Improving Validation Activities in a Global Software Development

Christof Ebert, Casimiro Hernandez Parro, Roland Suttels, Harald Kolarczyk
Alcatel, Switching and Routing Division, Antwerp, Belgium; Madrid , Spain; Stuttgart, Germany
Correspondence: Tel.: +32-3-240-4081; e-mail: christof.ebert@alcatel.be

Abstract
Global software development challenges traditional techniques of software engineering, such as peer reviews or teamwork. Effective teamwork and coaching of engineers highly contribute towards successful projects. We will evaluate within this case study experiences with validation activities in a global setting within Alcatel's Switching and Routing business. We will investigate three hypotheses related to effects of collocated inspections, intensive coaching, and feature-oriented development teams on globally distributed projects. As all these activities mean initial investment compared to a standard process with scattered activities, the major validation criteria for the 3 hypotheses is cost reduction due to earlier defect detection and less defects introduced. The data is taken from a sample of over 60 international projects of various sizes from which we collected all type of product and process metrics in the past 4 years.

Keywords
Global development, validation, inspection, defect detection, efficiency, cost of non-quality, coaching, feature development, incremental development, teamwork

1. Introduction

Working in a global context has along tradition for telecommunication suppliers. The primary drivers in the past were the need to be locally present in terms of customization, after sales service, and show to local (governmental) customers that new jobs were created which in turn could justify more contracts. A growing amount of acquisitions, which add new markets, products, engineers, and creativity to the existing team, is another contributor to global development. A third reason for even starting new development activities in countries where neither the market nor the acquisitions would justify such evolution is the simple fact that in the existing sites it's impossible to further hire young engineers with right skills at reasonable cost. The answer in such cases is to start business in countries such as Eastern Europe or India, which we did over the past years.

Obviously working in a global context has advantages but also drawbacks. In fact, the business case is surely not a simple trade-off of different cost of engineering in different regions or time-zone effectiveness. Working in a globally distributed project means overheads for planning and managing people. It means language and cultural barriers. It creates jealousy between the more expensive engineers being afraid of loosing their jobs, while forced to train their much cheaper counterparts.

We will focus in this case study on impacts of global development towards validation activities. Form a business perspective this means impacts on quality and cost of non-quality.

Cost of non-quality is the cost of not reaching the desired quality level at the first run. It is often referred to as "rework". We calculate cost of non-quality by summarizing respective (life cycle phase depending) cost for defect detection and correction across all defects found in the project.

Global software development obviously challenges traditional validation techniques of software engineering and asks for new solutions. We will try in this case study to summarize experiences and to share the best practices from projects of different type and size that involved several locations in different continents and cultures. Especially validation activities during development, such as inspections or unit test need to be adjusted to achieve results, which are both efficient and effective. As a basis, we will evaluate experiences with validation activities in a global setting within Alcatel's Switching and Routing business. Such complex software systems show the various dimensions of global development and offer practical solutions as they have been managed since years in a global context. The challenges, which we will address in this case study, involve:

- to support of validation in a global product line concept;
- to facilitate early defect detection with an improved validation process;
- to reduce overall cost of non-quality.

We will analyze three hypotheses related to those challenges in the context of over 60 projects developed in a three-year timeframe. This study is unique in that it provides insight in a running software process improvement (SPI) program within a large organization dealing with legacy software in a global context. Around two thirds of the projects had been handled before the changes were implemented, some projects implemented only parts, and several projects implemented all three mentioned changes. This allows comparing impacts directly linked to any of the three factors.

Few documented results quantitatively evaluate the effects of global development. A good summary on the evolution of concurrent engineering is provided in [1]. Several recent studies describe experiences gained from distributed projects that are not further described in detail [2,3,4]. They provide a huge set of project management and team management techniques, which we could also apply, but did not give quantitative evidence about the effectiveness. In fact, most evaluations of validation techniques happen in classroom settings or collocated projects [5,6,7].

Only few studies describe the problem and a solution how to handle remote inspections in large-scale software development [8]. Experiences are based on a tool that facilitates inspections and annotations of results, even if the checker is located remotely. There are however no concrete results available on efficiency and effectiveness compared to a collocated setting, which is what we were interested in.

The effects of coaching had not been studied in the area of software engineering besides the practical guidelines from change management and technology introduction gained from using the CMM [2,3,5,9]. Most results published so far describe the background of such a program with focus on the assessment and qualitative observations [9,10]. They are in many cases looking on rather small groups of engineers that act like a small- to medium-size company, even when embedded in a big organization [7].

Several qualitative lessons learned have been documented [11], but they are difficult to scale up towards large legacy based development projects. They try to set up a return on investment (ROI) calculation that however typically takes average values across organizations and would not show the lessons learned in sufficient depth within one organization. It has been shown in these studies that the CMM is an effective roadmap for achieving cost-effective solutions. Often these studies seem not to be related to quantitatively specified upfront expectations.

Several current best practices related to continuous build, configuration management, inspections and validations within a product line concept involving legacy software are elaborated in [2,8,12,13]. This involves the impacts of team management based on concrete measurable targets that are followed up [7,10,14] - sometimes even to the extreme when it comes to surviving a project running out of control [2]. Again, what is missing is a timeline study summarizing impacts before and after the introduction of such process.

