Incorporating Cost-Benefit Analyses into Software Assurance Planning

Martin S. Feather, Burton Sigal, Steven L. Cornford
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA 91109, USA
Martin.S.Feather@Jpl.Nasa.Gov
Burton.Sigal@Jpl.Nasa.Gov
Steven.L.Cornford@Jpl.Nasa.Gov

Patrick Hutchinson
Wofford College,
429 N. Church St.
Spartanburg, SC 29303
hutchinsonrp@wofford.edu

Abstract

The objective is to use cost-benefit analyses to identify, for a given project, optimal sets of software assurance activities. Towards this end we have incorporated cost-benefit calculations into a risk management framework. The net result is the capability to rapidly explore the costs and benefits of sets of assurance decisions.

We describe the cost-benefit aspects of our framework, and demonstrate them on a small illustrative example. We then address the issues raised by seeking to apply this approach to software assurance planning for large-scale software development efforts.

1. Introduction

Software assurance is the planned and systematic set of activities that ensures that software processes and products conform to requirements, standards, and procedures. Examples of such activities are: code inspections, unit tests, design reviews, performance analyses, construction of tracability matrices, etc.

In practice, software development projects have only limited resources (e.g., schedule, budget and availability of personnel) to cover the entire development effort, of which assurance is but a part. Projects must therefore select judiciously from among the possible assurance activities. In order to do so, they need means to quantitatively assess the cost/benefit of a suite of assurance activities as applied to their specific project. The ability to so is essential to allow:

- estimation of not just budget and schedule, but also quality and risk,
- determination of the optimal allocation of their limited resources,
- identification and evaluation of tradeoff opportunities (e.g., give up some functionality in order to be able to afford to achieve a high quality for the reminder).

In optimizing the allocation of limited resources, the goal might be to minimize risk with the available resources, or to reduce risk to an acceptable level, minimizing the resources needed to do so. In either case, it is an optimization problem to trade costs (of performing activities) and benefits (the value of risk reduction).

Cost-benefit analyses of *individual* activities have been reported (e.g., reinspections - [1]; regression testing – [4]). Studies of overall process improvement also exist (e.g., [8] that relate software defects, productivity, development cycle time and effort estimation to process ratings akin to the Software Engineering Institute (SEI)'s Capability Maturity Model (CMM). However, the middle ground, of quantitatively planning the suite of activities to apply to a given project, is relatively under explored. This middle ground is the focus of our ongoing efforts.

We have incorporated the key elements of cost-benefit calculations into a risk management tool. The net result is the ability to study the implications of multiple interrelated decisions on selection of software assurance activities.

Our starting point is a NASA-developed risk management framework, the effect Detection and Prevention (DDP) tool for risk assessment, planning and management [3]. DDP deals with requirements, risks and risk mitigations. Risks are quantitatively related to requirements, to indicate how much each risk, should it occur, impacts each requirement. Mitigations are quantitatively related to risks, to indicate how effectively each mitigation, should it be applied, reduces each risk. A set of mitigations achieves benefits (requirements are met because the risks that impact them are reduced by the selected mitigations), but incurs costs (the sum total cost of performing those mitigations). The main purpose of DDP is to facilitate the judicious selection of a set of mitigations, attaining requirements in a cost-effective manner. DDP has the capability to represent and reason simultaneously with a multitude of mitigations, and their effectiveness at reducing multiple risks. In actual usage, DDP application sessions have dealt with ranges of 50 – 150 each of requirements, risks and mitigations.

The remainder of this paper is organized as follows:

Section 2 – describes the cost-benefit calculations we have incorporated into the DDP framework.

Section 3 – demonstrates these cost-benefit calculations on a small illustrative example.

Section 4 – outlines the issues raised by seeking to apply this to software assurance planning for large-scale development efforts, and the approaches we are taking to address these issues.

Section 5 – conclusions.

2. Cost-benefit calculations in DDP

Motivated by the desire to aid software assurance planning, we have extended DDP's cost-benefit aspects in two key ways:

- classifying mitigations into three categories – preventions, detections, and alleviations, and
- associating with mitigations the time phase (e.g., the design phase of a project) at which those mitigations are applied.

