Modeling Software Defect Dynamics

RECENT ENHANCEMENTS TO THE CONSTRUCTIVE QUALITY MODEL (COQUALMO) HELP IN ASSESSING DEFECT DYNAMICS TO BETTER UNDERSTAND THE TRADEOFFS OF DIFFERENT PROCESSES AND TECHNOLOGIES FOR REDUCING DEFECTS.

by Raymond Madachy, Barry Boehm and Dan Houston

Software defects are not created equal and exhibit various dynamic behaviors that complicate project decisions. Interrelated factors that affect defect dynamics include: project environment and practices that impact the overall defect generation and detection rates, phases and the timing of activities, the types of defects, and mission circumstances.

The practices to find and remove defects have varying efficiencies with respect to the types of defects and the lifecycle timing. Difference classes of defects have mission-dependent risk profiles (e.g. real-time or not), some methods are better suited than others for finding certain types of defects, and there are overlaps between the methods with respect to the classes of defects.

Parametric modeling and simulation can help reason about strategies for reducing defects by quantifying the impact of different processes and technologies. This paper presents our ongoing work to extend and refine the COnstructive QUALity MOdel (COQUALMO) for assessing defect dynamics to better understand the tradeoffs. Using parametric cost and defect removal inputs, static and dynamic versions of the model help one determine the impacts of quality strategies on defect profiles, cost and risk. We describe the evolution of the models into increasingly detailed forms.

Software quality processes can be assessed with the models that predict defects introduced and removed. The models are calibrated with empirical data on defect distributions, introduction and removal rates; and supplemented with Delphi results for detailed defect detection efficiencies.

The basic version of COQUALMO [1] models medium grain defect introduction and detection rates. It uses COCOMO II [2] cost estimation inputs with defect removal parameters to predict the numbers of generated, detected and remaining defects for requirements, design and code. It models the impacts of defect reduction practices for the primary activities of automated analysis, peer reviews, and execution testing and tools on these defect categories. It is a static model with time-invariant factors and presents the final cumulative quantities of defects remaining.

However, the top-level decomposition by phase injection was not fine enough when considering the relative dynamics of different types of defects. Subsequently we refined the taxonomy for NASA to classify them using the Orthogonal Defect Classification (ODC) [3].

Though the static models demonstrated the rough tradeoffs between practices, we reformulated the ODC extension into a dynamic version that provides insight into time trends and is suitable for continuous usage on a project. It is more realistic because it simulates the changing of factors throughout a project (e.g. improved peer reviews), their interrelationships and time-dependencies. It uses system dynamics for simulation modeling.

Most recently, Dynamic COQUALMO is an improved system dynamics model [4] that further refines the defect practice impacts in terms of their varying efficiencies. It is also calibrated to project practices and empirical defect rates at The Aerospace Corporation, and serves for retrospective studies of completed projects for process improvement.

The original COQUALMO model was developed with industrial data supported by affiliates of the University of Southern California Center for Systems and Software Engineering (USC-CSSE). Further empirical data was used from manned and unmanned flight projects to tailor and calibrate the models for NASA. At The Aerospace Corporation we used empirical data for local calibration. This paper presents the latest developments in the ongoing empirical research.

COQUALMO Background

Cost, schedule and quality are highly correlated factors in software development. They essentially form three sides of a triangle, because beyond a certain point it is difficult to increase the quality without increasing either the cost or schedule, or both. Similarly, development schedule cannot be drastically compressed without hampering the quality of the software

product and/or increasing the cost of development. Software estimation models can (and should) play an important role in facilitating the balance of cost/schedule and quality.

Recognizing this important association, COQUALMO was created as an extension of the COnstructive COst MOdel (COCOMO) [2], [5] for predicting the number of residual defects in a software product. The model enables 'what-if' analyses that demonstrate the impact of various defect removal techniques. It provides insight into the effects of personnel, project, product and platform characteristics on software quality, and can be used to assess the payoffs of quality investments. It enables better understanding of interactions amongst quality strategies and can help determine probable ship time.

A black box representation of COQUALMO's submodels, inputs and outputs is shown in **Figure 1.** Additions to COCOMO II are shown in blue. Its input domain includes the COCOMO cost drivers and three defect removal profile levels. Defect introduction and removal is illustrated as a pipe and tank flow model in **Figure 2.** The defect removal profiles and their rating scales are shown in **Table 1**. More details on the removal methods for these ratings are in [2]. From these inputs, the tool produces an estimate of the number of requirement, design and code defects that are introduced and removed as well as the number of residual defects remaining in each defect type.