Within this paper specific terminology is used that should be briefly explained. CMM is the capability maturity model; SPI is software process improvement; PY is person years and Ph is person hours, r is the correlation coefficient measured in the observed data set (depending on distribution we use Spearman or rank correlation). Size is measured in KStmt, which are thousand delivered executable statements of code (incl. declarations). We prefer statement counting compared to lines because the contents of a program are described in statements and should not depend on the editorial style.

Failures are deviations from a specified functional behavior. They are caused by defects. Reviews detect defects and need to distinguish between what would cause a failure and what would be editorial or related to nonfunctional behavior. We summarize these review activities as validation as we compare to both functionality and design decisions. Coaching is used in this study as on the job support of engineers by more experienced peers on process and technology related aspects.

The paper is organized as follows. Section 2 introduces the environment and set-up of the study. Section 3 describes some essential lessons learned and best practices how to improve validation activities in a global software development. Finally, section 4 summarizes results and probes further.

2. Case Study Setting

Alcatel is a globally acting telecommunication supplier with a multitude of development projects, which typically involve different countries. Software development of switching and routing systems involves several thousand of software engineers. This development staff is distributed over the whole world in more than 15 development centers in Europe, the US, Asia (especially India) and Australia. Strong functional and project organizations, which interact in a classic matrix, facilitate both project focus and long-term skill and technology evolution.

The study describes projects within Alcatel's Switching and Routing business. The components within the product range from the proprietary S12 switching system to Corba/Java middleware and frontends. Alcatel is registered for the ISO 9001 standard. The majority of development locations are

ranked on CMM L2; few are on CMM L3. In terms of effort or cost, the share of software is increasing continuously and is currently in the range of 90 %. We focus in this study on switching projects as they are developed typically involving at least two to three sites, often in several continents. The projects vary in size between few person years and several hundred person years, depending how much new development is involved.

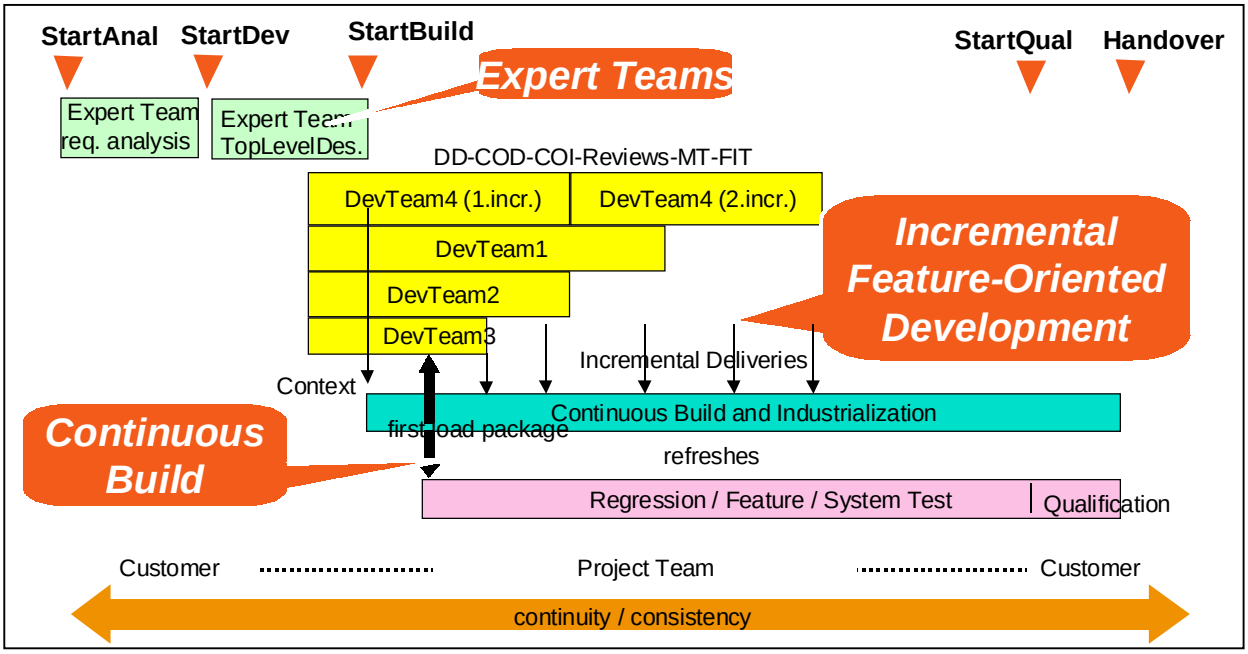


Fig. 1: Process Overview Building upon Feature-Oriented Teams and Continuous Build

To avoid overheads in terms of redundant skills and resource buffers in various locations; engineering is entirely globally managed. Having no option for working local, projects are organized to achieve the best possible efficiency by optimizing the trade-off between temporarily collocating teams, reducing overheads, and having the right skills in due time.

The development process focuses on teamwork and continuous build [2,12]. It is further outlined in fig.1, which distinguishes between upfront Expert Teams for analysis and system design, Development Teams which deliver increments to the continuous build, and the overlapping (sandwich) Test Team doing the regression, feature and system test activities. Some details related to making teamwork more efficient are discussed in the following section.

