periodica polytechnica

Electrical Engineering
54/1-2 (2010) 71–78
doi: 10.3311/pp.ee.2010-1-2.07
web: http://www.pp.bme.hu/ee

RESEARCH ARTICLE

Quality-driven optimized resource allocation

Ákos Szőke / *Orsolya* Dobán / *András* Pataricza

Received 2010-05-25

Abstract

The assurance of a good software product quality necessitates a managed software process. Periodic product evaluation (inspection and testing) should be executed during the development process in order to simultaneously guarantee the timeliness and quality aspects of the development workflow. A faithful prediction of the efforts needed forms the basis of a project management (PM) in order to perform a proper human resource allocation to the different development and QA activities. However, even robust resource demand and quality estimation tools, like COCOMO II and COQUALMO do not cover the timeliness point of view sufficiently due to their static nature. Correspondingly, continuous quality monitoring and quality driven supervisory control of the development process became vital aspects in PM. A well-established complementary approach uses the Weibull model to describe the dynamics of the development and QA process by a mathematical model based on the observations gained during the development process.

Supervisory PM control has to concentrate development and QA resources to eliminate quality bottlenecks, as different parts (modules) of the product under development may reveal different defect density levels. Nevertheless, traditional heuristic quality management is unable to perform optimal resource allocation in the case of complex target programs.

This paper presents a model-based quality-driven optimized resource allocation method. It combines the COQUALMO model as early quality predictor and empirical knowledge formulated by a Weibull model gained by the continuous monitoring of the QA process flow. An exact mathematical optimization technique is used for human resource, like tester allocation.

Keywords

software process · quality · resource allocation

Ákos Szőke

Multilogic Ltd., H-1023 Budapest, Frankel L. str. 45, Hungary
e-mail: aszoke@mit.bme.hu

Orsolya Dobán
András Pataricza

Department of Measurement and Information Systems, BME, H-1521 Budapest, Magyar tudósok krt. 2, Hungary

1 Introduction

The basic definition of quality demands that the target software conforms to the requirements and it is fit to be used from the customer's point of view. Solution providers face the demand of the efficiency of the design and implementation workflow from all the aspects of timeliness, costs and quality additionally to the conformance to requirements. As quality related problems necessitate additional iterations in the development cycle, continuous monitoring and supervisory control of quality assurance (QA) activities are critical tasks in software project management.

As a matter of fact, quality is a very subjective attribute of a SW system. However, "defects are not the only measure of quality, of course; but they are the most visible indicator of quality throughout a project" [6], so it is reasonable to measure quality by the occurrence of defects.

Management of QA related costs starts with the allocation of a quality budget [7] at the beginning of the project based on a forecast of the extent of QA related efforts. However, the actual efforts occurring during the progress of the development process frequently differ from the initial predictions. Accordingly controlled, adaptive quality management is needed to avoid cost or temporal penalties originating in additional iterations in the development workflow induced by quality problems. This paper presents a supervisory control policy providing an optimized resource allocation in adaptive way based on the monitoring of the QA process.

2 Background and definitions

SW quality has six views according to ISO/IEC 9126: functionality, reliability, usability, maintainability, efficiency, and portability. Many of quality factors used in formal quality models are considerably subjective [3]. Although, subjective ratings are better than not measuring at all, we focus on a much restrictive, but objective interpretation of quality, namely that of the lack of defects in the sense of conformance to the requirements. The IEEE standard [12] subdivides the notion of defects into the following categories: *failure* is a program behavior differing from user expectations; *fault* is an underlying cause that could

lead to failures, and *error* is a missing or incorrect human action that could inject faults into the SW.

2.1 Measuring quality

An objective PM approach needs a quantitative indicator of the quality in the form of a metrics. The most basic quality metrics is *defect density* expressing the ratio of defects measured in terms of KSLOC (kilo source lines of code) or pages (in documents):

$$\text{defect density} = \frac{\text{number of known defects}}{\text{product size}} \tag{1}$$

An important derived metrics is the *defect profile* which predicts the reliability of the product by describing the dynamics of defect removal during development, including document inspections, code inspections, and testing. It represents the efficiency of defect removal of the development corporation in each phase of development.

