A Case Study in Root Cause Defect Analysis

Marek Leszak
Lucent Technologies
Optical Networking Group
Thurn-und-Taxis-Str. 10
90411 Nuernberg, Germany
+49 911 526-3382
mleszak@lucent.com

Dewayne E. Perry
Electrical and Computer Engineering
The University of Texas at Austin
Austin TX 78712
+1 512 471-2050
perry@ece.utexas.edu
www.ece.utexas.edu/~perry/

Dieter Stoll
Lucent Technologies
Optical Networking Group
Thurn-und-Taxis-Str. 10
90411 Nuernberg, Germany
+49 911 526-2624
dieterstoll@lucent.com

ABSTRACT
There are three interdependent factors that drive our software development processes: interval, quality and cost. As market pressures continue to demand new features ever more rapidly, the challenge is to meet those demands while increasing, or at least not sacrificing, quality. One advantage of defect prevention as an upstream quality improvement practice is the beneficial effect it can have on interval: higher quality early in the process results in fewer defects to be found and repaired in the later parts of the process, thus causing an indirect interval reduction.

We report a retrospective root cause defect analysis study of the defect Modification Requests (MRs) discovered while building, testing, and deploying a release of a transmission network element product. We subsequently introduced this analysis methodology into new development projects as an in-process measurement collection requirement for each major defect MR.

We present the experimental design of our case study discussing the novel approach we have taken to defect and root cause classification and the mechanisms we have used for randomly selecting the MRs to analyze and collecting the analyses via a web interface. We then present the results of our analyses of the MRs and describe the defects and root causes that we found, and delineate the countermeasures created to either prevent those defects and their root causes or detect them at the earliest possible point in the development process.

We conclude with lessons learned from the case study and resulting ongoing improvement activities.

KEYWORDS
root cause analysis, defect prevention, process improvement, quality assurance, modification management

1 Introduction

1.1 RCA project overview

The product in our study is a network element (NE) that is a flexibly configurable transmission system in an optical network, consisting of circuit packs, ASICs, software units, and a craft terminal. Total head count for this release was 180 people and the development project lasted for 19 months.

The NE software is developed in teams of 5-10 people. A typical (large) NE configuration can consist of many different hardware board types and up to 150 different software components. A software team is responsible for a collection of functionally related components, which altogether form an architectural unit, called ‘subsystem’ within this paper. The overall size of the NE software product is around 900 K-NCSL (non-commentary source lines), 51% being newly developed software.

This release has been a very important and critical one especially for the European market. Management concern for process improvement enabled several project retrospective activities, one of them being the root cause defect analysis (RCA) project. Several improvement projects towards, for example, better effort estimation, more efficient development, and predictable and higher quality (measured in number of defect MRs) have been initiated recently.

The team has been constituted as cross-functional team: members represent the NE software and hardware subsystems, as well as the independent integration & certification department and quality support group. We also have been supported by members of the Bell Laboratories software productions research department who brought extensive experiences from other similar studies (e.g. [12]) into our team. The mission of the RCA project was

- analyze sample defect MRs; find systematic root causes of defects
- analyze major customer-reported MRs during the maintenance release (so-called post-GA MRs, GA = general availability of the product)
- propose improvement actions, as input for current development projects, in order to reduce number of critical defects (severity 1&2 MRs) and to reduce rework cost, e.g. MR fix effort.

1.2 Limitations

This study has focused on defect analysis and determining the underlying root causes of those defects. There are more general perspectives that one might take (for example, what went well and what went wrong) but they are out of scope for this study. Moreover, correlations with other product

ICSE 2000, Limerick, Ireland

metrics have not been considered either, though it would be very interesting to analyze, for example, defect distributions in relation to test effort and the number of defect MRs per component (an associated analysis on the same project, studying quality related inter-dependent process and product measurements, is described in another paper [14].) We have focused our effort analysis on the reproduction, investigation and repair of the defect MRs, not on the retest effort that makes up a significant part of the rework effort.

Due to time pressures resulting from limitations on team member availability, we were not able to implement the formal analysis training and testing to establish a defined level of inter-rater reliability. However, there are two mitigating factors. First, the analysts looking at the MRs were members of the team putting together and reviewing the analysis instrument. This participation resulted in project relevant aspects being included in the questionnaire. We argue that this participation also resulted in a shared understanding of the components of the analysis. Second, we did implement informal consistency checks during the analysis process. Where inconsistencies were found, the analysts made subsequent corrections to the defect analyses. Thus, we are relatively confident in the consistent ratings of the resulting data.

1.3 Relation to other Work

Prior work on software faults has generally been reported on initial developments and focused on the software faults themselves rather than their underlying causes. The work of Endress [5], one of the earliest papers to analyze software faults, based his error classification on the primary activities of designing and implementing algorithms in an operating system. Thayer, Lipow and Nelson [15] provide an extensive categorization of faults based on several large projects. Schneidewind and Hoffman [13] categorize faults according to their occurrence in the development life-cycle. Ostrand and Weyuker [8] introduced a novel attributed categorization scheme delineating fault category, type, presence and use. Finally Basili and Perricone [1] provided an analysis of a medium scale system.

