METHODS & TOOLS

Global knowledge source for software development professionals **ISSN 1023-4918**

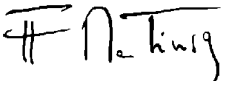
Fall 2002 (Volume 10 - number 3)

Process improvement and fear fighting

Software development process management and improvement are the main themes of this issue. It is true that the major problem with a new approach/technology is not really how to handle it, but how to make the transition to the new situation. Adoption is translated into change and the word you will the most likely see associated with change is fear (or opportunity if it appears on a vendor's slide...)

Change is an activity not easy to execute smoothly, mainly when it deals with leaving a reliable position. I am however not convinced that it is only the action of changing that is the main factor of resistance to software process initiatives. Under the generic word of change, there is also the menace that practices proposed by software process will often impose more control on the work of software developers. Programming is a very personal task. If you propose to learn a new language, like Java in the recent years, you will often see a limited resistance to change. On the contrary, many programmers will be enthusiasts to learn a "new" technology that increases their marketability. If you propose to these same programmers to implement a software inspection program, like the one described in the previous issue of Methods & Tools, I am quite sure that you will see less enthusiasm.

Thus, the problem is not really involved by the action of changing, but by what is changing. More specifically, developers will prefer freedom to formalisation and control. All changes that are perceived by developers as a mean for their company to increase the control on their own work will meet a high level of resistance. In many companies, adopting a process is already perceived as a menace to the freedom of development. The remarks on "unnecessary burdens" can be answered if there is enough automation to support the new procedures, like software configuration management tools for instance, but addressing the issue of a lack of trust from the employees towards their organisation is another topic. Change processes have the best chances to succeed when they are owned by employees. The first step toward process improvement is therefore to implement a culture of respect, because the desire to improve (and not just "play the process game") is influenced by this. How about an employee respect improvement initiative?



Inside

Software Development Process Misconceptions

Nathalie Gaertner, ngaertner@rational.com,
Rational Software, www.rational.com

Nowadays, many software developments are more about "hacking" than really about engineering. But isn't the real name of the software development discipline "Software engineering"? To be an effective engineering discipline some best practices must be applied, supporting guidelines must be provided and rules must be followed. A software engineering process can give the underlying support for it. So why are software engineering processes no so widespread and really used? One answer to this question is the wrong ideas software people have about such processes. This article shows some of these misconceptions to help people understand the real benefits of applying and living the software development with a well-defined and effective process.

What is a software development process?

A process should define the series of actions or operations that lead to the construction of something assigning tasks to people. A software engineering process should thus guide people in the software team in their daily work to specify, construct, deploy and maintain software.

Figure 1: Software engineering process - WHO has to do WHAT, HOW and WHEN

To provide this guidance, a process can take the form of a knowledge database (paper or electronic form) that will contain various information and guidelines to explain and detail what people have to do and how they can do it to achieve a goal. It will not only contain information about coding, but also about project management, managing requirements, testing, etc.

If we follow a process while developing software:

- we make software development activities predictable: all team members know what to do today and the following days to achieve a well defined goal; project manager know what the members have done, are doing and will do.
- we make software development activities repeatable and measurable. It is important to control and assess the development flow. We need means to measure where the project is, what has been done and how effective it has been done.
- we provide a common vision for the whole software team. Process help team members speak the same development language and understand common goals. Nowadays, software projects are not done by only one person. It is a real team sport; and for every sport we need some rules to follow to have people playing together. This will help people with different competencies to work together

and communicate better as well as help them to understand what the other team members are doing.

- we have a roadmap that will enable all the software people together develop the software that the customer will really use.

But to get all these benefits, a process should be first effectively used. To ensure that the software engineering process will be really lived, it should have the following qualities/characteristics:

- It should be understandable; the information provided by the process is understandable, clear and non ambiguous so that people can see the meaning and purpose behind it.
- It should be accessible; the information provided by the process should be available 24 hours a day, 7 days a week. If we need guidance, we will get it quickly in a form that we can use. No need to spend too much time to first find out where to look at.
- It should be easy to use; the information provided by the process is well defined, well formatted, with the right level of details and also defines how it can be applied.
- It should support customization; the process can be tuned according to the specificities of the project (technology, complexity, and team members). The tuning can be either modification of existing information or extensions to add new information.

The Rational Unified Process™ (RUP) is such a software engineering process. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. It is based upon a comprehensive set of industry best practices that provide proven methods, guidelines, templates and examples for developing software. RUP is a web-enabled, searchable knowledge base, which is integrated with Rational tools and provides project members with an online mentor covering the full lifecycle. It is an iterative and incremental process that mitigates risk, is use case driven and architecture centric.
For more details about the Rational Unified Process see the Rational Unified Process itself or refer to the article "Understanding the Unified Process (UP)" in the Spring 2002 issue of Methods and Tools.

Misconceptions

If we are looking at how many problems and money is still lost today in software development and maintenance, if we know all the benefits we could get from a software engineering process, it is really difficult to understand why so many companies still haven't adopted a development process.

One reason for this current situation often lies in the misconceptions people have about processes. The following sections will list some of these wrong ideas to show the real benefits of engineering processes.

Before I was only coding and it was also working

When people hear the name of *process* they often directly say or just thing "Why changing? Why making my work more complex? Before, without any process it was doing my job and I was doing it well because I was also able to write software that runs".

People fear that a process will imply just an overhead with no real benefits. It is perceived as being just a waste of time.