2.1. Categories of mitigations: preventions, detections and alleviations

Mitigations are classified into the following three categories:

- *Preventions* – assurance activities that reduce the likelihood of problems occurring, e.g., training of programmers reduces the number of mistakes they make. Preventions incur a cost of performing the prevention activity.
- *Detections* – assurance activities that detect problems, with the assumption that detected problems will be corrected, e.g., unit testing detects coding errors internal to the unit. The net effect of assuming that detected problems will be corrected means that detections effectively reduce the likelihood of those problems remaining present in the final product. Detections thus incur two costs: that of performing the detection activity, e.g., performing the unit test, and that of repairing the problems detected, e.g., correcting a problem found during unit test. Note that the cost of performing a detection is a function of the detection itself, while the cost of repairing a problem it uncovers is a function the problem, no matter how it is detected. DDP computes the total cost by summing the detection's performance cost and the problem repair costs for the expected number of problems that it detects.
- *Alleviations* – assurance activities that decrease the severity of problems should they occur, e.g., programming a module to be tolerant of out-of-bound values input to it from another module. Alleviations incur a cost of performing the alleviation activity.

2.2. Time phase of mitigations

Mitigations are associated with the time phase (e.g., the design phase of a project) at which they would be applied. For this purpose, time phases form an ordered set, e.g, a project's lifecycle might be subdivided into the "requirements phase", "design phase", "coding phase" and "test phase. We allow the user to define the elements of this ordered set. Some obvious examples of mitigations, and how they might be associated with time phases, are:

- requirements inspection – requirements phase
- design review – design phase
- code walkthrough – coding phase
- unit test – test phase

DDP's cost and benefit calculations take into account the time phasing of mitigations by splitting those calculations into steps, one for each time phase. In a given step, only the mitigations that are both selected for application and associated with that step's time phase are applied to effect reductions of likelihoods and/or severity of problems. The first stage (in our simple example, the "requirements phase") starts with all the possible problems initialized to their a-priori likelihoods (i.e., the likelihood of the problem occurring were nothing done to inhibit it) and with their impacts on requirements at their maximal (unalleviated) strengths. Thereafter, each successive phase inherits the problems in the state in which they emerge from the previous phase.

One consequence of this is that the DDP model can support project planning that must take into account the phase in which costs are incurred. In practice, large development efforts that span several years may well have their budget allocated on a yearly basis.

Another consequence is that the DDP model is capable of representing the phenomenon of cost of corrections escalating later in the development lifecycle. For example, it is often observed that a requirement flaw detected and repaired at requirements time costs (say) 10 times as much to repair if only detected and then repaired at coding time, 100 times as much at test time, etc. Such observations are used to argue for increased emphasis on early-lifecycle activities.

The DDP model accommodates this phenomenon by separating the cost of detecting a problem from the cost of repairing it, and furthermore, by allowing the cost of repair to be dependant on the time phase in which the repair is performed. DDP's cost calculation works as follows: when a mitigation detects the problem, it does so in the time phase associated with the mitigation. Assuming the problem is then repaired in that same time phase, the calculation of the cost of its repair takes into account the time phase itself. We therefore require that each problem has associated repair costs, one for each time phase, on which to base this calculation.

3. Demonstration of cost-benefit calculations on small illustrative example

The following example is based on the scenarios discussed by Gary Gack on the "Defect Tracking +

Inspections = $ in Your Pocket" portion of his website (http://www.iteffectiveness.com/defecttracking.html).

Gack considers the case of a 50,000 lines of code system containing 2500 defects (50 per KLOC). Gack presumes the compiler detects half the defects, leaving the remaining 1250 to be detected by other means. He presents four scenarios:

1. perfect walkthroughs/inspections
2. perfect testing
3. typical MIS organization's testing
4. aggressive inspections and structured testing

He subdivides activities into three phases, Inspections & Walkthroughs, Testing, and Production, for which he assumes the relative costs of finding and fixing defects to be 1:12:40 (a middle ground between the various ranges reported in published data). Scenarios 1 & 2 are idealistic scenarios to illustrate the calculations involved.

Scenario 1: perfect walkthroughs/inspections: these catch all defects that escaped the compiler, so the relative cost is $1,250.