Our current study is based in part on earlier work by Perry and Evangelist [10,11] on interface faults which, while cognizant of the earlier fault categorization work listed above derived its list of interface faults from the fault data rather than using a pre-existing categorization. It is based also on the work by Perry and Stieg [12] --- a study of one of the releases of one of Lucent's very large switching systems. The fault categorization used in this study was based in part on the published categorizations and part on the experience of the developers in the reported project.

The work here is similar in intent but differs in implementation details. First, the defect categories are an improvement on the original categories in that they are separated into three classes for better human factors reasons, breaking up a large set of defects to be selected from into three reasonable sized sets of defects. Second, the effort estimation scales are uniform here where they were different there; further we added investigation effort here. Third, we expanded the root causes over what we had in the original study. This expansion was done in conjunction with the knowledgeable developers from the project and reflects both the current state of their project and their processes. Fourth, and our most novel aspect from a research point of view, we allowed for multiple root causes to be defined as well as for no root causes (i.e., a simple mistake with no underlying, lurking cause). And finally, rather than surveying the entire set of defect MRs, we have randomly selected a statistically significant sample from each of the subsystems for detailed analysis.

Card's "Learning from out Mistakes with Defect Causal Analysis" [3] provides a generic process which is congruent with the process followed here.

There are several strands of related work that are similar to ours but which did not have a direct influence on our approach. Chillarege, et. al. in their paper "Orthogonal Defect Classification" [4] focuses on defect types and defect triggers as a means of feedback to the development process. In spirit, this is much the same as our approach except that their method is used throughout the entire development process for immediate feedback where ours is essentially a retrospective and 'end of development' feedback process.

More recent work on defect and root cause analysis by Weider Yu et. al. [16,17] has followed a process similar to ours, but using different defect and root cause classification schemes. They have not focused explicitly on the effort related to the defects. However, the general shape of their results are similar to ours.

In a larger context, our methodology applies key aspects of the defect prevention process area of the CMM [9] and is similar to the one described in [5].

1.4 Organization of the Paper

We first discuss our methodology for the root cause analysis study in section 2. After presenting our defect classification scheme and its interesting new aspects in section 2.1, we discuss the defect selection procedure and our method for deriving countermeasures and actions for improvement in section 2.2 and 2.3. We then present our data analysis focusing first on how we prepared the data (3.1), then on the results of our general analyses of the defects, effort and root causes (3.2), and finally on the selection of critical root causes to be either prevented or found earlier (3.3). We conclude in section 4 with lessons learned, defining the most promising countermeasures and outlining resulting ongoing improvement activities, and describing our future plans.

2 RCA Methodology

2.1 MR Classification Scheme

We designed and implemented a web interface tool for RCA which provides an on-line representation of the MR classification scheme. The purpose of this tool and its associated form is to be accessible easily and to provide a simple means of data collection with appropriate online human factors support. The form has the appropriate census and MR data extracted from the MR database. Beyond that there are the following subsections to the analysis form:

MR classification: phase detection information
An important fact needed if we are to find defects earlier is when the defect was in fact found. The analyst may choose any one of 10 process phases as the phase in which the defect is found. In addition, the analyst may indicate why the defect was not found earlier.

MR classification: real defect classification
We divided the defects into three classes of defects: implementation, interface and external. Within each of these classes there are a set of appropriate defects types, depicted in table 1.

Table 1: Classification of defect types

Implementation	Interface	External
1: data design/usage	9: data design/usage	16: development environment
2: resource allocation/usage	10: functionality design/usage	17: test environment (tools/infrastructure)
3: exception handling	11: communication protocol	18: test environment (test cases/suites)
4: algorithm	12: process coordination	19: concurrent work (other releases)
5: functionality	13: unexpected interactions	20: previous (inherited from prev. release)
6: performance	14: change coordination	21: other
7: language pitfalls	15: other	
8: other		

The third factor in the defect classification is the defect nature: *incorrect, incomplete, other.* These were to be applied wherever they were appropriate. Their use was to reduce the number of explicit defect types. The other part of the defect classification focused on reclassifying the severity if that was necessary and reporting the amount of effort to reproduce the defect, to investigate it, and to repair it. We used a uniform scale for these effort reports: zero (meaning negligible effort), less than one day, one to five days (i.e., up to a week), five to twenty days (one to four weeks), and more than 20 days (more than a month).

MR classification: real defect location
The real defect location specified either a document identifier, or whether it was software or hardware.
'Real' location characterizes the fact that in real projects some defects are not fixed by correcting the 'real' error-causing component, but rather by a so-called 'work-around' somewhere else.

MR classification: defect triggers
Our approach to defect triggers (root causes) is rather novel. There are a number of dimensions that may in fact be at the root of each of the defects - that is, there may be several underlying causes rather than just one. We therefore provided a set of four root cause classes: phase-related, human-related, project-related, and review-related. These four classes span a four-dimensional root cause space, i.e. each individual defect root cause is uniquely characterized by specifying a value in each of the four root cause dimensions.