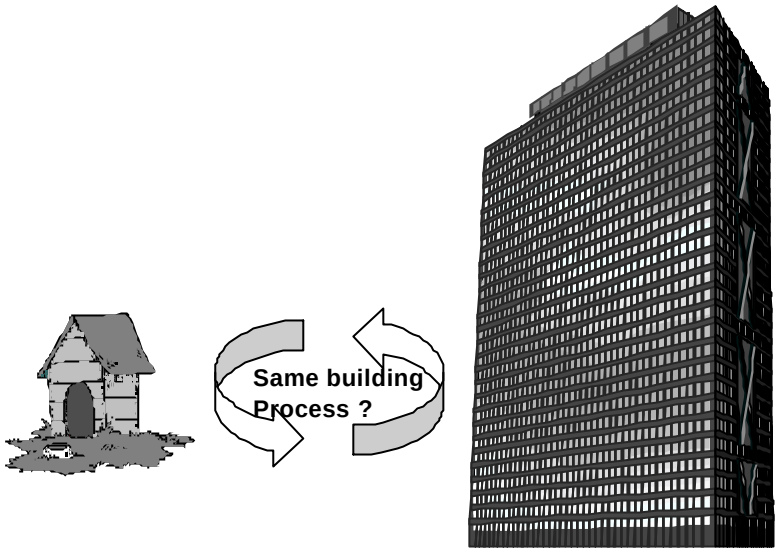


Figure 2: Complexity change implies changes in process

Yes probably people were able to write software that runs without any formal defined process. Probably they had some kind of process, but it was just in their own heads. Now the problem is that software is becoming more and more complex. Everybody sees and feels that. Today's software is not comparable to software done years ago. We adapted the programming language to give us better support, we improved tools like IDEs (Integrated Development Environments) to extend the development and debugging environment; we should then also improve the development process. Today we develop software in teams, we have people having different roles, we need to improve communication, and we need a common vision to achieve the expected goal.

Would it make sense that a house builder says: "yesterday I was building a dog house without any plan, blueprint or supporting process, so I will be able now to build a skyscraper without any plan, documents or supporting tools".

The real problem is that people are resistant to change. They will react against things because they fear the learning curve linked with the introduction of new things (in software engineering and also everywhere else). But if we do not learn new thing and try to improve ourselves and the way we work, we would not have nice cars, mobile phones, great software to support our business, etc. Therefore, yes, there is a learning curve, but there is also a return of investment associated with it.

Often the resolution of this wrong idea is more an issue about evangelization and explaining the motivation factors.

A process is not "sexy"

How often did I hear our corporate process or this commercial process is too complex? Probably a thousand times.

How often did I hear people complaining about their process that was too light and did not support them correctly in a real software development project because they were not so experienced people? Many times too.

All these complaints came from different people, in different companies using different processes. However, they all rely upon a common compromise: should the process be complete and thus be felt as complex or should it be lightweight but thus not always provides all the support expected?

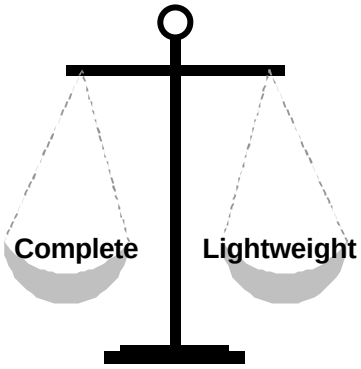


Figure 3: Completeness or Simplicity?

Now what does this mean? It shows us that new "thing" can be perceived as complex just because they are complete, whereas incomplete "things" are perceived easy at the beginning, but quickly they don't provide the expected support. It also means that a "good" process must be scaled to the size of the team and the size of the project; one method cannot fit all projects and a software engineering process must thus be tailored to each particular project. Finally the compromise between complete and lightweight implies that the process should contain all the information people need for a project (be complete) but should provide it in an understandable, readable and accessible form (show only the relevant parts in a simple and lightweight form).

Thus a software development process should be complete and also "sexy" enough to attract people and have them looking at it and living it. It should look simple and be well structured given the needed support or information on demand.

Iteration lead to never ending changes

Iterations rely on a concept that all of us know: we cannot cope with everything at ones; so we start doing a sub-part, then we add another part, and so on until we get to the expected result. For example, a programmer writes not the whole code for a whole project and then tests the whole project; he starts with a component or subpart of a component and then progressively adds more and more. But do we want chaos: everybody in the software development team does what he likes when he wants and we will see if we can integrate the work of all at some point in time? Or do we want to manage the sequence of activities of everyone to have an effective teamwork and to be able to mange the risks in our project?

An iteration is defined as a distinct sequence of activities with a base-lined plan and evaluation criteria defining the goal to achieve (i.e. an internal or external release for software development iterations). It contains coding activities but also requirements management, planning, designing and testing activities.

When people think about iterations, they sometimes see the picture of an uncontrolled spiral approach that leads to a never-ending iteration loop. So basically, we know when we start to iterate, but never when we will finish; and this incremental and iterative process is a mean to accept changes requested from stakeholders (users, designers, programmers, etc.) at any time in the development cycle.

Figure 4: Never ending iteration - a wrong idea about iterative development

Having people fighting about an iterative and incremental process is legitimate if this picture would be the reality. But real iterative development is different. It means managing what people are doing at the beginning, and what they should do later. It implies doing planning and controlling activities. We build an overall software development plan and also detailed iteration plans. So we know exactly when we start and when we finish and we know what are the goals for all iterations. We always assess our current position / status to keep on track.

Iterations are also not an excuse to accept all modifications and new ideas from the customer or marketing department. We accept modifications as long as they are managed (check the validity and priority, assess the impact of the change, and then accept, reject or postpone it). This enables us to take into account the dynamic aspects of requirements and to re-plan / re-design items along the way if needed.

Just theory invented by methodologists

A process created by a methodologist. Can this be useful? How could such a process be practical and effective if it is the ideas of a single man sitting all the day in a closed office having no experience of real life problems and no contact with people really doing the software? Statements like "This process is just nice theory; in our real live software development we cannot apply the principle described in it" are just an excuse for some people to resist to changes.

