A BPTrends Column May 2008

Human Processes

Keith Harrison-Broninski
CTO Role Modellers (www.rolemodellers.com)

harrison_broninski@acm.org

Hyper-productivity

Regular readers of this column will know of my view that the next significant step in both IT and management is to improve the way that people do collaborative knowledge work (what I call "interaction work"). You can find all sorts of evidence for this view on the Human Interaction Management Web site.[1] However, most people know from their own experience that action is needed and the sooner the better. In particular, working hours are getting out of control, a trend that is not helped by the current expectation that everyone is "always on" via their mobile phone.

A typical factor in people's inability to control their workload is the necessity to use inefficient workplace software, the main culprit being email. Across the board in industry there are major problems resulting from use of email. It is not at all clear when emails should be sent, what they should contain, who should be included, the validity of requesting actions from people, who has committed to do what, the correlation between different email streams, the accuracy of data included, etc. Almost daily, this lack of clarity causes material problems in organizations of every kind.

Problems such as email usage are complex to resolve, relating as they do to process awareness, new approaches to management, new forms of software, and more. Hence solutions for such problems can be expensive to implement in large organizations. How can one justify the expense? In other words, what is the basis of a business case for the introduction of more efficient human collaboration?

It is necessary to find a *quantitative* means of evaluating the impact. One can list ad nauseam aspects of organizational life that will be improved by rationalizing the way people work together, but the board have a duty to consider the bottom line in any major decisions they make. However swayed they may be personally - e.g., from experience of mobile phone calls at ungodly hours - they will not be acting with due diligence if they approve a program that has no means to demonstrate **Return On Investment (ROI)**. So it is necessary to find a metric that is applicable to human collaboration.

This is non-trivial, since there is a difference between efficiency and effectiveness.

When measuring the impact of process improvement in mechanistic areas such as transaction handling or manufacturing, the outputs of the process are probably going to remain similar after the changes have been implemented. The aim is simply to produce these outputs quicker and cheaper. In other words, one is aiming for increased efficiency.

[1] http://human-interaction-management.info

The same is not at all true for human collaboration. Free people up by helping them work better, and they will deliver more value to the organization. Once people are not struggling to keep afloat, they can help steer. In other words, improving interaction work delivers increased effectiveness. However, in advance one cannot always predict what form the increase will take. Take sales, for example. One might expect a salesperson who works better to make more sales, but it is quite possible that the actual improvement will be customer relationships that are longer-lasting, something that cannot be measured until enough time has passed.

So what metric should one use when proposing a program to improve human collaboration in the organization? Ideally, the metric should be as simple as possible, to reduce the degree of contentious debate around improvement initiatives.

A very simple metric

One particular metric for human collaboration is both very simple and very reliable. Work out how much you pay people per hour. For permanent employees, make sure to use "fully burdened FTE rates", which include salary and benefits plus other costs such as payroll taxes, education and travel. Then count how many hours your staff are spending at work, taking care to include work done out of the office. If your staff start taking less time to do their work, you are getting better value for money.

This applies even if you don't pay people overtime. There is a huge hidden cost to working long hours. Tiredness and stress not only make people miserable, but reduce their contribution to the organization and increase "churn" (the frequency at which staff leave the organization altogether). These negative impacts are at least partially offset by paying people decent overtime - and if you don't pay people overtime directly, you can be sure that your organization is paying the price somehow.

So a first step for a program to improve interaction work in your organization is to find out how much time in each day people are taking to do their work. Cost this time, and aim to reduce the total by a specific amount - say 1 hour per day per person. This effectively gives you a budget for the interaction work improvement program - spend any less than this in total, and you will have delivered ROI.

This approach is likely to ring bells with the executives who hold purse strings. One consultant wrote to me recently to say:

> *I have found talking FB FTE (fully burdened full time equivalents) to managers as a very useful, almost universal unit of measure to talk about process improvement and what it equates to in FTE and therefore $$ terms.*

Of course, this is using the narrowest of measures - a measure that takes no account of the significant improvements you can expect in *effectiveness*, which as discussed above will be the main benefit of the program. However, the aim here is simply to get the program approved by the board. Once you get going, material impacts of all kinds will become evident. It will be much easier to get approval for your second interaction work improvement program than for your first!