The project data has been collected in a standardized way since years by means of a history database [15]. Data is aggregated from the respective operational databases (e.g. effort or defect data) towards teams, and consolidated for the entire project. Per project data is collected in two ways. Obviously the first source is the online project information which is accessible from a project homepage. It is online available in operational databases and reflects real-time accuracy. Such online data includes defect reporting with status, test progress, requirements implementation status, etc. In addition to this operational online data there is from begin to end a project-specific consolidation data sheet capturing project and process metrics, such as size, effort, defects and their distribution, etc. This consolidated data is available on a monthly basis to track overall performance with respect to performance targets (e.g. quality, reliability, cost, cost per activity, cycle time, etc.)

Having done that since '96 allows us to study impacts of various process changes and other parameters on project performance. We will use in this study the data of 60 projects, which we combined with some qualitative information to link each project towards the three hypotheses. Pilot projects are not included in this study. Only those projects were considered that were linked to a clear customer contract from the beginning.

Having the independent variables closely linked to the over 60 projects in our study, we could extract impacts of each single variable - following the rule that there should be at least 10 samples for each variable. This avoids conclusions of the type described in experimental software engineering as shotgun approach with uncontrolled independent variables or the Hawthorne effect [16]. The subsets of projects used here to explain results are not overlapping. This means that effects attributing to one result would not attribute simultaneously to another and thus hide the real explanation.

To avoid long discussion on possible side effects as they could occur if we discuss the entire design process (e.g. tools impacts, skills impact), we narrowed down the design process towards validation activities, especially code reviews. Validation - or not doing it at the right time and with the right process - is a major cost driver and risk factor in almost all software projects.

Normalized cost of non-quality has been calculated based on actual defects detected and actual average cost per defect (incl. detection and correction) per activity where it is detected. Defects detected after handover are accounted with same cost as during test to avoid different types of market-specific opportunistic and penalty cost in the field. Then we calculated the average cost per defect for the entire project and normalized with size. The result is a project-specific average value of Ph/defect.

3. Improving Validation Activities

Competition, along with the customers' willingness to change suppliers whenever they are dissatisfied has resulted in huge efforts to provide switching software on time and with *exactly* the quality the customer has specified and expects to pay for. A study by the Strategic Planning Institute shows that customer-perceived quality is amongst the three factors with the strongest influence on long-term profitability of a company. Customers typically view achieving the right balance among reliability, delivery date, and cost as having the greatest effect on their long-term link to a company. We thus focus here on making validation activities more effective.

Since defects can never be entirely avoided, several techniques have been suggested for detecting defects early in the development life cycle [2,10,17]:

- design reviews and inspections;
- code inspections with checklists based on typical fault situations or critical areas in the software;
- enforced reviews and testing of critical areas (in terms of complexity, former failures, expected fault density, individual change history, customer's risk and occurrence probability);
- tracking the effort spent for analyses, reviews, and inspections and separating according to requirements to find out areas not sufficiently covered.

We will further focus on several selected approaches that are applied for improved defect detection before starting with integration test and system test.

To improve validation activities in global project settings we have embarked on three separate changes, which we will further discuss. Each change had been linked to a hypothesis (or management expectation) which we evaluated during the projects. Having many projects in parallel and being forced to carefully introduce change to avoid any confusion on which process is applied in a certain project, we had after three years enough data from projects having implemented none or several of mentioned changes.

Changes that impact entire development processes need good management sponsorship [10]. Although all three changes that we evaluate and describe in this case study have been discussed in literature before [2], it's not at all evident that such a big organization would easily go for it. Many people that are used to doing things their own way need to be convinced. Often this means heavy disputes that can only be resolved with hard facts. A case study (or extensive pilot) thus is key to successful (i.e. sustainable) process change.

The changes had been piloted first in a small and uncritical environment before starting to roll out the change. This allowed to carefully check results versus initial hypotheses (or assumptions) and to prepare the necessary changes to the management system (e.g. processes, rules, planning guidelines, budgeting guidelines, training materials, tools adaptations, etc.). Pilots were then not continued in increasingly broader settings, but the sponsoring management decided upon the good results and benchmarking evidence from other companies, that the changes become mandatory. In fact it's a risk to pilot too long, as the variety of processes confuses the engineering teams.

The following three hypotheses which we set upfront (i.e. as management decisions supported by history data) will be evaluated in the following three subsections:

- **Hypothesis 1:** Collocating peer reviews would improve both efficiency and effectiveness of defect detection and thus reduce cost of non-quality.
- **Hypothesis 2:** Providing a certain level of coaching within the project reduces cost of non-quality.
- **Hypothesis 3:** Reengineering the development process towards teamwork and continuous build allows to better managing globally distributed projects, and thus reduces cost of non-quality.

The hypotheses all center around facilitating early defect detection in the development of projects that are handled in a global context. They are ordered according to introduction effort. While collocation of peer reviews is rather easy to achieve, coaching means already a certain investment and the reengineering of a development process is certainly the biggest change described in this study.

3.1 Collocating Peer Reviews

The single most relevant techniques for early and cost-effective defect detection are inspections and module test. Detecting faults in architecture and design documents has considerable benefit from a cost perspective, because these defects are expensive to correct. Major yields in terms of

reliability however can be attributed to better code, for the simple reason that there are much more defects residing in code that were also inserted during the coding activity. We therefore provide more depth on techniques that help improving quality of code, namely code reviews (i.e. code reading and code inspections) and module test.