Figure 5: The great ideas of just one man?

A "real" process is based upon best practice. It comes from practices and knowledge that have emerged from several development efforts and that have been extracted, recorded and documented to make them useful and reusable. A process is therefore not only nice theory!

It is just about doing more paper work

Does a process imply producing tons of paper? No, not at all.

If we develop software internally for some internal users with no sub-contractors and with a small software development team, we will surely have a small set of work products (plan, requirements, etc.) and almost all will be kept small and in electronic form. No paper could be generated at all.

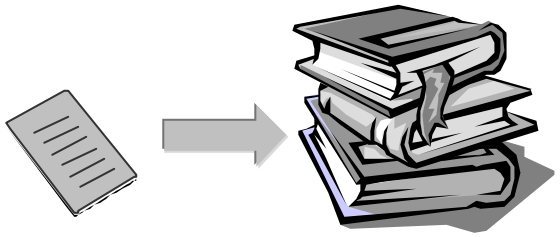


Figure 6: Paper overhead misconception

So following a software engineering process does not mean producing paper. Only formality requirements would drive more paper work. But this formality is only driven by the domains we are working in (i.e. regulated environments) or by the kind of projects (having third parties participating in the development).

A good software engineering process will provide adequate guidance and support for formal projects as well as non-formal ones.

A creativity killer

Another misconception about process is that the process is prescriptive and non-flexible. Some people really think they will loose their freedom and that the process will banish all initiatives and creativity.

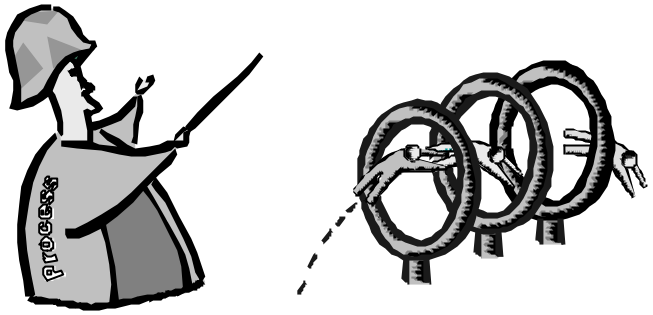


Figure 7: Is a process prescriptive?

The fact is that a process enables them to be even more creative because the process acts as a catalyst for the creativity of all. For example, the ideas of analysts and designers are combined to create an even better solution. What is also true is that this catalyst will only work if people in the software team are willing to communicate and play their role in their team.

Prescriptive is not a characteristic of a process. Yes, a process will define a roadmap or a kind of frame in which people can move. A software development process is neither a cookbook to be applied without thinking, nor just a rule set. People in the software team have to move within the frame the process defines but they have the freedom to decide which way to go.

Flexibility is also provided but depends on the kind of project and environment the project is in. Indeed different projects have different needs: a new development project is different from a maintenance project; the complexity (technology and management) can be different, the people in the software team can be more or less experienced, the kind of development (contract with third parties for example) can introduce differences, etc.

A practical process is therefore more like a framework that can be customized for each project. The Rational Unified Process (RUP), for example, is a framework and gives several guidelines and roadmaps to take into account different kinds of project and project characteristics. It also has a discipline called environment that guides users in the framework customization and it is shipped with a RUP Builder to build customized RUP web sites (even a RUP for small projects can be generated).

A process does not hinder. It is agile and improves the team member's work and creativity.

Ok… but not for my project

So we agree about the benefits of a process; we even agree that all the potential drawbacks are just misconceptions. Then comes the conclusion: people argue that there is too much time pressure in their project, so there is no time to implement a process.

Time pressure is something we will have anyway. We can allocate time to have programmers learning new technologies or languages, so we can also have time allocated to promote a common vision of how to develop software in teams.

Figure 8: Not for this project, because of time pressure

Other people will argue that their project is that big and complex. It would be too much to introduce a software engineering process to this critical project or they say that their project is too small to get any benefits. These kinds of arguments are often more linked back to reactions of people who are reacting against changes.

Processes can be adapted to each project and thus how much we want to do from a process can be decided. We can start small with a first project and progressively, on following projects, do more as people start to know the process. Advantages like predictability, risk management or quality from the beginning can be gained also if we start to introduce a small subset of a process.

Conclusion

A software engineering process contains all we need to quickly and safely deliver software. It increases predictability, reduces risks, is based upon best practices and promotes a common vision and culture in the software team.

If we look today at software development projects, we recognize that we still miss targets (time, money) and that we could greatly improve quality. A software engineering process is a means to more effectively develop quality software in a predictable way.

Unfortunately today there are many misconceptions about software engineering processes, which result in a low acceptance rate of them. A way to improve this situation is to address all wrong ideas people have and to explain what are the real purposes and benefits.

Software Quality Improvement for Global Organizations After the Merger

Linda Newsom-Ray, Sr. Consultant,
MKS Inc., www.mks.com

Change as the only constant - Mergers & acquisitions

With increasing market pressures on growing the business to attract market recognition and retain market share, expansion by merger has become the most common method of fighting off the competition. If you can't beat them, buy them.

Against this backdrop of expansion and the resulting organizational change, there is the unalterable fact that global organizations are 100% dependent on their IT systems in order to stay in business.

Given these two factors, it follows that such organizations, and particularly their IT departments, must be fully equipped and ready for the next sweeping change. This is not easy when, commonly, financial organizations are littered with elderly 'legacy' systems that are, in fact, still critical to the business.

Mergers inevitably create duplication of some underpinning customer-facing technology. It can take years to migrate such systems or to construct a viable solution for the new, enlarged organization. Where there are especially entrenched positions and complex systems, it can take years just to obtain agreement to the decisions among the stakeholders.