In the meantime, what should be the subject of your first interaction work improvement program? In this article I will outline ways in which you can improve human collaboration on an activity that is central to the success of most organizations, namely the management of large software projects. In particular, we will look at the most challenging kind of large software project: the kind in which all is not going according to plan. Readers with experience in this area might wonder whether there is another kind of large software project ☺

Small software applications have bugs – large software applications have defect density

In the last few months I have been approached several times for advice about large software projects in which the fault rate is out of control. This is not as simple as saying that faults are appearing much faster than they can be fixed; at certain stages of a project, one expects this. The real problem is more complex, and relates to the nature of large software projects.

It is important to understand in what ways a large software project is unlike a small one. As I often suggest, one can gain understanding here by comparing software engineering with traditional heavy engineering. For instance, engineers who deal with extremely large and complex systems in fields such as aerospace and the military have always been used to projects that go live with a high number of faults, many of which will remain open throughout the entire lifetime of a system. Fault control in such systems is as much about *managing* faults as about *fixing* them.

However, this is not to say that one can take a relaxed attitude towards fault control. There are well-established metrics available for the "defect density" one can expect at each stage of large software projects of certain types in certain fields. These metrics allow software project managers to use the current number of faults, the current fault rate, and the current fix rate to predict where their project will be in the future - and understand whether or not these predictions are worrying.

The situations about which I was asked were worrying. Predictions showed that build deadlines would not be met, and that by a certain point in the future, the number of faults would escalate beyond the managers' ability to control. What should one do in such circumstances?

Over-staffing

Budget and resource constraints often mean that, even if too many faults are appearing in a software project, it is not always possible just to add staff. Further, even if it *is* possible, it is often not a good idea.

Adding staff works on the seemingly common sense principle that if 10 developers can fix n faults per week, 20 developers can fix something not too far from 2n faults per week. Software project managers often seem to believe that having more people working on the job will inevitably lead to an improved delivery rate.

Unfortunately, however, this is not true. In fact, adding staff may well lead to a reduced delivery rate.

Many readers will be familiar with Fred Brooks' seminal book on software project management, The Mythical Man-Month: Essays on Software Engineering.[2] The concept of "The Mythical Man-Month" refers to Brooks' principle that "Adding manpower to a late software project makes it later". Brooks says this is because adding people to a project has 2 primary knock-on impacts.

First, the newcomers have to be trained in the skills they need, and familiarized with the project itself. This takes up not only their time but also the time of other people - often the people who know most, and who are hence most productive. Induction also takes up other resources of various kinds that can then bottleneck the overall work being done by the project.

Second, the number of communication channels increases dramatically with the number of staff. Brooks' formula is that for n developers, there are n * (n - 1) / 2 communication channels within

[2] http://en.wikipedia.org/wiki/The_Mythical_Man-Month

the group. For example, 50 developers implies 50 * (50 - 1) / 2 = 1225 channels of communication.

This second point is a simplification, of course. However, from my experience, Brooks' formula is useful whether or not it is exactly correct, in that it indicates the scale of the problem. If you observe any large project from the floor you will see that there is a great deal of human interaction taking place, of all different kinds - not just formal meetings, but emails raising issues, questions asked over the partition wall, casual discussions by the coffee machine, and so on.

It is not simple to quantify this interaction and even harder to assess its value. However, if you have any intention of adding staff, doing so is vital – and the effort is worthwhile even if you do not plan to add staff. Careful management can both reduce the number of communication channels and deliver increased value from the ones that are left.

Readers of this column will take little convincing that "quantifying this interaction" depends on first analyzing the business processes involved. For collaborative, innovative, adaptive human work such as software development, this means using the techniques of Human Interaction Management (HIM)[3] – techniques designed to capture "interaction work".

It is no use trying to use traditional business process notational techniques to capture such processes. Attempting to reduce interaction work to a set of "parallel flowcharts" is doomed to failure since people are not machines, and do not work like machines. Rather, you need to adopt techniques that cater to the 5 principles of HIM, that are based on object modeling:

1. **Connection visibility** - to work with people, you need to know who they are, what they can do, and what their responsibilities are as opposed to yours.

