Sana Abiad, Ramzi A. Haraty, Nashat Mansour, "Software metrics for small database applications", Proceedings of the 2000 ACM symposium on Applied Computing, Volume 2, 2000, pp. 866-870.

Article Abstract:

Known software metrics for estimating complexity and effort are usually based on lines of code or the program's flowgraph. Such metrics are suitable for large-scale procedural or object-oriented software applications. In this work, we propose a new complexity metric, called DataBase Points (DBP), that is suitable for small-scale relational database business applications developed in the MS-ACCESS (ACCESS is a trademark of Microsoft Corporation) or similar environments. DBP is constructed from components that are derived from typical ACCESS design. Further, DBP is used to estimate the effort needed to develop such software. The results of applying this new metric to a number of applications show that it is promising and that it captures the complexity features of small database applications.

Reviewer Abstract:

This paper proposes a new complexity metric DataBasePoints (DBP), that is suitable for small-scale relational database business applications developed in the MS-ACCESS or similar environments. DBP estimates the effort needed to develop such software using components from design.

The traditional metrics for measuring complexity and effort are not suitable for small database applications because line of code and program's control flowgraph are not as significant as they are in procedural or object oriented programs. Therefore traditional like cyclomatic complexity, function points, COCOMO, regression based models are not suitable.

Components used in the model are: Tables, Relationships between tables, Transaction, Forms, Reports.

For tables the important factors for classification are:

Number of Fields per Table, Properties per Field, Table properties

For Relationships t the important factors for classification are:

Type of relationship, Properties of relationship.

For Transactions t the important factors for classification are:

Type of transactions, Properties of transactions

Using the components an aggregate complexity metric is calculated.

Effort = DBP * (0.2 + 0.01 * å F i)

where F i is the weight given to each of the adjusting factors (i = 1, 2 ..... 8).

The DBP calculation is analogous to Function Points. In the equation of DBP Adjusting Factors also play a role.

For empirical results of the metric six small applications developed by senior university students are used.

This metric is mainly for project managers and developers. They can use this metric to estimate the complexity of the project and adjust resources accordingly.

Reviewer: Orcun Colak

Charu C. Aggarwal, Joel L. Wolf, Philip S. Yu, "Maximum Factor Queue Length Batching Scheme for Video-on-Demand Systems", IEEE Transactions on Computers, Vol. 50, No. 2, February 2001, p. 97

Article Abstract:

In a video-on-demand environment, batching of video requests is often used to reduce I/O demand and improve throughput. Since viewers may defect if they experience long waits, a good video scheduling policy needs to consider not only the batch size but also the viewer defection probabilities and wait times. Two conventional scheduling policies for batching are the first-come-first-served FCFS) policy, which schedules the video with the longest waiting request, and the maximum queue length (MQL) policy, which selects the video with the maximum number of waiting requests. Neither of these policies leads to entirely satisfactory results. MQL tends to be too aggressive in scheduling popular videos by considering only the queue length to maximize batch size, while FCFS has the opposite effect by completely ignoring the queue length and focusing on arrival time to reduce defections. In this paper, we introduce the notion of factored queue length and propose a batching policy that schedules the video with the maximum factored queue length. We refer to this as the MFQL policy. The factored queue length is obtained by weighting each video queue length with a factor which is biased against the more popular videos. An optimization problem is formulated to solve for the best weighting factors for the various videos. We also consider MFQL implementation issues. A simulation is developed to compare the proposed MFQL variants with FCFS and MQL. Our study shows that MFQL yields excellent empirical results in terms of standard performance measures such as average latency time, defection rates, and fairness.

Reviewer Abstract:

This study attempts to find a better video scheduling discipline for Video On Demand (VOD) systems. A VOD system allows users to request and view videos. The user shall view a requested video after an interval of time referred to as latency. In a batching VOD system, the server does not immediately send the video upon receipt of a request. Instead, the fulfillment of the request is delayed by a certain interval of time called the batching interval. At the expiry of the batching interval, the server must use a scheduling discipline to decide which video to send on the available stream. The major scheduling disciplines explored by the study are the First-Come First-Serve (FCFS), Maximum Queue Length (MQL), and the one invented by the study. The scheduling discipline invented is called Maximum Factor Queue Length (MFQL). The attributes of interest are the defection probability, the average latency time, and the unfairness.

The defection probability refers to the probability that a given user will decide not to view a requested video as a result of an intolerable latency. The average latency time is the average time between the request made by a user of the VOD system and the viewing of the first frame of video. Finally, the unfairness metric is a measure of the relative difference in latency experienced by various videos (as a result of varying popularity of these videos). This study should be of value to companies involved in VOD services and to researchers. The study employs simulation of various scheduling disciplines to compare other disciplines with the MFQL discipline. The study takes the approach of simplifying the optimization problem by assuming no defections (i.e. a zero defection probability for all videos). Defection is introduced and empirical results for practical MFQL solutions are given.

Reviewer: Suresh Aryangat

Alberto Avritzer, Elaine J. Weyuker, "Metrics to Assess the Likelihood of Project Success Based on Architecture Reviews", International Journal of Empirical Software Engineering, Volume 4, Number 3, September 1999, pp. 199-215.

Article Abstract:

Architecture audits are performed very early in the software development lifecycle, typically before low level design or code implementation has begun. An empirical study was performed to assess metrics developed to predict the likelihood of risk of failure of a project. The study used data collected during 50 architecture audits performed over a period of two years for large industrial telecommunications systems. The purpose of such a predictor was to identify at a very early stage, projects that were likely to be at high risk of failure. This would enable the project to take corrective action before significant resources had been expended using a problematic architecture. Detailed information about seven of the 50 projects is presented, and a discussion of how the proposed metric rated each of these projects is presented. A comparison is made of the metric's evaluation and the assessment of the project made by reviewers during the review process.

Reviewer Abstract:

The object of study of this paper is a risk prediction metric to assess the likelihood of project success based on architecture reviews. The attributes considered to define the metric are number of problems encountered during previous software architecture reviews in different areas of the project and how severe they were. The project areas included are: project management, requirements, performance, OA&M, design, technology, network, security, testing, other. The severity indicates how important it is to address the problem. The severity classifications from most severe to least severe are: project-affecting issues, critical, major, minor, recommendations and observations. The objective of the study is to develop a process that would allow the project to determine at a very early stage of system development, whether or not the project is at high risk of failure. The results of the study should be of value to the organization and project managers as they can assess the risks of a project at an earlier stage and either mitigate the risks before it is too late or cancel the project. The study was made using data collected during 50 architecture and discovery reviews that were performed over a period of two years for many

different projects developed by a telecommunications company and some questionnaire data. The paper shows the path followed to define a risk metric. The paper provides a detailed description of how to get the metric based on a questionnaire. The whole process is described in detail and the questionnaire is provided. This allows other organizations to use the process to assess the risk of their projects.

Reviewer: Patricia Costa

Basili, V.R.; Lanubile, F.; Shull, F., "Investigating maintenance processes in a framework-based environment", Proceedings of the International Conference Software Maintenance, November 1998, pp. 256-264.

Article Abstract:

The empirical study described in this paper focuses on the effectiveness of maintenance processes in an environment in which a repository of potential sources of reuse exists, e.g. a context in which applications are built using an object-oriented framework. Such a repository might contain current and previous releases of the system under maintenance, as well as other applications that are built on a similar structure or contain similar functionality. This paper presents an observational study of 15 student projects in framework-based environment. We used a mix of qualitative and quantitative methods to identify and evaluate the effectiveness of the maintenance processes.

Reviewer Abstract:

Entity = process of adaptive maintenance in an environment which is suitable for reuse
Attributes = what methods worked best in final product
purpose - understanding
stakeholder - developers
context - student projects (CMSC 435)
background - ET++ widget set with example applications, only one easily adapted

This paper describes the particulars of the process for creating an application in the presence of easily-reusable software components. The same application was developed by several teams, where some teams developed their project as adaptive maintenance of an existing application.

Different groups of teams used different processes in completing their project, with different degrees of success. Success was measured as completeness of functionality as originally assigned. The choice of certain processes seemed to correlate moderately with the success of the final application.

The purpose of this study was to understand the effect that framework availability had on the development of applications. Although the sample size of teams was small, the authors hypothesized that teams adapting existing applications to achieve similar goals (the "strictly adaptive" approach) have a higher success rate. However, if the components contained in the framework are dissimilar from development goals, then starting from "scratch" (the "ad-hoc" approach) is more appropriate.

The beneficiaries of this study are the development teams, who must choose what process to follow when developing or maintaing software. Based on the kinds of components in their repository and the goals of each new project, they may choose to modify existing high-level components or incorporate lower-level components into a new architecture.

This study took place in an upper-level software engineering class consisting of graduate and undergraduate students. They were not software professionals, but most had some knowledge of C++, the language used in development. None of them had knowledge of the framework before developing their projects.

The development language was C++, and the framework used was called ET++, which included many low-level components, a few simple example demonstrations, and several example applications. One example application was most like the students' projects, and the teams that followed the "strictly adaptive" approach chose it as the starting point for their software.

Reviewer Name: David C Rafkind

A. Bianchi, A., F. Lanubile, and G. Visaggio, "A controlled experiment to assess the effectiveness of inspection meetings", 7th International Software Metrics Symposium, 2001, pp. 42-50

Article Abstract:

Reviewer Abstract:

The overall objective of the paper is to investigate the contribution of the meetings in the software inspection process model. The context of the study is a controlled experiment which was conducted as part of a two-semester software-engineering course at a university, simulated in a classroom environment, with more than one hundred undergraduates, the preparation and meeting steps of an inspection process for requirements documents. The entity being studied is the inspection meetings in two runs of the experiment: real teams vs. nominal teams.

While a real team reports defects during a face-to-face meeting, defects are attributed to a nominal team by merging the preparation logs of the team individuals. The purpose of the study is to evaluate the effectiveness of inspection meetings, the differences between real and nominal teams, and the additional insight into the reasons behind the differences. The study should be of value to project managers as well as researchers who is interested in the inspection structure, effectiveness, cost, and/or interval. The important results of this study include that nominal teams are more efficient than real teams in software inspection because the nominal teams outperformed real teams and there were more meeting losses than meeting gains due to negative acknowledgement in the group dynamics in a real team.

The results also indicate that most of the meeting losses were defects found by only one individual in the inspection team.

Reviewer: Zhijian Pan

John Bible, Gregg Rothermel and David S. Rosenblum, "A comparative study of coarse- and fine-grained safe regression test-selection techniques", ACM Transactions on Software Engineering and Methodology, Volume 10, Issue 2 (2001), Pages 149-183.

Article Abstract:
Regression test-selection techniques reduce the cost of regression testing by selecting a subset of an existing test suite to use in retesting a modified program. Over the past two decades, numerous regression test-selection techniques have been described in the literature. Initial empirical studies of some of these techniques have suggested that they can indeed benefit testers, but so far, few studies have empirically compared different techniques. In this paper, we present the results of a comparative empirical study of two safe regression test-selection techniques. The techniques we studied have been implemented as the tools DejaVu and TestTube; we compared these tools in terms of a cost model incorporating precision (ability to eliminate unnecessary test cases), analysis cost, and test execution cost. Our results indicate, that in many instances, despite its relative lack of precision, TestTube can reduce the time required for regression testing as much as the more precise DejaVu. In other instances, particularly where the time required to execute test cases is long, DejaVu's superior precision gives it a clear advantage over TestTube. Such variations in relative performance can complicate a tester's choice of which tool to use. Our experimental results suggest that a hybrid regression test-selection tool that combines features of TestTube and DejaVu may be an answer to these complications; we present an initial case study that demonstrates the potential benefit of such a tool.

Reviewer Abstract:
Safe Regression Test-Selection techniques select subsets of existing test suites to test modified programs, without omitting cases which would reveal faults in the modified programs. Two Safe Regression Test-Selection techniques, implemented as the tools TestTube and DejaVu, are compared for inclusiveness, precision, efficiency and generality. The comparisons are based on three sets of test programs: the Siemens programs, the ‘space’ program, and the ‘player’ component of the Empire game.

DejaVu analyzes control flow (fine-grained) and TestTube operates at the coarser level of functions and global variables. Due to these fundamental differences, it is found that DejaVu proves effective only when a large proportion of the test suite is to be excluded, or if individual test cases have large cost compared to DejaVu’s analysis time. TestTube is found to be more resilient to significant code changes, but less precise. TestTube can be implemented using simple scripts whereas DejaVu requires substantially more effort. The authors attempt to take advantage of both methods by creating a hybrid tool, which uses TestTube to localize changes to functions and then applies DejaVu to these restricted subsets. The hybrid tool proved superior to either method for a significant number of the test cases, and this would be an interesting avenue for future research.

This study evaluates two existing safe-RTS techniques and would primarily be of value to researchers in the field. The evaluation is based upon simple laboratory test cases and a commercial test component. The study would also be of some value to guide commercial testers in selecting from these two techniques.

Reviewer Name: Abhijit Ogale

Stefan Biffl, Wilfried Grossmann, "Evaluating the accuracy of defect estimation models based on inspection data from two inspection cycles, International Conference on Software Engineering, 2001, pp. 145-154.

Article Abstract:

Defect content estimation techniques (DCETs), based on defect data from inspection, estimate the total number of defects in a document to evaluate the development process. For inspections that yield few data points DCETs reportedly underestimate the number of defects. If there is a second inspection cycle, the additional defect data is expected to increase estimation accuracy. In this paper we consider 3 scenarios to combine data sets from the inspection-reinspection process. We evaluate these approaches with data from an experiment in a university environment where 31 teams inspected and reinspected a software requirements document. Main findings of the experiment were that reinspection data improved estimation accuracy. With the best combination approach all examined estimators yielded on average estimates within 20% around the true value, all estimates stayed within 40% around the true value.

Reviewer Abstract:

This paper studies the accuracy of defect estimation models. The purpose of defect content estimation technique (DCET), based on data from inspection, is to estimate the total number of defects in a document to evaluate the development process.

Inspections that yield few data points, DCETs reportedly underestimate the number of defects. If there is a second inspection cycle, the additional defect data is expected to increase estimation accuracy.

One of the models for estimating defects using inspection is Capture-Recapture model, which estimates defects, based on the number of recaptured defects and assumptions on capture probabilities. It is assumed that defects have equal probability of being detected. Total defect size is estimated using statistical models and estimators.

The second model is Reinspection Models. Reinspection Models can be divided into these steps: individual inspection, collation of the individual defect lists to a team list, and correction of the defects found by the authors. The primary objective of a second inspection cycle is to reduce the number of remaining defects to an acceptable quality level.
Depending on the effectiveness and efficiency of an inspection to find target defects, e.g., major defects, a project manager has to choose a design for the second inspection cycle. In general three different approaches are feasible:

A1: Reinspection with no changes of inspection design.
A2: Reinspection with modified inspection design.
A3: Reinspection with new inspection design.

According to the design chosen, estimation is projected to find the total number of defects.

The experiment in the paper is set for reinspection with no changes in design. There were 169 students, who participated in the experiment. The participants had a varying degree of development experience. While all of them knew how to develop small programs, about 10% had professional experience with the development of larger systems.

This paper can be useful to project manager and researchers who want to estimate the total number of defects in a document to evaluate the development process.

Reviewer: Orcun Colak

Stefan Biffl, Bemd Freimut, Oliver Laitenberger, "Investigating the Cost-Effectiveness of Re-inspections in Software Development", International Conference on Software Engineering, 2001, 155-166.

Article Abstract:

Software inspection is one of the most effective methods to detect defects. Re-inspection repeats the inspection processor software products that are suspected to contain a significant number of undetected defects after an initial inspection. As a re-inspection is often believed to be less efficient than an inspection an important question is whether a re-inspection justifies its cost.