- the *phase triggers* are the standard development phases or documents: requirements, architecture, high level design, component spec/design, component implementation and load building. The phase related root causes could be qualified by the nature of the trigger: incorrect, incomplete, ambiguous, changed/revised/evolved, not aligned with customer needs, and not applicable (the default).
- The *human related triggers* are: change coordination, lack of domain knowledge, lack of system knowledge, lack of tools knowledge, lack of process knowledge, individual mistake, introduced with other repair, communications problem, missing awareness of need for documentation, and the default, not applicable. The "individual mistake" trigger is similar to the "Execution/ oversight" category of Yu et al [17]. It reflects the fact that sometimes you just make mistakes.
- The *project triggers* are: time pressure, management mistake, caused by other product, and not applicable (again the default).
- The *review triggers* are: no or incomplete review, not enough preparation, inadequate participation, and not applicable. (Note that a review is a formal moderator-controlled inspection of a document or code artifact. Our review process is described in [7].)
- *other triggers*: And, of course, there is the escape "other" allowing a different trigger to be specified.

MR classification: barrier analysis
Finally, the analyst may suggest measures for ensuring earlier defect detection and/or for preventing or avoiding the defect altogether.

2.2 MR Selection Procedure for Root Cause Analysis
As is usual in these kinds of studies, there is a problem with the magnitude of the amount of work that would have to be done to analyze all the relevant MRs to get a complete

picture of the defects and their causes. One way of reducing the amount of work is to randomly select a significant subset of the MRs to represent the whole set and carefully analyze that subset. Thus our selection procedure was as follows:

- define a set of MRs per subsystem (not per team), such that a subteam analyzes selected MRs per subsystem
- for each set of MRs per subsystem:
  * filter out inappropriate MRs
  * split into n MR subsets, such that each MR in a certain subset "Si" belong to exactly one subsystem
  * from the MR subset Si, select further a (typically much smaller) subset Si' such that

  1) one part (say 5 - 10) is selected manually by the sub-team analyzing Si', based on own selection criteria like "this MR hurt us a lot", long lifetime, overly complex problem solution, etc.

  2) the second part is a random sample of Si of order 40 MRs. In case Si is not `significantly larger' than 40, all MRs in Si are selected.

  Explicitly not excluded are severity 3 or 4 MRs (being not customer-visible), no-change MRs (false positives), and documentation MRs.

2.4 Methodology to derive Countermeasures & Improvement Actions

Our methodology entails four steps.

1. Selection of most significant MR subset

Critical for proposing countermeasures is to focus on a reasonable subset of all defects.

To arrive at such a set we apply a filtering mechanism. The filter cannot be defined beforehand, but is the outcome of a first analysis step. In this first step the goal is to identify selection criteria which filter those MRs that have together a significant part of rework effort and which to a large extend are found late in the development process. This way we arrive at a first subset of MRs that will be analyzed in more detail.

As second criterion for finding important defects we look into Post-GA MRs and search there for dominant contributions. If the defects that are found to be important differ in their characteristics from the first set, we get a second set of MRs for detailed study. These results provide the statistical input to the team for the selection of countermeasures that were suggested for each MR during its analysis.

2. Prioritization of Countermeasures

The RCA action team brainstormed proposals and weighted each proposal with overall consensus, according to three factors: *statistical weight as percentage of total effort*, *effectiveness of the suggested countermeasure* and *estimated cost of its implementation*, on a scale of 0 to 1. The product of the values is taken to get a first ranking of countermeasures. Finally this ranking is taken as basic, but not fully binding, input to select the countermeasures to tackle. Typically

- countermeasures with weight >0.5 should be selected
- the number of countermeasures should be 'small' e.g. <20, to remain manageable w.r.t. organizational changes

3. Definition of Improvement Actions

We conducted a two-day workshop with the analysts in which we focussed on the selected subset of defects and root causes to determine the appropriate actions to the defined and prioritized countermeasures. The results are summarized in section 4.1.

4. Deployment of Improvement Actions

Results were presented to our R&D Management Leadership Team and the development teams. Key improvements proposed have been approved and their implementation initiated, see details in section 4.1.

3 Data and Analysis

We first discuss the issue of preparing the data for analysis, then present our general analysis results and conclude with a discussion of how we selected the critical root causes as a focus for more detailed analysis and group discussion.

Before continuing we briefly clarify the terminology we use in our MR handling process. The following types of MRs are distinguished:

- initialization MR: used to add an artifact for the first time to the configuration management repository. Once an MR of this kind is closed, other MRs, of type enhancement or defect, may be created on the associated artifact.
- enhancement MR: used to add new functionality as part of a new release, i.e. as planned evolution of an existing system.
- defect MR: used to correct any fault in specification, design, or implementation. For each problem detected, a new MR is issued. If several artifacts are affected by the correction, this is handled by MR spawns. In this paper, we consider always the problem-related MRs, not their spawned sub-MRs.

3.1 Data Preparation

Data screening for the analysis proceeded in two steps.

1. Consistency checking. A comparison of results between different subsystems was conducted. The rationale being the elimination of misunderstandings of terms and verification of results. In particular the analysts have been asked to provide reasons for atypical behavior or to correct the classification if it was due to a misinterpretation of terms.
2. Preparation of a representative sample. We separated from those settled analysis data the MRs which have not been selected randomly. From the rest of the MRs we split off so-called no-analysis MRs which were erroneously classified as defects but in fact were initial MRs or enhancement MRs. The remaining MRs constitute a basis of 427 MRs belonging to 13 subsystems (Post-GA MRs counted as separate subsystem).