Scenario 2: perfect testing: this catches all defects that escaped the compiler, so the relative cost is $15,000.

Scenario 3: typical practice: typical testing is assumed to catch 75% of the defects that escaped the compiler, leaving the remaining 25% to be caught by customers, so the relative cost is nearly $24,000.

Scenario 4: aggressive use of inspections and structured testing:

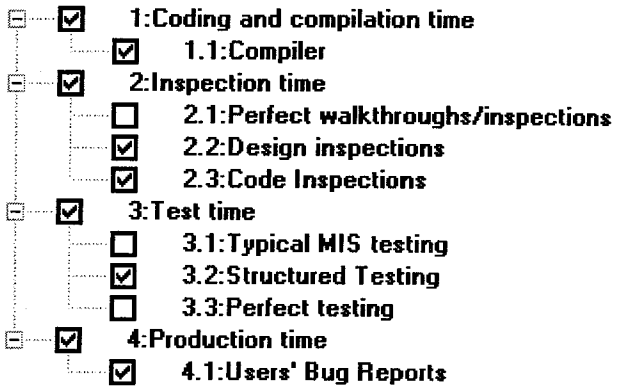
- design inspections catch 30% of the defects that escaped the compiler
- carefully chosen code inspections catch 80% of the defects emerging from the design inspections
- structured testing catches 70% of the defects emerging from the code inspections phase
- the (relatively few) remaining defects are detected by the customers.

Gack's calculations show the value of this approach, which allows only a modest number of defects (slightly more than 50) to remain for the customers to detect, and yet has a small total relative cost of approximately $5,000. This occurs when an earlier-phase activity (e.g, inspection) is selected, and leads to the early discovery, and therefore inexpensive correction, of problems that would otherwise be uncovered only in later phases (e.g, testing) when repair costs are higher. Cost considerations such as these are also discussed in [5].

3.1. DDP representation

To represent Gack's scenarios in DDP, we define a requirement "Defect-free code", a problem: "Coding defects" and, for each of the defect detection activities he discusses, a detection-type mitigation. For ease of understanding, we organize the mitigations into a tree structure whose parent categories correspond to each of the major time phases.

The DDP tree, extracted from a screenshot, is shown next. The check boxes alongside each line control whether or not the corresponding mitigation is active. This feature allows us to turn on and off the various mitigations so as



to quickly recreate any of Gack's scenarios (and many others, of course). Each time we do so, DDP automatically recomputes the defect detection rates and the costs.

We define four time values: "Coding and compilation", "Inspection time", "Test time" and "Production time". Each mitigation is associated with the appropriate such time value.

The effectiveness of each mitigation against the problem is expressed in DDP's effectiveness table, shown below. For example, Gack's code inspections that catch 80% of the defects remaining at that point are represented by the "Code inspections" row with a 0.8 effectiveness value. If we were to change these values, DDP would automatically recompute costs and benefits on the basis of the new values.

		Coding defects
[-]Coding and compilation time	Compiler	0.5
[-]Inspection time	Perfect walkthroughs/inspectio	1
	Design inspections	0.3
	Code Inspections	0.8
[-]Test time	Typical MIS testing	0.75
	Structured Testing	0.7
	Perfect testing	1
[-]Production time	Users' Bug Reports	1

Gack does not distinguish between cost of detection and cost of repair. One simple way to recreate his calculations is therefore is to associate his costs with just the repair cost of the "Coding defects" problem, assigning it to have the values 0, 1, 12 and 40 for respective time phases. Again, DDP would automatically recompute costs and benefits should we make changes to these values.