Different modules under development have frequently different defect densities originating in differences in their size, complexity and design personnel properties etc. As the overall quality of the system is dominated by the module of the worst quality, a fine granular PM solution is required to handle these differences. A *defect density profile* is a vector composed of the defect densities of the individual modules well-visualizable by means of radar charts (Fig. 1):

The principle of optimization of the use of QA resources is simply that more testing resources should be allocated to defect prone and/or mission critical modules to bring them to a uniform quality level of the system. In other words, the efforts dedicated to testing and repair should be harmonized with the expected number of defects in the individual modules under a constrained total quality budget.

2.2 Cost of quality

The two main types of software quality costs [8] are: *conformance*, and *nonconformance*. The cost of conformance includes prevention (do it right for the first time) costs and appraisal costs (quality improvement activities), while the cost of nonconformance includes all defect removal costs.

The ratio of defect removal costs during the different phases of the software lifecycle (design and code inspection, testing, and maintenance) follows typically the 1:10:100 rule. This emphasizes the importance of an optimized use of the quality budget: early defect removal is cheaper [4, 13].

The Constructive Quality Model (COQUALMO) [14] estimates the total number of defects remaining in a software system based on defect introduction (DI) and defect removal (DR) models.

The set of factors in COQUALMO are the same as in COCOMO extended by the factors related to the QA process. This model is capable to predict the expected numbers of defects at the end of requirement specification, design and coding phases separately.

3 Defect occurrence modeling

Defect occurrence models can be divided into two major categories [4]: *reliability models* predict end-product reliability by extrapolating the number of residual defects remaining undetected and uncorrected at the end of the testing process from the observed defect profile, while *quality management models* continuously monitor the entire workflow from the point of view of quality.

Model based defect occurrence prediction can provide the project manager with early warnings on quality related problems; therefore timely actions can be executed to reinforce the QA process.

3.1 Weibull model as a defect occurrence model

While COQUALMO is a proper approach for initial quality prediction and QA related cost allocation, it lacks the potential incorporating results gained from observations during the QA process and especially it is unable to model the dynamics of the testing process.

Many studies pointed out [7, 9] that a Weibull model is flexible enough to capture the dynamics of defect manifestations faithfully across a wide range of development situations. The Weibull model $\lambda(t)$ is formulated as:

$$\lambda(t) = N\alpha\beta t^{\alpha-1} \exp\left(-\beta t^{\alpha}\right) \tag{2}$$

where N represents the total number of events (e.g. defect occurrences), α determines the shape of the model (shape parameter), and β is the scale of the curve (scale parameter). In the practice, the parameters of the Weibull model are estimated by a best fit setting to the series of defect observation date clustered in a histogram-like form according to some user-defined temporal resolution (Fig. 2).

The area below the curve representing the entire workflow corresponds to the expected *total number of defects* introduced, and the area right to the instance of the termination of the QA activities to the predicted number of *residual defects* remaining undetected prior to issuing the software product.

A typical quality management model (Fig. 2a) characterizes the defect manifestation sequence over the entire QA lifecycle. The vertical line indicates the end of the production related QA activities. The area below the curve right to this line corresponds to the number of residual defects. The reliability model in Fig. 2b presents defect manifestations from the beginning of testing.

The Weibull formula can be splitted into two major parts having their own interpretation for QA activities. The monotonic increasing $\alpha\beta t^{\alpha-1}$ component dominates early phases when inspection and testing starts. The decreasing $\exp(-\beta t^{\alpha})$ factor corresponds to the fact that testing has to reveal less and less