In this paper we propose a cost-benefit model for inspection and re-inspection. We discuss the impact of cost and benefit parameters on the net gain of a re-inspection with empirical data from an experiment in which 31 student teams inspected and re-inspected a requirements document.

Main findings of the experiment are: a) For re-inspection benefits and net gain were significantly lower than for the initial inspection. Yet, the re-inspection yielded a positive net gain for most teams with conservative cost-benefit assumptions. b) Both the estimated benefits and number of major defects are key factors for re-inspection net gain, which emphasizes the need for appropriate estimation techniques.

Reviewer Abstract:

Repeating the inspection process on a software product is believed by many to be less efficient than the initial inspection. This paper attempts to shed more light on the relationship between costs and benefits of software inspections and reinspections. Theoretically, the rate of defect finding decreases over time, so the marginal cost per defect found increases. Furthermore, because inspectors usually work independently, the marginal benefit of increased investment diminishes with growing reading time, because more than one inspector may find the same defect. In order to measure the cost benefit relationship, two economic indicators are used: net gain (benefit minus cost) and return on investment (net gain per unit cost). An experiment was conducted with 31 student teams asked to find defects in a requirements document. The document contained seeded defects that were classified as trivial, minor, major or critical.

Because the software project was did not have an economic value per se, ten scenario cases with various cost and benefit characteristics were used to compute the economic indicators. Several conclusions were drawn: The efficiency of the teams was significantly lower for reinspections than for inspections, and also lower was the investment effort, the number of defects found and their benefit. Additionally, the net gain and return on investment diminished for the average with some exceptions. Assumptions on the level of indirect costs associated with inspections (delays etc.) had a significant impact on the economic indicators. Another significant source of influence on the results was benefit assumption for major and critical defects. Because of these, the authors believe that further empirical studies are necessary in this area.

Reviewer: Vasile Gaburici

Stefan Biffl, "Using Inspection Data for Defect Estimation", IEEE Software Magazine, Vol. 17, No. 6, Nov/Dec 2000, pp. 36-43.

Article Abstract:

This article pro-poses subjective team estimation models calculated from individual estimates and investigates the accuracy of defect estimation models based on inspection data.

Reviewer Abstract:

This paper examines existing defect estimation models (DEMs) for software projects, proposes new models, and measures their effectiveness. To validate the new methods, an experiment was performed as part of a university software development workshop that teaches 200+ undergraduate students to develop a real medium-size software project. The experiment's subjects were mostly novice software developers with little experience in inspection and estimation. Both objective and subjective DEMs, based on inspection data of a requirements document, were studied. One of the purposes of the study was to evaluate the relative accuracy of subjective and objective DEMs. Objective DEMs do not depend on personal opinion but require input from a complex data collection process. By contrast, subjective DEMs are relatively cheap and easy to obtain but depend on the knowledge and capability of the individual estimator.

The new DEM model is a capture-recapture model (CR). Four CR models are presented. They differ in their assumptions about the probability of defects to be found and of the ability of inspectors to find the defects. In the experiment, the estimator calculated the defect estimate's most likely value and a 95% confidence interval.

Some attributes of DEMs that are of interest are the relative error of the estimate (RE), the mean relative error (MRE), and the absolute relative error (ARE). The relative error characterizes a defects estimate's relative over- or under-estimation. The study found that both objective and subjective DEMs tend to underestimate defects in general. One particular objective estimate ('Mh') was found to perform better than the other objective estimates. However, subjective DEMs were more accurate as a group than objective DEMs. The confidence interval accuracy of all but the three best DEMs was much lower

than 95%. This study would be of interest to project managers, who use estimates of defects in a product to assess product quality with regard to the project plan.

Reviewer: Neal Bambha

Barry Boehm and Victor R. Basili, "Software Defect Reduction Top 10 List", IEEE Computer Magazine, Vol. 34, No. 1, January 2001, pp. 135-137

Article Abstract:

Software's complexity and accelerated development schedules make avoiding defects difficult. These 10 techniques can help reduce the flaws in your code.

Reviewer Abstract:

With the increase of software's complexity and accelerated development schedules, avoiding defects becomes very important in the process of software development.

In this paper, the process of software development is studied. The main contribution of this paper is to list 10 techniques to help reduce the flaws in the process of software development. This paper studied the cost of finding defects before or after delivery, the percentage of effort on avoidable rework, where most avoidable rework comes from, where most defects come from, where most downtime comes from, the peer reviews technique to catch most of the defects, using special reading technique to catch more defects than nondirected reviews, how to reduce most defect introduction rates, how much cost in per source instruction can effect dependability of software products, and how many user programs contain nontrivial defects.

The purpose of the study is to improve the process of software development. This study is very valuable for project managers to design the process and for developers to avoid defects.

This study is made in the Center for Empirically Based Software Engineering. This center seeks to transform software engineering as much as possible from a fad-based practice to an engineering-based practice through derivation, organization, and dissemination of empirical data on software development and evolution phenomenology.

Reviewer: Zhiyun Li

Lionel C. Briand, Tristen Langley, Isabella Wieczorek, "A Replicated Assesment and Comparison of Common Software Cost Modelling Techniques", International Conference on Software Engineering, 2000, pp. 377-386.

Article Abstract:

Delivering a software product on time, within budget, and to an agreed level of quality is a critical concern for many software organizations. Underestimating software costs can have detrimental effects on the quality of the delivered software and thus on a company's business

reputation and competitiveness. On the other hand, overestimation of software cost can result in missed opportunities to funds in other projects. In response to industry demand, a myriad of estimation techniques has been proposed during the last three decades. In order to assess the suitability of a technique from a diverse selection, its performance and relative merits must be compared. The current study replicates a comprehensive comparison of common estimation techniques within different organizational contexts, using data from the European Space Agency. Our study is motivated by the challenge to assess the feasibility of using multi-organization data to build cost models and the benefits gained from company-specific data collection. Using the European Space Agency data set, we investigated a yet unexplored application domain, including military and space projects. The results showed that traditional techniques, namely, ordinary least-squares regression and analysis of variance outperformed analogy-based estimation and regression trees. Consistent with the results of the replicated study no significant difference was found in accuracy between estimates derived from company-specific data and estimates derived from multi-organizational data.

Reviewer Abstract:

It is important for a company to underestimate softwar costs in order to delivering a software product on time, within budget and to an agrred level of quality. However, it is hard to estimate the cost befoer hand. Thia paper addresses the chanllange of assessing the feasibility of using multi-organization data to build cost models and the benefits gained from company-specific data collection. It replicates and expands on a previous study on MIS domain and it provides a procedure to perform and present cost model evaluations and comparisons. They selected some modeling techniques according to the following criteria: applied in Software Engineering; Automatable; Interpretable; Suitable Input Requirements. Thus they compared following methods: Ordinary Least-Squares Regression; Stepwise ANOVA; Analogy-based estimation; CART and some Combinations of Techniques. They are doing the comparation on the European Space Agency (ESA) multi-organization software project database.

They are using Magnitude of Relative Error (MRE) as a common evaluation criterion for these cost estimation models. Another criterion used is the prediction at level l, PRED(l)=k/N where k is the number of observations, where MRE is less than or equal to l. The actual criteria we used to assess and compare cost estimation models are the relative values of MMRE, MdMRE, and PRED(0.25) for the different techniques. They are also using cross-validation which involves dividing the whole data set into multiple train and test sets,calculating the accuracy for each test set, and then aggregating the accuracy across all the test sets. As their analysis of the evaluation results, ANOVA_e and OLS regression are better, across data sets, in terms of their predictive capability using cross validation. Then they conclude that ordinary least-squares regression is probably sufficient to get the most out of your data and help predict development effort.

Reviewer: Feng Peng

Lionel C. Briand, Jurgen Wust, Stefan V. Ikonomovski, Hakim Lounis, "Investigating Quality Factors in Object-Oriented Designs: an Industrial Case Study", International Conference on Software Engineering, 1999, pp. 345-354.

Article Abstract:

This paper aims at empirically exploring the relationships between most of the existing coupling and cohesion measures for object-oriented (OO) systems, and the fault-proneness of OO system classes. The underlying goal of such a study is to better understand the relationship between existing design measurement in OO systems and the quality of the software developed.

The study described here is a replication of an analogous study conducted in an university environment with systems developed by students. In order to draw more general conclusions and to (dis)confirm the results obtained there, we now replicated the study using data collected on an industrial system developed by professionals.

Results show that many of our findings are consistent across systems, despite the very disparate nature of the systems under study. Some of the strong dimensions captured by the measures in each data set are visible in both the university and industrial case study. For example, the frequency of method invocations appears to be the main driving factor of fault-proneness in all systems. However, there are also differences across studies which illustrate the fact that quality does not follow universal laws and that quality models must be developed locally, wherever needed.

Reviewer Abstract:

This paper aims at empirically exploring the relationships between most of the existing coupling and cohesion measures for object-oriented (OO) systems, and the fault proneness of OO system classes. The underlying goal of such a study is to better understand the relationship between existing design measurement in 00 systems and the quality of the software developed. The study described here is a replication of an analogous study conducted in an university environment with systems developed by students. In order to draw more general conclusions and to (dis)confirm the results obtained there, we now replicated the study using data collected on an industrial system developed by professionals. Results show that many of our findings are consistent across systems, despite the very disparate nature of the systems under study. Some of the strong dimensions captured by the measures in each data set are visible in both the university and industrial case study. For example, the frequency of method invocations appears to be the main driving factor of fault-proneness in all systems. However, there are also differences across studies, which illustrate the fact that quality does not follow universal laws and that quality models must be developed locally, wherever needed.

The purpose of this study is to understand the coupling and cohesion quality of object-oriented (OO) design for classifying the quality of systems, in particular measures for predicting fault-proneness. The design measures were used to build predictive models for evaluating the quality of systems in regards to fault-proneness. This study compares data that was collected from industrial projects with that of a study done at the University of Maryland (UMD) with student projects. The idea was to draw conclusions that could be span many systems. This paper is primarily for researchers for furthering work in this area, as well as for examining systems in the field that have implemented OO, albeit organizations that are more advanced in implementing software engineering methods. The measures for comparison that were employed for this study

were that of descriptive statistics, principal component analysis, univariate analysis, and correlation to size. The results that were derived from this investigation will go towards a holistic understanding of the impact on quality and predictability in regards to developing OO systems. The conclusions from this study indicate that the aforementioned design measures were effective is measuring quality. However, it must be noted that the validity of the resulting models is highly context sensitive because of the variability between the two projects examined. The context of the study was that of a laboratory environment as the conclusion states that the replication of similar studies is encouraged in order to build a credible empirical body of knowledge in regards to quality assessments of OO systems. It should be noted that the conclusions in this study state that the models derived are context sensitive.

Reviewer: Ronald Wilson

Lionel C. Briand, Khaled El Emam, Isabella Wieczorek, "Explaining the Cost of European Space and Military Projects", International Conference on Software Engineering, 1999, pp. 303-312.

Article Abstract:

There has been much controversy in the literature on several issues underlying the construction of parametric software development cost models. For example, it has been argued whether (dis)economies of scale exist in software production, what functional form should be assumed between effort and product size, whether COCOMO factors were useful, and whether the COCOMO factors are independent. Answers to such questions should help software organizations define suitable data collection programs and well-specified cost models. The only way to address these issues and obtain a generalizable conclusion is to investigate them on a large number of consistent data sets. In this paper we use a data set collected by the European Space Agency to perform such an investigation. To ensure a certain degree of consistency in our data, we focus our analysis on a set of space and military projects that represent an important application domain and the largest subset in the database. These projects have been performed, however, by a variety of organizations. First, our results indicate that two functional forms are plausible between effort and product size: linear and log-linear. This also means that different project subpopulations are likely to follow different functional forms. Second, besides product size, the strongest factor influencing cost appears to be team size. Larger teams result in substantially lower productivity, which is interesting considering this attribute is rarely collected in software engineering cost data bases. Third, although some COCOMO factors appear to be useful and significant covariates, they play a minor in explaining project effort. Overall, the most plausible model appears to be a log-linear model involving KLOC, team size, and a principal component influenced by three COCOMO factors: reliability requirements (RELY), storage constraints (STOR), and execution time constraints (TIME). High values for these factors are likely to be associated with embedded systems, which usually share these characteristics.

Reviewer Abstract:

There has been much controversy in the literature on several issues underlying the construction of parametric software development cost models. For example, it has. been argued whether (dis)economies of scale exist in software production, what functional form should be assumed

between effort and product size, whether COCOMO factors were useful, and whether the COCOMO factors are independent. Answers to such questions should help software organizations define suitable data collection programs and well-specified cost models. The only way to address these issues and obtain a generalizable conclusion is to investigate them on a large number of consistent data sets. In this paper we use a data set collected by the European Space Agency to perform such an investigation. To ensure a certain degree of consistency in our data, we focus our analysis on a set of space and military projects that represent an important application domain and the largest subset in the database. These projects have been performed, however, by a variety of organizations. First, our results indicate that two functional forms are plausible between effort and product size: linear and log-linear. This also means that different project subpopulations are likely to follow different functional forms. Second, besides product size, the strongest factor influencing cost appears to be team size. Larger teams result in substantially lower productivity, which is interesting considering this attribute is rarely collected in software engineering cost data bases. Third, although some COCOMO factors appear to be useful and significant covariates, they play a minor role in explaining project effort. Overall, the most plausible model appears to be a log-linear model involving KLOC, team size, and a principal component influenced by three COCOMO factors: reliability requirements (RELY), storage constraints (STOR), and execution time constraints (TIME). High values for these factors are likely to be associated with embedded systems, which usually share these characteristics.

The purpose of this study is to understand the influence of project elements in predicting the cost of software projects for the European Space Agency, in particular space and military projects. It focuses on investigation questions revolving around software cost modeling with four factors influencing cost, which are effort and product size, extent of economies of scale, the impact of team size, and various COCOMO factors. It is hoped that the results from this analysis go toward alleviating typical problems that often plague software projects, that of how to best manage projects, maximize productivity, and improve cost estimation. The analysis indicates that first the relationship between effort and product size produces two functional forms, that of linear and log-linear. Second, it is found that to the contrary the larger teams result in lower productivity and higher costs. Third, the COCOMO factors while useful do not contribute much in the way of explaining expended effort for a project. The results afford confidence to the authors stating that the final results were applicable to space and military projects and they can make practical recommendations. First is that the relationship between project effort and size should always be investigated. Second is that team size should be considered for use in the cost estimating models. This work is primarily aimed at large organizations, however, it can be used for further research in a laboratory environment. The context of the study is indicative of a large organization that has many projects of similar domain and large database from which to draw adequate data.

Reviewer: Ronald Wilson

Lionel C. Briand, Khaled El Emam, Dagmar Surmann, Isabella Wieczorek, Katrina D. Maxwell, "An Assesment and Comparison of Common Software Cost Estimation Techniques", International Conference on Software Engineering, 1999, pp. 313-322.

Article Abstract:

This paper investigates two essential questions related to data-driven, software cost modeling: (1) What modeling techniques are likely to yield more accurate results when using typical software development cost data? and (2) What are the benefits and drawbacks of using organization-specific data as compared to multi-organization databases? The former question is important in guiding the software cost analysts in their choice of the right type of modeling technique, if at all possible. In order to address this issue, we asses and compare a selection of common cost modeling techniques fulfilling a number of important criteria using a large multi-organizational database in the business application domain. Namely, these are: ordinary least squares regression, stepwise ANOVA, CART, and analogy. The latter question is important in order to asses the feasibility of using multi-organization cost databases to build cost models and the benefits gained from local, company-specific data collection and modeling. As a large subset of the data in the multi-company database came from one organization, we were able to investigate this issue by comparing organization-specific models with models based on multi-organization data. Results show that the performances of the modeling techniques considered were not significantly different, with the exception of the analogy-based models which appear to be less accurate. Surprisingly, when using cost factors (e.g., COCOMO-like factors, Function Points), organization specific models did not yield better result than generic, multi-organization models.