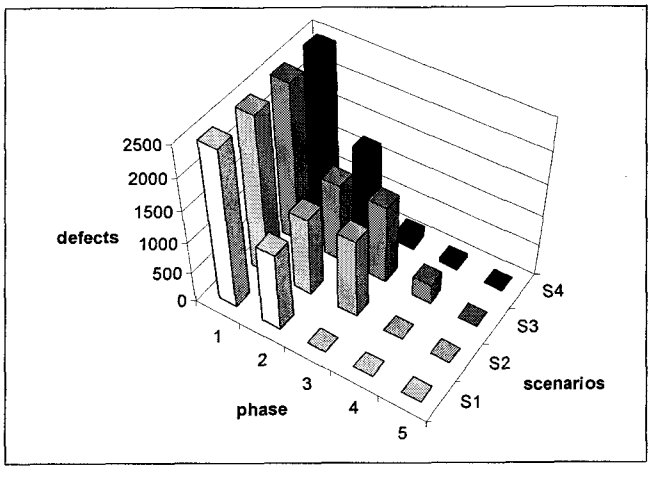
3.2. DDP calculations

We are able to quickly recreate all four of Gack's scenarios by checking/unchecking the appropriate mitigations. The tree shown earlier showed his 4th scenario of a suite of aggressive inspections and structured testing. To recreate his 2nd scenario of perfect testing, we would simply check and uncheck boxes to arrive at the tree shown below.

- ☑ 1:Coding and compilation time
  - ☑ 1.1:Compiler
- ☑ 2:Inspection time
  - ☐ 2.1:Perfect walkthroughs/inspections
  - ☐ 2.2:Design inspections
  - ☐ 2.3:Code Inspections
- ☑ 3:Test time
  - ☐ 3.1:Typical MIS testing
  - ☐ 3.2:Structured Testing
  - ☑ 3.3:Perfect testing
- ☑ 4:Production time
  - ☑ 4.1:Users' Bug Reports

DDP automatically computes the resultant defect rates and costs by phase. For visualization of multiple scenarios side by side, ee export the computed values in comma-delimited form, and use Excel to chart them. The results are shown below, where all the data values of this chart are those computed by DDP.

The vertical "defects" axis shows the



number of defects.

	phase 1	2	3	4	5
S1	2500	1250	0	0	0
S2	2500	1250	1250	0	0
S3	2500	1250	1250	312.5	0
S4	2500	1250	175	52.5	0

The "phase" axis shows the series of time phases (1 = initially 2 = compilation, 3 = inspection, 4 = testing, 5 = production).

The "scenarios" axis shows Gack's four scenarios (S1 = perfect testing, S2 = perfect walkthroughs/inspections, 3 = typical MIS testing, 4 = aggressive inspections and structured testing).

In all four scenarios, the compiler catches 50% of the initial defects, hence all the columns in row 1 drop identically, from height 2500 to height 1250.

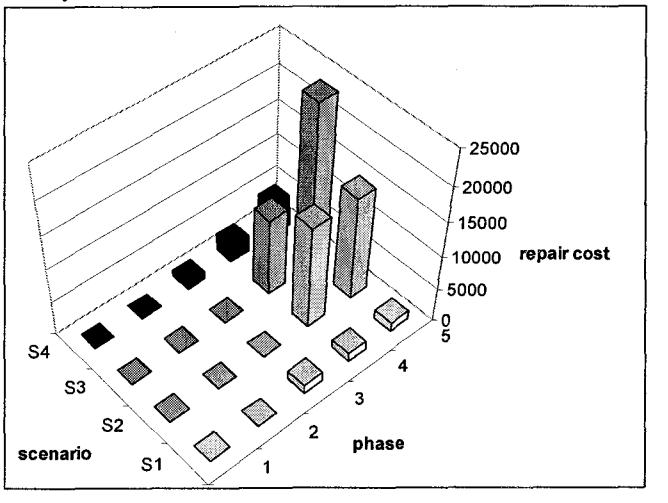
S1, the row of white columns towards the front, is Gack's Scenario 1. Perfect walkthroughs/inspections catch all defects that escape the compiler, so the column drops to zero height from phase 2 to phase 3.

S2, the row of light gray columns, is Gack's Scenario 2. There is no inspection time activity, so the defect columns in phases 2 and 3 are the same height. Perfect testing then catches all the remaining 1250 defects, so in phase 4 the column drops to zero height.

S3, the row of dark gray columns, is Gack's Scenario 3. Imperfect testing catches 75% of the 1250 defects that had escaped the compiler (as in Scenario 2, there is no inspection time activity), so the column drops from height 1250 to height 312.5 in phase 4. These remaining defects are caught by customers in phase 5.