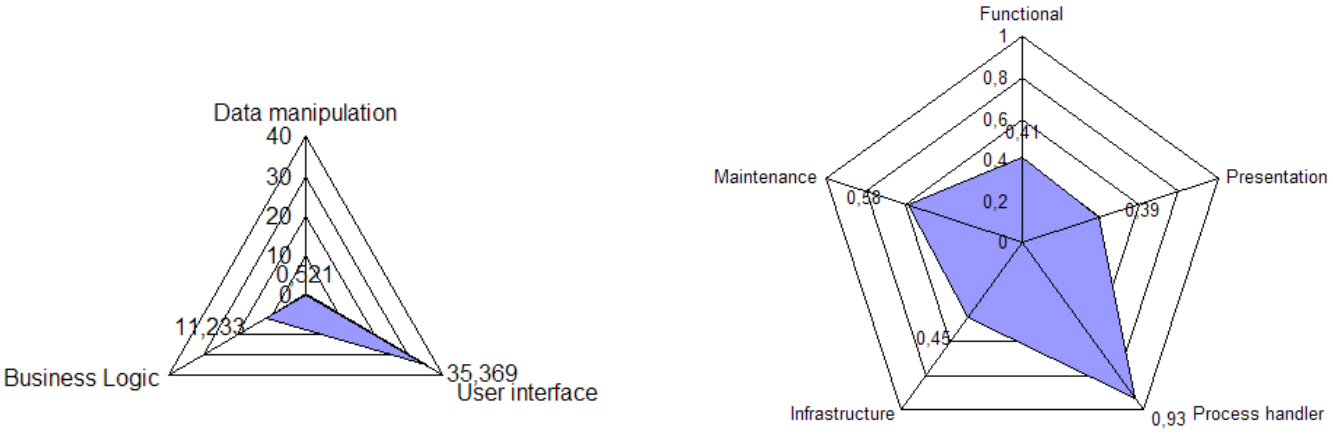
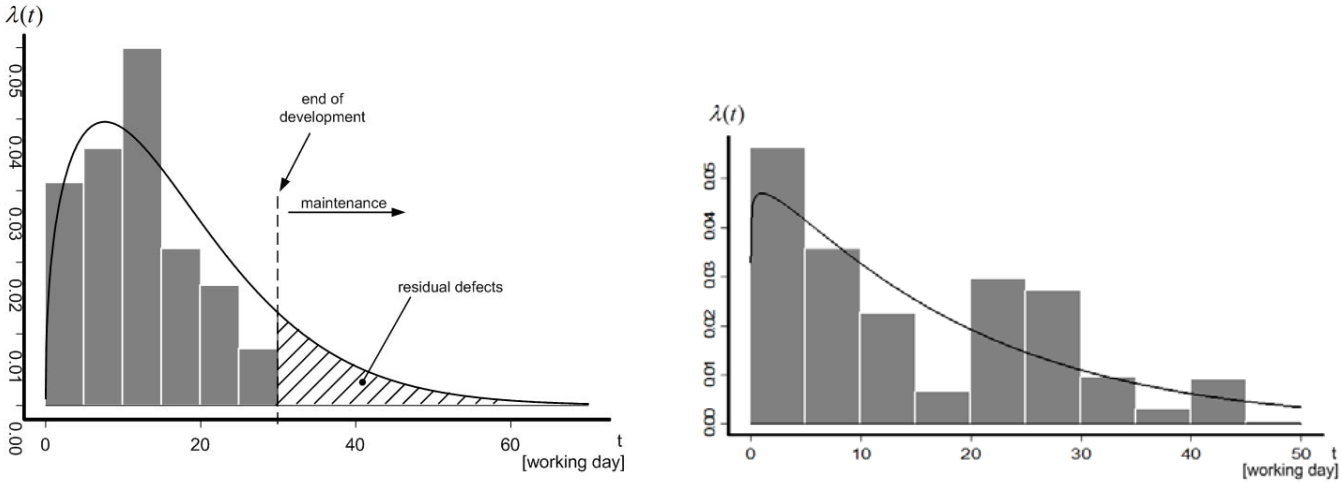


Fig. 1. Two different defect density profiles



a) Entire development lifecycle **b) Reliability model**

Fig. 2. Defect manifestation modelsa. Entire development lifecycle b. Reliability model

failures due to defect removal with the progress of the QA process.

The Rayleigh model is a special case of the Weibull model ($\alpha = 2$), which is used in quality management models. The necessary conditions to use this restricted model [7] are: unimodal staffing profile, life-cycle phases of a similar duration, reasonably complete reporting of defects, use of observable defects only in the statistics, and stable QA processes.

Defect manifestation is roughly proportional to the QA efforts (more efforts dedicated to QA shrink the time axis of the Rayleigh distribution and stretch the event rates). This observation will be utilized in the optimization of the QA resource allocation.

