Managing Collaborations

Mary E.S. Loomis
Software Technology Laboratory
HPL-97-117
September, 1997

collaboration,
program management,
project management

Managing software project teams can be quite a challenge; many of us bear scars we've earned in the process. Managing the particular kind of software project that I'll call a "collaboration" involves further complications. I want to take this opportunity to discuss some of the guidelines I've gathered about how to increase the probability that a collaborative software project will be successful.

Internal Accession Date Only

To be published in *Software Development*, Volume 6, Number 1, January 1998.

MANAGING COLLABORATIONS

Mary E.S. Loomis, Ph.D.
Hewlett-Packard Company

Managing software project teams can be quite a challenge; many of us bear battle scars we've earned in the process. Managing the particular kind of software project that I'll call a "collaboration" involves further complications. I want to take this opportunity to discuss some of the guidelines I've gathered about how to increase the probability that a collaborative software project will be successful.

What is a collaboration?

I'll use the term "collaboration" here to refer to a project in which

different parties are working together for some period to achieve a common goal.

Each phrase in this description is carefully chosen to focus our discussion on a particular kind of project.

- *different parties* -- The people involved in the project are from different organizations. Multiple management chains are involved, sometimes in different companies. Multi-organizational projects are especially challenging to manage, as each organization introduces its own motives, culture, and reward system (not to mention personalities and egos) into the soup.
- *are working together* -- Real energy is expended in cooperative effort. Together the parties produce some software result, ranging from a specification to a running system.
- *for some period* -- The joint effort has a finite life-span. The parties will eventually return to their separate efforts. Often the parties continue to pursue their separate efforts in parallel with their work on the collaboration. The life-span of a collaboration is typically relatively short, although depending on the circumstances, that lifetime may be measured in units of years rather than weeks. Nonetheless, a team member should not plan to make a career out of his or her participation in the collaboration.
- *to achieve a common goal* -- There are valid reasons that the parties are working together. There is some identifiable rationale for forming the collaboration and for management and participants to be willing to deal with its inherent complexities.

This topic – making collaborations successful – has been on my mind a lot over the last several years, and especially recently. I hope that the topic has some general appeal; my observation is that collaborative efforts are becoming increasingly common. Quite often, my success as a manager is largely determined by the ability of people in my organization to collaborate with people in other organizations.

Characteristics of successful collaborations

Here are some of the characteristics of successful collaborations that I've observed. They're listed in no particular sequence; all seem to be important.

1. Leadership. A successful collaboration has a leader, who is respected by the team. The leader should be either designated by some higher-ranking authority, or appointed by the team itself. Sometimes it works to have two co-leaders, but they must trust each other and communicate well. It typically does not work to have two leaders, unless they agree up-front on what to do to clearly resolve situations where they disagree
2. Goal. A successful collaboration has a clear goal and well-specified expectations. All the team members understand why they are working together, rather than having some other non-collaborative project pursue their goal. The team members together frequently assess their progress toward the goal. The leader keeps the goal highly visible, and uses the team members' commitment to pursue that goal to keep the effort on track. It is very easy for collaborative efforts to get bogged down in extra work, if the goal is obscured.
3. Roles. In a successful collaboration, each team member has a well-defined role. If multiple team members from the same organization have identical roles, then perhaps one of the team members is superfluous. In the most successful collaborations, each team member trusts the others to successfully accomplish their roles. Without this trust, team members may encroach on each others' territories in

attempts to get the overall task done. Territorial encroachment can threaten a collaboration's chances of success.

4. Buy-in. A successful collaboration has buy-in from all appropriate levels in the participating organizations. It is important to identify not only the stakeholders who have resources invested in the effort, but also the managers who can make strategic decisions that could torpedo the effort. The team should identify these people and communicate with them, asking them to state their endorsement of the work, both within their organizations and to the team. One technique that works well to ensure continued buy-in is to hold reviews of intermediate results, inviting the peers in the involved organizations (especially within the same company) to attend the same review. For example, an architectural review of an effort intended to provide an integration path for multiple software product lines might include the division managers from both the affected divisions, rather than conducting a separate review for each.

5. Schedule and interdependencies. A successful collaboration has a clearly visible schedule, with reasonable delivery points and visible recognition of dependencies across organizations. There is a process for handling changes and for tracking status. One sign of a software collaboration that is in trouble is one where there is no schedule reflecting the combined efforts of all the participants. The schedule is also a way to clarify the expected end of the collaboration. Successful collaborations have finite lifetimes and do not drag on forever.