Quality and service – the sole differentiators

With less and less tolerance for any downtime or errors, customer loyalty may only be as strong as the next click away from your website to a better one. So what are global organizations doing to improve customer retention?

It is certainly important to recognize that the quality of service does make a difference. To this end, many financial organizations have implemented positive policies regarding their customer-facing processes and staff. With 24/7 business operations, however, it is the systems that are directly customer facing: thus, the responsibility for excellent service is no longer the sole province of the branch network or customer service staff. So it follows that the systems development, management and operational staff all need to be wholly aware of what constitutes providing quality service.

Increasingly, global organizations are turning to major quality initiatives such as Six Sigma and TQM to provide some answers. These are designed to sweep away bad old practices and to bring in fresh new ways of ensuring that the customer's interests always come first. Ever more financial organizations are focusing their IT improvement initiatives, by utilizing well-respected benchmarking models, such as the Software CMM and, more recently, CMM Integrated. There are certainly some major success stories with all of these models, including claims of billions of dollars saved, enormous returns on investment and huge improvements in product quality.

Some of the obvious success stories are:

- Six Sigma: Motorola[i] and General Electric[ii],

- TQM/Malcolm Baldridge Award and Software CMM: Boeing[iii]; and
- Software CMM: Raytheon[iv]

What can be achieved by implementing Quality initiatives?

Successful quality initiatives:

- Can deliver huge improvements in the quality of the software; greatly reduce costs of software maintenance; help the IT department to deliver software better, faster and cheaper.
- Enable management to have greater visibility into the software processes, so that there are fewer surprises e.g. less likelihood of late delivery, cost-overruns, unstable systems/releases.
- Offer better options for making informed choices when business priorities change.

The fact is, however, that although large-scale quality initiatives have many good qualities, longevity is not normally one of them.

It is quite difficult to find companies that will publicly admit to it, but it is generally accepted within the software process improvements industry that at least 70% of large-scale quality initiatives fail, normally within two years, having delivered an average of 5% of the anticipated ROI. The Juran Institute estimates that 80% of all companies that tackled TQM in the 1980s failed.[v] More recently (1999) a DACS report demonstrates similar findings.[vi]

Getting started with an incremental quality initiative
Simple steps to help make it work

At this point, it should be noted that managing development projects is quite difficult. Managing software process improvement projects is even more difficult, even under normal organizational conditions, because of the nebulous concepts involved and a general lack of ability to either estimate accurately or to identify scope creep.

So, trying to get any kind of quality initiative to 'stick' when the organization is undergoing massive change, with all the resulting uncertainty, is next to impossible. The only way of tackling it is through the KISS principle (Keep It Simple, Stupid).

It is perfectly possible to use the principles of the quality models to provide an objective basis for improvement without investing hugely in training everyone at once. Models such as the Software CMM, or the newer CMM Integrated are excellent as an objective yardstick, for measuring organizational process maturity. Using these models can be very helpful in terms of prioritizing where improvements should start.

It is important, however, to make sure that goals are set for the relatively short-term and have SMART[vii] objectives that can demonstrably be achieved.

The first thing to do when faced with massive organizational change is to define the parameters of what is within the scope of the project and what is outside – in other words, the who, what and where of the people, processes and systems. The quicker this can be captured the quicker the duplication and waste can be eradicated. In any merger situation there will be redundant systems, redundant processes – that is, ones that are not facilitating a quality result – and, less pleasant to deal with, redundant jobs.

Next: define what is hurting. In the early days of a merger this is tricky as practically everything will feel as though it's hurting. Normally, it is a good idea to start by asking if there is a tendency for projects to overrun on dates and costs and under-perform on quality and/or delivered functionality. Looking critically at the cost of maintenance for each system is also a good indicator of quality (e.g. well-structured systems) and whether project and development practices are good or poor.

Once the causes of the pain are found, proper steps can be planned to define the desired values and how they will be achieved. Having an established quality improvement initiative demonstrates that there is a plan for the future and it will have the effect of providing some stability in what can be uncertain and, for some people, dispiriting times.

Early days

Living through a merger is like being caught up in a moving time-zone picture: there is a patch of sunshine – although one has to keep running to stay in it – and the rest of the world is in darkness. The future is unknown and the past is gone. During this period of uncertainty, people will gather around the coffee machine or water cooler and chatter, incessantly, about the situation. Be aware that 'the grapevine' is not only a source of misinformation and the subsequent generation of fear, but it will travel much faster than any formal organizational information and communication flows. It is therefore extremely important to put in place a policy of continuous communication of as much information as can be given at every stage. Even a statement of 'we do not yet have the full picture' will at least help to reassure staff that they are not being plotted against by bosses who know what the future will hold, but are not telling them.

Setting up a specific web site, dealing with the communication of plans and allowing for Q&A, can be very cost-effective and can be changed quickly as the plans develop.

In the early stages, it will be apparent to all staff that there is a plethora of systems providing more or less the same functionality: so some systems will have to be scrapped. Uncertainties hanging over the technical future add to the loss of morale because systems that are now declared surplus were once built and proudly maintained by staff whose historical contribution to the organization is being visibly eradicated.

Productivity can be close to zero, so it is important to put something positive in place quickly. This will normally mean putting together some sort of blueprint for the future and making sure that the resource planning allows for acquisition of new skills. For this to happen, a clear technical strategy needs to be in place based upon knowledge of the perceived needs of the enlarged organization. This will take time and require input from a wide cross-functional group. Anything that builds bridges across 'the divide' can only be useful at this stage to avoid some of the negative feelings that can occur.