Each subsystem was represented with at least 8% of its MRs. Typically 20%-40% of the MRs were analyzed. Taking also the total number of MRs per subsystem we had a multiplicity

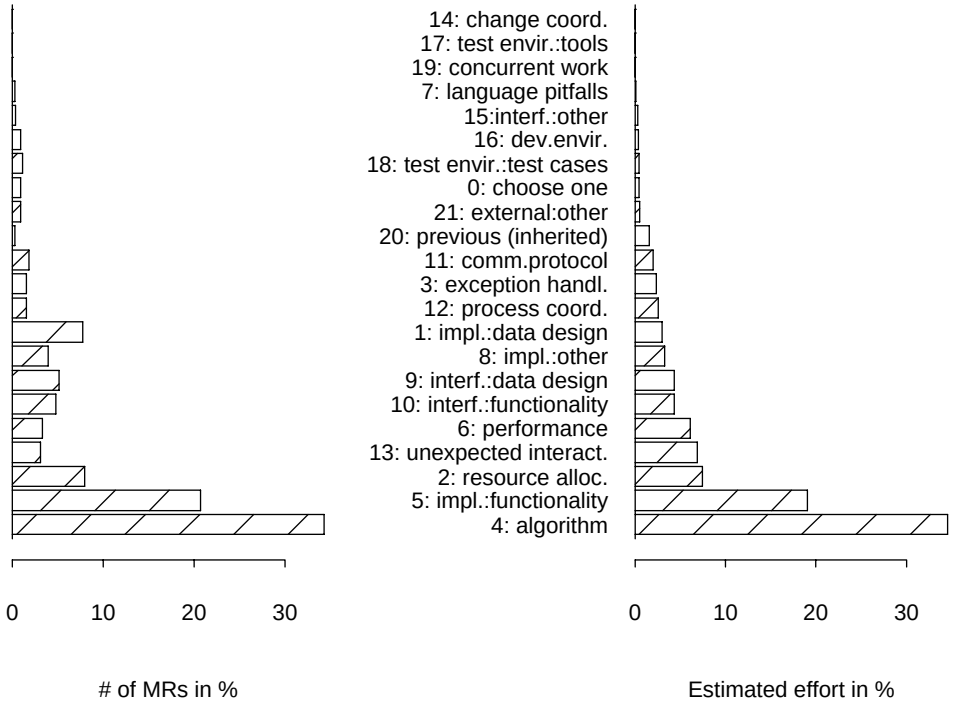

Figure 1: Distribution of defect types.

factor depending on the subsystem that each MR was weighted with during the analysis. E.g. each MR belonging to a sample that was represented with 20% of its MRs was counted as 5 MRs with all the characteristics the particular sample MR had like severity, defect type, etc. These weighted MRs were used throughout the following analysis, to extrapolate from the random sample to the total set of MRs.

Extra MRs that have been analyzed in addition to the random sample MRs were negligible in number except for one subsystem. The MRs of this subsystem have been included in the comparison between subsystems as separate group of MRs. The remaining extra MRs have not been considered in the statistical analysis but only in the manual evaluation of countermeasures.

3.2 General Analysis Results

Our statistical analysis is mainly descriptive in nature. Thus the bulk of evaluation consists in graphical or tabular aggregation of the results of our investigations and is done using the "S" software tool [2] in an exploratory way. Most results, depicted in figures 1 - 6, are presented using Pareto charts.

From the distribution shown in figure 1 we can derive several general results:

1. external defects are negligible, except for type "inherited from previous release".
2. interface defects consume about 25% of effort, the largest amount being caused by unexpected interactions, followed by functionality and data design.
3. implementation defects consume 75% of all effort and are dominated by defects of type algorithm and of type functionality. These defects will be studied in more detail below.

Interesting is the mismatch between number and effort which is most significant for defects of type previous, unexpected interaction, performance, and data design. This is reflected in the estimated effort (in person days) per MR of those defects. On the average we find 4.6 days for external, 6.2 for interface, 4.7 for implementation defects. Outliers are data design (at the low end) with 1.9 days and (at the high end) inherited defects 32.8, unexpected interactions 11.1 and performance defects 9.3.

As depicted in figure 2, system integration represents 50% of the distribution for defect detection, while the elimination of