6. Resolution process. A successful collaboration has a well-defined process for surfacing and resolving issues. It is important to decide on such a process, before the issues begin to arise. A well-defined issue resolution process is especially important when a collaboration is being co-led by two (or more) individuals. There must be some way to escalate and make progress. The process must fit the cultures of the organizations. Some organizations are more open and less threatening than others.

7. Communications. A successful collaboration has a well-designed system of communications. It matters not so much what tools are used, but rather that the team decides on how the communications will take place. Typical tools include regular meetings (in person or via teleconferencing or video conferencing, with notes distributed afterwards to the team members), shared workgroup databases, email and voicemail distribution lists, shared sets of presentation materials, source code control systems, and document version management systems. Agreeing on a common set of tools can make a huge difference to the operation of a collaboration. Paying significant attention to regular communications and being flexible are especially essential in collaborations that involve geographically dispersed teams. Finding mutually acceptable meeting times, when teams are many time zones apart can be a challenge. Either find an acceptable time and be consistent about using it, or move the time around so as to equally inconvenience everybody. Make it clear that participation in communications meetings is mandatory, and make them efficient.

8. Vocabulary and mindset. In a successful collaboration, the team members understand what other team members are talking about. It is worthwhile to invest the time required to develop a common terminology and vocabulary. Requirements and design models are important to nearly all software development projects, and especially so for collaborative software development processes. These models help clarify the semantics of the project area. They can be instrumental in helping the team members understand each other. However, vocabulary is not enough. In a successful collaboration, the parties understand each others' frames of reference, personal motivations and business reasons for being involved.

9. Credit. In a successful collaboration, there is plenty of credit given for each others' efforts. Team members should openly acknowledge each others' contributions, especially across organization boundaries. Managers of team members should give credit to the other organizations. After all, together the collaborating organizations are expected to achieve the goal better than any of the individual organizations could do alone. Extensive credit-giving works best when practiced by all the participating organizations. Even if you find your organization on the short-end of the credit-receiving process, I suggest you not scale back. Acknowledgement is noticed and can make a huge difference to the quality of the working relationships within the team. Consider your management role as being to support the team and to ensure the success of the overall effort, not to grab charter or to compete with your collaborators.

10. Don't force it. Don't collaborate merely for the sake of collaborating. These situations are typically doomed. The collaboration introduces more complexity than value, and the effort could have been pursued better by one (maybe any one) of the organizations by itself.

What makes software collaborations different?
These characteristics are mostly people-related and probably are not unique to software collaborations. What makes software collaborations different, anyway? There seem to be a few factors worth noting.

First, it can be difficult to measure the progress of any software project, and especially of a collaborative software project. Managing inter-organization dependencies and risks is complicated substantially by the uncertainties seemingly inherent in accurately predicting the schedules of software projects.

Perhaps the smoothest multi-organization project I have had the pleasure to be involved in recently was not a software collaboration; it produced a software conference. I am the Conference Chair for ACM's OOPSLA'97 Conference, which draws more than 2500 participants and over 100 presenters from all over the planet. It is a complicated conference and the efforts of more than 200 volunteers (from nearly as many organizations) and about four paid professionals, on over a dozen sub-committees, must be managed to pull this thing off. It's about an eighteen-month intensive effort. Contrary to most software projects, it has been really straightforward to assess progress. We're not producing code, so I never hear "it's 80% done" or "I've just fixed the last bug." This effort is also different from most software projects, in that the conference is redeveloped every year. Each year the new conference chair and committee get to build on a rich history of recorded experience.

Another difference inherent in software collaborations seems to be the extent of inter-dependencies. Much of software development tends to be side-effect rich. Unmanaged inter-dependencies have a way of showing up later as bugs. Management attention, careful modularization and encapsulation can help enormously.

The software business also lacks the kinds of blueprint languages that are typically used to ensure valid communications in other kinds of collaborations, e.g., in designing and implementing buildings, computing hardware, airplanes, automobiles, or toasters. No matter how conscientious we are about specifying software interfaces or functionality or expected quality of service (performance, reliability, security and so on), we're hardly ever quite precise enough. Also, unlike in these other industries, we in the software business don't have large stores of tested, standardized parts to draw from in constructing systems. Without standardization, communication of the details becomes even more essential.

Example collaborations
Let's look at the characteristics of several example software collaborations, some more successful than the others.