There are six possible paths between delivery of a module from design until start of integration test (fig.2). They indicate the permutations of doing code reading alone, performing code inspections and applying module test.

Although the best approach surely is from a mere defect detection perspective to apply inspections and module test, cost considerations and the objective to reduce elapse time and thus improve throughput, suggested to carefully evaluate which path to go in order to most efficiently and effectively detecting and removing faults. To our experience code reading is the cheapest detection technique, while module test is the most expensive. Code inspections lie somewhat in between.

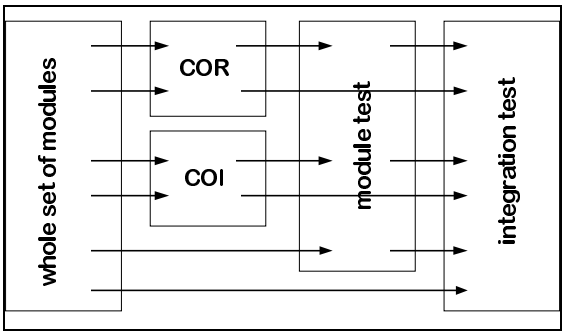


Fig. 2: Six possible paths for modules between end of coding and start of integration test (COR: code reviews; COI: formal code inspections)

Module test however, combined with C0 coverage targets has highest effectiveness for regression testing of existing functionality. Inspections on the other hand help in detecting distinct fault classes that can only be found under load in the field.

There is nevertheless a tendency not to perform these inexpensive validation techniques adequately. Fig.3 indicates the typical vicious circle of not validating when it's the right time, and as a consequence later having to detect defects at much higher cost thus again taking away unexpectedly resources during the next design activity of a project.

The target must be to find the right balance between efficiency (time to be spent per item) and effectiveness (ratio of detected faults compared to remaining faults) by making the right decisions to spend the budget for the most appropriate quality assurance methods. In addition, overall efficiency and effectiveness have to be optimized. It must be therefore carefully decided which method should be applied on which work product to guarantee high efficiency and effectiveness of code reading (i.e. done by one checker) and code inspections (i.e. done by multiple checkers in a controlled setting).

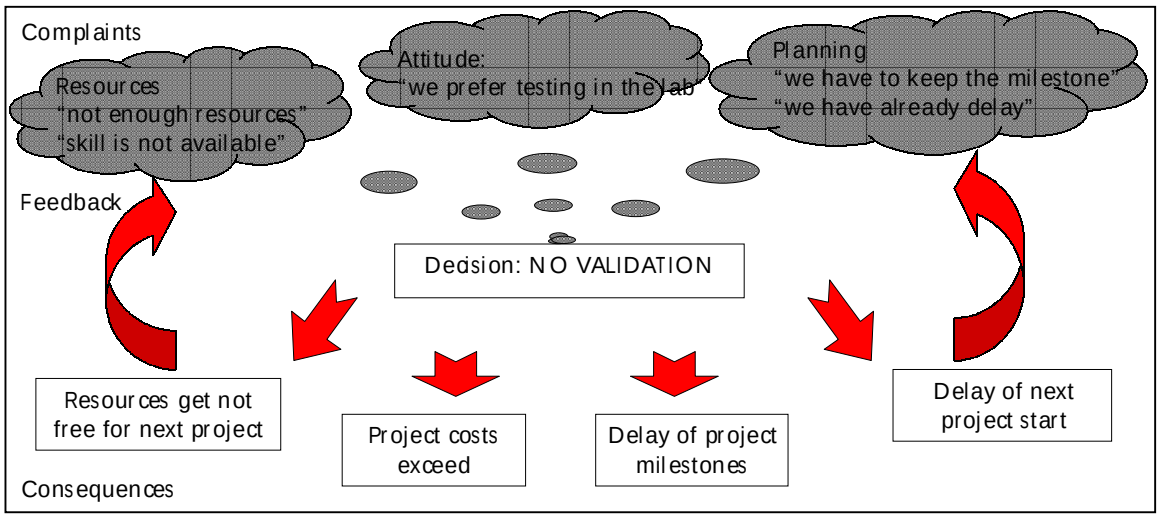
Wrong decisions can mainly have two impacts:

- Proposed method to be performed is too 'weak': Faults that could have been found with a stronger method are not detected in the early phase. Too little effort would be spend in the early phase. Typically in this situation, efficiency is high and effectiveness is low.
- Proposed method to be performed is too 'strong': If the fault density is low from the very beginning, even an effective method will not discover many faults. This leads to a low efficiency, compared to the average effort, which has to be spent to detect one fault. This holds especially for small changes in legacy code.

Globally distributed software development is highly impacted by work organization and effective work split. Often not all necessary skills to design a complex functionality are available at one location. While some authors recommend building virtual teams [3], we strongly advice to build coherent and collocated teams of fully allocated engineers. Coherence means that the work is split during development according to feature content, which allows assembling a team that can implement a set of related functionality - as opposed to artificial architecture splits. Collocation means that engineers working on such a set of coherent functionality should sit in the same building, if feasible in the same room. Full allocation finally implies that engineers working on a project should not be distracted by different tasks for other projects.