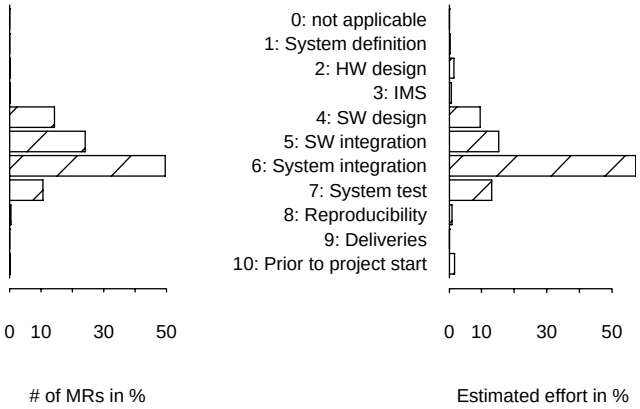

Figure 2: Distribution of phase where defect detected

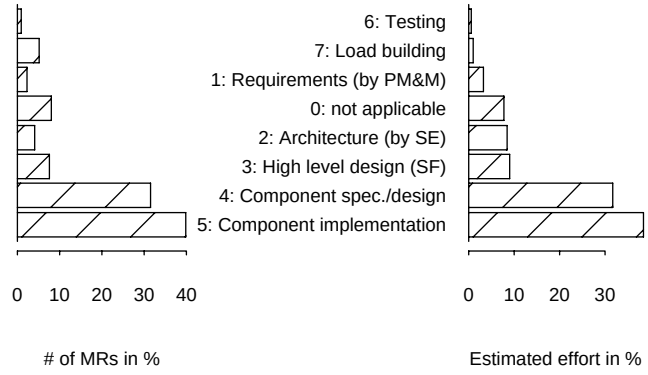

Figure 3:Distribution of phase where defect originated

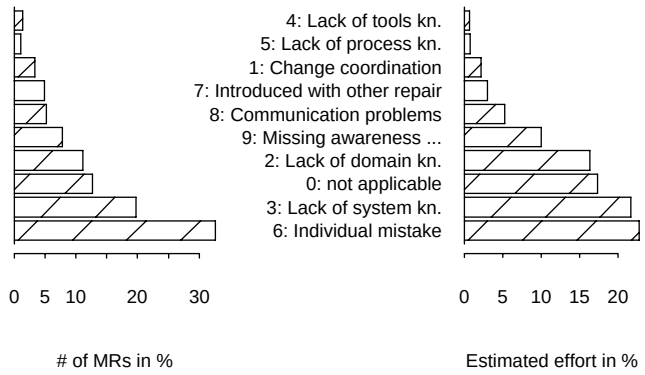

Figure 4: Distribution of human root causes

those defects taking up almost 60% of effort. As expected the data show a significant variation in effort per MR. Defects found in SW design and SW integration require less effort than those found in system integration and test. The estimated effort per MR (in person-days) as extracted from those figures are 3 days during SW design and integration, 6 days during system integration and system test and 9 days after delivery.

As shown in figure 3, defects are injected into the system predominantly (71%) within the component oriented phases of component specification, design and implementation. As an interesting outcome of the analysis we observe that defects from the requirements phase do not consume on the average tremendously more fix effort to be eliminated than others. Rather architectural mistakes turn out to require much more effort. It turns out that the required effort is for defects originating from requirements 6.5 days, from architecture 10 days, from high level design 5.8 days and from component spec./design 5 days.

An important study decision was to allow for several root causes to be specified during analysis of each MR. The intuition is that there may well be several factors contributing to the occurrence of a defect. Thus, in addition to phase, we have allowed human, project, and review root causes to be specified. These four-dimensional root cause classifications give indications as to what played a role in a defects occurrence. A useful way of looking at the data is to take the inverse percentages as an indication how many defects remain unaffected if the particular root cause were eliminated.

Viewing the data this way, we see from figure 4 that eliminating individual mistakes would have no effect on 67% of all defects, eliminating lack of system and domain knowledge would have no effect on 69%. If all communication related problems were to be solved, 87% of all defects remained unaffected.

For the selection of project root causes, while time pressure was chosen to be one affecting factor in 40% of all defects, mostly project root causes were not considered relevant.

Review-related root causes have been considered in 73% of all MRs and inadequate reviews have been specified as important in 48% of all MRs. Thus in 66% of all defects where review root causes have been considered at all, review deficiencies have been diagnosed.

3.3 Selection of critical Root Causes, to be improved or eliminated

As first step in figuring out dominant contributions, the distributions of MRs according to their defect type were studied with the result that defects of type algorithm, and of type functionality (defect class ‘implementation’) dominated by far all other defect types. ‘Functionality defect’ refers to missing or wrong functionality (w.r.t. requirements) in a design or code artifact whereas ‘algorithm defect’ refers to an inadequate (efficiency) or wrong (correctness) algorithmic realization In terms of numbers those defects represent 34% and 21%, respectively, of the defect population and 35% and 19% of the fix effort. The remaining defects are distributed over 16 other defect types.

Of particular interest are the Post-GA defects because they are typically detected by a customer. Of all Post-GA MRs 14% are classified as type algorithm and 68% of type functionality. This re-enforces our interest in those two defect types as deserving further detailed studies.

Specific Analysis of MRs with Defect Type ’Algorithm’ or ’Functionality’

MRs of defect class “implementation” and defect type “algorithm” or “functionality” have been correlated with the phase when they have been detected:

defect type	*SW design*	*SW integration*	*System integration*	*System test*
all MRs	14%	24%	49%	11%
algorithm	3%	14%	71%	12%
functionality	13%	23%	47%	14%
all other MRs	23%	32%	33%	9%

The table shows that of both defect types, more than 60% of the defects are detected late in the process, namely in system integration and system test. Since the average is 60% this also shows that the remaining 40% of all MRs is typically earlier detected. In particular, algorithm defects exceed the average finding in system integration by almost 50% and the finding of other MRs by 100%. Besides the pure numbers, late detection is a second strong argument for looking into the root causes of these particular defect types. To this end the correlations with various root causes were investigated.