S4, the row of black columns towards the back, is Gack's aggressive use of inspections and structured testing. Of the 1250 defects that escaped the compiler, inspection-time activities catch all but 175. Thereafter, structured testing catches all but 52.5, the number of defects making their way through to the customers to discover.

The escalating contribution of repair costs is shown vividly in the chart below:



	phase1	2	3	4	5
S1	0	0	1250	1250	1250
S2	0	0	0	15000	15000
S3	0	0	0	11250	23750
S4	0	0	1075	2545	4645

S3, the "typical" scenario, shows the high cost contribution of leaving defect detection until production time. Conversely, S4, the "aggressive" scenario that invests inspection effort early in the lifecycle, accumulates costs starting in phase 3, but because this results in fewer defects escaping to later stages, these costs are minor in comparison to the savings by the end. Note also that S3 results in a far lower final cost than S2, the "perfect testing" scenario. Gack stresses the point that inspections are the most cost effective improvements that most organizations can make.

4. Large scale software assurance planning

Software assurance planning for large-scale safety-critical projects typically involves consideration of dozens, possibly hundreds, of assurance-related activities. However, from project to project, many of the same software development risks will apply, and many of the same assurance-related activities will serve as risk mitigations. Hence it is both worthwhile and practical to pre-populate DDP with data pertinent to software development – risks that recur across many software development efforts, assurance-related activities that mitigate those risks, and estimates of how effective those mitigations reduce risk.

In addition to pre-populating DDP with relevant information, we collaborate with another NASA assurance planning tool to provide us an initial recommendation of which activities to perform.

We discuss this collaboration next, and then revisit the issue of determining mitigation effectiveness.

4.1. The DDP / Ask Pete collaboration for software assurance planning

Ask Pete (http://tkurtz.grc.nasa.gov/pete) from NASA Glenn Research Center is another effort that supports software assurance planning. Ask Pete uses a user-friendly electronic interview to elicit overall project characteristics. Behind the scenes, relevant portions of this data are automatically fed into COCOMO II to yield cost and schedule estimates, and into other matrices to yield control level determinations and IV&V estimates. The COCOMO estimates, and other portions of the user-provided data, are then combined with institutional practices and policies (for example, ISO 9001, CMM and site-specific principles), yielding estimates of cost, schedule, and risk; and plans for the development effort, oversight and risk mitigation while conducting software development.

We are involved in an ongoing collaboration with the Ask Pete project to jointly support software assurance planning [7]. Briefly, this joint approach has three main steps:

1. Ask Pete interviews the user to ascertain the characteristics of the project, and on the basis of this makes the first cut at a recommended set of assurance activities
2. DDP relates the recommended activities to the software development risks they mitigate, thus allowing for risk- and cost- based tuning and customization of the recommendations.
3. Ask Pete takes the recommendations as revised by DDP, and generates the final reports (including listing the revisions).

In order to support this, we pre-populated DDP with information pertinent to software assurance planning, as follows:

Mitigations: DDP is pre-populated with mitigations, namely the universe of all of Ask Pete's assurance activities (close to 200 in total). When Ask Pete recommends a set of assurance activities, the recommendation is automatically passed to DDP, causing the corresponding mitigations to be checked (akin to the way we checked particular mitigations to model Gack's various scenarios, in the earlier section).

Risks: DDP is pre-populated with risks pertinent to software development activities. For this purpose, we use a taxonomy of software risks from the Software Engineering Institute (SEI), namely those in the report [11].

Effectiveness: DDP is pre-populated with a set of values that say how effective each mitigation is at reducing each risk. We lack the data founded on experience to provide these values, so we use experts' estimates of what they might be.

4.2. Determining the effectiveness of risk mitigations

As mentioned above, we lack experiential data, and so make do with experts' *estimates* of the effectiveness of software assurance activities at reducing software development risks. We faced the same need in an earlier study [2], where we had used the Capability Maturity Model (CMM) for software [10] to suggest the set of mitigation activities. There too, we found the need to make estimates of the effectiveness of these activities. Indeed, we were unable to locate any description of even which KPAs addressed which risks, let alone by how much.