- Next-generation product architecture. Goal: to specify and prototype an architecture to enable multiple successful software product lines to converge and be extensible into the future. Scope: within one company, with three major divisions involved. Timeframe: 6 months. This collaboration has all the characteristics of successful projects listed above, and is judged a success by the team and by their management, all the way up to their point of convergence in the organization chart.
- Another next-generation product architecture. Goal: same as above, but for different software products. Scope: within one division, with three geographically distant operations involved. This effort failed and was eventually abandoned. Several characteristics of successful collaborations were missing. While there was a clear leader and a specified goal, the buy-in of affected parts of the organization was sorely lacking. Top-down endorsement of the effort was visible, but the team's results had no impact on the work or plans of the very independent (but to-be-affected) operations. The lower level managers did not buy in to making changes, but rather continued to focus individually on being successful with what they had (which, by the way, was also the way their performance was measured).
- Modeling language standardization. Goal: to reach convergence on a single specification for a language for object modeling. Scope: an industry consortium (the Object Management Group) of

companies with vested interest in object technology, either as vendors or users. Reaching convergence required achieving buy-in from about a dozen tools vendors and methodologists, endorsement from a task force of representatives from dozens of vendor and user companies, and votes from a committee of representatives from even more companies. The success of this effort falls into the bucket I label "Minor Miracle." As co-chair (with Jim Odell) of the OMG's Object Analysis & Design Task Force, I witnessed the importance of each of the characteristics listed above.

- Information system research pilot. Goal: to develop technologies for integration of applications in a particular domain (I'd rather not reveal which) and to pilot the resulting system with real users. Scope: within one company, with two major divisions involved, and with multiple development groups, then deployment at a large customer. While this effort eventually produced a small-scale deployable system, it definitely did not achieve its goal. The people involved learned a lot (and in that sense it was a successful experience), but it was not the success envisioned at the outset. Let's see why.
  - There was no clear leader. Sometimes the marketing group was in the lead (and making commitments) and sometimes the development group was in the lead (also making commitments).
  - The goal was really not very clear; it kept changing as the definition of the desired results evolved.
  - Although roles were stated, they weren't very clear and some team members didn't trust others to successfully execute their responsibilities. For example, one application team needed a fix to the infrastructure, but didn't trust the infrastructure team to be responsive, so made the fix itself without communicating with the infrastructure team. Then the application team frequently over the next months complained that it was overworked, relative to the infrastructure team, which apparently had less to do!
  - Buy-in was insufficient. Many levels of managers were involved. Nearly all peers communicated openly and well, except at the second-level manager level. Mixed signals were getting to the team members, and priorities became confused. Then the torpedo arrived: a high-level manager of one of the organizations dramatically changed that division's business strategy, making the success of the collaboration essentially irrelevant to the division.
  - Believe it or not, this effort went on for a very long time with no real schedule. There was a schedule, but it bore little resemblance to reality. Some of the first-level managers were quite frightened by schedules, couldn't seem to commit to realistic dates, and didn't report slippage when it did occur. The schedule had more than one instance of that hoped-for event, "And then a miracle occurs…"
  - There was a defined resolution process, but it lacked an effective way to identify issues. Team members were expected to surface issues by speaking up in rather large monthly status meetings. More frequent, one-on-one communication would have been much more effective for this group of people, with anonymous display of the issues for discussion at later meetings.
  - While most of the team members and their management chains were quite generous in giving each other credit, there was one manager who was notably less generous. The result was quite a bit of animosity directed toward that manager. This did not help morale at all.

  Need I say more?

The challenge of the obvious…

Actually, the characteristics of successful projects that I listed above appear to be fairly obvious. We probably all understand their essential contributions to team success. Perhaps I need not have repeated them here, but I hope there's value in capturing them in one place. Many of the principles of management are pretty obvious, yet we sometimes have trouble implementing them in the real world of our daily jobs. Perhaps there are other guidelines you'd like to add to the list. I'd really welcome hearing your ideas. Maybe we could collaborate!

Author's bio

Mary is Director of the Software Technology Laboratory of Hewlett-Packard Laboratories, at Palo Alto CA. She has many years of experience in both inter- and intra-company collaborations. Her primary technical interests are in object modeling, software development processes, database management, and

workflow. She earned her Ph.D. in Computer Science from U.C.L.A. in 1975. When not working or writing, she likes to run, cycle, and ski, and sometimes even manages to keep up with her daughters.