The correlation with the phase when the defect was

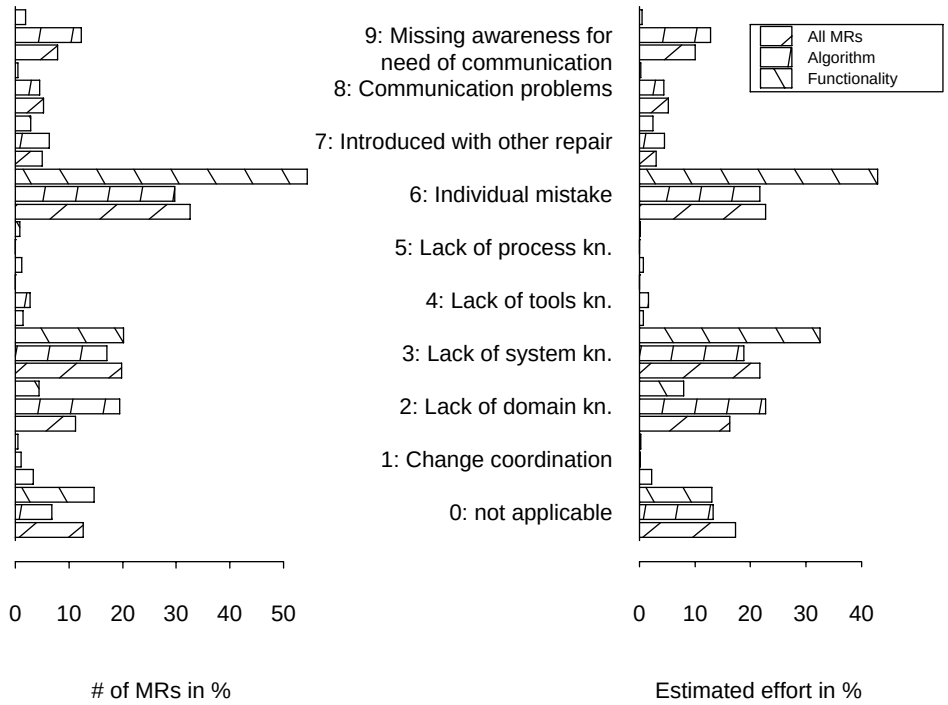

Figure 5: Human root causes found for defects of type algorithm and functionality, related to weighted mean number of MRs. [Three bars are displayed for each root cause. They show the respective results for all analyzed MRs and for the subsets of MRs classified as "algorithm" or "functionality" defects. The human root cause description starts off at the center-bar of the group of three bars.]

introduced

defect type	*Architecture*	*High level design*	*Comp. spec. / design*	*Component implementation*
all MRs	4%	8%	31%	40%
algorithm	0%	0%	48%	46%
functionality	4%	15%	25%	44%

shows no unexpected behavior. As one would assume, algorithm defects are introduced during design, specification and implementation, deviating significantly from the average distribution of defect defection. Functionality defects occur in rates close to the average behavior.

The correlation with human root causes (figure 5) shows significant contribution from lack of domain and system knowledge, and of individual mistakes. Not unexpectedly the contribution from lack of domain knowledge is smaller in case of functionality defects.

With respect to the correlation with review root causes we observe that of all MRs of defect type algorithm 75% are afflicted with inadequate reviews with a total of 84% reporting review root causes. For functionality the amounts are 30% inadequate reviews with a total of 67% reporting review root causes.

Contrasted with an average of 48% inadequate reviews on the basis of a reporting rate of 73%, we may infer that in particular algorithm defects escape earlier detection due to review deficiencies.

From all project root causes that have been available for selection, only time pressure constitutes a major part. On the average 40% of all defects are related to time pressure whereas this amount is 70% for defects of type algorithm and 17% for type functionality.

The analysis thus far indicates that important areas to look for improvements are reviews, domain and system knowledge, and test strategies (e.g. defined vs. achieved test coverage) because of late detection of defects. Means of prevention and earlier detection - provided by the MR analysts - are further evaluated manually to arrive at concrete counter measures.

Specific Analysis of Post-GA MRs

Although we intended to get a subset of MRs from a detailed analysis of Post-GA MRs, the outcome may be summarized very briefly. In fact we observe that defects of type algorithm and functionality again dominate by far all other defects. Thus having arrived at this subset already nothing must be added to cover defect causes that become visible to the customer. With respect to finding countermeasures this sample adds, however, the question why so many defects have been classified as "introduced by another repair".

4 Conclusion

We have described the origins of our study, delineated the process of our retrospective root cause analysis, and provided some of the analyses we performed on the data as illustrations and support for our subsequent improvement decisions. The primary novelty in our approach is the replacement of a one-dimensional root cause classification (that allows only for a single unique root cause to be selected) by a four-dimensional root cause space. The four dimensions are spanned by human, review, project, and life-cycle phase root cause classes. Unique root cause selection thus requires specification of a value in each of the four directions. This choice reflects the general richness that underlies most of the faults that occur in the building and evolution of large complex software systems. The rest of our contributions are incremental to the approach in [12].