The COQUALMO model contains two sub-models: 1) the defect introduction model and 2) the defect removal model. The defect introduction model uses a subset of COCOMO cost drivers and three internal baseline defect rates (requirements, design, code and test baselines) to produce a prediction of defects that will be introduced in each defect category during software development. The defect removal model uses the three defect removal profile levels, along with

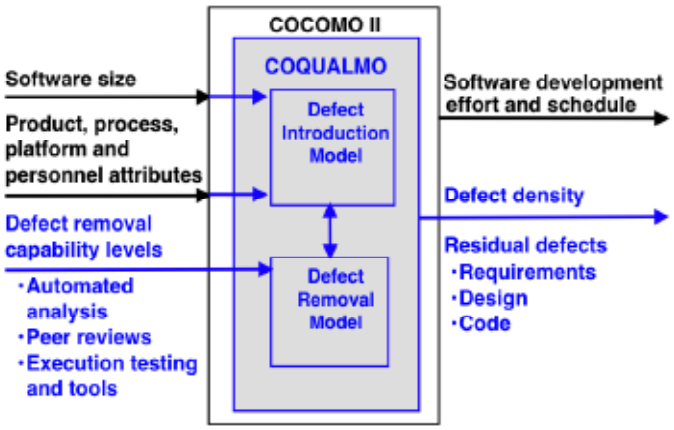

Figure 1: COQUALMO Extension to COCOMO

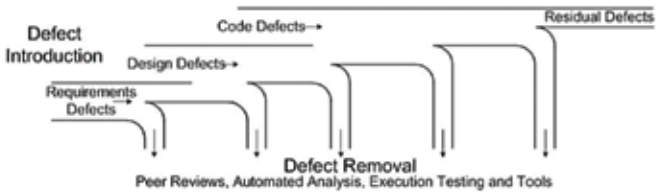

Figure 2: Defect Introduction and Removal Model

	Automated Analysis	Peer Reviews	Execution Testing and Tools
Very Low	Simple compiler syntax checking	No peer reviews	No testing
Low	Basic compiler capabilities for static module-level code analysis, syntax, type- checking.	Ad-hoc informal walkthroughs. Minimal preparation, no follow-up.	Ad-hoc testing and debugging. Basic text-based debugger.
Nominal	Some compiler extensions for static module and inter-module level code analysis, syntax, type checking. Basic requirements and design consistency, traceability checking.	Well-defined sequence of preparation, review, minimal follow-up. Informal review roles and procedures.	Basic unit test, integration test, system test process. Basic test data management, problem tracking support. Test criteria based on checklists
High	Intermediate-level module and inter-module code syntax and semantic analysis. Simple requirements and design view consistency.	Formal review roles and procedures applied to all products using basic checklists, follow up.	Well-defined test sequence tailored to organization (acceptance / alpha / beta / flight / etc.) test. Basic test coverage tools, test support system. Basic test process management.
Very High	More elaborate requirements and design view consistency checking. Basic distributed- processing and temporal analysis, model checking, symbolic execution.	Formal review roles and procedures applied to all product artifacts and changes. Basic review checklists, root cause analysis. Use of historical data on inspection rate, preparation rate, fault density.	More advanced test tools, test data preparation, basic test oracle support, distributed monitoring and analysis, assertion checking. Metrics-based test process management.
Extra High	Formalized specification and verification. Advanced distributed processing and temporal analysis, model checking, symbolic execution.	Formal review roles and procedures for fixes, change control. Extensive review checklists, root cause analysis. Continuous review process improvement. User and customer involvement, Statistical Process Control.	Highly advanced tools for test oracles, distributed monitoring and analysis, assertion checking. Integration of automated analysis and test tools Model-based test process management.