Projects are at their start already split into pieces of coherent functionality that will be delivered in increments to a continuous build. Functional entities are allocated to development teams, which are often based in different locations. Architecture decisions, decision reviews at major milestones, and test are done at one place. Experts from countries with minority contribution will be relocated for the time the team needs to work together. This allows effective project management based on the teams that are fully responsible for quality and delivery accuracy of their functionality.

Fig. 3: The vicious cycle of not reviewing results in due time.



Collocating a development team to stimulate more interactions is more expensive and might demotivate engineers due to traveling. We based our hypothesis on earlier qualitative studies telling that a team is only efficient when communication of team members happens whenever necessary and without long planning and preparation [4]. To base our decision on experiences within Alcatel, we studied projects where we could distinguish according to the factor of collocation degree.

The hypothesis we tested is that collocating peer reviews would improve both efficiency and effectiveness of defect detection and thus reduce cost of non-quality. Out of three recent projects that included several development teams that were globally distributed, we looked into the results of each single code inspection. 87 inspections were randomly selected.

We divided 2 data sets, one with collocated development teams and inspections, and one with distributed teams where inspections were conducted remotely. All other variables remained unchanged. The two sets showed normal distribution with average values of 25 vs. 13 defects/KStmt and 0,33 vs. 0,08 Ph/defect, respectively. A t-test shows for this data an evidence of more than 98% that indeed the two sets can be considered independent. Other impacts such as different knowledge of underlying code or skill level of engineers could not be found as explanation factor in this data, as the involved engineers had experience with the underlying baseline. The hypothesis was thus accepted on a level α = 0,02.

Looking into individual team performance, we could see that collocated teams achieve an efficiency improvement during inspections of over 50%. This means that with the same amount of defects in design and code, those teams, which sit at the same place, need less than half the time for defect detection. The amount of defects detected shows almost a factor 2 difference in terms of defects per KStmt. Looking towards the low cost of defect detection during inspections compared to subsequent testing activities and the cost contribution of validation towards total cost, we found an impact of >10% on project cost.

3.2 Effective Process Coaching

Continuous technical training and coaching seems natural for any engineering activity. Looking into post mortem studies of finished projects with respect to training activities, we realized that there are big differences in terms of phase-specific training that involves both technical and process aspects. Some project managers focus heavily on providing all necessary technical and process information at respective phase kick-off meetings (e.g. start of detailed design, or start of test), while others just present some rudimentary technical information and do not further bother with ongoing coaching.

Coaching in the context of validation processes is described in fig.4. First an initial process is defined and piloted. For peer reviews for instance this might include checking speed, duration of reviews, specific checklists, etc. This is also where such case studies appear as the one underlying this article. This process is then evaluated in real project settings. Actual data is compared with the original definitions and proposals. Practitioners are confronted with the results and together with experts for the respective process, they investigate results. The process is finally updated. Process coaches are the link between these elements of process refinement. They also guarantee consistency in applying the process. Finally they evaluate process data and propose together with practitioners updates for the respective processes.

Effective coaches consider main learning effects from past projects and relate them to available process expertise of the respective teams being coached. For validation activities which we embarked on within this case study there are the following elements being considered:

- General availability of a review or inspection leader: Only a trained and internally certified inspection leader is allowed to plan and perform inspections to ensure adherence to the formal rules and achievement of efficiency targets. The number of certified inspection leaders and their availability limits the number of performed inspections for a particular project.
- Upfront design effort (planned versus actually spent): The actual design effort per component (i.e. class or module) gives an initial estimate on how much code will be new or changed. This indicates the effort that will be necessary for validation tasks like inspections.
- Expertise of the checker: If specific knowledge is necessary to check particular parts of the software the availability of correspondingly skilled persons will have an impact on the planning of code reviews and code inspections.
- Checking rate: Based on the program language and historic experiences in previous projects the optimal checking rate determines the necessary effort to be planned.
- Size of new or changed statements: Relating to the checking rate the total amount of the target size to be inspected defines the necessary effort.
- Quality targets: If high-risk areas are identified (e.g. unexpected changes to previously stable components or unstable inputs from a previous project) exhaustive inspections must be considered.
- Balancing the cost-benefit trade-off of review activities: The intention is to apply code inspections on heavily changed modules first, to optimize payback of the additional effort that has to be spent compared to the lower effort for code reading. Code reading is recommended to be performed by the author himself for very small changes with a checking time shorter than two hours in order to profit from a good efficiency of code reading. The effort for knowledge transfer to another designer can be saved.
- Achieving the entry criteria: The inspection or review can start earliest if entry criteria for these procedures can be matched. Typically, at least error-free compilable sources have to be available.

Often coaching of engineers during the projects is reduced due to assumed negative impacts on total cost and duration. We found however the opposite. Reduced coaching harms overall project performance.

