the right time can be fatal.

I think the programming analogy is when you find yourself dependent on things you can't control directly. This includes new, untested technology, scheduled miracles, required violation of known laws of physics, and internal and external suppliers and subcontractors. Ignoring changes of weather in these areas can lead to death, either instantly or in a painful and protracted fashion.

Decisiveness

I have never met a good climber who was indecisive. I have known climbers who confused recklessness with decisiveness. There is a difference. (I might also add that I have known old climbers, and I have known bold climbers. I have not known any old, bold climbers.)

I think the main thing I learned when climbing was that you don't have the time to agonize over decisions. That doesn't mean that you can afford to shoot from the hip. You often have to consider alternatives that are difficult, situations in which one wrong branch can mean the difference between success and failure, or, in the extreme, between life and death. Sometimes the choice is not obvious.

However, what you cannot afford to do is to become paralyzed and continue to defer the decision -- the "paralysis by analysis" syndrome. You must take some time, consider the options, gather data, and then decide. Once you decide, you go forward and don't second-guess yourself.

This may mean walking a fine line in terms of team dynamics. It is critical to build consensus around important decisions. However, consensus building itself can lead to paralysis. At some point, it is the leader's responsibility to make the decision if there is an impasse. If the team has been correctly assembled, they will then execute, understanding that this "best" decision is better than no decision at all.

I once was perched on a rock trying to decide which of two ugly paths to

follow. My climbing partner humored me for a little while and then said, "Well, you can make a decision or you can sit here and freeze to death." It's the same thing in software development.

Common Goal and Focus

On a good climb, everyone agrees on the purpose. Usually, that means getting to the top. Everyone focuses on that. Anything that doesn't contribute to the goal is ruthlessly avoided: no side trips, no one going off to pick flowers, no one stopping for a half hour to take pictures, and so on. The team can agree beforehand that some of these activities are part of the climb, and thus sanctioned; however, it is very important that there not be confusion as to the principal objective and how it is to be attained.

The software development analog is staying focused on the objective, which is usually to build a software system. Interesting software side trips can sabotage the whole effort, especially if one sub-team ends up in a crevasse. On a related subject, don't worry about style points. Mountaineers don't award any. Getting to the top "ugly" beats an aesthetic retreat anytime. This is not a personal opinion; this is the way most of the world keeps score.

Taking the Long View

In talking about climbing a mountain, we often make the mistake of focusing on getting to the top, as though that were the only goal. Getting to the summit and then getting the whole team back down the mountain in one piece is the real objective. (In a similar vein, the objective of the space program in the sixties was not to put a man on the moon; it was to put a man on the moon and bring him back alive.) The software analogy for this mistake, to stretch the point a little, is to focus all the effort on the development necessary to ship the initial version of the product. This allows you to "plant the flag," as it were. (Sometimes we proclaim victory even earlier -- on shipping a first beta copy, for example.) I contend that the moral equivalent of climbing the mountain and getting back down safely is shipping a software product that you can support and maintain. It means putting together a product whose software is robust, whose documentation is complete and understandable, and whose support burden doesn't kill the rest of the organization. So when we plan a software project, we must plan to do the whole job, not just plant the flag at the summit.

Competition Can Cause Irrational Behavior

One of my sons, Dave, points out that the "purity" of mountaineering stems in part from the romantic notion of an isolated team striving against the elements to achieve a noble goal. He also points out that this ideal is often violated in the real world by competition between teams. There may be two or more groups striving for the same peak, all desiring to "plant the flag at the summit" first. (Some areas of scientific research suffer from a similar gap between the romantic ideal and the real world.) In software development, of course, the competitive pressures are even more intense, as the product under development is a competitive weapon that the rest of

the organization wants in its arsenal immediately. The effects of this pressure can be catastrophic; often, *en route* changes to plan in response to the competition cause the team to take risks that are too high, leading to failure or worse.

Common Causes of Failure

Why do some software projects fail? For some of the same reasons climbing expeditions fail:

- Trying to get to the summit too quickly.

  Analog: Unrealistic schedule from the start.
- Trying to get to the summit with clearly inadequate resources.

  Analog: Not enough good people or equipment.
- Climbing with a team that is too big; the logistical and communications burdens overwhelm the team.

  Analog: Too many average people.
- Taking too long; teams that stay on the mountain too long lose their verve, energy, and desire -- fatigue takes its toll. Also, they may flat run out of resources.

  Analog: Software projects that stretch out forever, taking so long that the requirements get changed, sometimes multiple times.
- Sticking to the wrong route in the face of new information.

  Analog: Ignoring data from early iterations; failure to adjust the plan during the course of the project.
- Getting wiped out by circumstances beyond the climbers' control.

  Analog: Supplier or subcontractor failure; failure of a key component that was really an R&D activity, not product-ready.
- Not having a reasonable plan that everyone understands, believes can succeed, and is totally committed to.

  Comment: Usually the result of a top-down, hierarchically-mandated plan.
- Failing to execute, within tolerances, against the plan.

  Comment: Sometimes results from the accumulation of many small slips, rather than any one spectacular failure.
- Losing gumption when the going gets tough; not understanding that adversity is part of the endeavor.

  Comment: Just as bad at the office as on the mountain.
- Not having any reserve for emergencies.

  Comment: Usually the experience of the senior players can provide

some of this reserve; they'll know what to do when the unexpected happens.

Ingredients for Success

What are the hallmarks of successful teams? Here are a few:

- Good planners, but not obsessive about it.

  *Comment:*A small amount of good planning beats a lot of detail every time.
- Ability to move fast with small teams; get on and off the mountain before Mother Nature changes her mind and decides not to let you climb this one this day

  .
  *Analog:*Get it done before the requirements change.
- Talent for assessing incoming data in real time and making appropriate changes to the plan at appropriate times.

  *Analog:*Use iterative development, integrate early and often, and use the information to adjust your plan.
- Good balance between top-notch individual contributors and good team players and leaders; the key here is usually very wide communications bandwidth.

  *Comment:*Need to have balance and shared mind-set.
- Monitor against plan at the appropriate granularity.

  *Comment:*Need a sense of when you are getting in trouble, and what to do about it.
- Leaders display maturity and good judgment.

  *Comment:*Knowing when to amplify and when to dampen is important.
- Stamina: Understand that overall extended performance is much more important than burst-mode performance. It is no surprise that most climbing leaders come into their own in their forties, not earlier.

  *Analog:*It would be interesting to see the statistics on software leads.
- Toughness: Ability to bear down when things go badly.

  *Analog:*Important when tracking down really hard bugs, for example.
- Focus: Team stays centered on a clearly-defined objective.

  *Comment:*All members know what, why, when, and how.
- Creativity: Willingness to experiment and to experience genuine joy in what they are doing.

  *Comment:*Just like the rest of life.

The Human Factor

Well, in some respects this comes down to saying that people are everything. For software projects, as in almost everything else worth doing, I'd like to paraphrase the novelist Irving Stone: "Give me men (and women) to match my mountains."

For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!