Table 1: COQUALMO Defect Removal Ratings

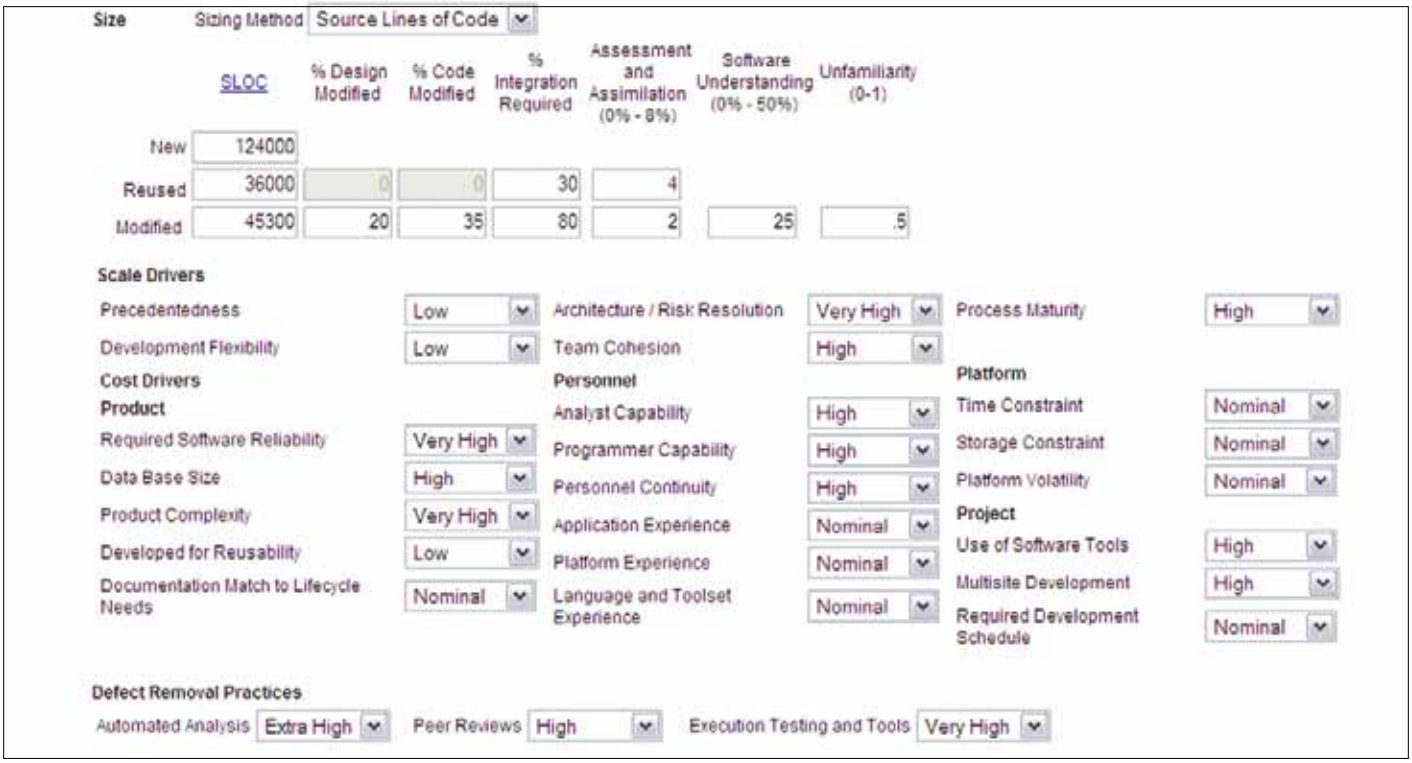

Figure 3: COQUALMO Inputs in COCOMO Suite Tool

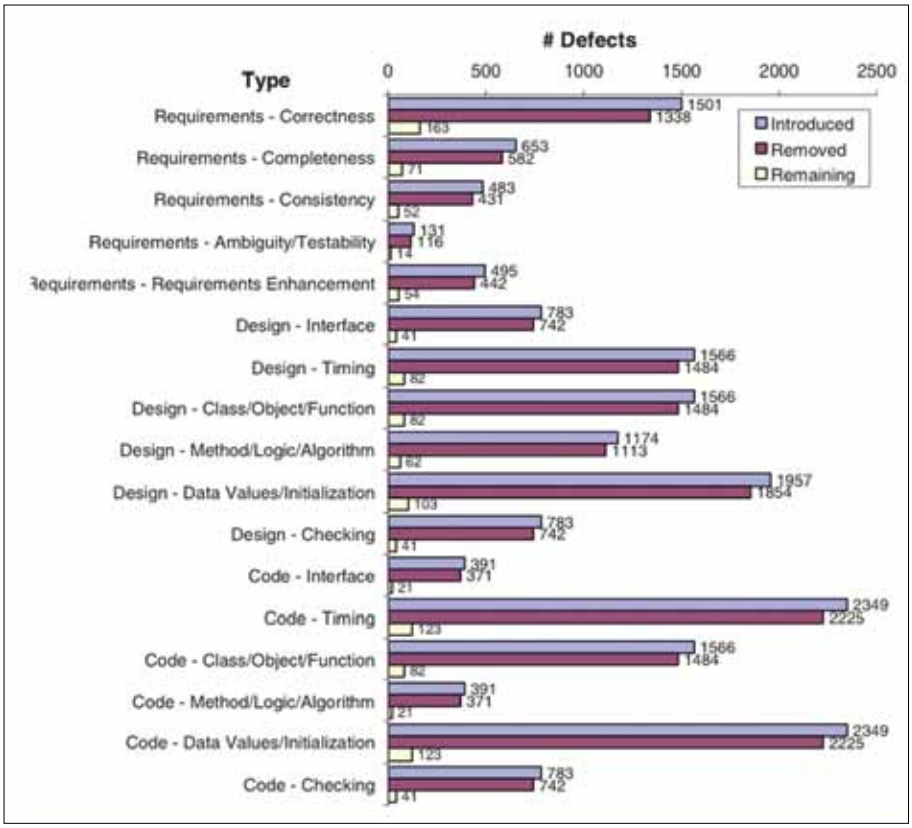

Figure 4: ODC COQUALMO Defect Outputs

the prediction produced by the defect introduction

model, to produce an estimate of the number of defects that will be removed from each category.

ODC COQUALMO Models and Tools

There are different implementations of ODC COQUALMO. Initially we created our static model in a spreadsheet and then transitioned to a web-based version as part of the integrated COCOMO Suite tool at http://csse.usc.edu/tools/COQUALMO.php or https://diana.nps.edu/MSAcq/tools/COQUALMO.php. The inputs to the static model are shown in **Figure 3** while **Figure 4** shows an example of ODC defect outputs. A dynamic simulation version models the defect generation and detection rates over time for continuous project usage, and provides continuous outputs as shown in the next section.