4 Case studies

The defect manifestation logs of two real-life projects were post-processed to check the faithfulness of the defect number estimators and correspondingly to test the feasibility of our approach. The first example was an e-business application (document registration system for a large enterprise) serving as a pilot application for calibrating the defect prediction measures vs. costs. The second one was a safety critical railway control system developed by means of upgrading the reliability of a traditional SCADA application by further tests. This later one was selected as a representative of the evolving class of applications integrating originally low or medium quality SW (like public domain SW) into critical applications after an exhaustive QA process.

Subsequently a "what-if" analysis was carried out in order to evaluate the practical importance of using mathematical optimization techniques in the QA related PM.

4.1 Commercial system

The pilot e-business application was developed by an internal team of 12 members supported by a consulting company during a period of 10 months from the project definition to its end.

The application was built over a typical three layered architecture composed of a presentation layer (Internet Explorer component embedded in a Visual Basic 6.0 host application), business logic (COM+ components developed in Visual Basic), and data

management layer (stored procedures and functions executed on a MS SQL 2000 Server database) implemented as three modules. The total size of the application was about 40kSLOC.

Our investigation was based on two major sets of data logs collected during the development process at the enterprise: the test result log database (based on Rational ClearQuest) containing all the defects detected during the development, and the COCOMO II and COQUALMO factors estimated by the local project manager.

The first step was the analysis of defect logs according to the three major development phases: requirements capture, design, and implementation indicating a moderately steep increase at the beginning and a slight decrease in the number of defects over the different phases of development.

The next step was a complete COQUALMO analysis checking the faithfulness of the estimation of the total number of faults. This emphasized the accuracy of the COQUALMO based predictions w.r.t. logged real-life data (the first three histograms in Fig. 3), as the two result set differ only by a linear factor. The difference can be significantly decreased after an appropriate calibration of the assignment of COQUALMO factors to the local process. The fourth histogram illustrates only the number of residual defects estimated by COQUALMO, since there was no available("?") project data in this phase.

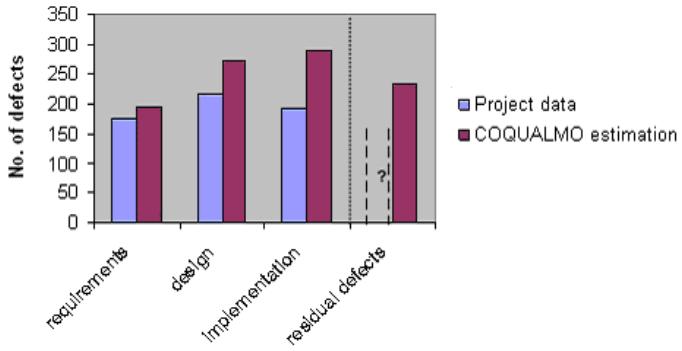


Fig. 3. Relationships between the Defect Log and the COQUALMO Estimators

Subsequently the model of the dynamics of the QA process was created by means of fitting a Weibull distribution to the observed defect manifestation log data (Fig. 2a).

Both COQUALMO and the Weibull model had shown a good correlation with the empirical data sets. COQUALMO can deliver a prediction on the total number of defects from the predefined product and project factors even prior of the start of the project, thus it can be used as a basic metrics in QA related resource allocation in the project planning phase. However, it can not reflex the temporal properties of the defect detection and removal process. The Weibull model delivers extrapolated estimators of the dynamics of defect occurrences based on the observations. It can be gradually refined as the QA process advances and delivers more and more log data.

4.2 Safety critical system

This pilot application was developed at a company specialized in embedded control system design. Their objective was to create a new product of a high required safety level (railway supervisory control system) based on a product developed originally for non-critical industrial applications. The company has decided on exhaustive re-inspection and re-testing of the existing application according to the domain specific standards related to safety critical system development (EN 5012x series) eliminating this way a completely new software development process.

The software consists of five major modules composed of totally more than 60 sub-modules by a project team of 20 persons. The applied software technology used UML based modeling tools extended by data flow models. Versioning- and a change tracking systems were applied during the project, from which the investigated defect log data were extracted. The size of the software was a total of 470 kSLOC written mainly in C, C++, Perl and Awk.

Our aim was the estimation of the correlation between the applied QA methods, efforts and the achieved safety level with respect to the logged defect removal data during the re-inspection and re-testing process. This experiment was a demonstrator of cases of software reuse, where additional QA activities aim to increase the reliability of the software.