Decide on what 'success' means for the organization – and measure it

To give a relatively quick and easy route out of the maelstrom created by mergers, it is recommended to hold one or more Goal / Question / Metric workshop(s), or some similar mechanism that will enable the group to determine, for example:

How the organization should be prepared for the future

- What business needs must be met

- What measures must be in place to improve:
  - time to market
  - code quality
  - system flexibility, etc.
- How to put together rapid response teams that are willing to work cross-functionally with the rest of the business
- How to reduce maintenance costs

It should be self-evident that the less effort spent on finding and removing bugs, the more money and resources will be available to position the organization for the future. In the new scheme of things, the 'techies' will need skill-sets that include analysis, architecture design and communication skills for Joint or Rapid Application Development, as well as technical ability and domain knowledge. Putting together a simple skills matrix, i.e. the required skills versus who has them, can be a helpful way of spotting the gaps.

Be very careful to avoid implementing a full-scale metrics program at this stage: they are notoriously difficult to make successful, even in with 'normal' levels of organizational change. Focus only on collecting metrics that are well-understood; that measure processes, not people; and when you know exactly what you intend to use them for. Collecting metrics for the sake of it has no benefits whatever and will create resentment.

What you need to make the right things happen

Looking to the future also involves analyzing the present and learning from the past. What is needed here is:

- The right vision and guidance
- Clearly defined business objectives and priorities
- Clearly defined technical strategy
- Clearly defined quality strategies
- The will and the resources to make it succeed
- The whole-hearted participation of the people who plan to stay in the company

Design action plans that prioritize and focus on results – and track, track, track. The Deming cycle of 'Plan, Do, Check, Act' is a useful aide-memoire and has now been incorporated into another well-respected benchmarking standard, ISO 9001: 2000.[viii] Ensure that as much as possible of this planning work is co-operative, using team-based activities. Using workshops across the enlarged organization to define and build towards common goals will reduce the tendency for attitudes of 'them and us'.

People, process and technology

'The underlying premise of software process management is that the quality of a software product is largely determined by the quality of the process used to develop and maintain it.'[ix]

It is well recognized that neither technology nor people alone will make high quality software systems. Well-defined and consistently applied supporting processes must also be present.

This interrelationship is normally represented thus:

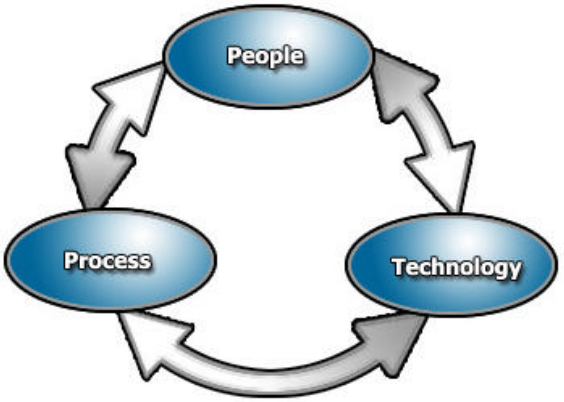


Figure. The people, process and technology triangle where the process and the underpinning technology support the people developing the software.

Each of the three factors is discussed below with particular emphasis on implementing quality initiatives against a background of constant organizational change.

People

Dealing with uncertainty

If all that is offered in the new organization is uncertainty, people will leave - usually the best people leave first. With any merger situation, or indeed any major organizational upheaval, people fear downsizing or losing out on job-leveling (i.e., ironing out the inconsistencies in rewards for similar roles). Suspicion and resentment can be rife and territorial battles flare up with little provocation. The 'not invented here' attitude can be a killer to any initiative. This is the base level of Maslow's Hierarchy of Needs[x] where survival is key.

It is essential to get people to believe that they have a future in order to progress from this stage. People cannot think at the 'self-actualization' level if they live in fear of not being able to support their family.

'Take hope from the heart of man, and you make him a beast of prey.' - Ouida

Dealing with resistance

It is likely that there will be people whose skills are no longer required and who are unable or unwilling to move on once their usefulness has been exhausted. Such people can be destructive to any improvement or change management initiative: they are the 'laggards' that Dr Everett M Rogers[xi] identifies in his Adopter Categories. Sometimes the only way of dealing with them is to move them out of the organization.

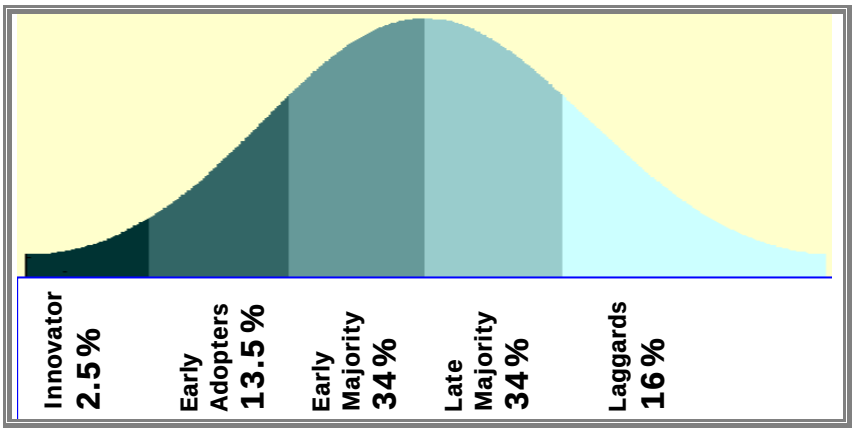


Figure - The 'Adopter Categories', Dr Everett M Rogers

To overcome resistance, it is vital to enlist the help of people who view change in a positive light. The best-suited candidates for supporting quality initiatives and making things happen are at the Innovator and Early Adopter end of the bell curve. These people need to be nurtured as they are the ones who can effect the required changes in organizational attitudes and aspirations. It is only possible to make an organization extraordinary by capitalizing on their extraordinary people. Extraordinary organizations will succeed where others fail. The average never wins:

'To succeed we have to surprise people. We have to attract and addict them. Attention is all.' [xii]

There is a checklist of 'Principles of Managing Organizational Change' in Appendix A, which provides further guidance.