   You need Role and User objects, both instances and types, each with its own properties and responsibilities.

2. **Structured messaging** - if people are to manage their interactions with others better, their communications must be structured and goal-directed.

   You need Interaction objects, in which there are multiple asynchronous channels, each for a different purpose.

3. **Support for mental work** - organizations must learn to manage the time and mental effort their staff members invest in researching, comparing, considering, deciding, and generally turning information into knowledge and ideas.

   You need Entity objects that can be created, versioned and shared in a structured way.

4. **Supportive rather than prescriptive activity management** - humans may not sequence their activities in the manner of a software program, but there is always structure to human work, which must be understood and institutionalized so that it can be managed and improved.

   You need State objects that can both enable and validate Activity objects, along with the Roles that contain them.

[3] http://www.human-interaction-management.info

5. **Processes change processes** - human activities are concerned often with solving problems, or making something happen. Such activities routinely start in the same fashion - by establishing a way of proceeding. Before you can design your new widget, or develop your marketing plan, you need to work out how you are going to do so - which methodology to use, which tools are required, which people should be consulted, and so on. In other words, process definition is an intrinsic part of the process itself. Further, this is not a one-time thing - it happens continually throughout the life of the process.

   You must be able to manipulate not only objects but also user interfaces and integration mechanisms via the process that contains them.

These HIM principles may seem like common sense, but many people are still trying to capture collaborative human work using notations like UML and BPMN, notations that stem from software design techniques – techniques intended for processes carried out by computers rather than by humans.

Only by using process design techniques, and an accompanying process notation, intended to capture human behavior is it possible to describe “interaction work” in a way that *reflects what is really going on*. The process design techniques and notation of HIM let you both measure such work and manage it better.

Measuring it, in particular, enables you to find out what bang you will get for your buck – for example, what improvement in delivery rate you can expect from adding extra staff. In fact, only by such means can you be sure you will actually *get* an improvement from adding extra staff, rather than the deterioration expected by Brooks.

The lessons of war

Capturing interaction work is the necessary precursor not only to scaling it up via extra staff, but also to improving it without adding staff at all. Clearly a preferable solution to adding staff is to increase the productivity of existing staff to the necessary level – to improve their efficiency as a team to the point where additional team members are not needed.

To achieve this, one can take a leaf out of the military's book, and apply 3 approaches in combination: operational, tactical and strategic. If you prefer, think of these approaches as short-, medium- and long-term. Taken together, they provide a means for even large software projects to become "hyper-productive" - a phenomenon well recognized in certain small teams, but rarely achieved on the biggest projects. Hyper-productivity often makes it unnecessary to add new staff members at all.

Here are details of how I deal with each of the 3 approaches in my own consultancy work.

Operational

One needs to do something in the short-term, not only to stop the situation getting further out of hand, but also to raise morale. Developers do not like to work on failing projects, and once your developers start leaving (or just giving up), you are really in trouble. Even if management can somehow conceal the true situation from developers - which is never easy - there is the morale of managers to consider as well.

So in the short-term, it is advisable to set up an emergency room - a lab environment in which some of the best developers can work free from disturbance to knock some of the worst faults on

the head as soon as possible. It is often surprising how much benefit accrues simply by reducing the amount of interruption that people are subject to. People should not be allowed to enter the emergency room without permission, and if the lab fault fixers need to talk to someone from outside, that person should be brought into the room specially.

A manager should be appointed for the lab, a person who is able to create a team atmosphere as well as to maintain a sense of urgency. A key part of the manager's role is to ensure that work is allocated in the most efficient manner possible from hour to hour - sometimes from minute to minute. Everyone should be fully loaded, the whole time, and distractions such as checking email discouraged. The aim is to maintain a sense of flow.

Lab work of this kind is exhausting, so the participants should be rewarded with overtime payments and pampered with free, luxury food and drink. However, the work is also rewarding due to the awareness of visible progress and the team spirit that should be engendered. Further, such a lab is a powerful means of knowledge sharing - after working in the lab, a developer will not only have more system knowledge but may well have acquired new ideas and techniques that will be useful going forward.