Our first step was to create a COQUALMO based estimator of the number of defects removed during the quality improvement process and to compare it with the data logged during this additional QA phase. Two separate COQUALMO calculations were performed to predict the number of residual defects in the original product and in the enhanced one, respectively. The number of defects detected during re-inspection and re-testing was estimated as their difference (Fig. 7).

The majority of the factors influencing the quality (platform, personal, project quality etc.) were identical in these two cases. The difference between the two estimators originates in the different defect removal methods applied reflected as different COQUALMO defect removal factor assignments (Table 1). Only simple automated analysis (syntax-, type-checking in the static software code), complemented by the usual basic level tests, peer reviews and execution tests was used during the development of the original software. QA was supported by basic test coverage tools and ISO 9000 compliant test process management.

The additional QA activities producing the improved software product were carried out under the strict checking and verification rules prescribed by the standard by using, for instance:

1 *Sophisticated automated analysis tools,* including syntax and semantics analyzers, requirements and design consistency and traceability checkers, formal verification and validation tools etc.;

2 *Peer reviews* covering all important aspects according to the

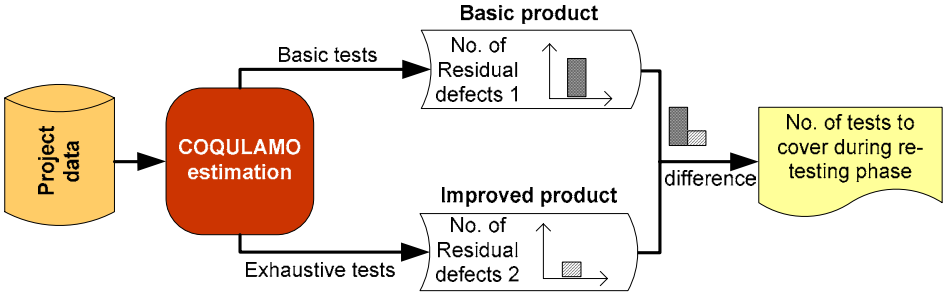


Fig. 4. COQUALMO estimations

standardized scenario;

3 *Execution testing* of 100% instruction-coverage (driven by Cantata).

Tab. 1. COQUALMO defect removal factor assignments

	Original SW development	Safety critical SW re-testing
Automated analysis	Low	Very High
Peer reviews	High	High
Execution testing	High	Extra High

COQUALMO estimated the number of residual defects as 763 in the original product (traditional QA), and as 263 in the improved one (strict QA), respectively. Accordingly, the detection of 500 defects was predicted for the re-testing phase. The project test log database contained a total of 399 defects detected during re-testing, thus the relative error of the prediction was by about 20%.

While the results in the current experiment were better than expected from the point of view of the accuracy of the estimates with respect to the real log data, its general use requires some caution. One of the potential dangers in this approach is that the difference of two uncertain large numbers may have a large relative error. Overestimation of the difference (a large number of residual faults) leads to redundant test resource allocation, while underestimation may result in underpowering of the QA process resulting in a large number of residual faults under a deadline constraint.

Uncertainty in COQUALMO has two roots. The first one is that the COQUALMO quality estimator formula itself is only a best fitting extrapolation based on empirical data from a large set of historical pilot projects and expert predictions used for the actual project.

The second one is the so-called calibration problem, as a potential misquantification of COQUALMO factors by a misjudgment of the project manager potentially resulting in a large deviation from the real data observed later.

A simple form of sensitivity analysis was performed for the most critical three factors (level of automated analysis, execution testing and peer reviews) in order to assess the impact of a miscalibration of the estimation by a wrong categorization (Fig. 5) of the factors.

We estimated the maximal impact of the deviation of each individual factor by one category up or down onto the cost of the re-testing. The set of potential neighboring estimates includes $2^3 = 8$ for each of the number of defects in the original and improved software. The largest relative error out of these 64 combinations was by about 60% clearly indicating the need of a continuous monitoring of the number of defects and not to rely only on initial estimates.

Please note, that relative error related to the estimation of the number additional defects may drastically increase, if only a minor increase in quality is indented, thus the initial and objective number of defects are near.