ODC Extension

At USC-CSSE we were evaluating and updating software cost and quality models for critical NASA flight projects [3]. A major focus of the work was to assess and optimize quality processes to minimize operational flight risks. We extended the COQUALMO model for software defect types classified with ODC. ODC COQUALMO decomposes the top-level defect types into more granular ODC categories.

The ODC taxonomy provides well-defined criteria for the defect types and has been successfully applied on NASA projects. The ODC defects are then mapped to operational flight risks, allowing "what-if" experimentation to determine the impact of techniques on specific risks and overall flight risk. The tool was calibrated to ODC defect distribution patterns per JPL studies on unmanned missions. A Delphi survey was completed to quantify ODC defect detection efficiencies, gauging the effect of different defect removal techniques against the ODC categories.

The approach is value-based [6] because defect removal techniques have different detection efficiencies for different types of defects, their effectiveness may vary over the lifecycle duration, different defect types have different flight risk impacts, and there are scarce resources to optimize. Additionally the methods may have overlapping capabilities for detecting defects, and it is difficult to know how to best apply them. Thus the tools help determine the best combination of techniques, their optimal order and timing.

ODC COQUALMO can be joined with different risk minimization methods to optimize strategies. These include machine learning techniques, strategic optimization and the use of fault trees to quantify risk reductions from quality strategies. We have demonstrated the integration with automated risk minimization methods to design higher value quality processes, in shorter time and with fewer resources, to meet stringent quality goals on projects [7].

ODC COQUALMO decomposes defects from the basic COQUALMO model using ODC [8]. The top-level quantities for requirements, design and code defects are decomposed into the ODC categories per defect distributions input to the model. With more granular defect definitions, ODC COQUALMO enables tradeoffs of different detection efficiencies for the removal practices per type of defect. **Table 2** lists the ODC defect categories used in the model, and against which data is collected.

This more detailed approach takes into account the differences between the methods with specific defect pairings. Peer reviews, for instance, are good at finding completeness defects in requirements but not efficient at finding timing errors for a real-time system. Those are best found with automated analysis or execution and testing tools.

Requirements	Design/Code
- Correctness	- Interface
- Completeness	- Timing
- Consistency	- Class/Object/Function
- Ambiguity/Testability	- Method/Logic/Algorithm
	- Data Values/Initialization
	- Checking

Table 2: ODC Defect Categories

The model also provides a distribution of defects in terms of their relative frequencies. The tools described in the next section have defect distribution options that allows a user to input actuals-based or expert judgment distributions, while an option for the Lutz-Mikulski distribution is based on empirical data at JPL [9].

The sources of empirical data used for analysis and calibration of the ODC COCOQUALMO model were described in [10]. The quality model calculating defects for requirements, design and code retains the same calibration as the initial COQUALMO model. The distribution of ODC defects from [9] was used to populate the initial model with an empirically-based distribution from the unmanned flight domain at JPL. The Lutz-Mikulski distribution uses the two-project average for their ODC categories coincident across the taxonomy used in this research for design and code defects. Their combined category of "Function/Algorithm" is split evenly across our two corresponding categories.

A comprehensive Delphi survey [11], [12] was used to capture more detailed efficiencies of the techniques against the ODC defect categories. The experts had on average more than 20 years of related experience in space applications. The ODC Delphi survey used a modified Wideband Delphi process and went through two rigorous iterations [12]. The results are summarized separately for automated analysis, execution testing and tools, and peer reviews in [12].

The values represent the percentages of defects found by a given technique at each rating (sometimes termed "effectiveness"). The different relative efficiencies of the defect removal methods can be visualized, in terms of the general patterns between the methods and against the defect types within each method. For example, **Figure 5** shows the results for Peer Reviews, where it is evident that Timing defects are more difficult to find than the other types.

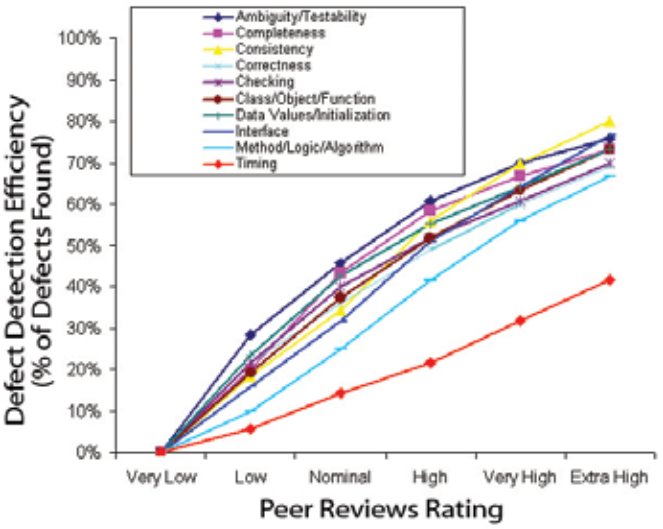

Figure 5: Peer Reviews ODC Defect Detection Efficiencies

ODC COQUALMO and Risk Minimization

Different methods for risk analysis and reduction have been performed in conjunction with ODC COQUALMO, which can produce optimal results in less time and allow for insights not available by humans alone. In [13] machine learning techniques were applied on the COQUALMO parameter tradespace to simulate development options and measure their effects on defects and costs, in order to best improve project outcomes. Another technique to reduce risks with the model is a strategic method of optimization. It generates optimal risk reduction strategies for defect removal for a given budget, and also computes the best order of activities [14].