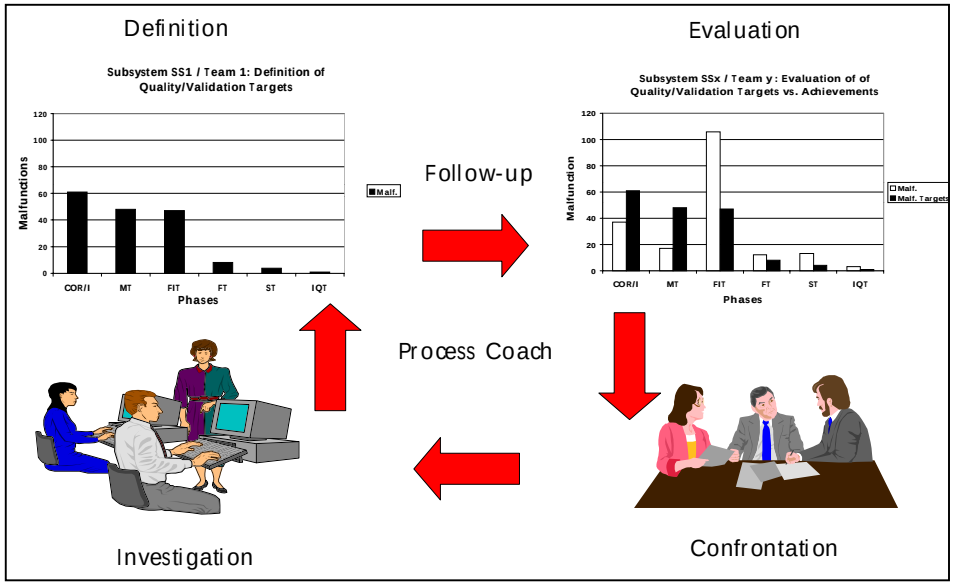


Fig. 4: Successfully introducing and sustaining validation activities is a culture change requesting adequate process management.

The hypothesis we tested is that providing a certain level of coaching within the project reduces cost of non-quality. Coaching in this study is the amount of on the job support by experienced engineers. Coaching comes on top of regular technical training and happens entirely on the job by means of allocating experienced engineers to teams of less experienced engineers. We compared a set of projects within one culture (i.e. Europe) and similar skill background (i.e. engineers had sufficient technical knowledge of the software package) that received a coaching effort of ca. 1..2% of total project budget with a second set of projects that received no coaching. This data set was evaluated in more depth to see not only the coaching degree but to also directly evaluate the type of coaching and the phases where coaching was intensified.

We found in this specific data set that coaching intensive projects had an average of 24 Ph/defect, while those with no coaching had an average of 29 Ph/defect. A t-test shows for this data an evidence of more than 90% that indeed the two sets can be considered independent. Based on the careful selection of the projects we could not see other impacts such as engineering skills or different stability of baselines. To see also the bigger picture of all projects in the history database, we did a simplified check based on a ranking of coaching effort for 68 projects over 4 years. Both parametric and non-parametric tests show similar results with significance level of > 90% The hypothesis was thus accepted on a level $\alpha = 0,1$.

Intensive coaching (ca. 1..2% of accumulated phase effort) could reduce the cost of non-quality in the project by over 20%. We found that for our own process and defect detection cost a break-even would be reached at ca. 5% coaching effort. Obviously, this is much more than what we usually consider necessary. This also means that there are quantifiable limits towards involving too many inexperienced engineers in one project.

3.3 Introducing Teamwork and Continuous Build

An essential factor in managing a global project is to create responsibility for results. We faced in the past often a situation where distributed projects were heavily impacted by the functional line organization or even some local legacy organization. However, nobody felt responsible for achieving results. The result was poor productivity and unpredictable delivery accuracy.

Due to not having a product perspective, work products were handled inefficient. Results were forwarded to the next in the chain, and cost of non-quality as well as delays accumulated. For instance, inspections were considered finished when the respective milestone date appeared, instead of applying reasonable exit criteria, before continuing the defect detection with the next and more expensive activity. Test was conducted with a rather static set of test cases that was not dynamically filtered according to real feature impacts. The root causes were obvious but so much embedded in the culture that a complete process reengineering was necessary to facilitate global development at competitive cost.

The major change, which we implemented to allow for global development, combine concurrent engineering, continuous build, and teamwork. They are supported by the respective workflow techniques. Concurrent engineering means that we assemble cross-functional teams focussed on customer requirements. Even before project kick-off a first expert team is assembled to ensure a complete impact analysis which is prerequisite to defining increments. Concurrent engineering also means that for instance a tester is also part of the team as experience shows that designers and testers look at the same problem very differently. Testability and reduced cost of test can only be ensured with a focus on test strategy and the potential impacts of design decisions already during the initial phases of the project.

Teamwork was reinforced to the degree that a team has sole responsibility for realizing a set of customer requirements. This means that not anymore a designer would leave the team when her work product is coded, but would stay to test the work products in the context of those changes provided by other team members. Feature-orientation clearly dominates artificial architectural splits [2,12]. The targets of the team are based on project targets and are shared by all team members. They are followed up based on delivered value, i.e. feature content [14]. Periodic reviews of team progress with the project lead are necessary to follow up and help in case of risks that cannot be mitigated inside the team.