Technology

Setting business strategy

The technology that underpins today's customer-facing 24/7 applications may still have some roots in its legacy systems. These are usually many years old, written in 3rd generation languages, rather inflexible to change and require highly specialized staff to maintain the code. When mergers happen, the new and enlarged organization may find itself with two or more systems essentially addressing the same business requirements. A way forward must be found, whether this is migration, re-engineering, re-writing, outsourcing or some combination of these.

Defining and gaining agreement to that plan can be an uphill struggle. Decisions should always be based on a rational analysis of the ability to meet the business needs in the future and the capacity, flexibility, and potential for longevity that each of the systems provides. Any decisions that are based on feelings, partial information or supposition will quickly fuel suspicion and claims of favoritism among aggrieved staff. One such mechanism for analyzing and defining the best way forward is Constructive Evaluation[xiii]. Using the principles of Quality By Design, the process involves identifying critical business goals, then gradually and iteratively drilling down to define what needs to be in place to meet the defined goals. In the hands of an experienced practitioner there can be no doubts regarding 'fair play'.

Similarly, some best guess must be made as to the main business functions the technology must support now and in the future. Assuming a range of up to 3 years is reasonable. Shorter periods run the risk of

purely tactical moves that damage the overall systems architecture, while longer periods are increasingly difficult to predict.

Strategy needs to be based on current knowledge and the most likely scenarios. It is probably a good assumption that the next two years will consist of almost constant change and that there will be another significant change after that. It is good practice to assess the most likely directions and then put in contingency plans and agreed checkpoints to mitigate the risk of potentially driving everyone down the wrong route. Whatever else happens, the need to have flexible systems (and flexible staff and processes) will become ever more of a driving factor.

Adding value over a period of time

Given the huge investment in most systems that support financial organizations, it is normally not viable to start completely from fresh with critical systems. If the old systems are still supporting at least some of the critical business functionality then the code has to be re-used or recovered in some way. It is, therefore, essential to discover what systems are in place. An audit must be performed to determine what systems are where, what applications are running on each system, and what hardware, operating systems, databases, and languages are being supported.

To move forward from here one must apply a policy of continuously removing or transitioning the systems that:

- Are a positive risk to the business (zap)
- No longer support the business's future (scrap)
- Need a GUI front end to give the look and feel that is wanted (wrap)
- Can, if transitioned to better technology, provide useful data/services (tap).

Once the physical audit has revealed what physical systems are present, the next step is to perform a physical to functional mapping, to determine to what extent the systems are supporting the required future business goals. Mapping the systems against the following simple matrix can help spot obvious candidates for zapping, scrapping, wrapping or tapping.

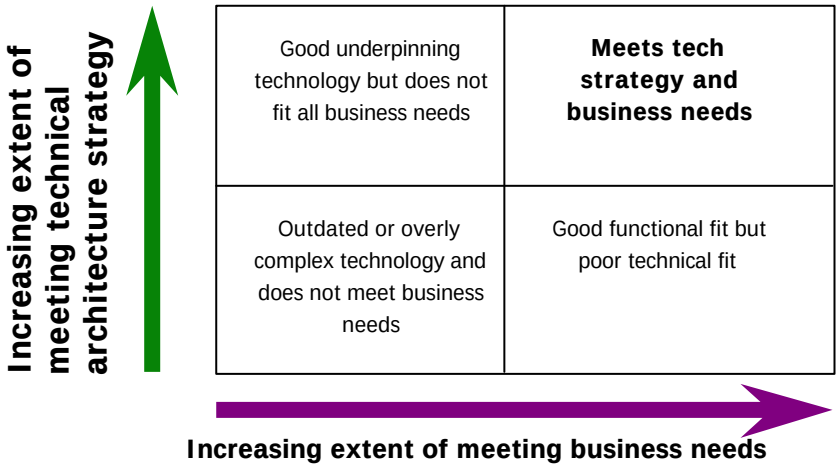


Figure. - 'Value to business' matrix; aim to zap or scrap the systems in lower left quadrant; consider wrap or tap for lower right quadrant. Systems in top left quadrant may, dependent on scalability, be candidates for further development, to enhance business fit.

It is also important to recognize the current levels of quality, including maintainability as measured by cyclomatic complexity, visibility of structure, etc. Unsatisfactory ratings against these measures could have a bearing on the need to replace sooner rather than later. Extensive examination of the systems against objective acceptance criteria, known as the 'ilities', (as in portability, flexibility, security, capacity, etc.) may also prove beneficial in determining the need to replace early.[xiv]

Define detailed mapping of existing/near future systems, against the blueprint

- The 'ilities' can help describe the increased requirements of the systems as they may be used in the enlarged organization or in a more customer focused way. Consider Constructive Evaluation Methodology[xv] for this detailed work.
- It is essential to define the future as clearly as one can even if it is prone to change. If it is documented and well thought out, it will form a solid basis for determining the risks involved in moving both towards and away from the blueprint when circumstances change.[xvi]

Process - Coping with differing process maturity across different sites

The third corner of the triangle is process. There is no possibility of building truly great systems, except by accident, unless the people who are designing and engineering them are working to well-defined, agreed processes.

With mergers and massive changes in global organizations, the likelihood is not that there are no development and project management standards, but rather too many disparate ones. There may also be different IDEs across various sites, which add to the complexity of developing a single project with multiple methods.

In addition to this, the speed of changeover between developing the old legacy systems and suddenly having to put together brand new systems with collaborative development techniques may mean that there are no defined processes for constructing the newer types of systems. This can mean that trouble is building up for later when these newer systems themselves advance to legacy status. Analyzing the code and re-engineering it to a cleaner design may be required to make these systems maintainable and capable of being deployed to meet the needs of the expanded organization.