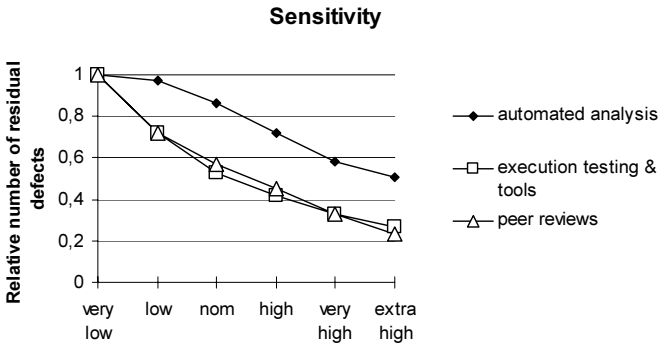


Fig. 5. Sensitivity analysis

The situation is essentially better, if the number of defects in the original software to be improved is already known from the logs of the initial QA process, thus only the number of defects in the target product has to be estimated. In our case, such data were unavailable due to historic reasons (the roots of the original software date back as long as a decade, parts of it were developed already prior to the introduction of the ISO9000 QA system at the enterprise). Moreover the lack of testing related data logs is typical in reusing third party (including public domain) software, as well.

5 Optimized resource allocation

Defect manifestation rates are roughly proportional to QA efforts, like project staffing. The underlying approximation is that the allocation of more resources (e.g. testing personnel) working in parallel results in a rate of defects removal proportional to

the amount of resources allocated.

The extraction of a Rayleigh model of defect occurrences from the QA logs during the development process helps allocating the proper amount of resources by predicting number of the residual defects at the end of the development process according to the current resource allocation scheme and development deadline.

Frequently several modules have to be developed differing in complexity and size, in implementation technologies (and/or even in their required reliability levels). Each module may have its specific defect density value and the overall quality of the system is dominated by the module of the worst quality.

If the estimation of residual defects happens for each module individually, defect prone ones can be identified. Adaptive quality management may re-allocate QA resources from non quality-critical modules to defect prone ones in order to reach a homogenously good overall quality (same residual defect density) in the system.

The process of resource allocation can be further refined by introducing defect and resource types into the clustering of the records in the historical defect databases and the subsequent calculations. For instance, defects can be classified according to their severity and source into groups like "minor specification deviation" (e.g. usability defects), "minor programming defect" (e.g. bad input checking) or "major programming/design defect" (e.g. improper result of a computation). Similarly, the removal of each defect type may require the involvement of a different kind of staff (e.g. analyst, designer, and programmer) and time.

We have the feeling, that this is the maximal granularity of cost factor allocation which delivers meaningful results in optimization, given the uncertainty of the different estimators.

5.1 Optimization of resource allocation

The availability of a mathematic model describing the QA process facilitates an optimization of the QA process from several aspects. By summing up the ideas described above, this mathematical model is composed of the following factors:

- Variables representing the expected number of defects for each individual module (for a given time instance this set consists of three values: the empirical constant for already detected defects and the two estimators of defects detected until the end of the project and residual ones) majorized jointly by the expected total number of defects in the module;
- Variables describing the amount of resources for QA activities to be allocated to the individual modules bounded by the amount of resources available (eventually these entities are decomposed into resource type);
- Cost factors contain the required amount of resources (e.g. analyst, designer, programmer) to remove defects (cost calculations are based on statistical investigation).

A variety of alternate optimization goals can be formulated by selecting an appropriate objective function.

One category aims at quality maximization constrained by the resources available and the cost budget, for instance by (i) minimizing the total number of residual defects in the system (representing a measure for the overall quality in a system composed of tightly integrated modules); or (ii) minimizing the maximal residual defect density over the modules (balanced quality of alternatively used modules). The other category aims at cost minimization constrained by the resources available and the required minimal quality level in the terms of quality indicators formulated in a similar fashion as the objective function above.

5.2 Results of the method

In our pilot experiment the maximization of the number of defects removed from the entire system was selected as objective function in order to balance the different defect densities and therefore maximize the overall quality of the entire system. The amount of resources per type and the QA cost budget were selected as constraints. This formulation led to a simple linear optimization problem producing the following results:

1 Adequate resource allocation was made by incorporating the costs of defect removal (testing + correction) into the factors of the objective function.