Reviewer Abstract:

The objects of study from this paper are software cost estimation techniques. Twenty different variables from software projects are considered in the analysis: total project effort, system size measured in function points, organization type, application type, target platform and other 15 productivity factors like customer participation and development environment The purpose of the study is to evaluate and compare cost modeling techniques trying to answer two questions: (i) What modeling techniques are likely to yield more accurate results when using typical software development cost data? and (ii) What are the benefits and drawbacks of using organization-specific data as compared to multi-organization databases? The study should be of value to organizations and project managers as it can help them to select the best software cost estimation techniques. The study also identifies the benefits and drawbacks of using organization specific data giving project managers a better understanding of what their cost estimation mean if they consider either organization specific or multi-organization data. The study was conducted using data from 206 software projects from 26 different companies. The projects are mainly business applications in banking, wholesale/retail, insurance, public administration and manufacturing sectors. The techniques considered in the study are: ordinary least-squares regression (OLS), a standard analysis of Variance approach for unbalanced data sets, CART, an analogy based approach. Combinations of these modeling techniques are also considered: CART and OLS regression and CART and analogy based approach. The paper describes the study and conclusions answering both questions in details. Overall, whether local or generic models are built, simple CART models seem to be a good alternative both from an accuracy and interpretability point of view. Local models developed using the one-company database do not perform significantly better than the models using external multi-organization data, when applied to the one company data set.

Reviewer: Patricia Costa

Broy, M., "Toward a mathematical foundation of software engineering methods", IEEE Transactions on Software Engineering, Vol. Year, Issue -Volume: 27 Issue: 1, Jan. 2001, pp. 42-57

Existing abstract:

The development of large software systems consists of a sequence of modeling tasks. It requires the modeling and description of the application domain, software requirements, software architecture, software components, their internal structure, and their implementation. Technically, in software engineering, we work with a development method and description techniques with modeling, refinement, and implementation concepts. Today, much of the modeling is carried out by informal text and graphical description techniques. The development is organized in a development process and supported by CASE tools. We show how mathematics can provide a scientific foundation for the modeling aspects, description techniques, and development methods of software engineering. Such a scientific foundation leads to a deeper understanding of the development process and to a basis for a more powerful tool support.

Modified Abstract:

We are trying to study the use of mathematics to provide a scientific foundation for the modeling aspects, description techniques and development methods of software engineering.

This study was sponsored by Bayerische Forschungsstiftung and Siemens Nixdorf and partially sponsored by Deutche Forschungsgemeinschaft, in order to come up with a different approach to developing method and description techniques. The conventional usage of informal text and graphical description techniques for this purpose, now has an alternative of using mathematical models that aim at providing a scientific foundation.

Methods, notations and concepts used in practice for the modeling and description of digital systems can be scientifically based on the more foundational and theoretical work in computing science. This way we obtain a mathematical basis for modeling languages and software and system engineering methods.

The purpose of the study is to provide a theoretical basis to the techniques in software engineering using a mathematical foundation.
Developing a mathematical theory that reflects the theoretical core of a method and that allows not only giving semantics to the description techniques but in addition explains and justifies the method as a mathematical rational.

A mathematical model is developed that comprises of system view. This model is supported by a complementary description technique. This is to show that the system can be used practically for development methods.

A much better understanding, conceptual consistency, guideline for cleaner methods of description and development methods is achieved using mathematical models. Moreover a scientific foundation is needed to study software engineering methods in a systematic manner.

This study aims to show what a mathematical basis of software engineering methods may look like. To show that a mathematical foundation help to design proper methods, especially when constructing the description techniques and their mathematical foundation hand-in-hand.

Reviewer: Nitin Dinesh Anand

Calzolari, F.; Tonella, P.; Antoniol, G., "Dynamic model for maintenance and testing effort", Proceedings of the International Conference on Software Maintenance, November 1998, pp. 104-112.

Article Abstract:
The dynamic evolution of ecological systems in which predators and prey compete for survival has been investigated by applying suitable mathematical models. Dynamic systems theory provides a useful way to model interspecies competition and thus the evolution of predator and prey populations. This kind of mathematical framework has been shown to be well suited to describe evolution of economical systems as well, where instead of predators and prey there are consumers and resources. Maintenance and testing activities absorb the most relevant part of the total life-cycle cost of software. Such economic relevance strongly suggests to investigate the maintenance and testing processes in order to find new models allowing software engineers to better estimate, plan and manage costs and activities. We show how dynamic systems theory could be usefully applied to maintenance and testing context, namely to model the dynamic evolution of the effort. When programmers start trying to recognize and correct code defects, while the number of residual defects decreases, the effort spent to find out any new defect has an initial increase, followed by a decline, in a similar way as prey and predator populations do. The feasibility of this approach is supported by the experimental data about two real world software projects.

Reviewer Abstract:
Entity = Model of maintenence and testing effort vs # defects
Attributes of interest = non-linear, dynamic nature of relationship between effort and defects eliminated
purpose = understanding + prediction
stakeholder = planner of software project; person who decides how much effort should be expended on certain tasks
context = corporate/government software with a long maintenance/testing lifetime and multiple releases
background - nonlinear dynamic model of biological predator/prey relationships

This paper examines a model of the relationship between defects and the effort spent eliminating them in the testing and maintenance phases of long lived software projects. It is similar in scope to Boehm's Constructive Cost Model (COCOMO).

The model is patterned after dynamical predator/prey systems from biological models. Considering defects as prey and maintainers/testers as predators that consume the prey (fix the defects), a system of two differential equations is proposed to relate them.

The purpose of this model is to predict and plan allocation of testing and maintenance resources. After the raw data has been passed through some smoothing function, the accuracy of the model's prediction of required effort ranges from 7-30 percent of the actual.

This model would be most useful to software project managers who wish to know in advance what the impact of succesive releases and varying maintenance effort would be on the quality of software products. However, due to the nature of the model, at least four data points are necessary for calibration, which implies that the usefulness of this model is not immediate.

This paper was written in the context of large software projects created by corporate or govenmental entities. The nature of the projects require a periodic release schedule, and guarantee that testing and maintenance last for periods of many months, as well as periodic increases in the number of defects.

Dynamical predator/prey systems in biology are based on the fact that without predators, prey populations would increase to the carrying capacity of the environment, and that the growth of the predator population is proportional to the size of the prey population. Such systems have successfully been applied to other domains such as economics.

Reviewer Name: David C Rafkind

Jonathan E. Cook and Alexander L. Wolf, "Software process validation: quantitatively measuring the correspondence of a process to a model", ACM Transactions on Software Engineering and Methodology, Volume 8, Issue 2 (1999), Pages 147-176.

Article Abstract:
To a great extent, the usefulness of a formal model of a software process lies in its ability to accurately predict the behavior of the executing process. Similarly, the usefulness of an executing process lies largely in its ability to fulfill the requirements embodied in a formal model of the process. When process models and process executions diverge, something significant is happening. We have developed techniques for uncovering and measuring the discrepancies between models and executions, which we call process validation. Process validation takes a process execution and a process model, and measures the level of correspondence between the two. Our metrics are tailorable and give process engineers control over determining the severity of different types of discrepancies. The techniques provide detailed information once a high-level measurement indicates the presence of a problem. We have applied our processes validation methods in an industrial case study, of which a portion is described in this article.

Reviewer Abstract:
This paper aims to measure the conformance of a software process to its model. The result can be used to evaluate either how well a model can simulate a real world process or to what extend are the people in project following the model.

The study defines an execution and model behavior as a series of events. Events usually correspond to important stages in the process like executing a development tool or a decision point. Authors acknowledge that it is not reasonable to expect every event to be tracked, but

argue that most development tools and management practices produce a satisfactory log of events.

The accumulation of events in the order they take place produces an event stream. The execution event stream and the corresponding model estimation are then compared using variants of string edit distance. The total cost of event deviations is divided by a projected maximum deviation total cost to come up with a number between 0 and 1.0. A low number denotes high degree of conformance. In addition to a simple quantity to summarize the evaluation, the output of the evaluation can be drilled down deeper to visually see where the problems take place.

An important part of the study is to come up with the lowest cost event stream prediction of a model. This is a complex problem and finding an optimal solution is regarded as infeasible. Instead the study uses a search tree, and heuristics to explore only promising parts of the tree.

The paper includes results from a telecommunications software system. The process starts with a client submitting a bug and labeled as successful if the client accepts the fix. The results show that successful processes have a higher conformance rates compared to unsuccessful ones.

Reviewer Name: Cengiz Celik

Delamaro, M.E.; Maidonado, J.C.; Mathur, A.P., "Interface Mutation: an approach for integration testing", IEEE Transactions on Software Engineering, Vol. Year, Issue -Volume: 27 Issue: 3, March 2001. Pp. 228-247

Existing abstract:

The need for test adequacy criteria is widely recognized. Several criteria have been proposed for the assessment of adequacy of tests at the unit level. However, there remains a lack of criteria for the assessment of the adequacy of tests generated during integration testing. We present a mutation based interprocedural criterion, named Interface Mutation (IM), suitable for use during integration testing. A case study to evaluate the proposed criterion is reported. In the study, the UNIX sort utility was seeded with errors and Interface Mutation evaluated by measuring the cost of its application and its error revealing effectiveness. Alternative IM criteria using different sets of Interface Mutation operators were also evaluated. While comparing the error revealing effectiveness of these Interface Mutation-based test sets with same size randomly generated test sets, we observed that in most cases Interface Mutation based test sets are superior. The results suggest that Interface Mutation offers a viable test adequacy criteria for use at the integration level.

Your rating for this paper - high

Modified Abstract:

This paper presents a mutation-based interprocedural criterion, named Interface Mutation (IM) criterion to support, complement, and improve integration testing. Also, the case study is reported to evaluate the IM criterion's cost measure, which are the number of mutants and the number of test cases required. This paper should be value of project managers and testers of any real world software industry. To evaluate a test set using Interface Mutation, the first step is

to perturb the system under test by making a simple syntactic change, determined by an Interface Mutation operator. The perturbation process is also known as mutation and the perturbed program as a mutant. The syntactic changes are made only at the interface ralated points--or connections--between units. The second step is to execute the mutants, to evaluated test set adequacy, and to decide mutant equivalence.

The Interface Mutation operators can be catagorized into 2 broad groups, (1) Operators Applied Inside the Called Function, and (2)Operators Applied Inside the Calling Function.

In the case study, the sort Unix utility was selected and several incorrect versions were created by seeding 11 most-difficult-to-reveal errors, selected by Hardness Index. Test sets for each function connection were then generated using Interface Mutation. Once an IM-based test set is obtained, its effectiveness is measured by checking whether it reveals the seeded error or not by comparing the execution of the erroneous version and the execution of the version with no errors with the test cases in the IM-based set. Interface Mutation has been shown effective in this case study by revealing almost 100 percent of error for all the IM-based test case sets, and the size of IM-based test sets is not the primary factor that controls the effectiveness. However, the cost of Interface Mutation might still constrain its use in practice. The alternative mutation criteria, such as random and constrained mutation or selective mutation, may be considered to reduce the associated cost.

Reviewer: Nuengwong Tuaychroen

Dewar, R.; Lloyd, A.D.; Pooley, R.; Stevens, P., "Identifying and communicating expertise in systems reengineering: A patterns approach", Software, IEEE Proceedings, Vol. 146, Issue 3, June 1999, Pg 145-152

Article Abstract: The reengineering of legacy systems-by which we mean those that have value and yet 'significantly resist modification and evolution to meet new and constantly changing business requirements'-is widely recognized as one of the most significant challenges facing software engineers. A complex mixture of business and technical factors must be taken into account to ensure success, and there is a wide range of different contexts each with its own problems. Moreover, the business needs do not stay constant while the technical factors are dealt with. In the paper it is argued that the main problem is not that the necessary expertise does not exist, but rather, that it is hard for software engineers to become expert in all of the necessary areas. It is proposed that systems reengineering patterns may help to codify and disseminate expertise, and that this approach has some advantages over conventional methodological approaches. This contention is supported by means of some candidate patterns drawn from the author’s experience and supported by information from elsewhere.

Reviewer Abstract: Reengineering, especially incremental reengineering in the presence of a changing business environment is hard chiefly because of the wide range of factors (or forces) which must be taken into account in evaluating candidate solutions. The wide scope of the reengineering problem and the absence of a commonly agreed classification of its areas exacerbate the problem. More typically, the reengineering expert must consider system(s) which interact with the business's procedures in complex ways. These require heavy modification of the

software, which can ultimately eliminate the economic advantage. Unfortunately, however, for a high proportion of large legacy systems such an approach is utterly infeasible. The present paper takes a wider, less purely software engineering view of the subject, and gives more detail. In summary, we believe that the most important problem is not an absence of expertise but the difficulty of transferring that expertise to those who need it.

In particular, we want to address the problem of synthesizing expertise in 'pure' systems reengineering with that which concentrates on the organization.

We think that patterns will complement overarching methodologies for reengineering. We prefer a broad interpretation of what might constitute a valid reengineering pattern. We recommend that pattern descriptions should include a Status section describing the confidence which may be placed in the pattern. The ultimate aim is to develop a pattern language: that is, a collection of patterns which work together effectively in documented ways.

We mention four candidate reengineering patterns, drawn (in one case) from interviews with people in a large company which undertakes many reengineering projects, and (in three cases) from our own experience of working on reengineering projects in business and academia.

Reviewer Name: Liu Hanbin

C.R. Douce and P.J. Layzell, "Evolution and errors: an empirical example", Journal: Proceedings of the IEEE International Conference on Software Maintenance 1999 (ICSM '99), August 30-September 3, 1999, pp. 493 - 498.

Article Abstract:

An empirical study is presented that explores how dangerous errors can be introduced into software by the simplest of maintenance tasks. Six different maintenance problems were given to eight intermediate level programmers. Each maintenance problem was coded using the C++ language and were selected from a domain that was deemed to be familiar to all subjects. An alarming level of error was discovered. Analysis of the maintained software showed that one third of all the resulting programs contained an error of some form. Reasons why the error rates were so high are considered and two broad approaches to the reduction of maintenance error are described.

Reviewer Abstract:

This study aims at characterizing and understanding the maintenance process by observing how a set of intermediate programmers performing different maintenance tasks. This study is performed in a laboratory environment where distraction and extraneous variables can be cleansed. The 6 maintenance problems were created to explore the comprehension of different facet of OO design.

The data collection is performed through four channels. Talk aloud, an editor that records every file change in a log, the observation notes made by the experimenter and finally any programming notes or drawing that the subjects made. The subjects were intermediate

programmers in C++. They were provided with some material: the source code to change and a description of the required change.

The code then was analyzed for internal or external errors. External errors are difference with the requirement, or any visible mistake. Internal being some latent mistakes that could be fore example an unused function, messy variable names, broken classes hierarchies,...

More than one third of the programs contained some sort of error ! Their type varied widely depending on the studied problem. Different explanation are proposed as that the problems were too difficult, the programmers didn't have appropriate skills levels, and finally that the definition of an error is subjective to the opinion of the experimenter.

Another observation is that half of the subject stated that problem solving performance was affected by talk aloud technique.