Of necessity, our approach to large scale software assurance planning is founded upon these estimates. This raises some concerns. In this section we argue that our approach is beneficial nevertheless, describe the steps we are taking to further validate it, and finally outline our hopes for being able to replace estimates with data based on evidence.

DDP has been successfully applied to advanced technologies (hardware and/or software), where the focus has been on increasing the infusion rates of these technologies by identification and mitigation of risks prior to delivery to a project [3]. In these kinds of applications, DDP is used to gather and combine data from a group of experts. Given that the subjects of these studies have been

advanced technologies, it is common for the data those experts provide to be estimates (e.g., of the effectiveness of mitigations at reducing risk!). Nevertheless, the studies have been revealing. Often it was seen that substantial efforts in non-technology development areas were necessary to sufficiently mature the product prior to technology pull, and funding could now be sought (and justified) in non-R&D programs to get this work done. The opposite was true as well where it was realized that the R&D efforts currently being worked were not necessarily the highest priority, and a course correction was necessary. The general point is that when a disciplined process is followed to gather and combine estimates from experts, it is often possible to draw substantial benefit from that information.

In collaboration with Tim Menzies we are pursuing an approach that, if successful, will indicate the *sensitivity* of DDP's answers to its data. For a particular application of DDP, we would ascertain the degree to which its answers would be changed (or not) were changes made to the data (estimates) from which they were derived. Menzies' suggestion is to make random variations around the supplied values, and derive sensitivities from this. He has a machine-learning based approach that appears successful at identifying key factors in surprisingly large models [9] from DDP data.

Lastly, we are looking to the broader software engineering community to see what data they have gathered on the effectiveness of software assurance activities. During the summer of 2001 we performed a pilot study to determine whether we could locate such data, and whether we could incorporate such data into the DDP framework. In areas of inspection and testing, data does seem to be available. For example, we located the paper ""Investigating the cost-effectiveness of reinspections in software development" [1] from its promising title. This paper turned out to hold data on the effectiveness of requirements inspections and reinspections. We note that ongoing efforts of the consortium http://www.cebase.org may gather additional data.

We collected the data we found into a spreadsheet before attempting to incorporate into DDP. The spreadsheet serves as a way to organize the data we find but leave it in its original form (e.g., retain the exact way the source describes the cost). An example fragment of our spreadsheet appears in the table, based on the data extracted from the aforementioned paper.

When we tried to incorporate the data into DDP, we rarely found all the aspects of information that we require. For example, the table below indicates that the Cost figures do not differentiate between cost of detection and cost of repair.

It seems that for the foreseeable future we will have to continue to rely predominantly on estimates of effectives values for software assurance activities. Our current approach has DDP pre-populated with estimates of the generic effectiveness values. In a given DDP application, we rely on the experts at hand to scrutinize these generic values, tailoring them as they deem appropriate to the problem at hand. The success of DDP when applied to identification and mitigation of risks in advanced

	Data				
	action	*type of action*	*when*	*cost*	*effectiveness*
Sources					
[Biffl, 2000]	Formal Requirements Inspection: a process for the verification of software requirements documentation by a team of inspectors incl. Defect detection meeting, and repair.	Detection	Requirements Phase	cost in terms of person hours to do an inspection for the given document mean 51.4 hours std.dev. 15.7 for a 35 page document with 80+ major defects. These figures appear to include a repair phase. The per page cost is 1.47 hours.	gives data from thirty teams with a mean of 45.2%, std dev. 16.6% of errors detected. actual results range from 8% to 74%. Will impact Requirements risks.
[Biffl, 2000]	Formal Requirements Reinspection: Same process as Inspection, but is performed after the errors reported by the first inspection are fixed	Detection	Requirements Phase	cost in terms of person hours to do a reinspection for the given document mean 35.5 hours std.dev. 11.3 for a 35 page document which had 80+ major defects before the first inspection. These figures appear	gives data from thirty teams with a mean of 36.6%, std dev. 15.1% of errors detected. actual results range from 5% to 69%. Will impact Requirements risks.

technologies (hardware and/or software) suggests that this approach has considerable value.