The changes we introduced towards achieving real incremental development can be summarized as follows (see also fig.1):

- Analyze requirements from the beginning in the view of how they could be clustered to related functionality, which later could be delivered as an increment.
- Analyze context (data structures that are common for all modules) impacts of all increments upfront before start of development. The elaboration phase is critical to make incremental development and a stable test line feasible.
- Provide a project plan that is based on these sets of clustered customer requirements; allocate each requirements set to a development team. Depending on the impact of the increments, they can be delivered to the test line more or less frequently. For instance, if a context impact is detected too late, a new production cycle is necessary which is taking more effort and lead time, than regular asynchronous increments of additional code within the originally defined context.
- Each increment is developed within one dedicated team, although a team might be assigned to several increments in a row. Increments must be completed until end of unit and feature integration test to avoid that the various components later cannot be accepted to the test line. An important criterion for the quality of increments is that they don't break the build.
- The progress tracking of development and test is primarily based on the integration and testing of single customer requirements. This for the first time gives visibility on real progress because a requirement can only be checked off if it is successfully integrated in the test line. Traceability is improved because each customer requirements links to the related work products.
- Increments are extensively feature tested by the independent test line.

Increments towards a stable build proved one of the key success factors in global development. We realized that cycle time of projects is heavily impacted by whether continuous build is globally applied or not.

The hypothesis we tested is that reengineering the development process towards teamwork and continuous build allows to better managing globally distributed projects, and thus reduces cost of non-quality.

Obviously one can argue that this hypothesis actually contains two different pieces, namely teamwork and continuous build. They are however closely related especially in global development or in large projects. It's practically impossible (or very cumbersome, error-prone, time-consuming and thus inefficient) to work with individual empowered development

teams, if no continuous build is applied. This would for instance mean that each team runs the risk of reworking their delivery because another team has invalidated specific design decisions. On the other hand, in big projects a continuous build is meaningless if no teamwork is involved to distribute work into chunks.

We evaluated the effects of this reengineered process carefully over the past 2 years. Consequently, we see two effects contributing to the hypothesis. Response time and thus overall cycle time is reduced as defect correction happens in the team (fig. 5). Field defects are reduced due to focus on an optimized test strategy, longer overall test-time and end-to-end responsibility of a development team.

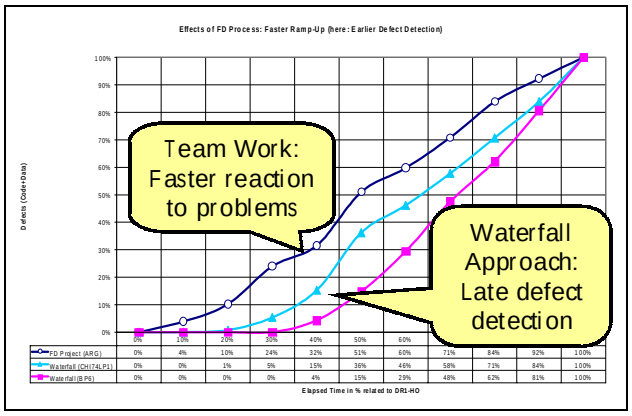


Fig. 5: Effective Team Management Scales Directly up to Faster Reaction Time

Cost of non-quality in the overall project has been reduced significantly due to this earlier defect detection. The hypothesis was tested in a set of 68 projects over the past 4 years (i.e. before and after the change). Consequently we can accept with a significance level of >95% in a t-test that the change towards team work and continuous build indeed reduces cost of non-quality.

4. Lessons Learned

Managing global software development is not easy and has high-risk exposure to lowering overall productivity. By introducing some changes towards improving the sense of responsibility for end results, we are able to keep quality and productivity standards and to improve performance - even with this changing work environment.

Looking into the overall project history database, we can also summarize the effects with few quantitative charts (fig. 6-8). Over the past 4..5 years defect detection has been shifted increasingly to the activity when defects are actually introduced into the project (fig. 6). Consequently, we could embark on the improvement activities mentioned in this article.

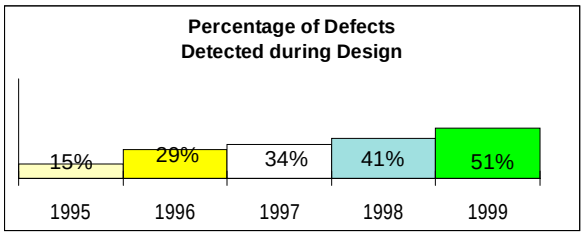


Fig. 6: Earlier Defect Detection during Design Allowed to Focus on Collocated Peer Reviews and Sufficient Coaching

With the attention on collocating peer reviews, embarking on better process coaching, and introducing incremental build principle into the projects, we could improve cost of non-quality by a factor of over 50% (fig. 7). Although the direct impact is visible on defects and quality level, we could achieve both cost and cycle time improvements (fig. 7,8). This is obvious if teams are encouraged to focus on the right quality level as long as they are in full control of the results, and not to wait until the next project phase.

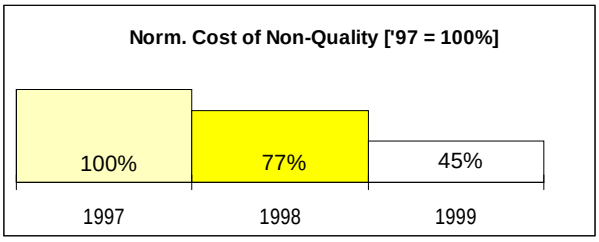


Fig. 7: Reduction of Normalized Cost of Non-Quality over past 3 Years by Combined Focus on Collocating Peer Reviews, Effective Process Coaching, and Introduction of Teamwork and Continuous Build

We will summarize those best practices related to improve validation activities, which we identified over the past years that clearly support global software development:

- Agreeing and communicating at project start the respective project targets, such as quality, milestones, content, or resource allocation. Similarly, at phase or increment start team targets are adjusted and communicated to facilitate effective internal team management.
- Making teams responsible for their results
- While having one project leader who is fully responsible to achieve project targets, assign her a project management team that represents the major cultures within the project.
- Defining at begin of projects which teams are involved and what they are going to do in which location. This includes a focus on allocation rules, such as scattering or collocation.

