

# HOW TO Manage Collaborative Software Projects

Phil Dwyer and Rob Purdie, PMP

Commissioned by Katrin Verclas, Aspiration

Project management is, at its heart, the art of managing communications between all parties involved in a project. It's about creating a network through which communications can flow, and rules (or protocols) to allow them to flow more quickly and efficiently. The more people involved in a project, however, the more complex it becomes to manage the communications between them, and as we know from network theory, that complexity scales very quickly. Mathematicians will tell you that network complexity scales exponentially – networks get really complex really quickly, and chaos is often the result.

Why then do we bother to undertake *collaborative* software projects? Why take on projects

1. Design a Structure
2. Establish a "Leader"/Facilitator
3. Define Roles and Responsibilities
4. Consider User Needs and Types
5. Prioritize Requirements
6. Identify Common Goals
7. Leverage Experience
8. Plan for Training

*The eight things this article argues you should do when managing collaborative software projects.*

involving multiple organizations, where communications are exponentially more complex and difficult to manage? Because when we collaborate we can create more value. We can, for example, share scarce resources (human capital as well as cold hard cash) and expertise; we can get scale benefits (i.e. we don't have to "re-invent" the application for every partner involved); and we can share project risks across all involved parties. But the added complexity *does* make collaborative projects much more difficult to manage.

Some of the communications issues that tend to arise on collaborative projects are the same as those on any project – poorly defined/understood scope; poorly defined/understood roles and responsibilities; unidentified project risks/poorly defined risk response

plans; poorly coordinated activities/confusion around who is doing what and by when; poorly communicated status/confusion around what work has been done and what work remains. For the

purposes of this article, though, we've tried to focus on issues specific to collaborations between loosely allied sets of partners. Our "lens" for this analysis is that of two collaborative software projects involving NGOs which ran into significant problems and roadblocks typical of ventures of this nature.

The Food and Commodity Tracking System (FACTS) is a web-based logistics application designed to improve the delivery of international food aid. Its development involved five not-for-profit humanitarian relief organizations wanting better tools to manage their warehouses, co-ordinate shipping, and to issue reports to funding agencies and head offices. It was developed with the help of Microsoft and Aspiration.

The system took three years to develop (between 2000 and 2003) and was pilot tested in five field offices (Kyrgyzstan, Indonesia, Bolivia, Ghana and Guatemala). Today, very few of those offices are still using the system and it is unclear at this point how the product can be moved forward.

According to an evaluation of the project by Mark Surman (now with IRDC in Canada, and formerly of the Commons Group), the problems encountered included:

- Confusion about leadership and responsibilities
- Lack of follow-through
- Funding problems
- Complaints that the software is headquarters-centric and lacks features needed in the field
- Complaints about the software's lack of flexibility
- No provision for ground integration support

## 1. Design a Structure

The first point in Surman's list is surely one of the less welcome outcomes of collaborative ventures: projects need champions, or leaders: without them they are likely to become mired in confusion or worse, apathy and inaction. When you set out on a collaborative project of this sort, you *must* hammer out an organizational structure for the project with your partners. It's essential that you collaboratively answer the question, "What will this collaboration look like?" This structure will allow the participants to begin the process of fleshing out roles and responsibilities.

Before getting to that stage, however, it's important that you establish a leader or facilitator for the project.

## 2. Establish a "Leader"/Facilitator

NGOs often manage collaborations by consensus and sometimes find the notion of leadership unpalatable. Though understandable, this reaction should be resisted: projects need someone to champion them to their goal.

Every project is different: projects are made up of teams of individuals, and these teams develop group dynamics unique to themselves, so it's impossible to prescribe a "best practice" way of settling on your leader.

However, it is absolutely vital that everyone involved understands that the leader's job is simply to steer the project to a successful completion, and to keep the team focused on achieving the objectives of the project. It's also important to stress that the selection of a leader in no way devalues the input and work of the other contributors. The leader's role should be to make it possible for everyone involved to contribute their best work.

The best venue for this discussion is face-to-face. Every participant should be prepared to sit around a table and work out who would be the most appropriate leader or facilitator for the project.

### 3. Define Roles and Responsibilities

As part of this process, you'll have to define the roles and responsibilities with each partner involved. They will need to be even more clearly defined in a collaborative project: we often make implicit assumptions about roles and responsibilities based on our knowledge and past experience of working with colleagues. In collaborative projects, where you don't have the same pool of implicit knowledge to draw upon, such assumptions have to be made explicit, if the collaboration is to succeed. If you can't clarify who should do what and by when, the end result will be number 2 on Surman's list – lack of follow through.