An integration of ODC COQUALMO was also been prototyped with the DDP risk management tool [15], [16], which uses fault trees to represent the overall system's dependencies on software functionality. These experiments to optimize quality processes are described in more detail in [7].

Dynamic Simulation Model

This section summarizes a continuous simulation model version using system dynamics [17] to evaluate the time-dependent effectiveness of different defect detection techniques against ODC defect categories. As a continuous model, it can be used for interactive training that demonstrates the effects of changes midstream or for tracking product quality through continuous updating with project actuals [17].

The model uses standard COCOMO factors for defect generation rates and the defect removal techniques for automated analysis, peer reviews and execution testing and tools. The model can be used for process improvement planning, or control and operational management during a project.

COQUALMO is traditionally a static model, which is a form not amenable to continuous updating because the parameters are constant over time. Its outputs are final cumulative quantities, no time trends are available, and there is no provision to handle the overlapping capabilities of defect detection techniques. The defect detection methods and the defect removal techniques are modeled in aggregate, so it is not possible to deduce how many are captured by which technique (except in the degenerate case where two of the three methods are zeroed out).

In this system dynamics extension to ODC COQUALMO, defect and generation rates are explicitly modeled over time with feedback relationships. It can provide continual updates of risk estimates based on project and code metrics. This model includes the effects of all defect detection efficiencies for the defect reduction techniques against each ODC defect type per Figure 4.

The defect dynamics are based on a Rayleigh curve defect model of generation and detection. The buildup parameters for each type of defect are calibrated for the estimated project schedule time, which may vary based on changing conditions during the project.

The defect detection efficiencies are modeled for each pairing of defect removal technique and ODC defect type. These are represented in graph functions for defect detection efficiency against the different ODC defect types.

Scenarios are demonstrated with dynamic responses to changing defect removal settings on different defect types. The variable impact to the different defect types can be visualized

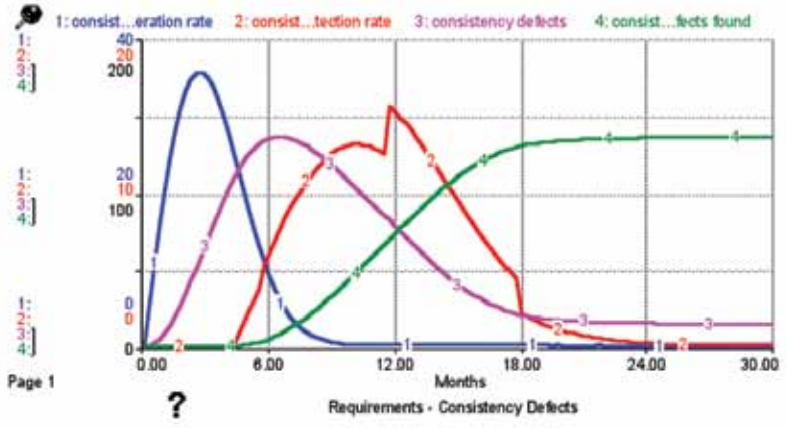

Figure 6: Requirements Consistency Defect Dynamics (1: Generation Rate, 2: Detection Rate, 3: Current Defects, 4: Defects Found).

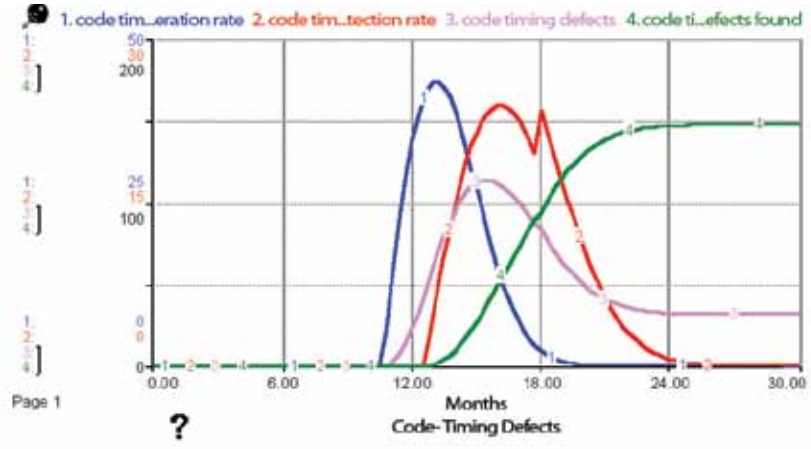

Figure 7: Code Timing Defect Dynamics (1: Generation Rate, 2: Detection Rate, 3: Current Defects, 4: Defects Found).

in the curves. An example for requirements consistency defects are in **Figure 6**, showing the perturbation from the defect removal changes. Another is the graph in **Figure 7** from the simulation model showing representative dynamics for code timing defects, including the impact of changing the defect removal practices in the midst of the project at 18 months. At that time the setting for execution testing and tools goes high, and the timing defect detection curve responds to find more defects at a faster rate.