- Setting up a project homepage for each project that summarizes project content, progress metrics, planning information, and team-specific information.
- Collocate as much as possible teams to facilitate effective teamwork.
- Provide the necessary coaching on the job and friction-free by mixing different level of expertise.
- Provide the necessary tools and technology to manage workflow and workspaces around the world (e.g. CM, problem management, test environments).

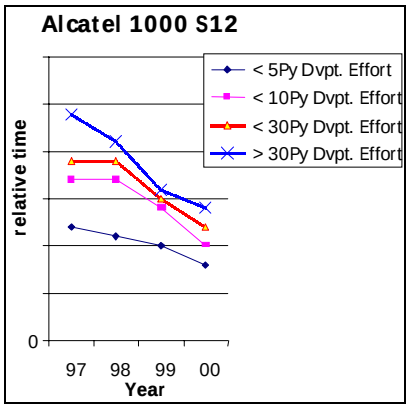


Fig. 8: Cycle Time Reduction is a Desired Side Effect of Earlier Defect Detection and Continuous Build

5. Conclusions

We had embarked on three specific changes that we first piloted in smaller settings to optimize the underlying process, and then rolled out over the past 3 years. All changes met the expectations in terms of cost versus benefit, although we cannot yet tell that they are fully implemented in all projects. The three underlying hypotheses could be validated in this case study covering more than 60 projects over a timeframe of 3 years:

- Collocating peer reviews improves both efficiency and effectiveness of defect detection and thus reduce cost of non-quality.
- Providing a certain level of coaching within the project reduces cost of non-quality.
- Reengineering the development process towards teamwork and continuous build allows to better managing globally distributed projects, and thus reduces cost of non-quality.

Literature

[1] Smith, R.P.: The Historical Roots of Concurrent Engineering Fundamentals. IEEE Transactions on Engineering Management, VOL 44, N0. 1, February 1997.

[2] McConnell, S.: Software Project Survival Guide. Microsoft Press. Redmont, USA, 1998.

[3] Karolak, D.W.: Global Software Development. IEEE Computer Society Press. Los Alamitos, USA, 1998.

[4] DeMarco, T. and T.Lister: Peopleware. 2nd ed. Dorset House, New York, 1999.

[5] Jones, C.: *Software Quality. Analysis and Guidelines for Success*. Thomson, Boston, USA, 1997

[6] Tagaki, Y. et al: Analysis of Review's Effectiveness Based on Software Metrics. *Proc. Int. Symp. on Software Reliability Engineering '95*. IEEE Comp. Soc. Press, Los Alamitos, Ca, USA, pp. 34-39, 1995.

[7] Humphrey, W.S.: *Introduction to the Personal Software Process*. Addison-Wesley, Reading, USA, 1997.

[8] Perpich, J.M., et al.: Anywhere, Anytime Code Inspections: Using the Web to remove Inspection Bottlenecks in Large-Scale Software Development. *Proc. Int. Conf. on Software Engineering*, IEEE Comp. Soc. Press, pp. 14-21, 1997.

[9] Wigle, G.B.: Practices of a Successful SEPG. *European SEPG Conference 1997*. Amsterdam, 1997. More in-depth coverage of most of the Boeing results in: G.G.Schulmeyer and J.I.McManus, Ed.: Handbook of Software Quality Assurance, 3. ed., Int. Thomson Computer Press, 1997.

[10] Grady, R.B.: *Successful Software Process Improvement*. Prentice Hall, Upper Saddle River, 1997.

[11] McGarry, F. et al: Measuring Impacts Individual Process Maturity Attributes Have on Software Products. Proc. 5. Int. Software Metrics Symposium. IEEE Comp. Soc. Press, pp. 52-60, 1998

[12] Karlsson, E.A. et al: Daily Build and Feature Development in Large Distributed Projects. Proc. ICSE 2000, pp. 649-658. IEEE Comp. Soc. Press. Los Alamitos, USA, 2000.

[13] Perry, D.E. et al: Parallel Changes in large Scale Software Development: An Observational Case Study. Proc. ICSE 1998, pp. 251-260. IEEE Comp. Soc. Press. Los Alamitos, USA, 1998.

[14] Royce, W.: Software Project Management. Addison-Wesley. Reading, USA, 1998.

[15] Ebert, C.: Technical Controlling in Software Development. *Int. Journal of Project Management*, .Vol. 17, No.1, pp. 17-28, Feb. 1999.

[16] Fenton, N. E. and S.L. Pfleeger: *Software Metrics: A Practical and Rigorous Approach*. Chapman & Hall, London, 1997.

[17] Ebert, C., T.Liedtke and E.Baisch: Improving Reliability of Large Software Systems. *Annals of Software Engineering*. Vol. 8, pp. 3-51,1999.