2 Optimization balanced the different defect densities of the modules; therefore the defect density profile is roughly a regular polygon (Fig. 6) indicating a homogenous defect density (quality) in the system.

3 Optimization revealed an a priori improper resource allocation in the case of the commercial system: some modules were over-tested. The optimal resource allocation in the what-if experiment would use by 20.5 % fewer resources, as indicated in the logs. This fact can be interpreted as 20.5 % of costs actually spent could have been saved or reallocated to defect removal in the most critical modules in order to decrease later maintenance costs.

Naturally, more sophisticated cost functions can be realized, for example, which functions deal with defect severity, priority etc., but our presented solution does not deal with these parameters.

6 Architecture of the resource allocation optimization method

The architecture of the resource allocation optimization method is presented in Fig. 7:

Upon starting the project:

1 COCOMO and COQUALMO models are built based upon the product, requirements and the knowledge on the workflow delivering among others an estimate of the number of total and residual defects in each module.

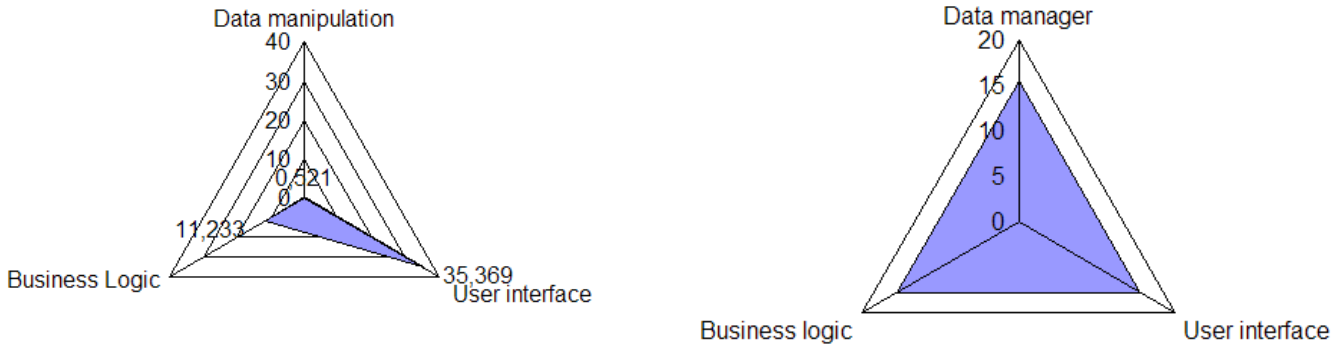


Fig. 6. Defect density profiles: before (left) and after (right) optimization

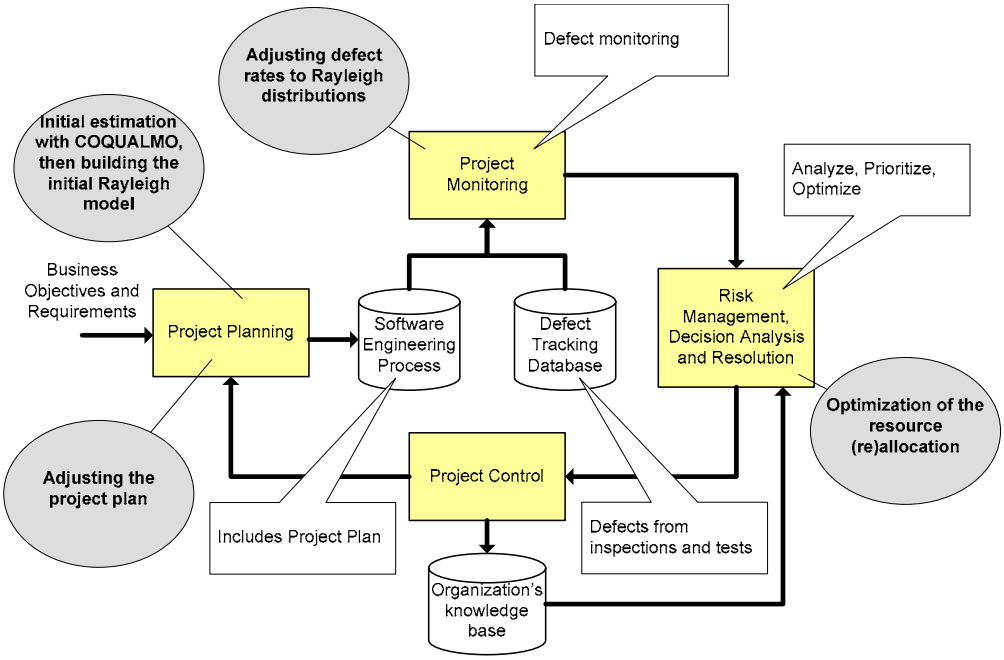


Fig. 7. Architecture of the Resource Allocation Optimization Method

2 Initial quality budget and initial resource allocation is performed according to an initial Rayleigh model derived from the result of the COQUALMO-based estimation. The parameters of this Rayleigh distribution are estimated in such a way, that the number $N$of defects is taken as the COQUALMO estimator, while the shape parameter β describing the dynamics of defect manifestations is reused from former projects carried out under similar circumstances. The main underlying assumption is that projects characterized by similar COCOMO factors share a similar dynamics at an organization.

3 The project management monitors and analyzes the Rayleigh model periodically updated from the defect tracking database. This model can provide early warnings on evolving quality problems.

4 The project management prioritizes and optimizes the quality budget with the help of optimized resource reallocation thus takes corrective actions (project control).

This algorithm iterates the steps from 3 to 4 during the project at stated intervals.

7 Conclusion and future work