Dynamic COQUALMO

Dynamic COQUALMO was independently calibrated to software development projects at The Aerospace Corporation for estimating counts of residual defects [4]. The simulation also provides a basis for considering the effects of re-planning, project revision, and process improvements. In this section we show how two project defect profiles were replicated and a project retrospective was performed on one of them.

Description

In Dynamic COQUALMO the duration of each phase is specified. It then provides a decomposition of defect estimates

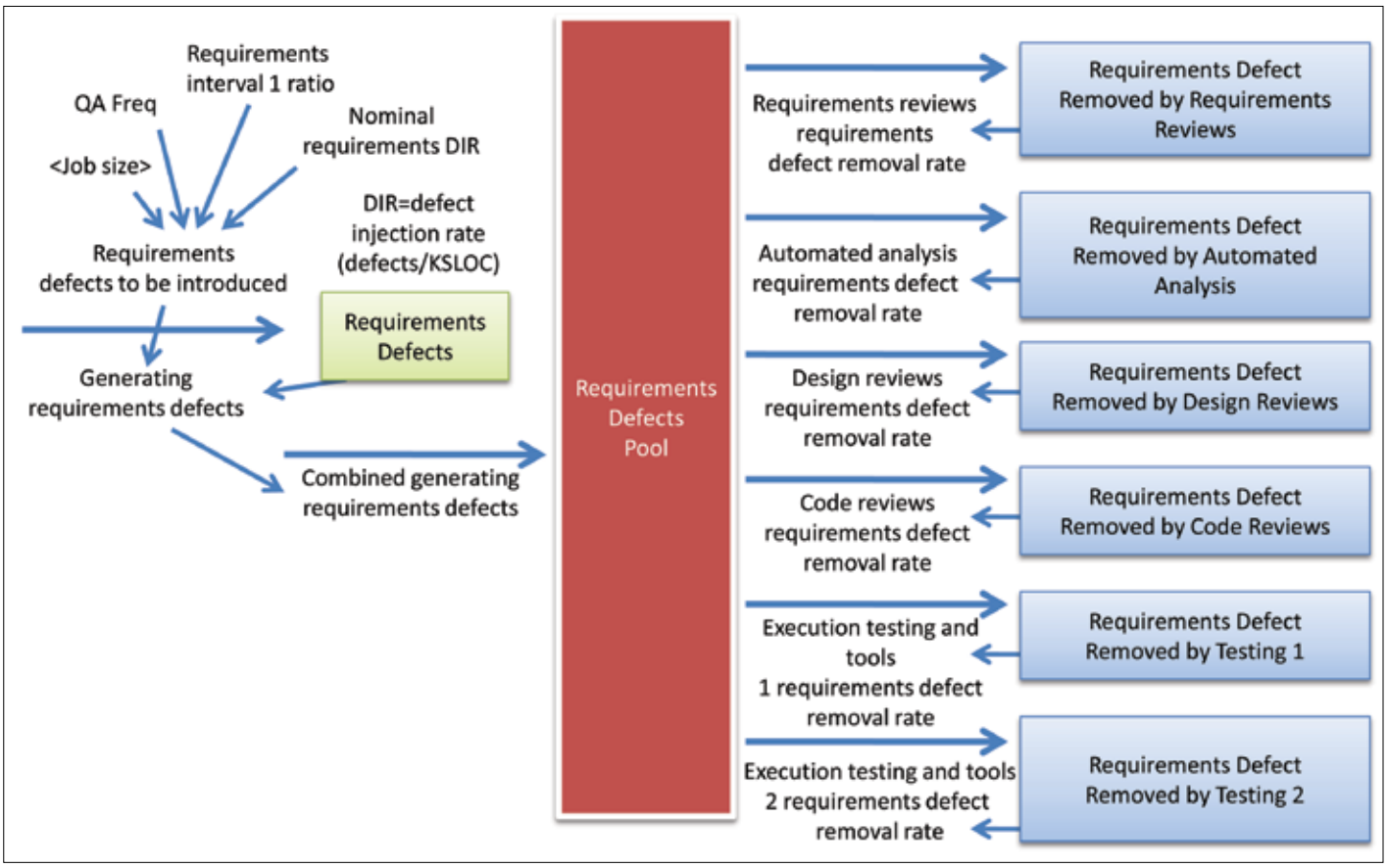

Figure 8: Requirements Defect Flows

across phases so as to achieve a project defectivity profile. A Rayleigh curve generator was used for the defect patterns [4] that is more generalized and calibrated than the ODC COQUALMO dynamic version previously described.

To translate COQUALMO into a simulation model, it was necessary to decompose some of its components. First, peer reviews were separated into requirements, design, and code reviews by decomposing and allocating the peer review DRFs, under the constraint that they aggregate to those specified in COQUALMO. Also, COQUALMO assumes the same quality of practice for each type of quality activity, for example, all peer reviews at performed at a nominal level. Realistically, quality activities are performed to varying degrees within a project. The model accommodates this variation by weighting the quality levels of each activity and taking a weighted average.