This study is of definite interest to the project manager or anyone responsible to supervise maintenance tasks. It highlights many aspects of the problem that may be responsible for the introduction of new errors during the process.

Reviewer Name: Laurent Eschenauer

Matthew B. Dwyer, John Hatcliff, Roby Joehanes, Shawn Laubach, Corina S. PasSreanu, Robby, Hongjun Zheng, Willem Visser, "Tool-supported Program Abstraction for Finite-state Verification", International Conference on Software Engineering, 2001, pp. 177-187.

Article Abstract:

Numerous researchers have reported success in reasoning about properties of small programs using finite-state verification techniques. We believe, as do most researchers in this area, that in order to scale those initial successes to realistic programs, aggressive abstraction of program data will be necessary. Furthermore, we believe that to make abstraction-based verification usable by non-experts significant tool support will be required.

In this paper, we describe how several different program analysis and transformation techniques are integrated into the Bandera toolset to provide facilities for abstracting Java programs to produce compact, finite-state models that are amenable to verification, for example via model checking. We illustrate the application of Bandera's abstraction facilities to analyze a realistic multi-threaded Java program.

Reviewer Abstract:

This study described an approach to user-guided abstraction of programs written in Java. The goal was to enable non-experts in program verification to generate abstract program models that are compact enough for tractable verification. The abstraction would be accomplished by a toolset called Bandera - an integrated collection of program analysis and transformation components. The researcher states that the greatest obsticle to scaling finite-state verification

technology to reasonign about realistic programs is the increase in the size of a finite-state model as the number of program components grow.The perspective of this study was characterization.

The study would be of interest to other researchers.

The underlying principles on which the abstraction techniques are built is described as 1) define an abstraction mapping that is appropriate for what is being verified, 2) use the mapping to transform the temporal property into an abstract property, 3) use the abstraction mapping to transform the concrete program into an abstract program, 4) verify the abstract program satisfies the abstract property, and 5) inter that the cotnract4e program satisfies the concrete property.

The researcher applies the abstraction tools to a variety of small Java programs. The researcher feels that the main contribution of his work was the integration of different techniques into a coherent program abstraction toolset that has the ability to greatly extend the range of programs to which model checking techniques can be effectively applied. Evidence to support this statement was the Java program that had a total run-time for all tool components of less than one minute on a 444Mhz Sun Ultra 10 with 1 Megabyte of memory. The checks of the original program could not be completed with four times the amount of memory.

Reviewer: Patricia Larsen

Stephen G. Eick, Member IEEE, Todd L. Graves, Alan F. Karr, Member IEEE Computer Society, J.S. Marron, and Audris Mockus, Member, IEEE, "Does Code Decay? Assessing the Evidence from Change Management Data", IEEE Transactions on Software Engineering, Volume 27, January 2001, Page(s) 1-12.

Existed Abstract:
A central feature of the evolution of large software systems is that change-which is necessary to add new functionality, accommodate new hardware, and repair faults-becomes increasingly difficult over time. We approach this phenomenon, which we term code decay, scientifically and statistically. We define code decay and propose a number of measurements (code decay indices) on software and on the organizations that produce it, that serve as symptoms, risk factors, and predictors of decay. Using an unusually rich data set (the fifteen-plus year change history of the millions of lines of software for a telephone switching system), we find mixed, but on the whole persuasive, statistical evidence of code decay, which is corroborated by developers of the code. Suggestive indications that perfective maintenance can retard code decay are also discussed.

Modified Abstract:
The study addresses the phenomenon of code decay both scientifically and statistically. Code Decay is measured by examining span of changes, breakdown of modularity, fault potential and prediction of effort required to make a change. The purpose of the study is to be able to answer whether the code decay is real, how can it be characterized, and the extent to which it matters. A change is any alteration to the software recorded in the change history database including adaptive, corrective and perfective changes. The study is aimed at software developers, project managers and empirical researchers. The research is based on the entire change management history of a large, fifteen-year old millions of lines of real-time software system for telephone

switches. The study uses a medical metaphor to understand code decay: Software suffering from decay can be thought of as diseased. It explores the causes of code decay disease, symptoms exhibited by decayed software and risk factors that increase the likelihood of code decay. It proposes a number of measurements i.e. Code Decay Indices (CDIs) like history of frequent changes, span of changes, age of a software unit, fault potential and effort required to implement a change. Four specific analyses were performed which demonstrated: 1) the increase over time in the number of files touched per change to the code, 2) the decline in modularity of a subsystem of the code, 3) contributions of factors to fault rates in modules of the code, and 4) that span and size of changes are important predictors of the effort to implement a change. The results yield very strong evidence that code does decay and show that change-based conceptual model of decay is correct. That change is agent of decay is borne out by the data, which is crucial since there are then actionable means to retard or reverse decay.

Reviwer; Kanta Jiwnani

Khaled El Emam, "Benchmarking Kappa: Interrater Agreement in Software Process Assessments", International Journal of Empirical Software Engineering, Volume 4, Number 2, June 1999, pp. 113-133.

Article Abstract:

Software process assessments are by now a prevalent tool for process improvement and contract risk assessment in the software industry. Given that scores are assigned to processes during an assessment, a process assessment can be considered a subjective measurement procedure. As with any subjective measurement procedure, the reliability of process assessments has important implications on the utility of assessment scores, and therefore the reliability of assessments can be taken as a criterion for evaluating an assessment's quality. The particular type of reliability of interest in this paper is interrater agreement. Thus far, empirical evaluations of the interrater agreement of assessments have used Cohen's Kappa coefficient. Once a Kappa value has been derived, the next question is "how good is it"? Benchmarks for interpreting the obtained values of Kappa are available from the social sciences and medical literature. However, the applicability of these benchmarks to the software process assessment context is not obvious. In this paper we develop a benchmark for interpreting Kappa values using data from ratings of 70 process instances collected from assessments of 19 different projects in 7 different organizations in Europe during the SPICE Trials (this is an international effort to empirically evaluate the emerging ISO/IEC 15504 International Standard for Software Process Assessment). The benchmark indicates that Kappa values below 0.45 are poor, and values above 0.62 constitute substantial agreement and should be the minimum aimed for. This benchmark can be used to decide how good an assessment's reliability is.

Reviewer Abstract:

The reliability of software process assessments is an important rasearch topic of the software engineering area. There are different types of reliability that can be evaluated such as the internal consistency of instruments. Interrater agreement is concerned with the extent of agreement in the ratings given by independent assessors to the same software engineering practices.High interrater

agreement is desirable to give credibility to assessment results,for example, in the context of using assessment scores in contract award decisions. If agreement is low, then this would indicate that the scores are too dependent on the individuals who have conducted the assessments.The statistic that has been employed almost exclusively in the evaluation of interrater agreement has been Cohen's Kappa which is quite popular with researchers for evaluating intra and inter observer agreement. It is also necessary to be able to decide whether the value obtained is good enough. Therefore, the most important issue is to develop a software process assessment benchmark for interpreting Kappa values.

The interpretation of measured values can be norm-referenced or criterion-referenced. Since there is no clear general basis for defining a Kappa threshold (or series of thresholds), the criterion-referenced approach would be difficult to operationalize. One common technique for constructing norm-referenced scores is to use percentiles. This has the advantage of deriving scores that are easily understandable, and it does not make distributional assumptions. The auther uses data obtained from process assessments of 70 process instances during the SPICE Trials. The benchmark has four ranges or levels based on the quartiles of the Kappa distribution. A detailed description of the assessment process is described then. And the author also gives some analysis of the result from the tests.

The main distribution of the paper is that it present the constructing of a new benchmarks specific to the software process assessments for interpreting the obtained values of Kappa. It can be used to decide the extent to which the reliability of new assessments is good or bad compared to assessments conducted within the SPICE Trials.

Reviewer: Feng Peng

W.M. Evanco, "Analyzing change effort in software during development", 6th International Software Metrics Symposium, 1999, pp. 179-188

Article Abstract:

Reviewer Abstract:

The overall objective of the study is to explain the effort associated with non-defect changes of software during development. The context of the study is a university expecrimental project where some ordinal response models were developed for analyzing change effort in software during development, and the models were calibrated on the basis of a single software system and then validated on two additional systems. The entity(s) being studied are three existing Ada software systems, and the developed and then calibrated ordinal response models. The purpose of the study is to understand the factors that govern change effort during development. The study shall be of value to software engineering researchers and developers who are interested in change effort model development and applications. Important results of the study include the identification of several categories of determinants for change effort: change locality, the internal complexity characteristics of the software components associated with the change, and the type of the change. The concordancy or discordancy statistics were comparable to those of the

calibrated models, when the calibrated models were applied to rank order software systems developed in the same environment.

Reviewer: Zhijian Pan

M.W. Godfrey and Qiang Tu, "Evolution in open source software: a case study", Proceedings of the International Conference on Software Maintenance, 2000, 11-14 Oct. 2000, pp. 131 - 142.

Article Abstract:
Most studies of software evolution have been performed on systems developed within a single company using traditional management techniques. With the widespread availability of several large software systems that have been developed using an "open source" development approach, we now have a chance to examine these systems in detail, and see if their evolutionary narratives are significantly different from commercially developed systems. The paper summarizes our preliminary investigations into the evolution of the best known open source system: the Linux operating system kernel.

Because Linux is large (over two million lines of code in the most recent version) and because its development model is not as tightly planned and managed as most industrial software processes, we had expected to find that Linux was growing more slowly as it got bigger and more complex. Instead, we have found that Linux has been growing at a super-linear rate for several years. The authors explore the evolution of the Linux kernel both at the system level and within the major subsystems, and they discuss why they think Linux continues to exhibit such strong growth.

Reviewer Abstract:
This case study aims at characterizing the software evolution model of an open source development. Its goal is to improve our understanding of the software evolution process in new scenario like the one provided by an OSD. Such development is highly collaborative and geographically distributed and is different to the traditional in-house development process in many ways. The goal is not to fulfill a commercial void or a customer request, but to please the developer. There is no economic pressure to release the code and developers are volunteers. The consequence is that they prefer to work on new code and ideas than to correct bugs and defects or perform essential tasks such as testing or code restructuring.

This paper investigates the evolution of the Linux system. It analyses the two development path over 6 years and 96 kernel versions. Different measurement are made: total size, counted lines of code, global functions count, etc... The subsystem view of the kernel is also taken into account. Each subsystem is a separate directory in the distribution tree.

The main observation is that the growth is super-linear, when we were expecting a sub-linear rate over time. This contradicts previous hypothesis stated by Lehman and Turski. When looking at the subsystem, one can see that this growth is mainly in the driver subsystem. Because of the popularity of Linux, many third parties contributed to implement new drivers in the system.

This study is of real interest for the researcher. It opens new paths of research on the evolution model. Showing that the classic model for closed development fails in OSD. It also demonstrates the necessity of subsystem analysis for a large project analysis.

Reviewer Name: Laurent Eschenauer

Dennis R. Goldenson, Anandasivam Gopal, Tridas Mukhopadhyay, "Determinants of Success in Software Management Programs: Initial Results", 6th International Symposium on Software Metrics, 1999, Page(s) 10-21.

Existed Abstract:

"While a great deal is known about technical issues of data gathering and applied statistics, less is known about what it takes to implement a successful software measurement program. Indeed, a good deal of anecdotal evidence suggests that such efforts often fail. In this paper, we report the initial results from a large-scale survey of practitioners and users of software measurement programs. A preliminary multivariate analysis examines differences in the use of software measurement results in organizational decision making. Three variables (alignment with intended users, management commitment and use of analytic methods) account for two thirds of the observed variance. "

Modified Abstract:

The study is about determining what it takes to implement a successful software measurement program. By success, the authors mean to what extent are measurement and analysis regularly used to inform management and technical decision making and to what extent can improvements in an organization's performance be attributed to the use of measurement and analysis in that organization. An active program of measurement and analysis is often regarded critical to the success of software development, maintenance, and acquisition efforts. The study should be of value to practitioners and users of software measurement programs. The study collected data from a broad based survey of practitioners and users of software measurement programs administered via the World Wide Web. The sample of 228 includes representatives of defense and other government organizations, defense contractors, and commercial enterprises to ensure sufficient variation to allow multivariate analyses. This study examined three explanatory variables: 1) alignment of the measurement program with wider business and organizational goals; 2) organizational commitment and resource sufficiency; and 3) the technical characteristics of the measurement program itself. The three predictor variables that most strongly relate to the study's proximate criterion of success in implementing software measurement programs are 1) user stakeholder involvement in setting the organization's measurement agenda, 2) management commitment, and 3) the extent of use of varying data and analytic methods. These three variables account for two thirds of the observed variance. While these results may seem fairly intuitive, this study added a better quantitative description than was available previously.

Reviwer; Kanta Jiwnani

R. Grable, J. Jernigan, C. Pogue, and D. Divis, "Metrics for Small Projects: Experiences at the SED", IEEE Software Magazine, Vol. 16, No. 3, March/April 1999, pp. 21-28

Article Abstract:

An organized, comprehensive metrics program can bring order to the chaos of small-project management and form the foundation for a concerted process improvement effort. The authors describe their experience in applying metrics to one such US Army organization.

Reviewer Abstract:

This paper is a case study of software development at the Software Engineering Directorate (SED) of the US Army Missile Command. This group designs, builds, and maintains small embedded applications consisting of 10,000 to 50,000 lines of source code. The authors describe how they applied metrics to these projects, and the results they achieved. The paper characterizes the metrics and the process for collecting the metrics at this organization. Before undertaking a coordinated effort for its software metrics process, SED had inaccurate, inconsistent metrics data on its software projects. Little or no validation was done on the data that was collected. The main goal of the study was to show productivity increases at the SED as a result of the software improvement efforts. The four basic metrics measured were effort, schedule, defects, and size. The authors calibrate a cost model similar to Barry Boehms Constructive Cost Model (Cocomo) using the data they collected. This plots effort in person-months vs. size in lines of code. They also plot development time vs. effort. For both these measures, the authors develop the best nonlinear fit to the data. The effort equations in the SED data produced exponents in the effort equation less than one. With exponents less than one, the model predicts that larger projects cost less per line of code than do small projects. This contradicts the original Cocomo model's "diseconomy of scale" findings, which had an exponent greater than one. The statistical modeling was based on small sample sets, where N was less than 30. The output of the metrics effort is a series of graphs and charts of size, defects, and schedule metrics that aid project leaders in managing their projects. The data showed an increase in productivity after implementing the metrics collection process. This study would be of interest to managers in government organizations, to serve as an example of how a systematic measurement process can improve productivity.

Reviewer: Neal Bambha

Todd L. Graves, Mary Jean Harrold, Jung-Min Kim, Adam Porter, and Gregg Rothermel , "An empirical study of regression test selection techniques", ACM Transactions on Software Engineering and Methodology, Volume 10 , Issue 2 (2001), Pages 184-208.

Article Abstract:

Regression testing is the process of validating modified software to detect whether new errors have been introduced into previously tested code and to provide confidence that modifications are correct. Since regression testing is an expensive process, researchers have proposed regression test selection techniques as a way to reduce some of this expense. These techniques attempt to reduce costs by selecting and running only a subset of the test cases in a program's existing test suite. Although there have been some analytical and empirical evaluations of individual techniques, to our knowledge only one comparative study, focusing on one aspect of

two of these techniques, has been reported in the literature. We conducted an experiment to examine the relative costs and benefits of several regression test selection techniques. The experiment examined five techniques for reusing test cases, focusing on their relative ablilities to reduce regression testing effort and uncover faults in modified programs. Our results highlight several differences between the techiques, and expose essential trade-offs that should be considered when choosing a technique for practical application.