Customers are most likely to buy the first product on the market that offers all the features they want, at a reasonably low price and promising a high end-product and maintenance quality. Apart from these expectations, software development organizations have to improve their project and in-process activities. The statistical quality model presented here allows the formulation of a mathematical model reflecting the quality of the entire software in a given phase of development.

Gradual refinement of the model by using in progress empirical data may compensate the high uncertainty in initial predictions of defect numbers and correspondingly resource allocation.

This paradigm combined with exact mathematical optimization techniques allows optimal resource (re)allocation to get the highest quality level keeping the given cost and schedule related constraints.

References

1 **Object Management Group**, *Software Process Engineering Metamodel Specification*, OMG, 2002.

2 **Paulk M, Curtis B, Chrissis M, Weber C**, *Capability Maturity Model for Software*, Tech. Report 1993 CMU/SEI.

3 **Fenton N E, Pfleeger S L**, *Software Metrics 2nd Edition*, PWS Publishing Company, 1997.

4 **Kan S H**, *Metrics and Models in Software Quality Engineering*, Addison-Wesley, 2002.

5 **Luenberger D G**, *Linear and Nonlinear Programming*, Addison-Wesley, 1989.

6 **Yourdon E**, *Software Metrics*, Application Development Strategies (newsletter) (November 1994), 16.

7 **Card D N**, *Managing Software Quality with Defects*, Proceedings of the 26^{th} Annual Computer Software and Application Conference (COMPSAC'02), 2002 IEEE Computer Society, DOI 10.1109/CMPSAC.2002.1045048, (to appear in print).

8 **Slaughter S A, Harter D E, Krishnan M S**, *Evaluating the Cost of Software Quality*, Communication of the ACM **41** (August 1998), no. 8, 67-73, DOI 10.1145/280324.280335.

9 **Li P L, Shaw M, Herbsleb J, Ray B, Santhanam P**, *Empirical Evaluation of Defect Projection Models for Widely-deployed Production Software Systems*, SIGSOFT 2004/FSE-12, posted on November, 2004, DOI 10.1145/1029894.1029930, (to appear in print).

10 *ISO/IEC Standard 9126. Information Technology - Software Quality, Part 1*, 1995.

11 **Putnam L, Myers W**, *Measures for Excellence*, 1992. Yourdon Press Computing Series.

12 *IEEE Standard Glossary of Software Engineering Terminology, IEEE STD 610.12-1990*.

13 **Remus H**, *Integrated Software Validation in the View of Inspections/Review*, Proceedings of the Symposium on Software Validation, Darmstadt, Germany, Amsterdam: North Holland, 1983, pp. 57–64.

14 **Center for Software Engineering**, *Constructive Cost Model (COCOMO)and Constructive Quality Model (COQUALMO)*, available at `URL: http://sunset.usc.edu`.

15 **R Development Core Team**, *R Language Definition*, 2004, available at `URL:http://www.r-project.org`.