Nevertheless, this approach is not intended to be a normal part of the project. If it is necessary to carry it on for more than a month, the participants should change regularly - weekly, for example - to avoid burnout. Both developers and managers can learn lessons from such a lab, but the emphasis should be on carrying these lessons into regular daily practice rather than on sustaining the lab itself.

Tactical

In the medium-term, there are various means to reduce pressure on a project, with which most project managers are familiar:

- Deadlines and the content of deliverables can be renegotiated with the client;
- Workarounds can be developed for areas of fault;
- End-user expectations can be managed, such as when to expect the introduction of particular aspects of system functionality;
- Training can be planned so that end-users acquire system knowledge gradually, in order to allow time for the corresponding functionality to be released;
- Easy-to-use techniques can be developed for end-user fault reporting and control;
- And so on.

When proposing such measures to the client(s), one should take care to present a positive picture. Don't emphasize the trouble the project is in! Rather, explain how your advanced project management techniques have identified potential problem areas well in advance, and prepared a range of compensating solutions. The client(s) should come away from such a meeting feeling glad that *you* are in charge, not someone who might have let the problems go unnoticed until it was too late.

Strategic

However, it is only fair to admit that, if the project has got to this state, someone has slipped up - even if it was your predecessors. Clearly the scope of the work was misjudged, inadequate development techniques have been used, project management has not been optimal, and so on. So it is necessary to take remedial action to get the project properly back on track in the long-term.

There is a wealth of advice available here, particularly with regard to "Agile" techniques. I recommend the book Organizational Patterns of Agile Software Development by Coplien and Harrison[4]. If you can't face a long read (and it is a big book), here are summaries by the authors of "the ten patterns that research has found most strongly correlate to business success".[5][6] In each case, the pattern name is followed by a typical situation in which it is useful, description of the problem, and an outline of the pattern itself. The text below is only slightly altered (for clarity) from the original author's own words:

1. UNITY OF PURPOSE

   ...the team is beginning to come together. Team members may come from different backgrounds and may bring many different experiences.

   Many projects have rocky beginnings as people struggle to work together.

   The leader of the project must instill a common vision and purpose in all the members of the team.

2. ENGAGE CUSTOMERS

   ...an organization is in place, and its Quality Assurance function has been generally shaped and chartered. The Quality Assurance (QA) function needs input to drive its work. Many people in the enterprise are concerned about quality.

   It's important that the development organization ensures and maintains customer satisfaction by encouraging communication between customers and key development organization roles.

   Closely couple the Customer role to the Developer and Architect, not just to QA or marketing. In short, developers and architects must talk freely and often with customers. When possible, engage customers in their environment rather than brining them into your environment.

3. DOMAIN EXPERTISE IN ROLES

   ...you know the key atomic process roles (FormFollowsFunction) including a characterization of the Developer role.

   Matching staff with roles is one of the hardest challenges of a growing and dynamic organization.

   Hire domain experts with proven track records, and staff the project around the expertise embodied in the roles.

4. ARCHITECT CONTROLS PRODUCT

[4] http://www.amazon.com/gp/product/0131467409

[5] http://users.rcn.com/jcoplien/Patterns/OrganizationalPatterns.pdf

[6] http://users.rcn.com/jcoplien/Patterns/Top10OrgPatterns.html

...an organization of developers exists and needs strategic technical direction.

Even though many individuals design a product, a project must strive to give the product elegance and cohesiveness.

Create an Architect role as an embodiment of the architectural principles that define an architectural style for the project, and of the broad domain expertise that legitimizes such a style.

5. DISTRIBUTE WORK EVENLY

...an organization is working to organize in a way that makes the environment as enjoyable as possible and which makes the most effective use of human resources.

It is easy to depend on just a few people to carry most of the organization's burdens.

Try to keep the communication intensity ratio to two or less.

6. FUNCTION OWNER AND COMPONENT OWNER

...the functions in your project tend to cut across the components of your project. You recognize the importance of organizing according to the architecture of the project (ConwaysLaw), but things aren't simple.