Reviewer Abstract:
Regression testing is the process of applying test cases to evaluate correctness of a change made to a software system. There are cases when application of the whole test suite is too expensive. There are three basic families of test selection techniques in the literature that finds a subset of the test cases which is relevant to the kind of modification.

The paper aims to evaluate minimization, dataflow, safe, random and retest-all techniques. The cost of a technique is defined by the reduction in the number of test cases, therefore it is assumed that test cases have similar costs. The amount of analysis cost to come up with a candidate subset is omitted. Another important variable shows reliability and defined as one minus the ratio of defects failed to be discovered by the selected subset but could be found by using the whole test suite.

The experiments were carried on 9 programs written in C varying in size from 138 lines to 49316 lines. 7 of the smaller programs come from a previous experiment where bugs were inserted deliberately. The two other programs are "space", an interpreter for an array definition language and "player", a large subsystem of internet game Empire. They have been subjects of other studies in the literature.

The results of the experiments pointed out relative strengths of different techniques and lays out a primitive guide to test selection. Specifically it was found out that minimization is unreliable but relatively cheap. Dataflow and safe techniques show similar results, but the former is more expensive to compute.

Reviewer Name: Cengiz Celik

Andrew R. Gray, Stephen G. MacDonell, "Software Metrics Data Analysis: Exploring the Relative Performance of Some Commonly Used Modeling Techniques", International Journal of Empirical Software Engineering, Volume 4, Number 4, December 1999, pp. 297-316

Article Abstract:

Whilst some software measurement research has been unquestionably successful, other research has struggled to enable expected advances in project and process management. Contributing to this lack of advancement has been the incidence of inappropriate or non-optimal application of various model-building procedures. This obviously raises questions over the validity and reliability of any results obtained as well as the conclusions that may have been drawn regarding the appropriateness of the techniques in question. In this paper we investigate the influence of various data set characteristics and the purpose of analysis on the effectiveness of four model-

building techniques--three statistical methods and one neural network method. In order to illustrate the impact of data set characteristics, three separate data sets, drawn from the literature, are used in this analysis. In terms of predictive accuracy, it is shown that no one modeling method is best in every case. Some consideration of the characteristics of data sets should therefore occur before analysis begins, so that the most appropriate modeling method is then used. Moreover, issues other than predictive accuracy may have a significant influence on the selection of model-building methods. These issues are also addressed here and a series of guidelines for selecting among and implementing these and other modeling techniques is discussed.

Reviewer Abstract:

Whilst some software measurement research has been unquestionably successful, other research has struggled to enable expected advances in project and process management. Contributing to this lack of advancement has been this incidence of inappropriate or non-optimal application of various model-building procedures. This obviously raises questions over the validity of any research obtained as well as the conclusions that may have been drawn regarding the appropriateness of the techniques in question. In this paper we investigate the influence of various data set characteristics and the purpose of analysis on the effectiveness of four model-building techniques -- three statistical methods and one neural network method. In order to illustrate the impact of data set characteristics, three data sets, drawn from the literature, are used in this analysis. In terms of predictive accuracy, it is shown that no one modeling method is best in every case. Some consideration of the characteristics of data sets should therefore occur before analysis begins, so that the most appropriate modeling method is then used. Moreover, issues other than predictive accuracy may have significant influence on the selection of model-building methods. These issues are also addressed here and a series of guidelines for selecting among and implementing these and other modeling techniques is discussed.

The purpose of this study is to evaluate modeling methods employed in software engineering to build models for measuring software projects and processes, in particular statistical methods. Software development efforts are notorious for having a reputation of having cost and schedule overruns. Therefore, collected measures should lead to the development of models that allow for greater predictability and insight into ever increasing complex software development efforts. The authors note that for more accurate predictability in the model, data from the organization are used for comparison. This study uses data sets that were previously published for a cross-comparison of each of the following types of methods, that of Miyazki (1994), Li and Henry (1993), and Dolado (1997). The methods involved were of two types. The first was that of regression analysis, which consisted of least-squares, least-median-squares, and least-trimmed-squares. The second was that of neural networks, which is back-propagation as applied to feedforward. The study is aimed primarily at project managers, however, organizations at a higher level could make use of this study for examining and/or evaluating their projects. The authors emphasize that not only assessment of each methods advantages and disadvantages should be done, but also the amalgamation of two or more methods should be implemented more appropriate to the variables and environment at hand. The examination of these methods will go towards recommending the most appropriate model. The context of the study was that of a laboratory environment as the conclusion states that this work should be a precursor to more

rigorous analysis in the future. It should be noted that other issues not described in the paper such as data characteristics, expertise, and interoperability ought to be examined in conjunction with the aforementioned methods when modeling the software process.

Reviewer: Ronald Wilson

Harrold, M.J.; Rosenblum, D.; Rothermel, G.; Weyuker, E., "Title -Empirical studies of a prediction model for regression test selection", IEEE Transacations on Software Engineering, Vol. Year, Issue -Volume: 27 Issue: 3, March 2001, pp. 248-263

Existing abstract:

Regression testing is an important activity that can account for a large proportion of the cost of software maintenance. One approach to reducing the cost of regression testing is to employ a selective regression testing technique that: chooses a subset of a test suite that was used to test the software before the modifications; then uses this subset to test the modified software. Selective regression testing techniques reduce the cost of regression testing if the cost of selecting the subset from the test suite together with the cost of running the selected subset of test cases is less than the cost of rerunning the entire test suite. Rosenblum and Weyuker (1997) proposed coverage-based predictors for use in predicting the effectiveness of regression test selection strategies. Using the regression testing cost model of Leung and White (1989; 1990), Rosenblum and Weyuker demonstrated the applicability of these predictors by performing a case study involving 31 versions of the KornShell. To further investigate the applicability of the Rosenblum-Weyuker (RW) predictor, additional empirical studies have been performed. The RW predictor was applied to a number of subjects, using two different selective regression testing tools, Deja vu and TestTube. These studies support two conclusions. First, they show that there is some variability in the success with which the predictors work and second, they suggest that these results can be improved by incorporating information about the distribution of modifications. It is shown how the RW prediction model can be improved to provide such an accounting.

Your rating for this paper - high

Modified Abstract:

This paper presents results from new empirical studies that were designed to evaluate the effectiveness and accuracy of the Rosenblum-Weyuker (RW) model for predicting cost-effectiveness of a selective regression testing method. This study should be of value to project managers and testers in order to plan for regression testing which is an important activity that can account for a large proportion of the cost of software maintenance. Selective regression testing techniques reduce the cost of regression testing if the cost of selecting the subset from the test suite together with the cost of running the selected subset of test cases is less than the cost of rerunning the entire test suite. The RW model was originally framed solely in terms of code coverage information and evaluated empirically using the TestTube method and a sequence of 31 versions of KornShell. In the new studies in this paper, two selective regression testing tools were used (TestTube and DejaVu), and seven different C programs, that had been previously used in a study by researchers at Siemens corporate Research, were used as subjects. For the experimental subjects used in the new studies, the RW predictor was quite successful for both the DejaVu and TestTube selection method. However, the predictions of the model occasionally

deviated significantly from observed test selection results. Moreover, when individual modified versions are considered, the percentage of test cases for these programs, especially selected by DejaVu, varies significantly from the percentage predicted. These results suggest that, not only code coverage, but also the distribution of modifications made to a program can affect significantly in determining the accuracy of a predictive model of test selection. Therefore, to achieve improved accuracy both in general and when applied in a version-specific manner, prediction models must account for both code coverage and modification distribution.

Reviewer: Nuengwong Tuaychroen

H. Hofmann, F. Lehner, "Requirements Engineering as a Success Factor in Software Projects", IEEE Software Magazine, Vol. 18, No. 4, July/August 2001, pp. 58-66

Article Abstract:

Requirements engineering might be the single most important part of any software project. Based on their field study of 15 RE teams in nine software organizations, the authors identify the RE practices that clearly contribute to project success, particularly in terms of team knowledge, resource allocation, and process. They then summarize the best practices exhibited by the most successful teams.

Reviewer Abstract:

This paper examines the requirements engineering (RE) process in 15 RE teams at 9 software companies in the telecommunications and banking industries. The development projects studied were recently released critical business applications. The purpose of the study was to characterize the RE processes used by these teams, and to understand which RE practices clearly contribute to a software project's success. In this case, success was a weighted performance measure that considered three dimensions: quality of RE service, quality of RE product and process control. The study focused on three factors - team knowledge with respect to the application domain, resources allocated to the RE effort, and the RE process used. Stakeholders were questioned as to their perception of the RE team's knowledge, and a score was given by weighting the answers to the questions. Resources were measured by number of person-hours assigned to the RE effort at each project. In order to characterize the RE process used, stakeholders were questioned and the authors used the answers to fit the process into the appropriate category.

The main result of the study is a table, which shows, for each focus area, what were the best practices, the cost of introduction for the practice, and the key benefit realized. It was shown that successful projects allocate a significantly higher amount of resources to RE (28 percent) than the average project in this or previous field studies, and that successful projects expend these resources according to a well-defined process.

This paper would be of value for project managers and senior managers at software companies, since it demonstrates the effects, in real-world projects, of placing sufficient resources and maintaining good practices for RE.

Reviewer: Neal Bambha

Hsiang-Jui Kung; Cheng Hsu, "Software Maintenance Life Cycle Model", Proceedings of the International Conference on Software Maintenance, November 1998, pp. 113-121.

Article Abstract:
The Software Maintenance Life Cycle Model (SMLC) was developed providing a basis to help software maintenance planning. The management framework was developed based on the SMLC concept to operationalize the model. This paper provides results from two cases to validate the model and the framework.

Reviewer Abstract:
Entity = Model of maintenence phases of big software project
Attributes of interest = different phases in life cycle, ways to discern them from data
purpose = understanding - no prediction
stakeholder = software maintenance planner
context = corporate/government software with a long maintenance lifetime and multiple releases
background - life cycle model to describe software development, no analogous maintenance model

This paper concerns itself with a model of different stages in the maintenance phase of deployed software's life. The paper also provides a management framework based on the model to operationalize the life cycle concept.

The model seperates software maintenance into a "life cycle" consisting of four phases: introduction, growth, maturity, and decline. By tracking and classifying maintenance changes made to the system, the timeline of stages can be determined over the system's lifetime.

The purpose of the model is to provide close and effective management of the maintenance process. It is hoped that with a feasible model of software maintenance, the total cost of the system will decrease as managers are more effectively able to focus their attention.

This model is geared towards software maintenance planners, who up to now have had a fairly monolithic model of software maintenence. The model should allow them to accurately distribute their resources, depending on what stage of maintenance their project is in.

This paper was written in the context of large software projects created by corporate entities. The nature of the projects require a periodic release schedule, and guarantee that testing and maintenance last for periods of many months.

The authors contend that one of the main problems with software maintenance is that it is largely reactive; that management of maintenance activity lacks perspective or a "holistic view". It is assumed that by characterizing different parts of the maintenance phase, managers will have more control over effort spent to maintain the project.

Reviewer Name: David C Rafkind

Taghi M. Khoshgoftaar, Edward B. Allen, "A Comparative Study of Ordering and Classification of Fault-Prone Software Modules", International Journal of Empirical Software Engineering, Volume 4, Number 2, June 1999, pp. 159-186.

Article Abstract:

Software quality models can predict the quality of modules early enough for cost-effective prevention of problems. For example, software product and process metrics can be the basis for predicting reliability. Predicting the exact number of faults is often not necessary; classification models can identify fault-prone modules. However, such models require that "fault-prone" be defined before modeling, usually via a threshold. This may not be practical due to uncertain limits on the amount of reliability-improvement effort. In such cases, predicting the rank-order of modules is more useful. A module-order model predicts the rank-order of modules according to a quantitative quality factor, such as the number of faults. This paper demonstrates how module-order models can be used for classification, and compares them with statistical classification models. Two case studies of full-scale industrial software systems compared nonparametric discriminant analysis with module-order models. One case study examined a military command, control, and communications system. The other studied a large legacy telecommunications system. We found that module-order models give management more flexible reliability enhancement strategies than classification models, and in these case studies, yielded more accurate results than corresponding discriminant models.

Reviewer Abstract:

A convincing case for usefulness of quality prediction is the successful detection and prevention of problems in software modules, for instance the prediction of reliability based on software product and process metrics. The difficult task of predicting the number of faults can be eschewed by employing classification models. Instead of aiming for absolute figures, classification models attempt to discriminate between groups of modules based on their predicted degree of faultiness. Classification based models suffer from an inherent problem in that a threshold is usually employed in order to obtain the classification. This may lead to inaccurate results, due to the large variance in input data for this kind of problem. An alternative to a definite classification model is a rank model. This type of model outputs a rank for each module based on a given factor, e.g. the number of predicted faults. Such a model assumes an underlying quantitative software quality model as the basis for the prediction. The paper focuses on rank-based models, and explores their accuracy in comparison with discriminant-based classification models. The actual modeling technique for rank computation is based on stepwise multiple regression, optionally using principal components analysis to decrease the number of variables and improve stability. This model is evaluated against nonparametric discriminant analysis on two large-scale projects: a military command, control, and communications system, and a large legacy telecommunications system. They conclude that rank based models have several advantages over discriminant-based models: they avoid choosing an arbitrary threshold, facilitate user acceptance because a ranking is easier to understand than the result produced by a black box formula, and finally they provide better accuracy.

Reviewer: Vasile Gaburici

Jung-Min Kim, Adam Porter, Gregg Rothermel, "An Empirical Study of Regression Test Application", International Conference on Software Engineering, 2000, pp. 126-135.

Article Abstract:

Regression testing is an expensive maintenance process used to revalidate modified software. Regression test selection (RTS) techniques try to lower the cost of regression testing by selecting and running a subset of the existing test cases. Many such techniques have been proposed and initial studies show that they can produce savings. We believe, however, that issues such as the frequency with which testing done have a strong affect on the behavior of these techniques. Therefore, we conducted an experiment to asses the effects of test application frequency on the costs and benefits of regression test selection techniques. Our results expose essential tradeoffs that should be considered when using these techniques over a series of software releases.

Reviewer Abstract:

After modifying software developers want to know that unmodified code has not been adversely affected. When unmodified code is adversely affected it is called regression error.

Developers do regression testing to search for regression errors. The simplest regression testing is rerunning al existing tests. This method is simple but can be expensive. An alternative is regression test selection (RTS). With RTS a subset of tests are selected and rerun. But there is a cost-benefit tradeoff with RTS.

This paper advocates a hypothesis, which states that the amount of change made between regression testing sessions strongly affects the costs and benefits of different regression test selection techniques. This is because test selection techniques will select increasingly larger test suites and because these suites will become increasingly less cost-effective at finding faults.

The experiment performed to prove this hypothesis uses eight C programs with a number of modified versions and test suites for each program. Experiment has four independent variables.

1. The subject program (each program has a variety of modified versions)
2. The test selection technique.
3. Test suite composition
4. Test interval

This paper is mainly useful for researchers who want to learn the relation between test selection techniques, code modification and test intervals. Developers and project managers may not use it in real projects because the validity of the experiment's results is limited.

Reviewer: Orcun Colak

Lesley Pek Wee Land, Chris Sauer, Ross Jeffery, "The Use of Procedural Roles in Code Inspections: An Experimental Study", International Journal of Empirical Software Engineering, Volume 5, Number 1, March 2000, pp. 11-34.