Testing is also decomposed to distinguish software development testing from system testing, in which reliability is usually measured. This decomposition was also accomplished by weighting the two sets of testing defect removal factors under the constraint that they aggregate to COQUALMO values.

Due to the many differences in testing processes across software types and organization, the two testing phases are simply called Testing 1 and Testing 2, thereby allowing a user to define the differences between them and weight their effectiveness accordingly.

Dynamic COQUALMO has six defect flows, an inflow and an outflow for each artifact type: requirements, design, and code. Each of these flows has a single source, but multiple outflows, one for each quality-inducing activity as shown for requirements defects in **Figure 8**.

Model inputs include the following:

- Estimated job size in KSLOC.
- Settings for COCOMO II factors, including effort multipliers and scale factors.
- Estimated phase durations and degrees of phase concurrency (sum to the project duration).
- Usage profile of quality levels for each defect removal activity.
- Relative effectiveness estimates:
    - Effectiveness of requirements, design, and code reviews in finding requirements defects.
    - Effectiveness of design and code reviews in finding design defects.
    - Effectiveness of the two test phases in finding defects (requires definition of the differences between the two phases).

Results

It was found the default COQUALMO values for nominal defects introduced were high. Values between .5 Defects/KSLOC (Project C requirements) and 6.1 Defects/KSLOC (Project C code) were used to produce the modeled curves such as in Figure 9, from [18] loosely constrained by limited knowledge of the ratios of the actual defect sources.

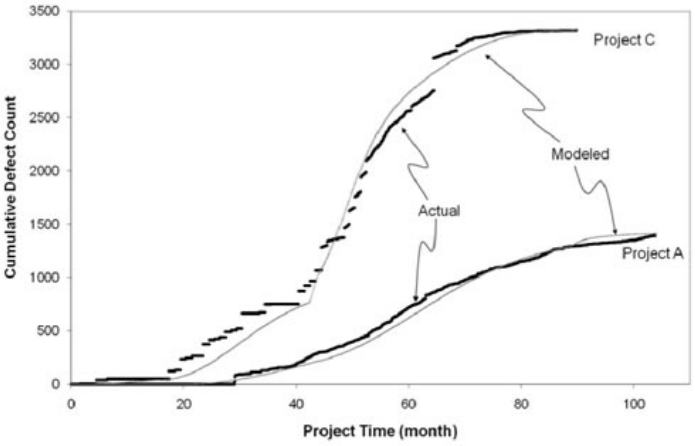

Figure 9: Calibrated Dynamic COQUALMO for Two Projects

For the retrospective studies we looked at the factors affecting defect introduction and used them to perform a cost benefit analysis. The cost of improving the factors was estimated and Dynamic COQUALMO provided a measure of the benefits in terms of reduced numbers of defects. These were plotted and those with the highest benefit against cost were selected for action in the next project.

A combination of four cost factors was selected as the most cost effective and beneficial set of improvements [4]. This set, plus Requirements Review, was modeled as the Improved Project A. **Figure 9** displays the modeled defect profile for the actual project as well as for the two alternate cases. The Dynamic COQUALMO modeling suggested two conclusions:

1. Revising the project earlier would not have made a large difference in the final product quality, but would have saved finding and fixing about 600 defects.

2. Starting the project with the selected set of improvements, would have both reduced the effort of finding and fixing defects and improved the final product quality. The number of defects introduced for the Improved Project would have been about

1000 less than in the actual project and the residual in the final product would have been less than half that of the actual final product.

Conclusions and Future Work

Software estimation models can and should play an important role in facilitating the right balance of activities to meet quality goals. By predicting software defect introduction and removal rates, COQUALMO variants are useful for identifying appropriate defect reduction strategies.

The extension for ODC defect types provides more granular insight into defect profiles and their impacts to specific risks. We have shown that the ODC COQUALMO model can be used in different ways to reason about and optimize quality processes.

The use of value-neutral software engineering methods often causes software projects to expend significant amounts of scarce resources on activities with negative returns on investment. The use of models and risk minimization techniques can be used to optimize the scarce resources. Results of experiments combining ODC COQUALMO with various methods show they can produce optimal results in less time and allow for insights not available by humans alone.

We will continue to integrate the ODC COQUALMO model with complementary techniques to support risk management practices, and to compare their performances.

Software development projects seem to have characteristic defect discovery profiles. Dynamic COQUALMO can replicate a discovery profile and, by inference, produce a realistic defect profiles for use in managing quality effort in future projects.

Organizations should consider local calibration, because the usage profiles of default COQUALMO may require adjustment.

Using Dynamic COQUALMO in a retrospective cost-benefit analysis for product quality provides credible results.

One of the next steps for Dynamic COQUALMO will be to use the model prospectively to assess project options as part of either initial planning or planning a revision.

We are analyzing more empirical data for further calibrations and improvements, We are continuing to gather data from case studies at the Naval Postgraduate School, The Aerospace Corporations, and CSSE industrial affiliates. There are also affiliates in commercial domains undertaking ODC defect analysis for which specialized calibrations are being done.