4.1 Countermeasures and Improvement Actions

Our strategy was to find and deploy an effective set of

improvement actions, so that for future development projects

- the overall number of defect MRs is significantly reduced
- the defects are detected earlier in the lifecycle
- the mean effort to fix a defect is reduced
- the actions are really effective, i.e. focussing on systematic errors which result in a small number of process changes, promising at the same time to affect multiple defect root causes

In the countermeasure definition meeting the team decided to select 10 focus areas on the basis of the statistical data evaluation. The areas chosen from the effort distribution over the phase defects have been introduced, are component specification and design, component implementation and architecture. Note that this selection covers all algorithm defects and 73% functionality defects which were found to dominate all other defect types. Review root causes were selected as one single category. From the human root cause data the team selected 'individual mistake', 'lack of system knowledge' combined with 'lack of domain knowledge' as one area, and selected 'introduced with other repair' and 'communication problems' together with 'missing awareness for need of communication' as another area. The reason for selecting categories with small contributions was the insight that only human and review root causes can be addressed by countermeasures directly. Finally the team added the categories 'subcontracted software components' and 'project management', due to specific proposals for means of prevention found in the analysis data. Within these categories a set of countermeasures was distilled from the suggestions provided by the analysts during the analysis. The same procedure was followed to arrive at a set of measures for earlier detection of defects found in phases system integration, system test, and maintenance. All countermeasures thus assembled have been weighted to arrive at a ranking.

As ranking criteria the team decided to use three inputs

- Savings potential per RC area, represented by portion of total bugfix effort (this the maximum effort that can be saved by avoiding defects of this particular category),
- effectiveness per countermeasure, i.e. estimated percentage of MRs of this RC area which can maximally be influenced by a countermeasure,
- cost per countermeasure, i.e. estimated additional cost to implement the countermeasure. In order to combine the cost measured in 1000 US-$ with the other ranking criteria we mapped cost ranges onto factors in the interval [0,1] in the following way:
  [0 ,6[-> 1.0
  [6 ,30[-> 0.8
  [30 ,130[-> 0.6
  [130 ,600[-> 0.4
  [600, [-> 0.2

The following main countermeasures (CM) and associated improvement projects/activities (IP) have been defined, with increasing potential benefit in the ordered list below. All proposed IPs have been started and are ongoing. Note also that countermeasures are defined even in areas where SW development is already comparatively mature. Since our organization satisfies CMM level 3 criteria in several key process areas, we understand the improvement activities as one of the means that allow us to reach level 3 fully. Potential savings, effectiveness and cost (S/E/C) shown in brackets.

CM1: component specification & design documentation

- extend ensure required contents especially include compliance to non-functional and performance requirements (32% / 20% / 0.8)

IP: improve requirements management and systems engineering process w.r.t. traceability process, capturing non-functional SW requirements.

IP: introduce performance engineering (i.e. performance modeling, budgeting, and measurements).

CM2: component implementation

- increase usage of static & dynamic code analysis tools (coding standards checking, memory leak detection, code coverage analysis) (40% / 15% / 0.8)
- better unit tests (higher test coverage, complete test specification, systematic case selection, better host test environment, test bed, test automation) (40% / 35% / 0.4)

IP: code analysis tools and unit test tools usage as standard procedure in development process, fully integrated with load build environment. (Note: other product improvements on implementation level, e.g. cleanroom software engineering, sophisticated coding standard based on pre-/post-conditions etc. have not been tackled: this would mean a major paradigm shift - considered too risky for ongoing development of releases within the same product line in a highly competitive market.)

CM3: system & domain knowledge

- extend training offers and attendance on architecture and application domain, improve systems design skills (38% / 35% / 0.6)

IP: enhanced training program, assigned training coordinator

CM4: document & code reviews

- analyze review culture and performance, improve review process (66% / 30% / 0.8)

IP: include total effort for reviews in realistic planning, ensure sufficient review participation, increase awareness for review importance by e.g. better training, establish review process control: strict entry conditions, scheduling, timing. An review improvement project has been started in cooperation with Fraunhofer Institute for Experimental Software Engineering (IESE). First analysis results appeared in [7].

CM5: project management

- increase process compliance, i.e. completeness of exit conditions of systems and software development process (100% / 30% / 0.5)

IP: study correlation of component measurements (size, defects, complexity) and process compliance (see significant results in [14]).

IP: Implement database system for all project-related data,

supporting project tracking and reporting early warnings on process issues.

4.2 Lessons learned

1. Bugfix costs *do not grow* exponentially by phase, but rather linearly. (Note, however, that we don't consider re-testing effort which would have added a significant amount to total rework costs.)
2. The majority of defects *do not originate* in early phases
3. Within the same project, the defect attribute distribution per SW subsystem revealed large differences. (To our knowledge this has not been reported in other studies.)

The number of defects per subsystem found in system test ranged from 0% to 55%, the respective range is 5% to 95% for defects found in system test and system integration together. Although intriguing, we learned that these numbers cannot simply be attributed to differences in the quality of SW artifacts. To a significant extent they are due to architecture caused differences of subsystems. Thus, some have been targets of requirement changes, others could reuse existing functionality, others have higher operation profiles due to belonging to a lower architectural layer, etc.. Although interesting, the data available did not permit a detailed comparison along these lines which therefore is left for future studies. In particular it would be interesting to disentangle the architectural aspect from the team cultural one, e.g. how unit testing or reviews are done, because it would permit identification and promotion of best practices. An interesting side-result of the comparison is the fact that post-GA defects are to a much larger extent (30%) than on average (5%) caused by another repair which may be traced back to a project's 'end-game' pressure.

4. There is a significant influence of human factors on defect injection

Our study extended similar ones with regard to human factors for defect infection, and it made them more explicit. We recognized that this was in fact a particularly important attribute. It allowed us to separate randomly inserted defects due to unavoidable (human) mistakes and systematically introduced defects due to mismatches in required and available technical and/or soft skills. In software engineering work the "human factor" should receive higher focus.