Article Abstract:

Software inspections are important for finding defects in software products (Fagan, 1976; Gilb, 1993; Humphrey, 1995; Strauss and Ebenau, 1994). A typical inspection includes two stages: individual preparation followed by a group review with roles assigned to each reviewer. Research has shown that group tasks typically result in process loss (Lorge et al., 1958; Steiner, 1972). In software defect detection also, considerable defects found during individual preparation are subsequently not reported by the group (Porter and Votta, 1994; Porter et al., 1995, 1997; Land et al., 1997a, 1997b; Siy, 1996; Votta, 1993). Our objective is to study whether procedural roles (moderator, reader, recorder) affect group performance, particularly in terms of process loss. At the same time, the use of roles in software reviews has also not been empirically validated, although there are wide claims for their benefits. Procedural roles made a limited difference to group performance. Further analyses provide possible explanations for the results and a deeper understanding of how groups make their decisions based on individual reviewers? findings. Limitations of the research are discussed. We also suggest how procedural roles may greater impact group performance.

Reviewer Abstract:

The object of study from this paper is the process of making code inspections. The attribute of interest of code inspections is the use of roles. The purpose of the study is to evaluate whether procedural roles (moderator, reader, recorder) improve group performance in detecting errors and to test whether they do so in part through reducing unnecessary defect loss. The results of the study should be of value to practitioners as they guide them to select the use of roles or not when conducting code inspections. The paper revisits two experiments conducted on 101 different subjects in two consecutive years. The first group didn't use roles and the second did. Both groups appeared to have the similar background and experience. The subjects were third year science undergraduates undertaking a Software Engineering course at the time of the experiments. The paper describes the study, analyses the results and discuss the results. The paper concludes that review meetings that employed procedural roles make a difference to group reviews mainly by reducing the defect loss in meetings. As a collateral effect of the study, the authors have a better understanding of how review groups behave, in particular how decisions are made based on their individuals' findings. The paper also describes what the authors learned about how the review groups behave.

Reviewer: Patricia Costa

Patricia K. Lawlis, Kathryn E. Mark, Deborah A. Thomas, and Terry Courtheyn, "A Formal Process for Evaluating COTS Software Products", IEEE Computer Magazine, Vol. 34, No. 5, May 2001, pp. 58-63

Article Abstract:

A software product evaluation process grounded in mathematics and decision theory can effectively determine product quality and suitability with less risk and at lower cost than conventional methods.

Reviewer Abstract:

In this paper, a formal process for evaluating COTS software products is introduced. This process is using mathematics and decision theory to effectively determine product quality and suitability with less risk and at lower cost than conventional methods. Recently, government agencies and businesses often use COTS software products as their components of their system. However, there is no well-defined software evaluation approach to make right product choices. The authors propose the requirements-driven COTS product evaluation process (RCPEP) to ensure a quality outcome.

The RCPEP is different from ad hoc product evaluation in: RCPEP relies on user-defined requirements instead of a short list of evaluator-generated criteria, RCPEP identifies every product that possibly addresses the requirements instead of an arbitrarily chosen short list, RCPEP uses decision theory to make effective evaluation process, RCPEP uses strict controls to ensure that the evaluation does not compromise the validity of the results.

The PCPEP approach has been demonstrate in a case study that resulted in the selection of a product by the US Air Force for a large, component-based training management system. This study is useful for government agencies and businesses who plan to use COTS software to build their systems.

Reviewer: Zhiyun Li

Luigi Lavazza, Giuseppe Valetto, "Requirements-Based Estimation of Change Costs", International Journal of Empirical Software Engineering, Volume 5, Number 3, November 2000, pp. 229-243

Original Abstract:

We present a case study that aims at quantitative assessment of the impact of requirements changes, and quantitative estimation of costs of the development activities that must be carried out to accomplish those changes. Our approach is based on enhanced traceability and an integrated view of the process and product models. The elements in the process and product models are quantitatively characterised through proper measurement, thus achieving a sound basis for different kinds of sophisticated analysis concerning the impact of requirements changes and their costs. We present the results of the application of modeling and measurement to an industrial project dealing with real-time software development. The ability to predict the impact of changes in requirements and the cost of related activities is shown.

Reviewer Abstract:

This article describes a method of estimating the cost of change requests, using a description of the request change as the primary source of estimation. While other methods of estimating costs rely on analyzing the change request in terms of function points or lines of code, the method described instead uses the actual change request itself.

The method relies heavily on the original implementation of the requirements and resulting artifacts. The requirements must thoroughly describe the product at a fine level of granularity, the code should be highly modularized, and traceability relations between the artifacts and requirements must have been be maintained. In the project studied, each requirement was uniquely labeled, and a custom database application has been written to maintain the relationship of the labels with the artifacts.

The authors used the method on a real-time, safety-critical system, where each module was implemented by a 1 or 2 person team. Only limited data was available for study, in particular, code size measures, subjective measures of code complexity, and defects discovered in the testing process.

The method divides the process into three phases, the "requirements understanding phase", the "implementation" phase, and the "testing" phase. The implementation and testing phases are further divided into an "impact" function, where the scope of the necessary changes are determined, and a "cost" function, where the effort cost is estimated. These phases are then combined to yield the cost estimate.

The results of the case study indicate that the method gave reasonably good estimates of the cost of change requests. By basing the estimation method on the change requests, an early estimation of the effort/cost is possible.

This paper is aimed at characterizing and evaluating a change request cost estimator. While the range of applicability would seem to be rather narrow, it could be useful to managers in that range.

Reviewer: David P. Steelman

Lindvall, M, "Measurement of change: stable and change-prone constructs in a commercial C++ system", Sixth International Software Metrics Symposium 1999, pp. 40-49

Existed Abstract:
"Our previous studies of developers' ability to predict software change revealed a great potential for improvement of the change management process. What we consider most beneficial is characterizing and understanding software change by measuring it and identifying what kind of changes take place and how frequent they are. With such knowledge, it is possible to build change models that help developers make better predictions regarding future requirements. An analysis is presented of a commercial object-oriented system-the PMR (Performance Management traffic Recording) system of Ericsson Radio Systems-that was changed significantly due to the implementation of a set of new requirements. Measures are used in order

to identify stable and change-prone constructs of the system. The results from the analysis are complemented with results from interviews with developers about what changes most frequently in a C++ system. "

Modified Abstract:
The study of a industrial object oriented PMR(performance management traffic recording) is being done in this paper. An analysis of this system is presented with the context that this system was changed significantly due to the implementation of a set of new requirements to its original form. This system operates and supports systems of Ericsson's cellular telecommunications system.

This study is part of a larger study that monitors, measures, and analyzes current practices at the workplace. This is an empirical study in the area of change management related to systems developed in C++. A look at what changes in the source code structurally with a complementary study - a survey in which developers indicated their perception of how frequently certain kinds of changes occur.

The goal is to better understand what constructs are stable and remain unchanged and what constructs are change prone and change frequently. The results from the analysis are complemented with results from interviews with developers about what changes occur most frequently in an object oriented system.

The source codes of the two releases R3 and R4 of the system This study aims to understand how to manage change for a system developed using object-oriented technique, to better understand the of the kinds of changes that are made to modern commercial software systems. This study helps us understand how to make improvements in predicting future changes and how to design new systems that are resilient to change. The results from this study will be used as input to the next step, which is to develop methods for linking change measures to effort.

Reviewer: Nitin Dinesh Anand

Yu-Liang Liu, Yue-Li Wang, and D. J. Guan, "An Optimal Fault-Tolerant Routing Algorithm for Double-Loop Networks", IEEE Transactions on Computers, Vol. 50, No. 5, May 2001, p. 500

Article Abstract:

A weighted double-loop network can be modeled by a directed graph G(n; h1, h2; w1, w2) with vertex set Zn = {0, 1, , n-1} and edge set E = E1 U E2, where E1 = {(u, u + h1) | u in Zn}. Assume that the weight of each edge in E1 is w1 and the weight of each edge in E2 is w2. In this paper, we present an optimal routing algorithm on double-loop networks under the case where there is at most one faulty element. Our algorithm is based on the fact that the shortest path from a vertex to any other vertex in a double-loop network is in the L-shape region.

Reviewer Abstract:

This study improves upon a previously existing algorithm used for routing messages in a double-loop network under fault conditions. A double-loop network, G(n; h1, h2; w1, w2), is a directed graph consisting of two sets of edges, E1 and E2, connecting vertices in a circular fashion. For each vertex u in the graph, E1 consists of edges (u, u+h1) where h1 is a positive integer. Similarly, E2 consists of edges (u, u+h2) where h2 is a positive integer. The addition in the previous statements is modulo n, where n is the number of vertices in the graph. Edges in edge set E1 have weight w1 and edges in edge set E2 have weight w2, where w1 and w2 are both positive integers. The previously existing algorithm was developed by Guan and is optimal in the absence of a fault. The algorithm has a fault-mode which succeeds in routing messages but can be sub-optimal. The paper gives an example of a case in which Guan's algorithm gives a sub-optimal routing result. The sub-optimal result occurs for a double loop network G(14; 4, 3; 1, 2) for source vertex 0, destination vertex 8, and faulty vertex 4. The paper goes on to show how to obtain the routing parameters P, Q, S, and T. Using these parameters, the paper presents its algorithm and proves that its algorithm is optimal. This study is useful for researchers and those interested in implementing double-loop network architecture.

Reviewer: Suresh Aryangat

Marcos, M.; del Pobil, A.P.; Moisan, S., "Model-based verification of knowledge-based systems: a case study", Software, IEEE Proceedings-Vol. 147 October 2000 Issue 5 Pg.163-167

Article Abstract: As knowledge-based systems become a standard in software development. The interest in verification and validation techniques to ensure their reliability has grown. For this purpose the verification of knowledge bases is fundamental. Numerous references in the literature involve verification by detection of implementation-dependent anomalies in the knowledge base such as inconsistency, incompleteness and redundancy. This approach has several limitations. The main one is the lack of significance of these anomalies in relation to the task of the system. To overcome this problem an approach that exploits the information on the organization of the knowledge involved in the task, and of the utilization of this knowledge for solving the task is proposed. This approach referred to as model-based verification is described, and a case study on the verification of classical planning systems is presented.

Reviewer Abstract: As knowledge-based systems become a standard in software development, the interest in verification and validation (V&V) techniques adapted to their particularities have grown. The use of implementation-dependent anomalies for verification have some problem, the main problem is their lack of significance in relation to the task that the system is required to perform.

To overcome the problem ,In this paper we describe our approach, which we have referred to as model-based verification , and present a case study on the verification of classical planning systems. We illustrate our approach with a knowledge-level analysis of different planning systems, together with a collection of model-based verification issues that we have identified from it. In spite of the simplicity of the analyzed systems, the case study illustrates the different kinds of model-based issues that can be used for verification.

In this paper we have presented a case on the verification of planning systems that describes the benefits of our model-based approach. It consists in the verification of properties defined at the knowledge level. Such properties make reference to the task and use the terminology of the task.

Reviewer Name: Liu Hanbin

McDermid, J.A.; Bennett, K.H., "Software engineering research: a critical appraisal", Software, IEEE Proceedings- Vol. 146, Issue 4, August 1999, Pg 179-186

Article Abstract: It is argued that computer science and software engineering should be regarded as separate disciplines, and that the long term success of academic computer science departments will only be secured by adopting a stronger engineering stance in research, enhanced by closer links with industry. A number of ways are suggested to revitalize the software engineering community and forge stronger links with industry, particularly through a better and more coherent research base.

Reviewer Abstract: It is becoming more widely recognized that computer science and software engineering are separate disciplines. The primary motivation behind this document is to address the common criticism that academic work in computer science is of little direct relevance to industry. We identify below some ways to focus the software engineering research community and to start to bridge this gap.

There are several definitions of software engineering and computer science. We conclude the computer science and software engineering is related disciplines, but that they are very different in character and they require very different research methodologies.
A tenet of this paper is that research in CSD should be industrially relevant, at least more so than it is now. We outline some of our reasons for holding this belief. We also argue that greater relevance will only come about through a change in attitudes in CS departments in universities, and through the development of a mope coherent and extensive SE research community. Then we consider steps which we believe are necessary for the creation and expansion of such a community. In subsequent sections we outline what we consider should be the technical aims of an expanded and revitalized SE research programmer, then discuss means of realizing such a programmer before drawing conclusions.

By way of conclusion we would stress three points:
(i) SE is a distinct discipline of strategic importance; it cannot legitimately be viewed as an aspect of CS.
(ii) Having a strong and effusive SE research programs important to many developed and developing nations especially to international competitiveness.
(iii) Establishing a strong community is a long terms challenge -- but action is needed now whilst there is E research base on which to build.

Reviewer Name: Liu Hanbin

Osamu Mizuno, Tohru Kikuno, Yasunari Takagi, Keishi Sakamoto, "Characterization of risky projects based on project managers' evaluation", International Conference on Software Engineering, 2000, pp. 387-395

Original Abstract:

During the process of software development, senior managers often find indications that projects are risky and take appropriate actions to recover them from this dangerous status. If senior managers fail to detect such risks, it is possible that such projects may collapse completely.

In this paper, we propose a new scheme for the characterization of risky projects based on an evaluation by the project manager. In order to acquire the relevant data to make such an assessment, we first designed a questionnaire from five viewpoints within the projects: requirements, estimations, team organization, planning capability and project management activities. Each of these viewpoints consisted of a number of concrete questions. We then analyzed the responses to the questionnaires as provided by project managers by applying a logistic regression analysis. That is, we determined the coefficients of the logistic model from a set of the questionnaire responses. The experimental results using actual project data in Company A showed that 27 projects out of 32 were predicted correctly. Thus we would expect that the proposed characterizing scheme is the first step toward predicting which projects are risky at an early phase of the development.

Reviewer Abstract:

This article presents a method of identifying "risky" projects. A "risky" projects is intuitively defined as a project that "appeared to be out of control" from the viewpoint of senior management. The author's notion of "risky" is similar to Yourdon's more catchy "death march project", although the authors note that many of the projects defined as "risky" were in fact successfully completed. The method discussed attempts to quantitative characterizations of these intuitive notions. This may be useful to project managers, and does not seem to be overly restricted to a particular domain or development method.

The method consists of a questionnaire given to project managers. The questions were grouped into five "viewpoints": Requirements, Estimations, Team Organization, Planning Capability, and Project Management activities. The questions were answered using responses from the Likert scale: "Strongly agree", "Agree", "Neither agree nor disagree", "Disagree". The questionnaire is fully described in the paper.

The projects studied were drawn from the work at a single company, whose main products are software for ticket vending machines, ATM terminals, and point-of-sale systems. Each project studied was developed using the waterfall model. The authors had two sets of projects to analyze. Both sets were historical, i.e. the projects had already been completed (or abandoned, in case of failed projects). It is the intended that this method be used on "live" projects, and there is some discussion of the possible implications.

The first set of 32 projects was used to derive a baseline of what constituted a "risky" project. A "risky" project was quantitatively defined as a project that exceeded both cost estimates and time estimates by fixed percentages (the exact percentages were not presented). The questionnaires were then analyzed using logistic regression to find correlations between "risky" and "no problem" projects. The authors were able to find correlations which enable them to accurately classify 27 of the 32 projects correctly, and 9 out of 10 of the actual risky projects.

The authors then applied their method predictively to a second set of 8 projects. In this test, they successfully predicted the status of all projects (5 "no problem" projects, 3 "risky" projects). The authors are careful to state, however, that such accuracy could not be expected in "live" projects. They attribute their "idealized" result to the fact that as these were all completed projects, the managers to whom the questionnaire was distributed tended to respond more honestly than might be the case in a live project.