If you organize teams by components, functions suffer, and vice versa.

Make sure every function has an owner and that every component has an owner.

7. MERCENARY ANALYST

... you are assembling the roles for the organization. The organization exists in a context where external reviewers, customers, and internal developers expect to use project documentation to understand the system architecture and its internal workings. (User documentation is considered separately). Supporting a design notation, and the related project documentation, is too tedious a job for people directly contributing to product artifacts.

Technical documentation is the dirty work every project must do.

Hire a technical writer, proficient in the necessary domains, but without a stake in the design itself.

8. ARCHITECT ALSO IMPLEMENTS

...an organization is being built to serve an identified market (OrganizationFollowsMarket) or markets. Going forward, the project needs the necessary architectural breadth to cover its markets and to ensure smooth evolution, but it can't be blindsided by pragmatic engineering and implementation concerns. Furthermore, the project needs to carry through a singular architectural vision from conception to implementation if it is to have conceptual integrity.

A software project must broaden the scope of leadership without sacrificing depth and attention to pragmatics.

Beyond advising and communicating with Developers, Architects should also

participate in implementation.

9. FIREWALLS

   ...an organization of developers has formed in a corporate or social context where peers, funders, customers, and other “outsiders” scrutinize them. Outsiders who feel a need to offer input and criticism often distract project implementers.

   It's important to placate stakeholders who feel a need to "help" by having access to low levels of the project, without distracting developers and others who are moving towards project completion.

   Create a ManagerRole, who shields other development personnel from interaction with external roles.

10. DEVELOPER CONTROLS PROCESS

    ...an organization has come together to build software for a new market in an immature domain, or in a domain that is unfamiliar to the development team. An InformalLaborPlan will mark progress. The necessary roles have been defined and initially staffed.

    A development culture, like any culture, can benefit from recognizing a focal point of project direction and communication.

    Make the Developer the focal point of process information.

One should take the time to understand agile project management. Unfortunately, though, agile techniques tend to work much better with small projects than with large ones. In my experience, agile techniques typically have most success in projects with less than 50 staff, and projects with over 100 staff need something else.

When agility is not enough

So here are some strategic techniques that I have found useful either to complement or replace agile project management for getting large projects back on track.

1. **Food**

   Provide ample, high-quality food free to all project staff daily. A small investment in pampering of (say) £10 per person per day pays huge dividends in morale and commitment.

   Don’t take this lightly – and don't be mean! The aim is to make people feel valued, by providing them with luxury food that they would not necessarily buy for themselves. A few expensive berries, pastries and smoothies will generally deliver a lot more value than overtime payments of an equivalent monetary amount.

2. **Team ownership of code**

   You can't act responsibly if you're not responsible for anything. Coders must own their bugs and work as a team to reduce them. Develop continuous integration practices and put up highly visible flags in each team area every morning: green, amber, red to indicate code quality for that team based on automated testing of last night's build. Institute a daily trawl through all open faults by each Work Package Manager with a view to escalating or

delegating faults or actions as appropriate. Introduce a weekly quota of fault fixes based on size of team.

Careful synchronization with the emergency room is clearly necessary here while it is running.

3. **Knowledge management**

   Build system knowledge. Black box: give developers a user's perspective - terminology, operation, concerns, and so on. White box: make sure they understand what all the different parts of the system do. Do both these things as early and as widely as possible to get maximum benefit.

   It doesn't take as much time, or encroach as much on other work, as you might think - lunchtime seminars could be the way to go, for instance.

4. **Testing**

   Give developers a means to test the system as a whole (not just to run unit tests). In particular, they need a framework with which they can construct and run end-to-end tests themselves, for example to reproduce informal user fault reports or check particular behavior relevant to a fault.

   Development of, or just input into, such a framework may be one very useful activity of the emergency room.

5. **Pair programming**

   This is an established agile technique that scales very well to large projects. Pairs in which the members are of roughly equal ability but mixed knowledge work best. Change the pairs often. Check to ensure that both people are contributing rather than having one driver and one passenger. Convince people to give it a go - nearly everyone finds they like it. In some circumstances, trios or quartets work even better than pairs.