Think of this as a two-phase process: first, allocate the general roles and responsibilities of the team members (i.e. Rob will be project manager, Phil will be tester, etc.), and then go on to the specifics – define responsibility for the completion of each project deliverable.

It's useful, at this point, to have a tool which helps you break the project down into its component parts – the specific deliverables which must be produced in order to fulfill the project requirements. A very popular tool used for this purpose is the Work Breakdown Structure (WBS) – a tool that we would argue should be employed on all projects, but *especially* on collaborative projects.

Now, referencing a completed WBS, map each deliverable to the individual team member who will be

responsible for its delivery. A popular tool used for this purpose is the Responsibility Assignment Matrix.

#### Related resources at:

<http://importantprojects.co.uk/resources>

Work Breakdown Structure (WBS) Template (.dot )

Responsibility Assignment Matrix (RAM) Template (.dot)

### 4. Consider User Types and Needs

You might argue that the 4<sup>th</sup> item on the list, which comes down to a failure to consider different user needs (and types) when defining the requirements for a project, is not unique to collaborative ventures: and that's true, to some extent. It's an issue that plagues many software projects, collaborative or not.

However, collaboration doesn't just amplify this problem, it changes its nature. In the case of the FACTS project, the team defining the requirements of the software did consider one dimension of the complexity – they made sure that all the partners in the venture were consulted at an early stage. But they forgot that there are (at least) two dimensions of users – "down" the organization as well as "across" the alliance. Field workers were not involved in the requirements definition phase of the project, with the result that the software, when it was released, lacked many of the features needed in the field.

This failure to involve users led to complaints about a lack of flexibility in the system from field workers who needed to add their own data fields because, for them, the software had been poorly designed. In the end, the pilot users who were testing the system went back to their previous habits (Excel spreadsheets or paper) after trying, and failing, to work with FACTS.

What should they have done? Clearly, for complex tasks, it's not enough simply to define what needs to be done. It's often the case that there simply aren't enough resources to do everything you

want to do. The second collaborative software project we studied illustrates this point very well. The Hands on Tech (or CityCares National Technology Initiative) project aimed to use technology to make it easier to match up volunteers with appropriate NGOs. The 12 affiliates involved in the project planned to use a Web-based data management tool to achieve this aim.

This was an ambitious goal. Managed by an experienced not-for-profit project manager who had no software development experience, however, in the end it was a little too ambitious. The requirements defined for the project grew until the list of requirements began to outstrip the resources of the project.

A project is, by definition, a *temporary* endeavor; it has a definite beginning and a definite end, and is always constrained by scope, time and cost (these three constraints are often referred to in

**Related resources at:**  
<http://importantprojects.co.uk/resources>  
 Scope Statement Template (.dot )

project management as the Iron Triangle or the triple constraint). Project scope cannot be changed without impacting time, cost, or both. So, before diving into a detailed requirements definition process, set boundaries for the project team and stakeholders by completing a Scope Statement. While doing so, spend some time documenting the "known exclusions" to the project scope as well (a step that is often overlooked but very valuable in

setting expectations with project stakeholders).

Second, math doesn't lie. Use a spreadsheet to work out what resources you have for requirements definition, and then track your "spending" regularly against your budget.

## 5. Prioritize Requirements

Of course, simply defining your requirements is not enough. Suppose you have 127 requirements. Now imagine that each requirement is vital to one of the partners in the venture. But you don't have the resources to build all 127 requirements into the final product. What do you do?

Failure to prioritize requirements is another of those issues which may not seem particular to collaborative projects until you stop to consider the social dynamics at play in a project involving multiple organizations.

Hands on Tech had initially come together to achieve economies of scale. The idea was logical: pooling resources would allow everyone involved in the project to get what they wanted without having to invest what it would have cost them to develop the entire application themselves. But, since everyone now has a stake, the natural inclination of stakeholders is to place their own requirements at the top of the list, and push for less than vital requirements to be included.