Morisio, M.; Stamelos, I.; Spahos, V.; Romano, D., "Measuring functionality and productivity in Web-based applications: a case study", Sixth International Software Metrics Symposium 1999, pp. 111-118

Existed Abstract:
"We explore the variation of the cost of writing code when object oriented framework based development of Web applications is encountered for the first time. Managers need such information to justify their investments in innovative development strategies. Size measurements are essential in this task and a number of metrics, namely: lines of code, classical function points, and object oriented function points, are employed. It is argued that lines of code and object oriented function points are more suitable in this case. Data analysis reveals that learning influences mainly the cost of writing new code, consisting of continuous calls to components provided by the framework. We also explore the applicability of an already proposed effort prediction model that is based on different reuse types. A cost estimation model is the by-product of this study, providing a helpful tool for managing the first projects in which the framework is employed."

Modified Abstract:
This paper describes a study for which the purpose is to construct a cost estimation model that takes into account the learning factor for adopting Object Oriented frameworks. The model is intended to assist managers in predicting and tracking development costs during the first exploratory projects that have adopted the framework by calculating the relative cost of writing the new code under the framework. This study is intended to be valuable to managers who are asked to invest in technology adoption, direct such software development projects, monitor productivity growth, and estimate return on investment for framework adoption. The authors study the impact of the learning effect, which results in lower productivity initially due to the need for the designer to understand the complex class hierarchies and objects embodied in the framework in order to use the framework effectively.

This study utilizes an experiment from a companion paper; using empirical data collected from the development of five web-based applications. These applications were developed using an

object-oriented framework composed of a service platform to abstract common functions from a network, and components to abstract control services. The development of applications consisted of the verbatim reuse of the framework plus the development of new code (Java), which implements calls to the components provided by the framework. The paper does not describe whether the development work was done in an industrial or academic setting.

The attributes of the cost model are size, reuse level, effort (number of person hours), and the learning rate parameter. Three different size metrics were investigated: Function Points, Object-Oriented Function Points, and Lines of Code. Function Points is not a useful metric because it does not distinguish between new code and reused code. Reuse level is equal to the size of the reused software divided by the size of the total software. The learning rate parameter is calculated based on the project data and validated against the Wright curve, the earliest industrial learning curve representation. The model shows that for the first applications developed, the relative cost of writing new code decreases exponentially as the number of cumulative projects increases.

Reviewer: Kathleen Dangle

F. Niessink and H. van Vliet, "Two case studies in measuring software maintenance effort", Proceedings of the International Conference on Software Maintenance, November 1998, pp. 76 - 85.

Article Abstract:
In this paper we present the results of two measurement programs, that were aimed at investigating possible cost drivers for software maintenance. The two measurement programs were implemented in the software maintenance departments of two different organizations. Both programs were set up in roughly the same way. We use standard statistical techniques-principal component analysis and multiple regression analysis-to analyse the datasets. Surprisingly, with one of the datasets we are able to explain a fair amount of variance in the effort, while with the other dataset we can explain much less. From a closer inspection of the different environments we conjecture that the existence of a consistently applied process is an important prerequisite for a successful measurement program. In addition, if the process exists in multiple variants, it is important to know which variant is applied when.

Reviewer Abstract:
This paper presents the result of a measurement study whose goal was to gain insight into maintenance cost drivers, in order to support the estimation and planning of software maintenance tasks.

This research was performed in two different organizations. Both were immature in that there was little or no a priori qualitative or quantitative data on maintenance cost drivers. More over none of the organization applies one defined, maintenance process. Two different students designed the measurement program, each of them implemented a study program to characterize the maintenance process, determine likely maintenance cost drivers, collect data, etc...

The data then was analyzed through principal component analysis and regression analysis. Those methods were useful to explore the relationships between characteristics of change requests and the effort to implement them.

The method worked well for one organization but not the other. It appeared that organization B is completely lacking of standardization in maintenance process, it is well possible that this lack of standard has a large impact on the usefulness of the data gathered.

This paper is aimed at the corporation trying to implement a maintenance process. It will help the company understand the necessity of standards. It will also helps the researcher with new data sets and application of different statistical tools.

Reviewer Name: Laurent Eschenauer

Barshar Nuseibeh, "Weaving Together Requirements and Architectures", IEEE Computer Magazine, Vol. 34, No. 3, March 2001, pp. 115-117

Article Abstract:

Twin Peaks intertwines software requirements and architectures to achieve incremental development and speedy delivery.

Reviewer Abstract:

In software development, people often start from one of two points: requirements or architectures. This waterfall model limits the process of development in many ways.

In this paper, the twin peaks model is introduced to intertwine software requirements and architectures to achieve incremental development and speedy delivery. This model is based on the author's experience in industrial software-development projects and it is an adaptation of the spiral life-cycle model. This model gives equal status to requirements and architectures. What makes this model different from other models is that while this model develops requirements and architectural specifications concurrently, it continues to separate problem structure and specification from solution structure and specification, in an iterative process that produces progressively more detailed requirements and design specifications. In this way, the model develops more detailed requirements and architectural specifications concurrently. This model addresses three management concerns: IKIWISI, COTS and rapid change.

The purpose of this study is to understanding the existing, but implicit, state of the practice in software development. This model is especially useful for project managers to obtain more insight of a process of development.

This study is based on accepted research in its evolutionary development. Before this model, software-development organizations often use waterfall model. Then, the spiral life-cycle model is introduced to addresses many drawbacks of waterfall model. It provides an incremental development process. The twin peaks model improves the spiral life-cycle model further.

Reviewer: Zhiyun Li

Ohlsson, M.C.; Wohlin, C., "An empirical study of effort estimation during project execution", Sixth International Software Metrics Symposium, 1999, pp. 91-98

Existed Abstract:

"Presents an empirical study of effort estimation in software engineering projects. In particular, this study is focused on improvements in effort estimations as more information becomes available. For example, after the requirements phase, the requirements specification is available, and the question is whether the knowledge regarding the number of requirements helps in improving the effort estimation of the project. The objective is twofold. First, it is important to find suitable measures that can be used in the re-planning of the projects. Second, the objective is to study how the effort estimations evolve as a software project is performed. The analysis is based on data from 26 projects. The analysis consists of two main steps: model building based on data from part of the projects, and evaluation of the models for the other projects. No single measure was found to be a particular good measure for an effort prediction model; instead, several measures from different phases were used. The prediction models were then evaluated, and it is concluded that it is difficult to improve effort estimations during project execution, at least if the initial estimate is fairly good. It is, however, believed that the prediction models are important for knowing that the initial estimate is of the right order, i.e. the estimates are needed to ensure that the initial estimate was fairly good. It is concluded that the re-estimation approach will help project managers to stay in control of their projects. "

Modified Abstract:

This paper presents an empirical study of effort estimation, focused on evaluating the usefulness of using proxies. A proxy is an indirect measure of the entity of interest, which is project size in this paper. Proxies can help in creating mental pictures and provide another level of abstraction when estimating project effort. The data used in the study comes from 26 similar projects from 1996 to 1998 software development courses conducted at the department of Communication Systems, Lund University, Sweden.The problem is to find suitable proxies, which reflect and correlate with project effort. Base on the data from 1996 and 1997, the study includes eight proxies (requirements, tests, sub-test included, processes, flowcharts outputs, inputs, and total signals) from three different documents (Software Requirement Specification, Software Verification and Validation Plan, and Software Top-Level Design Document). The documents relate to different phases. The paper analyzes the proxies by using a correlation analysis and a principal component analysis. The analysis results show that it is difficult to find a proxy with good correlation to project effort because seven of the projects used to build the models had negative correlation between the proxies and the effort. Since the study cannot identify one superior proxy, the authors decide to investigate three of the proxies-- requirements, tests, and outputs, which are from different documents. By using multiple linear regression analysis with those three proxies, the results show that the mean value and standard deviation of the estimation error do not change significantly, as more information becomes available. This is probably due to that the initial estimate is fairly good, and the uncertainty remains throughout the project. After re-planning and comparing with 1998 projects, the paper recommends using proxies as a means for supporting project managers confidence in the effort estimation in order to control software

projects. The paper's proxy-based approach provides the opportunity to regularly re-estimate software projects to ensure that they proceed according to plan.

Reviewer: Nuengwong Tuaychroen

Andy Podgurski, Wassim Masri, Yolanda McCleese, Francis G. Wolff and Charles Yang, "Estimation of software reliability by stratified sampling", ACM Transactions on Software Engineering and Methodology, Volume 8 , Issue 3 (1999), Pages 263-283

Article Abstract:
A new approach to software reliability estimation is presented that combines operational testing with stratified sampling in order to reduce the number of program executions that must be checked manually for conformance to requirements. Automatic cluster analysis is applied to execution profiles in order to stratify captured operational executions. Experimental results are reported that suggest this approach can significantly reduce the cost of estimating reliability.

Reviewer Abstract:
The new approach of stratified sampling is suggested for the problem of operational testing and software reliability estimation. It involves instrumenting the software so that the inputs and execution profiles generated by beta testers are available for replay. The large number of execution profiles are recreated in a laboratory setting by utilizing the captured information. Cluster analysis is used to stratify the executions based primarily on control flow information. A stratified random sample is selected and rigorously checked by skilled personnel for conformance with requirements, and it is suggested that the number of executions that need to be manually examined is reduced as compared to other approaches. Ultimately, a measure of the software’s reliability is obtained by using variance estimates.

The technique is tested using five small size student projects and a medium size commercial project scheduling system, all of which demonstrate a significant improvement over random sampling techniques. The need for extensive further research to analyze the efficacy of the approach is acknowledged, and the presented examples merely illustrate the feasibility of the approach.

On the whole, the study aims at improving operational testing and reliability estimation of software by suggesting a new technique. This paper would be of importance primarily to researchers since the approach needs to be tested in larger practical environments, and of some value to commercial testing teams who may be able to induct some of the methods into their own testing procedures.

Reviewer Name: Abhijit Ogale

Adam Porter, Harvey Siy, Audris Mockus and Lawrence Votta, "Understanding the sources of variation in software inspections", ACM Transactions on Software Engineering and Methodology, Volume 7, Issue 1 (1998), Pages 41-79.

Article Abstract:

In a previous experiment, we determined how various changes in three structural elements of the software inspection process (team size and the number and sequencing of sessions) altered effectiveness and interval. Our results showed that such changes did not significantly influence the defect detection rate, but that certain combinations of changes dramatically increased the inspection interval. We also observed a large amount of unexplained variance in the data, indicating that other factors must be affecting inspection performance. The nature and extent of these other factors now have to be determined to ensure that they had not biased our earlier results. Also, identifying these other factors might suggest additional ways to improve the efficiency of inspections. Acting on the hypothesis that the "inputs" into the inspection process (reviewers, authors, and code units) were significant sources of variation, we modeled their effects on inspection performance. We found that they were responsible for much more variation in detect detection than was process structure. This leads us to conclude that better defect detection techniques, not better process structures, are the key to improving inspection effectiveness. The combined effects of process inputs and process structure on the inspection interval accounted for only a small percentage of the variance in inspection interval. Therefore, there must be other factors which need to be identified.

Reviewer Abstract:
This paper provides more analysis on a previous experiment on improving software inspections. The original study studied the affects of varying process structures. The variables under study was number of individuals in the team, the number of meetings, and whether the teams worked independently.

The original study concluded that the structure of inspection process had little to do with the number of defects it uncovered, and in most cases the extra time period a particular process type induced wasn't worth the extra defects found. The results therefore called for focusing on better inspection techniques.

The experiments were done on six team members working on a compiler for a telephone switching system at Lucent Technologies. 5 other developers joined them from time to time. The developers all had similar backgrounds and had been working for the same company for at least 5 years.

This study focuses on input parameters of inspections to find out if they can explain the large amounts of variance observed in the original study. A level of verification of the first analysis's conclusions can be done buy finding out if they were biased by these extra parameters.

The input parameters under study were code size, author of the code, development phase, functionality of the software, performance of individual reviewers, preparation time and meeting duration. By using these variables, the authors try to come up with a model that contains as few basic variables as possible. The paper shows that it is possible to explain the performance variance of software inspections by input parameters to a satisfactory extent. Furthermore, they verified that their original experiment wasn't biased by these external factors.

Reviewer Name: Cengiz Celik

Ramesh, B.; Jarke, M., "Toward reference models for requirements traceability", IEEE Transactions on Software Engineering, Vol. Year, Issue -Volume: 27 Issue: 1, Jan. 2001, pp. 58-93

Existing abstract:
Requirements traceability is intended to ensure continued alignment between stakeholder requirements and various outputs of the system development process. To be useful, traces must be organized according to some modeling framework. Indeed, several such frameworks have been proposed, mostly based on theoretical considerations or analysis of other literature. This paper, in contrast, follows an empirical approach. Focus groups and interviews conducted in 26 major software development organizations demonstrate a wide range of traceability practices with distinct low-end and high-end users of traceability. From these observations, reference models comprising the most important kinds of traceability links for various development tasks have been synthesized. The resulting models have been validated in case studies and are incorporated in a number of traceability tools. A detailed case study on the use of the models is presented. Four kinds of traceability link types are identified and critical issues that must be resolved for implementing each type and potential solutions are discussed. Implications for the design of next-generation traceability methods and tools are discussed and illustrated.

Modified Abstract:
Requirements tractability keeps in check if the development process is alignment with the requirements. These traces should have a fixed and well-defined framework. This paper takes a empirical approach to developing this framework, in contrast to the conventional way of modeling based on theoretical considerations or analysis.

This paper was sponsored by the US office of Naval Research project on engineering of complex systems along with other groups.

Reference models are standard systems that can be modified according to the application, and then analyzed. These reference models save huge costs in terms of being used instead of making the real system and making it from scratch.

This paper discusses the use of reference models for requirements tractability.
This paper presents the reference models together with their grounding in the empirical studies and relates the finding s to requirements for future tractability mechanism.

Tractability gives essential assistance in understanding the relationships that exist within and across software requirements design and implementation. These relationships allow designers to show that the design meets the requirements and help early recognition of those requirement not satisfied by the design,

One of the biggest challenges in managing large, complex systems is due the way requirements are constantly evolving and changing.

A simple Meta model of requirements tractability was developed that will define the language in which tractability models can be defined and can be customized within the scope defined by the meta model.

This tractability scheme should help document and understand the evolution of requirements. Empirical studies have been done in a broad range of software development organization to come up with reference models for the objects and tractability links to be recorded.

Reviewer: Nitin Dinesh Anand

Pierre N. Robillard, "The Role Of Knowledge in Software Development", Communications of the ACM, Vol. 42, No. 1, January 1999.

Article Abstract: Not Available

Reviewer Abstract:

An insight into various factors involved in software development is presented. Gaps between the viewpoints of software practitioners, software scientists and cognitive scientists are intended to be bridged. Basic idea about knowledge is presented and classification of knowledge into two major types, namely, procedural and declarative along with further classification of declarative knowledge into topic, semantic and episodic is presented and discussed at length. Relationship of software development to specific types of knowledge discussed above is studied with example. Notion of "schema", proposed in 1975, is introduced in the context of this literature and its applicability to the current study is thoroughly discussed. An example of operating system, to explain notion of schema, is given. The usefulness of the schema concept is presented by illustrating a scenario. Schema concept is emphasized by presenting its use in Artificial Intelligence.

Studying the concept of schema to a greater detail, and extending it to the subschema level, the notion of "proposition" is introduced. Concept of "proposition" is discussed at length. Well definedness and ill-definedness of a given problem along with the relationship of definedness to software design is presented. Appropriateness of formal specifications in software design and development is discussed considering various facts. Next, chunking phenomenon is explained and its relevance to software methodology is presented.