5. Conclusions

We have described the capabilities for cost-benefit calculations that we have embodied within a risk management tool. These capabilities allow us to differentiate risk mitigations into three categories: preventions, detections and alleviations. This distinction permits the calculations to take into account important phenomena such as the escalating cost of bug fixes as projects progress through their lifecycle. By embodying these capabilities into the DDP risk management tool, we are able to scale them to handle dozens, and even hundreds, of risks and mitigations. For software assurance planning in large-scale projects, we collaborate with another software assurance planning tool whose output serves as a good first cut at a suitable suite of assurance activities. Pre-populating our tool with risks and mitigations that recur in many software development efforts allows a mode of use in which experts need not start from scratch, but can instead scrutinize this preformulated information, and tailor it as needed to the problem at hand. Key to our approach is to quantitatively state the effectiveness of mitigations on the risks they mitigate (i.e., state how much they mitigate those risks). We performed a pilot study to locate date on which to base these effectiveness values, but concluded that for the foreseeable future we must continue to rely predominantly on experts' estimates to serve as these values. Successes with this approach based on experts' estimates suggest that this is a viable and worthwhile approach.

Preceding sections have discussed some related work. In addition, we mention the following:

We see some relationship to the cost/benefit work of [6]. 2-D graphical plots of requirements, where the dimensions indicate value of attainment against cost of attainment, appear to serve as guides to software release planning. Our work involves risk as the key factor that underlies all our calculations, and in this respect is quite different.

Risk estimation approaches (e.g., fault tree analysis, baysian methods) appear very well suited to the assessment of a *single* design. However, our application is to the planning of entire, often quite large, *suites* of mitigations, were the driving concern is the cost-benefit-guided selection from among a large set of such mitigations.

6. Acknowledgements

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology. Discussions with Julia Dunphy (JPL), John Kelly (JPL), Tim Kurtz (NASA Glenn), Tim Menzies (Univ. British Columbia) and Peter In (Texas A&M) have been most useful in helping us formulate our ideas.

7. References

[1] S. Biffl, B. Freimut and O. Laitenberger. "Investigating the cost-effectiveness of reinspections in software development", *23rd Int. Conference on Software Engineering*, 2001, pp. 155-164.

[2] S.L. Cornford, M.S. Feather, J.C. Kelly, T.W. Larson, B. Sigal, J. Kiper. "Design and Development Assessment", Proceedings of the 10th International Workshop on Software Specification and Design, San Diego, California: 105-114, November 2000.

[3] S.L. Cornford, M.S. Feather & K.A. Hicks. "DDP – A tool for life-cycle risk management", *IEEE Aerospace Conference*, Big Sky, Montana, Mar 2001, pp. 441-451.

[4]. T. Graves, M. Harrold, J. Kim, A. Porter and G. Rothermel. "An Empirical Study of Regression Test Selection Techniques". *20th Int. Conference on Software Engineering*, 1998, pp. 267-273.

[5]. C. Kaner. "Quality Cost Analysis: Benefits and Risks", Software QA Vol 3, #1, p. 23, 1996.

[6] J. Karlsson & K. Ryan. A Cost-Value Approach for Prioritizing Requirements. *IEEE Software*, Sept./Oct. 1997, 67-74.

[7] T. Kurtz & M.S. Feather. "Putting it All Together: Software Planning, Estimating and Assessment for a Successful Project", in Proceedings of 4th International Software & Internet Quality Week Conference, Brussels, Belgium, Nov 2000.

[8] F. McGarry, S. Burke & B. Decker. "Measuring the impacts individual process maturity attributes have on software products", in Proceedings, *5th* International *Software Metrics Symposium*, 1998, pp. 52-60.

[9] T. Menzies & Y. Hu. "Constraining Discussions in Requirements Engineering via Models", *1st International Workshop on Model Based Requirements Engineering*, San Diego, California, Dec 2001.

[10] Mark C. Paulk, Bill Curtiss, Mary Beth Chrissis, Charles V. Weber. Capability Maturity Model for Software, Version 1.1. Technical Report CMU/SEI-93-TR-024, Software Engineering Institute, Carnegie Mellon University, February 1993.

[11] F. Sisti & J. Sujoe. "Software Risk Evaluation Method Version 1.0" Technical Report CMU/SEI-94-TR-019, Software Engineering Institute, Carnegie Mellon University, 1994.