   In general, pair programming is more than twice as productive in terms of elapsed time - i.e., it is worth doing.

6. **Systematic fault fixing**

   Research into fault fixing indicates that most software developers spend over 50% of their time fault fixing. Yet few ever learn how to do it properly - probably because there is no standard methodology or even practices.

   I have compiled a systematic fault diagnosis procedure based on the research papers and books that *are* available. For faults that take less than 2 hours to diagnose, the procedure does not always add value – but for faults that take more than 2 hours to diagnose, it makes a dramatic difference. The systematic procedure has enabled faults to be solved in hours on which experienced developers previously spent weeks and got nowhere.

7. **Collaborate better**

The BBC reported on 13 August 2007[7] that "Workers are 'stressed out' by e-mails". Regular readers of this column will be aware of my view that email in its raw form is a pernicious problem across the board in the modern workplace, preventing people from making best use of their time rather than aiding them to do their work, and that the solution lies in new tools such as HumanEdj.[8] Gartner describes these tools as "high-performance office systems", and says that the market for them is lucrative if not yet mainstream.[9]

Adoption of a high-performance office system such as HumanEdj generally takes place in 3 stages. In the first stage, people use the tool simply to reduce the time taken to deal with email and other communications, and to improve their effectiveness. In the second stage, people start to understand how it lets them structure their collaboration with colleagues into processes – what HumanEdj calls "Stories". In the third and final stage, people start to make positive, systematic input into the design of these Stories – they structure the Stories in which they take part according to well-known HIM patterns, in order to improve their efficiency and integrate the Stories better with organizational goals, practices, and standards.

TAKE AWAY: Quantifying hyper-productivity

A consultancy client recently evaluated the effects of applying the techniques described above to fault fixing. Their conclusion was that when using my techniques for fault fixing, faults were fixed in approximately half the normal length of time, or in other words, that developers were twice as productive as usual.

Most literature on fault fixing estimates that the average developer spends over 50% of their time fault fixing. So if you are looking to improve the efficiency of your development staff, it must be worth considering techniques that double the productivity of this time.

Nevertheless, it is often tempting to sweep problems under the rug when they seem too huge to solve. Large software projects that have gone out of control can seem like a many-headed hydra - as soon as you solve one problem, ten more problems appear. However, there is always a way through the labyrinth. The key is to adopt a military approach, and divide remedial measures into operational, tactical and strategic.

Further, take care to explain what you are doing to others and get them on board. Developers as well as managers should understand what is going on and what you are doing about it. A typical but very unhelpful approach to managing crisis situations is to take everything behind closed doors. Resist this temptation! People are surprisingly good-natured, even in times of stress, if they are made to feel a valued part of something - as opposed to being excluded from important discussions.

There is a route to hyper-productivity, and it is not hard. You just have to take it in stages.

---

[7] http://news.bbc.co.uk/2/hi/health/6944747.stm

[8] http://humanedj.com

[9] http://www.information-age.com/reports/276941/riding-the-fourth-wave.thtml

Author

Keith Harrison-Broninski is a consultant, writer, researcher, and software developer working at the forefront of the IT and business worlds. He is author of the landmark book **"Human Interactions: The Heart And Soul Of Business Process Management"** (Meghan-Kiffer Press, 2005)[10], described by a BPTrends review as *"the overarching framework for 21st century business technology,"* and by the BPM Group as *"a must read for Process Professionals and Systems Analysts alike."* Keith is also a contributing "thought leader" to the BPM Group book "In Search Of BPM Excellence" (Meghan-Kiffer Press, 2005)[11].

Along with his research and consulting work, Keith is the CTO of Role Modellers Ltd, whose company mission is to develop understanding and support of collaborative human work processes across industry, a field that Keith has pioneered with his work on Human Interaction Management.[12] Role Modellers' free software, HumanEdj, is the reference implementation of a Human Interaction Management System.

Find out more about Keith and his work at http://keith.harrison-broninski.info.

[10] http://www.mkpress.com/hi

[11] http://www.mkpress.com/bpmg.html

[12] http://human-interaction-management.info