In the second part of the literature, planning, its properties and importance is discussed. The impact of planning on design process is presented with an example. Systematic planning vs serendipitous planning is analyzed. Recent study results on expert plan structures are presented. Usefulness of CASE tools in planning is listed. Finally, it is concluded that software development is processing of knowledge in a focussed way. Possible ways of achieving new directions in software engineering is discussed.

Reviewer Name: Vaddadi P Chandu

Chris Sauer, D. Ross Jeffery, Lesley Land, and Philip Yetton, "The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research", IEEE Transactions on Software Engineering, Volume 26, January 2000, Page(s) 1-14.

Existed Abstract:
Software engineers use a number of different types of software development technical review (SDTR) for the purpose of detecting defects in software products. This paper applies the behavioral theory of group performance to explain the outcomes of software reviews. A program of empirical research is developed, including propositions to both explain review performance and identify ways of improving review performance based on the specific strengths of individuals and groups. Its contributions are to clarify our understanding of what drives defect detection performance in SDTRs and to set an agenda for future research. In identifying individuals' task expertise as the primary driver of review performance, the research program suggests specific points of leverage for substantially improving review performance. It points to the importance of understanding software reading expertise and implies the need for a reconsideration of existing approaches to managing reviews.

Modified Abstract:
The entity under study is software development technical reviews (SDTRs) used for detecting defects in software products. SDTRs are characterized as: an organizational device for detecting defects in software products at any stage of the life cycle and for obtaining secondary benefits through a two-stage process in which software engineers first independently inspect the software product for defects and then combine their efforts in a group meeting in which the participants adopt roles with the goal of producing a report in which all the defects agreed upon by the group are identified. The purpose of this study is to understand the effectiveness of SDTRs in detecting defects and also to find out what makes them effective as they are, in order to establish a systematic basis for discovering ways to improve defect detection performance. This study applies the behavioral theory of group performance developed through laboratory studies on a two-staged task that includes an individual preparation stage and a stage during which the group meets. The laboratory study subjects were third year undergraduates. The study aims to answer three questions: First, what makes SDTRs effective at defect detection? Second, how can SDTR performance be improved within the current review design? And third what design alternatives would the theory predict to be more effective? The study concludes stating that the most important factor in determining effectiveness is the level of expertise of the individual reviewers. It highlights three ways of improving performance and also predicts more effective designs may be achieved by taking best advantage of relative merits of individuals and groups. It also suggests a three-stage design. The study provides the researcher with relevant research issues like the exploration and extension of authors' understanding of defect detection expertise. It also provides grounds for the manager to review certain established management assumptions about reviews and introduce new, more effective management practices.

Reviwer; Kanta Jiwnani

Srinarayan Sharma and Arun Rai, "CASE deployment in IS organizations", Communications of the ACM, Volume 43 , No. 1 (Jan. 2000) , p. 80

Article Abstract:

Reviewer Abstract:

The overall objective of the paper is to assess the impact of software engineering development tools (CASE) with an innovative set of measures for implementation. The context of the study is a national survey to collect the empirical data for the study. A principle component factor analysis was followed by a varimax rotation to emprically assess groupings of the tasks. The entity being studied is the constituent processes of production, coordination, and organization. The purpose of the study is to provide insights on demographic differences and similarities between adopters and non-adopters of CASE technology. The study should be of value to information systems managers who is interested in the productivity gains from CASE usage. The important results of this study include an innovative framework to measure the CASE usage and impacts in IS organizations. The resuts show that organizations have adopted CASE for only a few more than half of development tasks considered, and the use of CASE in these tasks are only in a small subset of development projects. It therefor explains the fact CASE technology has not shown much effect on system development performance or that there is no broad acceptance about the plausible impacts of the technology. Venders should examine the degree to which the support packaged with CASE tools plays a role, if any, in reducing knowledge barriers and increasing CASE usage.

Reviewer: Zhijian Pan

A. von Mayrhauser, C. Wohlin, M. C. Ohlsson, "Assessing and Understanding Efficiency and Success of Software Production", International Journal of Empirical Software Engineering, Volume 5, Number 2, June 2000, pp. 125-154.

Article Abstract:

One of the goals of collecting project data during software development and evolution is to assess how well the project did and what should be done to improve in the future. With the wide range of data often collected and the many complicated relationships between them, this is not always easy. This paper suggests to use production models (Data Envelope Analysis) to analyze objective variables and their impact on efficiency. To understand the effect of subjective variables, it is suggested to apply principal component analysis (PCA). Further, we propose to combine the results from the production models and the analysis of the subjective variables. We show capabilities of production models and illustrate how production models can be combined with other approaches to allow for assessing and hence understanding software project data. The approach is illustrated on a data set consisting of 46 software projects from the NASA-SEL database (NASA-SEL, 1992). The data analyzed is of the type that is commonly found in project databases.

Reviewer Abstract:

The study was performed to gain a better understanding of software productivity and project success. Since numerous factors influence productivity within a single project, it is very difficult to understand the sources of inefficiencies. The study was broken down into parts which included how production models work and how they analyze data, methods to analyze subjective data, an approach to combined analysis, and finally a case study using the combined analysis approach. The main objective of the case study was to illustrate how to use a combined approach that evaluates both productivity and success of projects based on objective and subjective variables. In using different types of methods to analyze a combination of objective and subjective project data, numerous purposes for the study could be derived, including understanding, evaluating, and predicting. Similar to the multiple purposes of this study, numerous entities were studied including process, model, and metrics.

This study if of value to not only researcher by analyzing objective and subjective data, but to a company and it's project managers who would value results in understanding how well a project did and what could be corrected in the future.

The case study was performed using data from 46 software projects collected by NASA-SEL. Although this was most likely some of the best data available, it still lacked quality quantitative data.

Although the combined analysis showed that there was much to learn from combining different analysis approaches, challenges existed. Combined analysis of production models and subjective variables provided a better understanding and assessment than could have been achieved by only one of the approaches. It would, however, be helpful to look deeper into project specifics to interpret the results.

Reviewer: Patricia Larsen

Fiona Walkerden, Ross Jeffery, "An Empirical Study of Analogy-based Software Effort Estimation", International Journal of Empirical Software Engineering, Volume 4, Number 2, June 1999, pp.135-158.

Article Abstract:

Conventional approaches to software cost estimation have focused on algorithmic cost models, where an estimate of effort is calculated from one or more numerical inputs via a mathematical model. Analogy-based estimation has recently emerged as a promising approach, with comparable accuracy to algorithmic methods in some studies, and it is potentially easier to understand and apply. The current study compares several methods of analogy-based software effort estimation with each other and also with a simple linear regression model. The results show that people are better than tools at selecting analogues for the data set used in this study. Estimates based on their selections, with a linear size adjustment to the analogue's effort value, proved more accurate than estimates based on analogues selected by tools, and also more accurate than estimates based on the simple regression model.

Reviewer Abstract:

The conventional approach taken to estimate software cost is based on algorithmic models, where the effort is the result of a computation based on a mathematical model. Analogy-based estimation, is considered by the authors an emerging and promising alternative to conventional approaches. Their claim is that analogy-based estimation has accuracy comparable with conventional methods, while being potentially easier to understand, being more useful where the domain is difficult to model and offering a chance to learn from past experience. This type of estimation involves two steps: an initial selection of a baseline project considered closely related to the target project, followed by a linear size adjustment to compensate for the size of the target. The size was estimated using function points. The paper compares two main methods for the selection of similar projects from a small database of projects: human selections versus software. The human subjects were trained, but otherwise inexperienced. Several software tools were tested, but they all relied on computing the Euclidean distance of several metrics to determine the closest analogue. The authors conclude that humans are better than tools when selection is to be performed from a small database. The main reason behind this was that human subjects picked only one or few characteristics for similarity, but intuitively choose those that induced the highest correlation in effort. Tools that used unweighted Euclidean distance gave unwarranted importance to some characteristics. A comparison between analogy-based estimation and a simple conventional linear regression model was also performed. Simple linear regression was found to be less accurate than analogy-based estimation.

Reviewer: Vasile Gaburici

David R. Webb, "Managing Risk with TSP", Crosstalk Magazine, June 2000, <http://www.stsc.hill.af.mil/crosstalk/2000/jun/webb.asp>

Article Abstract:

One of the most important aspects of applying the Team Software Process (TSPSM) to software projects of any size is the increased success of identifying, tracking, and mitigating risk. The Mission Planning Software Section of the Software Engineering Division of Hill Air Force Base (TISHD), has found the TSP's simple strategy for identifying, tracking, and handling risks to be extremely effective. In fact, many common software project risks are managed purely by adopting the TSP.

Reviewer Abstract

The article discusses the effective strategies of the Team Software Process for risk management. The development schedule consists of phases, and the risk assessment is performed at the launch of each phase. The goals, quality criteria and risks for the phase are identified, and the tasks are defined and estimated using the rigorous Personal Software Process (PSP). Management roles such as Design Manager, Customer Interface Manager, etc. are assigned to team members to monitor individual risk categories. An earned value plan is produced against which the project is tracked. Risks are reviewed in weekly meetings and issues which require immediate actions are identified.

An experienced TSP team can effectively handle the most common risks: schedule overruns, requirements creep and quality problems. Two projects called TaskView and A-10 AWE which

utilized the TSP principles are discussed. In addition to the above, TaskView experienced great reduction in risks associated with defects and test time. In the case of the A-10 AWE, the risk management and tracking strategies of the TSP worked well, but the task schedule estimates were highly inaccurate, since the engineers were not PSP trained. This identified the crucial importance of the PSP in data gathering, without which the TSP would not work.

This article describes the TSP process in brief, and its application to two commercial software projects, in a professional team environment. The article would serve as an outline of the TSP process for corporations and commercial software teams which are examining its utility. However, the article serves only an introductory purpose, and detailed references will have to be consulted if the TSP process is to be actually adopted.

Reviewer Name: Abhijit Ogale

Anders Wesslén, "A Replicated Empirical Study of the Impact of the Methods in the PSP on Individual Engineers", International Journal of Empirical Software Engineering, Volume 5, Number 2, June 2000, pp. 93-123.

Article Abstract:

The Personal Software Process (PSP) has during the last couple of years gained attention as a way to individual improvements in software development. The PSP is introduced to students and engineers through a course, which introduces a personal software development process. The personal software development process is improved in steps during the course and a collection of methods is introduced to support the personal development process. The question is, however, how do these methods influence the performance of an individual engineer? This question has been studied in a study made at the Software Engineering Institute, and the study has shown that the methods in the PSP have a positive effect on the performance of the individuals. There is however a need to replicate this study to confirm the findings in other settings and with other individuals. This paper describes a replication of the study made at the Software Engineering Institute. Both the original study and this replication are made on data reported from the students taking the PSP course. The differences between the two studies are the programming languages used, which held the courses, the class sizes, and the experiences of the students. In summary, the results from this replication confirm the results in the original study: Size estimation accuracy gets better, the defect density gets lower, the defects are found earlier and that the pre-compile yield gets better during the PSP course. Basically, the two studies show that the methods in the PSP help engineers to improve their performance.

Reviewer Abstract:

This study was a replication of a previously performed study of the Software Engineering Institutes' (SEI) Personal Software Process (PSP) in order to evaluate whether engineers performance improved with its use. The original study yielded positive results, but was questionable due to the objectivity: either the training or the study was performed at the SEI. The current study is of value to those who are users of PSP (current or potential) and to the SEI

since the PSP is their intellectual property and replication of the original study with similar results would be very favorable to selling their product.

PSP is a concept, framework, and a course. The concept is that engineers manage their own software development process, including measuring and analyzing performance for improvement. The framework consists of techniques and tools to accomplish the measurement and analysis. A course has been created at the SEI describing the process and was the focus of the original study by Hayes and Over (1997) being replicated. The course is designed to take a programmer through a 7 step development process for improvement. Steps are broken down to be major improvements or minor improvements. The baseline is PSP0.

The current study was performed using data collected during three different PSP courses: two graduate courses at Lund University and the third a PhD course at Linkoping University. Measurements were collected at each major step of the PSP, such as program size, development time, defects removed in phase, defect density, pre-compile defect removal yield, and productivity.

The initial study results showed improvement in numerous areas such as size and time estimation accuracy, overall defect density, and compile defect density. The replicated study showed very similar results to the original study. Basically, both studies support the objectives of the methods in the PSP.

Reviewer: Patricia Larsen

Karl Wiegers, "A Software Metrics Primer", Software Development Magazine(Online source), http://www.sdmagazine.com/documents/s=756/sdm9907b/

Article Abstract: Not Available

Reviewer Abstract:

This article presents the efficacy of using metrics to help find areas requiring improvement in a team, methods to use the metrics efficiently and means to implement them.

A general idea on various reasons to measure software, measurable quantities and drawbacks of not measuring software are discussed. Quantities that are measurable are presented quoting some of them. A method to select balanced a set of metrics for any particular organization, GQM(Goal-Question-Metric) method is presented and explained at length. A pseudo example is given to show how to work with the GQM method. A definite set of metrics and related groups, which should include any set of balanced metrics is tabulated followed by a brief description of those metrics.

The main hindrances to creating a measurement culture in an organization are presented. Methods to overcome these hindrances are discussed with an example illustrating how to do it. Making software measurement a habit is emphasized upon. Various reasons are presented

supporting software measurement to become a habit with no or little loss of time. Finally tips to make software measurement a success are presented.

Reviewer: Vaddadi P Chandu

Marvin V. Zelkowitz, Ioana Rus, "Understanding IV&V in a Safety Critical and Complex Evolutionary Environment: The NASA Space Shuttle Program", International Conference on Software Engineering, 2001, pp. 349-360.

Article Abstract:

The National Aeronautics and Space Administration is an internationally recognized leader in space science and exploration. NASA recognizes the inherent risk associated with space exploration; however, NASA makes every reasonable effort to minimize that risk. To that end for the Space Shuttle program NASA instituted a software independent verification and validation (IV &V) process ii 1988 to ensure that the Shuttle and its crew are not exposed to any unnecessary risks. Using data provided by both the Shuttle software developer and the IV &V contractor, in this paper we describe the overall IV&V process as used on the Space Shuttle program and provide an analysis of the use of metrics to document and control this process. Our findings reaffirm the value of IV&V and show the impact IV &V has on multiple releases of a large complex software system.

Reviewer Abstract:

This paper presents an overview of a multi-level complex process for the NASA Space Shuttle software IV&V. The major results of their study show that: In spite of the complexity of the software being produced, the resulting product is effective and safe. IV&V is able to provide an independent means to certify flight readiness of this software. Yet, in spite of the care in developing such software, defects are still an unavoidable consequence of today software development process.

In the NASA Space Shuttle program, software releases, called operational increments(OI), are used for repeated missions on all four orbiters. Mission safety and reliability are the most important criteria for all missions and for each new software release. Because of this, changes to either the software or hardware are made with great care.

An IV&V, independent verification and validation, process requires the IV&V team have technical, financial and managerial independence from the development group. These activities occur during three phases in the development process: requirement analysis, product evaluation, flight certification. The IV&V team interacts with Shuttle software development in four distinct ways: Issues tracking,Flight software readiness assessments,Special studies,Facilitates channels of communication. Also as part of this evaluation process, IV&V issues are tracked by the IV&V team in both informal interactions with the developer and in a series of formal reports called Issues Tracking Reports (ITRs). Once discovered, an issue is tracked until it is resolved and the ITR is closed. The ITRs can be catrgorized as following: No change;Change;CR;Process. It is

found that an ITR does not necessarily reflect a single issue in the development of that OI. Some other analysis has also been done on the ITRs.

IV&V has been proven to be a very good method for the software quality control. It can discover the hidden defects in the early stage and reduce the cost greatly.

Reviewer: Feng Peng