As with technology and people, the only way through this is to find out what methods and processes are in use across the organization. Some analysis is needed to determine the level of use, the maturity of the processes, and the degree of success as evidenced by the visibility of the project process and the quality of the resulting software products. Only when this health check across the organization is done can any decisions be made as to the next steps.

Suspicion and competition can be rife between previously separate organizational areas. It is, therefore, essential that only objective measurements be used to determine the relative maturity and value of the processes in use.

Using an objective yardstick, like the CMM, can prove beneficial in such cases, as there can be no arguments if one part of the organization has already been assessed as a Level 3 and another part has no formal project management or software engineering processes in place.

If none of the component parts of the newly enlarged organization have a history of using CMM as a benchmark, then it is probably not a good idea to start full-scale assessments at this time. Full-scale

assessments can be a major investment and the organizational flux experienced post-merger will not form a good basis for the training and process development that will be required.

The most cost-effective approach to understanding the process maturity at each site is to organize a series of lightweight process reviews. This will avoid over spending on too-sophisticated mechanisms. With a knowledgeable software CMM practitioner, for example, a good 'feel' for the maturity of a development site can be obtained in a low number of hours, given some astute questioning and access to project and process documentation. Once this understanding has been obtained and documented, then the CMM can be used as a mechanism to prioritize the areas for potential improvement.

Global organizations frequently fall into the trap of aiming for a particular CMM level as their main goal. Whilst this is admirable in itself, it should not be the only aim: any improvement should be based on real business goals, not on some arbitrary notion of 'goodness'. It should be noted that CMM initiatives are also highly prone to failure: the main reason for failure – apart from the normal factors to do with resistance to change – is a lack of focus on meaningful business objectives.

Major CMM-based initiatives should not be attempted unless it is known what measurable (SMART) results are required to be achieved: for example – fewer defects into production, better predictability of project timeliness and delivery of functionality.

Aiming purely for a particular level means that too much emphasis is placed on the assessment itself, rather than improving the way things are done: once the assessment is over, the staff may just heave a sigh of relief and heave the process documentation out at the same time. If the right business goals are set, however, then the CMM level will be attained as a useful by-product – and the staff and management will want to continue working in such as way as to continue to reap the business benefits.

Should an attempt be made to bind together all the development groups?
Or is it best to allow the differences to co-exist?

There will almost certainly be some differences in methods of project management and software engineering methods used across previously disparate organizations.

Performing an initial review of the processes to discover the differences and then mapping some common terms for each site can be a useful first step, as a basis to understand how each site operates. Allowing differences in processes to allow for different styles of project, is something to be encouraged – provided that there are consistent and visible control points for projects, which can be understood and used when developers and project managers from different sites work on the same team. The results of the initial process reviews can form the basis for defining the terminology and processes to be used on the projects that follow.

Different development units will probably employ different IDEs. This is acceptable, provided that the quality processes are recognizable and they are being used to ensure that the right functionality is being delivered to the anticipated level of quality.

The most benefit can be obtained by ensuring that source code re-use is possible. To do this requires full visibility across the IT organization via consistent enabling technology. The most obvious candidates for supporting technology would be for the critical processes such as: requirements management; workflow through the project with quality reviews at each stage of deliverable; full software

configuration management throughout the development and testing phases; fault-fixing; and deployment. Rewarding developers every time they re-use code can also be an efficient and cost-effective way of reducing maintenance overhead in the longer term.

Judging by metrics - Cost

Any attempt to understand the relative cost of development and maintenance, versus the quantity of deliverable functionality, may be regarded with some suspicion, and thus fraught with problems. On the other hand, it will not be easy to determine what is to be done with a particular development site, until one understands it in objective terms.

One needs to determine, in bald terms, what it costs to develop systems in this site, as opposed to that site. Any comparison is only meaningful if the size (including complexity) of the development is taken into account. Descriptions of effort or duration are not verifiable as a measure of size.

Installing Function Point or Feature Point analysis is likely to be too time-consuming and expensive to be of benefit in the shorter term. Instead, consider a simple mechanism, using self-defined size/complexity attributes, which can be easily trained and used. The same mechanism can be used for estimating all project size metrics, which will also be of benefit in the longer term.

Right quality products and projects

The following are some easy ways of finding out whether the projects are delivering what is wanted:

- Judge by the quality of the resulting code; number of defects per KLOC, measured at intervals: e.g. on implementation, and at 6 months, 12 months, 18 months
- Judge by project timeliness, and whether the project is on-budget
- Examine those teams that are consistently delivering late and/or over-budget – and find out why this happens
- Examine those who are consistently good – and find out why: reward and promote their achievements

Define the goals for process quality and process adherence

Simple mechanisms, like defining a set of artifacts to be used in each project, can make a big difference to the quality of software products. If there are no consistent artifacts across the organization, consider implementing some or all of these: quality plans, risk management plans, configuration management plans, test strategy plans, standardized progress reports, standardized time-sheets, requirements change request forms, quality assurance and customer sign-off forms for deliverable work-products or documents, and post implementation reviews. Consider implementing formal peer reviews of critical work-products, as they can be a cost-effective way of reducing defects early in the lifecycle. Even rapid delivery projects can benefit from extremely concise versions of all these control-mechanisms.

Alternatively, consider implementing a process management and workflow system to ensure that each work product (i.e., software or management artifact) conforms with the required flow for that type of project. For example, a workflow rule can be put in place to make sure that Quality Assurance reviews a risk management plan before it goes out to a project sponsor.

Process improvements and quality initiatives in general can generate enormous rewards both in tangible forms (such as a reduction in maintenance costs) and in intangible ways (staff satisfaction). The only way to reap the benefits, however, is to make sure that the improvements are sustainable in the long term. This requires planning, persistence and a proactive interest in the processes to make sure that the improvements become – and remain – a way of life, or 'the way we do things around here'.