5. RCA has a low and tolerable effort, relative to its apparent benefits

Two technical insights that we think are worthwhile mentioning, as well. In spite of starting the activity several months after project completion, and that the defects to be analyzed were on the average about a year old, the mean time for analysis was just 19 minutes. Thus such activities are even cheaper if they are performed during the project when the detailed knowledge about defects can be recalled easily. In-process RCA is a cost effective mean to identify deficiencies and improvement areas. When combined with statistical analysis, which of course is only possible in rather large development projects, conclusions about countermeasure selection can be made sound and put on a solid basis with regard to costs and potential benefits.

4.3 Current State and Future Plans

We plan to deploy RCA as continuous activity within all future development projects. Since RCA has been a post-mortem project activity so far, our RCA concept has been generalized to be applied as in-process RCA, i.e. during the development project. Concept extensions, already implemented in a successor release to the one under study in this paper, include

- MR selection criteria - which MRs need to be analyzed to find root causes
- adapt MR analysis input tool - interfaces to our configuration management tools is being built
- improved RCA analysis scheme, e.g. adding a test root cause area

As future aspects, we plan for

- installation of a permanent defect prevention / RCA team
- evaluation of cost-benefit of RCA, by comparing quality gain vs. RCA costs of different product releases before/ after RCA introduction. Gain will be measured by post-GA MRs, both absolute number and defect density until one year after GA.

In-process RCA is an important step towards full integration of the RCA methodology into the standard development process. It makes sense to require the additional RCA information from the bug fixer, prior to MR resolution. The RCA team should assess the individual proposals and provide feedback for organization change and process change at regular intervals, e.g. after each development phase or each major project milestone [3].

Since most engineers can then get involved in RCA activities, an extensive training on defect prevention and RCA should be performed, one essential step to make RCA a collaborative, continuous, and best-in-class improvement activity.

ACKNOWLEDGMENTS

We gratefully acknowledge the continuous commitment and support of our R&D Director Warren Koontz to the RCA project. The qualified contributions of the many RCA team members is also largely appreciated.

REFERENCES

1. V. R. Basili and B.T. Perricone: Software Errors and Complexity: An Empirical Investigation, CACM 27:1 (January 1984), 42-52.
2. R. A. Becker, J. M. Chambers and A.R. Wilks: The new S Language. Chapman and Hall, 1988
3. D. N. Card: Learning from our Mistakes with Defect Causal Analysis. IEEE Software 1/1998, p. 56-63
4. R. Chillarege et al: Orthogonal Defect Classification - A Concept for In-Process Measurements. IEEE Transactions on SW Engineering, vol. 18(11), 11/1992
5. A. Endress: An Analysis of Errors and Their Causes in Systems Programs. IEEE TSE, SE-1:2 (June 1975), 140-149.
6. C. Kaplan, R. Clark and V. Tang: Secrets of Software Quality - 40 Innovations from IBM. McGraw-Hill, 1995 (Defect Prevention Process in chapter 15)
7. O. Laitenberger, M. Leszak, D. Stoll and K. El-Amam: Causal Analysis of Review Success Factors in an Industrial Setting. 6th International Symposium on Software Metrics, West Palm Beach, Florida 11/1999
8. T. J. Ostrand and E.J. Weyuker: Collecting and Categorizing Software Error Data in an Industrial Environment, The Journal of Systems and Software, 4 (1984), 289-300.
9. M. C. Paulk, B. Curtis and M. B. Chrisis: Capability Maturity Model for Software (CMM) Version 1.1. SEI Report, CMU/SEI-93-TR, 1993 (RCA requirements in key process area “defect prevention”, CMM level5)
10. D. E. Perry and W. M. Evangelist: An Empirical Study of Software Interface Faults. Proc. of the International Symposium on New Directions in Computing, IEEE CS, August 1985, Trondheim Norway, 32-38.
11. D. E. Perry and M. Evangelist: An Empirical Study of Software Interface Faults - An Update. Proc. of the 20th Hawaii Int. Conf. on System Sciences, 1987, 113-126.
12. 1D. E. Perry and C. S. Stieg: Software Faults in a Large Real-Time System: A Case Study. 4th European SW Engineering Conf., Garmisch-Partenkirchen, 10/1993
13. N. F. Schneidewind and H.M. Hoffmann: An Experiment in Software Error Data Collection and Analysis. IEEE TSE, SE-5:3 (May 1979), 276-286.
14. D. Stoll, M. Leszak and T. Heck: Measuring Process and Product Characteristics of Software Components - A Case Study. 3rd Conf. on Quality Engineering in Software Technology (Conquest-99), Nuremberg, 27-29 Sept. 1999. ISBN3-00-004774-3
15. T. A. Thayer, M. Pipow, and E.C. Nelson: Software Reliability - a Study of Large Project Reality. TRW Series of Software Technology, Vol 2, North Holland, 1978
16. W.D. Yu, A. Barshefsky and S.T. Huang: An Empirical Study of Software Faults Preventable at a Personal Level in a Very Large Software Development Environment. Bell Labs Technical Journal, 2:3 (Summer 1997), 221-232
17. W.D. Yu: A Software Prevention Approach in Coding and Root Cause Analysis. Bell Labs Technical Journal, vol. 3, no. 2, April-June 1998, 3-21