With more comprehensive data the quality model will be further improved, tailored for government projects and other organizations, and integrated with complementary methods for value-based decision making on quality strategies.

Acknowledgments

This work was partially supported by NASA AMES Cooperative Agreement No. NNA06CB29A for the project Software Risk Advisory Tools. The authors wish to thank the CSEE affiliates who contributed data, the ODC Delphi participants, Dr. Mike Lowry and John Powell at NASA. At the Aerospace Corporation this work was conducted with Douglas Buettner and Myron Hecht.

References

1. Chulani S., Boehm B.: Modeling software defect introduction and removal: COQUALMO (COnstructive QUALity MOdel), University of Southern California Center for Software Engineering, USC-CSE Technical Report 99-510 (1999)
2. Boehm, B., Abts C., Brown A., Chulani S., Clark B.,Horowitz E., Madachy R.,Reifer D., Steece B.: Software Cost Estimation with COCOMO II, Prentice-Hall (2000)
3. R. Madachy, B. Boehm, Assessing Quality Processes with ODC COQUALMO, Proceedings of the 2008 International Conference on Software Process, Liepzig, Germany, 2008
4. Houston D., Buettner D., and Hecht M.: Dynamic COQUALMO: Defect Profiling Over Development Cycles, Proceedings of the 2008 International Conference on Software Process, Vancouver, Canada (2009)
5. Boehm, B.: Software Engineering Economics, Prentice-Hall (1981)
6. Biffl S., Aurum A., Boehm B., Erdogmus H., Grünbacher P. (eds.): Value-Based Software Engineering, Springer (2005)
7. Madachy R., Boehm B., Richardson J., Feather M., Menzies T.: Value-Based Design of Software V&V Processes for NASA Flight Projects, In: AIAA Space 2007 Conference, (2007)
8. Chillarege R., Bhandari I., Chaar J., Halliday M., Moebus D., Ray B., and Wong M.: Orthogonal Defect

Classification - A Concept for In-Process Measurements, IEEE Transactions on Software Engineering, Vol. 18, No. 11, pp. 943-956 (1992)

9. Lutz R., Mikulski I.: Final Report: Adapting ODC for Empirical Analysis of Pre-Launch Anomalies, version 1.2, NASA Jet Propulsion Laboratories, JPL Caltech report (2003)
10. Madachy R.: Risk Model Calibration Report, USC Center for Systems and Software Engineering, Report to NASA AMES (2006)
11. Madachy R.: JPL Delphi Survey for V&V Defect Detection Efficiencies, USC Center for Systems and Software Engineering, Report to NASA (2006)
12. Madachy R.: Calibration of ODC COQUALMO to Predict V&V Effectiveness, USC Center for Systems and Software Engineering, Report to NASA AMES (2007)
13. Menzies, T., and Richardson J.: "XOMO: Understanding Development Options for Autonomy", In: 20th International Forum on COCOMO and Software cost Modeling, USC (2005)
14. Port D., Kazman R., Polo B., Nakao H., and Katahira, M.: Practicing What is Preached: 80-20 Rules for Strategic IV&V Assessment, Center for Strategic Software Engineering, Technical Report, CSSE-TR20051025, University of Hawaii at Manoa (2005)
15. Feather M., and Cornford S.: Quantitative Risk-based Requirements Reasoning, Requirements Engineering, Springer, Vol. 8, No. 4, , pp. 242-265, (2005)
16. Feather M., Cornford S., Hicks K., and Johnson R.: Applications of Tool Support for Risk-informed Requirements Reasoning, Computer Systems Science and Engineering, CRL Publishing Ltd., Vol. 20, No. 1, pp. 5-17 (2005)
17. Madachy, R.: Software Process Dynamics, IEEE-Wiley, Hoboken NJ (2008)
18. Buettner DJ. Designing an Optimal Software Intensive System Acquisition: A Game Theoretic Approach. Doctoral Dissertation, University of Southern California (2008)

About the Authors

Raymond Madachy is an Associate Professor in the Systems Engineering Department at the Naval Postgraduate School. His research interests include systems and software cost/quality estimation and measurement; process simulation; risk management; integrating systems engineering and software engineering disciplines; and integrating empirical-based research with process simulation.

Barry Boehm is the TRW Professor of Software Engineering and Director Emeritus of the USC Center for Systems and Software Engineering; and Director of Research at the Systems Engineering Research Center for the DoD. He is an AIAA Fellow, an ACM Fellow, an IEEE Fellow, and a member of the National Academy of Engineering.

Dan Houston is a Senior Engineering Specialist with The Aerospace Corporation where he researches and develops software process simulation models for providing quantitative decision support to system program offices. Prior to joining Aerospace, he spent 16 years at Honeywell as a software developer, Six Sigma Black Belt, and software metrician. He is a Senior Member of both the IEEE and the American Society for Quality. His publications address software development economics and process improvement, software process simulation, and statistical methods in software engineering.

Authors Contact Information

Email: **Raymond Madachy:** rjmadach@nps.edu

Email: **Barry Boehm:** boehm@usc.edu

Email: **Dan Houston:** daniel.x.houston@aero.org