Conclusions

In situations where an organization is dealing with the aftermath of a merger or is going through rapid transition and organizational churn, it is not recommended to commence large-scale quality initiatives such as Six Sigma or TQM. Such initiatives are unlikely to survive long enough to provide a genuine ROI. For most organizations, there is too much risk, too high a financial investment in the required level of training without sufficient focus on results. It is not completely unheard of for these initiatives to survive in such conditions, but it is quite rare and demands an exceptional level of leadership and commitment.

There can be similar issues with wholesale adoption of benchmarking methods to achieve 'a level' for the sake of it: sensitive and selective use of the models, aimed at making improvements to reach particular business goals, is an approach which is much more likely to succeed.

To reap rewards from quality initiatives during the early stages of a merger, ensure that you have clear and visible mechanisms in place for focusing on quick wins with SMART objectives and verifiable results in each of the following areas:

- Technology – make it future-proof
- People – ensure that their skills are honed
- Processes – make them visible and consistent

Consider combining your processes with underpinning technology so that there is more likelihood of synergy, compliance and more predictable and better quality results.

Remember that, until the post-merger integration process is well under way, there will be suspicion as to the 'hidden agenda' of any attempt at measuring software quality, team productivity or maturity of processes. Overcome this by ensuring that there is frequent, clear communication as to future plans, successes and failures; opting for team-based workshops 'across the divide'; and ensuring that all measurement or assessment of quality is done as objectively as possible.

Above all, keep it simple, keep it forward-looking and focus on positive, achievable results.

Appendix A

Principles for managing organizational change – a quick checklist

- Expect & allow for a period of mourning for the old
- Analyze the past
  - Take forward what is good
  - Throw away the no-longer relevant
  - Balance-sheet AND a people-based approach
  - Cost reductions you need to achieve
- Define business goals for the skill-sets you need in the future
  - Define what skills are most appropriate for the future needs: technical and interpersonal
  - Consider People-CMM[xvii] as a basis for understanding the maturity of the processes relating to human capital, as evidenced across different parts of the organization
- Audit the skill-sets that you have now
  - What technical skill-sets do people possess?
  - Are these skills relevant to the new organization?
  - What personal/communication skills do they have?
  - Are they willing/able to move forward?
- Define the gap & make action plans to move forward
  - Use the skills audit to form a skills-matrix, against the technologies that you intend to employ in the future
  - Gaps in knowledge will demonstrate where to train people, or whether their combined skills and experience demonstrate that they should move on
- Encourage everyone to think and act as a leader
  - Help everyone to gain the most from group synergy
  - Help ensure that the action plans are tracked and actively progressed
  - Review progress in light of changing organizational needs
- Communicate early and frequently
  - Intentions, concepts
  - Plans; even communicating a lack of firm plans can help
  - Successes and failures
  - Be as open as possible, to avoid negativity created by uncertainty

Bibliography

[i] http://www.motorola.com/mediacenter/news/detail/0,1958,467_243_23,00.html

[ii] At GE, Six Sigma is 'the way we work around here' http://www.ge.com/sixsigma/

[iii] http://www.boeing.com/news/feature/baldrige/14mar_s84617.pdf and

for CMM: http://www.boeing.com/news/releases/2002/q1/nr_020225m.htm

[iv] http://www.raytheon.com/press/1999/jan/seilev5.html

[v] http://www.juran.com/research/articles/article004.html

[vi] 'A Business Case for Software Process Improvement Revised - Measuring Return on Investment from Software Engineering and Management', an updated DACS State-of–the-Art Report; Thomas McGibbon, Sept 30, 1999; http://www.dacs.dtic.mil

[vii] SMART: Specific, Measurable, Achievable, Realistic and Time-Bound

[viii] Lloyd's Register Quality Assurance http://www.lrqa.co.uk/Default.htm?trainingservices/pdca.htm~mainFrame

[ix] 'The Capability Maturity Model', Paulk *et al*, CMU SEI, Addison Wesley, 1994

[x] Abraham H Maslow: 'Motivation and Human Personality', 1954

[xi] 'Diffusion of Innovations'; Dr Everett M Rogers, The Free Press, 1995

[xii] 'Funky Business' *ibid* please see reference above

[xiii] 'Constructive Evaluation Method', based on the principles of Quality By design. Developed by Professor Vic Stenning, *et al*, as a practical, iterative, method of ensuring that software and systems are developed according to rigorous principles, to meet their stated objectives. Contact Prof Stenning for consultancy and advice on using the method. Downloadable copy of method from http://www.anshar.com

[xiv] Software Quality Engineering principles, as proposed by Tim Kasse, one of the contributing authors of the Software CMM who continues to provide inspiration and guidance on software process improvements: http://www.kasseinitiatives.com

[xv] 'Constructive Evaluation Method', *ibid* please see reference xiii

[xvi] For a useful discussion of developing a technical architecture, see: 'Re-shaping IT for Business Flexibility', Mark Behrsin, Geoff Mason, Trevor Sharpe; IBM McGraw Hill Series, 1994

[xvii] 'The People Capability Maturity Model': Curtis, B., Hefley, W.E., & Miller S; CMU Software Engineering Institute, 1995. Drs Curtis and Hefley both provide consultancy and assessment services around the People CMM and also the Software and Integrated CMMs. See the SEI website for further details - http://www.sei.cmu.edu

METHODS & TOOLS is published by **Martinig & Associates,** Rue des Marronniers 25, CH-1800 Vevey, Switzerland Tel. +41 21 922 13 00 Fax +41 21 921 23 53 www.martinig.ch
Editor: Franco Martinig; e-mail franco@martinig.ch ISSN 1023-4918
Free subscription: www.methodsandtools.com