To deal with this problem, you first have to come to an agreement on the overall priorities: what requirements does everyone agree on? This discussion should provide you with core functionality everyone agrees is crucial to the final product. This will ease you into the more contentious territory of prioritizing the requirements that remain. Try to draw the emotion out of these discussions by focusing on the key issue: how well does each piece of functionality fulfill the most important requirements of the project. If you can, reduce this "fitness factor" to a number – score each "candidate" by how well it achieves the project objectives, if necessary giving each partner in the venture the chance to score each one separately (as long as all parties agree to abide by the resultant priorities ranking). And

remember at this point to involve the fieldworkers who will be using the product as well as all the partners in the coalition.

## 6. Identify Common Goals

Here's where the specific features of collaborative projects become important. Economies of scale only work if participants have common goals. If everyone comes to the project with a completely different set of requirements, there are unlikely to be huge cost savings.

There is no easy solution to this problem. The only way to reach an agreement is to get all of the involved parties together in a room and brainstorm on them. Be sure to keep the discussion focused: projects help organizations achieve their strategic objectives. When a coalition is involved it's more difficult because each partner may have different objectives.

Ask yourself if your organization, and the other partners in the project, are good at working with other people. If the answer is no, bring in an external facilitator to mediate these discussions.

Also, beware of the phenomenon known as "scope creep." In a collaborative environment, this is likely to be fuelled by unreasonable expectations. For example, it is not uncommon for features which had been on a "nice to have" list, when an agency is the sole funder of the project, to graduate to a "must have" list, if it gets turned into a collaborative venture. Because resources are being pooled, it is assumed that resources are (for practical purposes) infinite.

It is unlikely that any of the organizations involved will want to sacrifice any of their own requirements in order that others can have theirs met: charities don't always behave charitably when spending their hard-earned funds. In the Hands on Tech project, this factor alone presented a significant hurdle.

The 12 affiliates had an initial budget of USD \$500K, and a timeline of 9 months. But they had unrealistic expectations of what could be achieved. They worked on a list of requirements which, because of its sheer size and scope, had no hope of being fully implemented.

When this became obvious, the list should have been prioritized and pared down, so they were only working to define requirements which had a chance of being implemented. In the end, despite the fact that the list of requirements was eventually halved, in an attempt to achieve a more realistic and achievable list, the initial project overspent its budget. The organizations involved had kept on defining requirements until all the money available was spent and had failed to prioritize any of them.

And yet, despite these almost catastrophic problems with the initial project, Hands on Tech is now making money. How was it turned around? The partners in the venture realised that they didn't collectively have the right expertise to manage a project of this size and complexity, and brought in an experienced software project management team to help them.

## 7. Leverage Experience

Additional experience on either of these projects may have helped both avoid some of these pitfalls. It is starkly evident that, despite the size and complexity of these projects, neither team sought expert help initially.

If you have nobody who has the experience you need in your organization, you should find someone who does. You may think that bringing in expert help will be more expensive, but as both our case studies demonstrate, mistakes can be far more costly.

Who should you look for when bringing in extra help? It's certainly true that the more experience someone has in an industry, the more effective they can be when running a project in that industry.

But get your definitions right: what experience are you really looking for? If your project is a software development project, our advice would be to look for someone with that specific experience: experience in project management in the not-for-profit world is no guarantee of success in software development. Software development, and especially complex, multi-stakeholder projects, requires experience specifically focused in this field. At the same time, project management with knowledge of and experience in the not-for-profit sector with its multiple bottom lines and often different organizational and management cultures is advisable to ensure a good fit.

## 8. Plan for Training

Finally, when you are allocating time and budget for your project, remember to include some resources for training. FACTS was not widely adopted because the pilot users had no training: they had to learn to use the system and integrate it into their day to day work by themselves. Many simply gave up.

## Conclusion

Building effective collaborative networks (human, organizational, technological) offers us huge benefits. They allow us to achieve complex and difficult tasks we could not have dreamed of achieving – they offer us a treasure trove of information, knowledge, insight, shared costs, skill and experience. But we also have to be acutely aware of the complexities they introduce to *already* complex tasks and projects. Simply adding one new partner (or node to your network) will scale its complexity as well as the benefits it offers you.

With the right planning and execution, collaborative software projects hold the potential to help NGOs achieve efficiencies of scale and cost savings on their software projects. But don't assume those benefits will be achieved as a natural outcome of collaboration: they have to be earned by managing and nurturing the network you create, or your project will quickly spin out of control.

---

For more information, <http://importantprojects.co.uk>