Requirements, Architectures and Risks

James D. Kiper
Dept. of Computer Science and Systems Analysis
Miami University
Oxford, OH 45056
kiperjd@muohio.edu

Martin S. Feather
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Dr
Pasadena CA 91109-8099
Martin.S.Feather@Jpl.Nasa.Gov

Abstract

There is wide agreement that architecture plays a prominent role in large, complex software systems. Selection of an appropriate architecture – one that matches the system requirements and implementation resources – is a critically important development step.

We advocate the use of risk-based reasoning to help make good architectural decisions. In this paper, we explore the adaptation of a risk management process and tool to this purpose.

1. Introduction

Software design for complex software systems is difficult. The past decade has seen a convergence of opinion about the importance of using established architectures and design patterns. At the system level, styles of software architecture [1, 11] like pipes-and-filters or event-driven provide a starting point for design of complex software systems. At the more detailed level, architectural treatments capture well-reasoned decisions whose strengths and weakness are understood, e.g., software design patterns like wrapper or builder. [7] This paper will focus on the system level use of architecture, although the approach should also be applicable to the finer grained use of design patterns.

Choosing a *good* architecture is a critically important step in the design of a system. A poor choice at this level is difficult to repair at a more detailed design level. We define the adjective *good* with respect to architecture to mean an architecture that matches system requirements *and* can be implemented within the resources allocated to it. The implementation itself is a non-trivial task, and induces a further set of critical decisions.

The primary thesis of this paper is that risk can be used to guide these decisions. Use of risk-based reasoning enables software engineers and managers to make choices of software architecture and architecture implementation that satisfy both criteria – meeting system requirements and adhering to resource limitations.

This paper is organized as follows: section 2 describes the current risk-based design process and the tool that has been developed to support this process; section 3 discusses some shortcomings in this process that are caused by the failure to capture of explicit design and, in particular, architectural aspects; section 4 describes ways in which we are incorporating software architectural decisions into this process and tool.

2. Basis for the approach - risk-based design

The approach advocated herein begins from an existing risk-based design process and its accompanying tool support. This is the “Defect Detection and Prevention (DDP)” process [4], developed and used at JPL to help engineers manage the trade space of choices in designing spacecraft and associated technology.

DDP has three primary sets of issues that it captures and tracks: requirements, risks, and mitigations. The DDP tool is typically used to collect and maintain decisions and information discussed in several meetings with a group of experienced engineers and domain experts. The process used in these DDP sessions is diagrammatically explained in figure 1. The first step is the collection and weighting of requirements. Given the requirements, the domain experts determine the risks that these system requirements entail. Each of these risks is then scored as to its impact on each of the requirements. After risks are determined in step 2, the activities that can mitigate these risks are then listed. Each of these mitigations is scored as to its effectiveness at reducing each risk.

DDP is unique in bringing a quantitative risk-based approach to bear at early stages of decision-making. The scoring of the links between risks and requirements, and between mitigations and risks, are given a quantitative, probabilistic interpretation. This allows DDP to add up the cumulative impact of all risks, compare an individual risk’s cumulative impact, compute how much of requirements are being attained, compute how much net benefit the use of a mitigation conveys, etc. [5]

This information is used together with budget information on the cost of mitigations to make choices about which mitigations to select. The goal is to reduce the risks to sufficient levels (and so adequately attain requirements) while remaining within resource limitations.

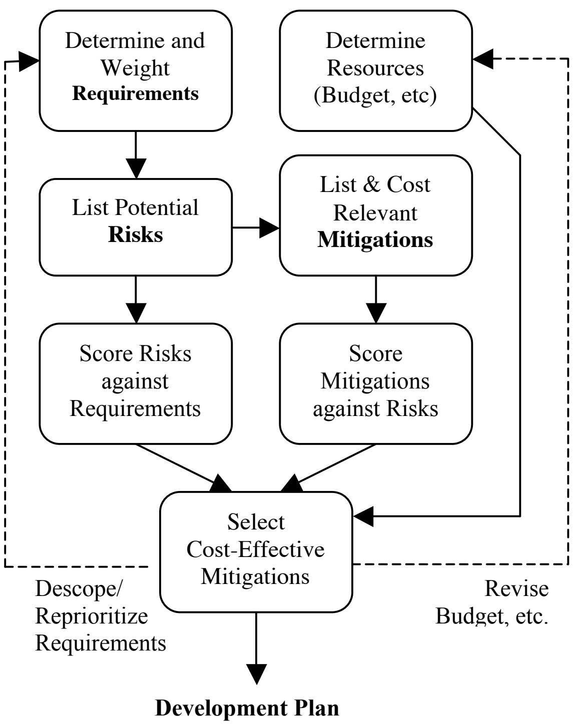
Determine and Weight **Requirements**
Determine Resources (Budget, etc)
List Potential **Risks**
List & Cost Relevant **Mitigations**
Score Risks against Requirements
Score Mitigations against Risks
Select Cost-Effective Mitigations
Descope/ Reprioritize Requirements
Revise Budget, etc.
Development Plan

Figure 1: Standard DDP Process

In this process, risk is used as the intermediary through which link requirements are indirectly linked to mitigations. Our experience is that this indirection is particularly useful. For example, the phenomenon of “diminishing returns” as more and more mitigations are applied to the same risks falls out naturally from this approach. In contrast, attempts to link requirements directly to solutions (development plans) often fail to capture the multiplicity of problems and solutions.

3. Shortcomings in the current process

The standard DDP process depicted in figure 1 involves the gathering and linking of three major concepts: system *requirements* (weighted to reflect their relative importance), *risks* that threaten to detract from attainment of those requirements, and *mitigations* to help quell those risks (and so lead to improved attainment of requirements). We have found this risk-centric approach to be quite effective in guiding experts to make their choices of mitigations. (The reader may wonder why choices have to be made among mitigations. The answer is one of resource limitation. Choosing to do all mitigations is typically not possible from a budget and time perspective.)

We have observed that in use of the DDP tool and process on JPL applications, there is some additional structure to the concepts involved that the current process is not adequately capturing. We describe how these observations lead us to now propose to include *architecture* as a first-class concept within the DDP process.

Our first step in this direction stemmed from the observation that some mitigations induce and/or exacerbate risks. For example, a vibration test may be used to check that a piece of hardware will operate correctly when subject to vibration, thus decreasing the risk of launching a spacecraft that is unable to operate under mission conditions. However, there is some risk that the test itself will cause problems (e.g., break something). The risk of those problems we term *induced* risk. Another example is of a protective coating applied to a piece of circuitry, say. Its purpose is to protect the circuitry from future damage, i.e., decrease those kinds of risks. However, should there be need to modify the circuit, that protective coating will make it harder, perhaps even impossible, to effect the modification. We describe the risks that would lead to the need for modification as *exacerbated* by the protective coating (i.e., while their likelihoods remain the same, their impact, should they occur, is increased). Software analogies of these phenomena are well known – fixing one bug may introduce new ones; introducing monitoring code may aid testing, but decrease performance (or lead to changed timing behavior when that test-time code is dropped from the final delivered code).

The standard DDP process (and its tool support) was evolved to accommodate these phenomena by extending the allowable range of the values attached to the links between mitigations and risks. Initially all such values were restricted to being positive proportions (i.e., in the range (0, 1]), indicating the proportion by which application of the mitigation would eliminate risk. Lack of a link between a risk and a mitigation indicated that the mitigation would have no effect whatsoever on that risk. The extension was to allow the expression of negative values as measures of effectiveness, where a negative value in the range [-1, 0) indicated induced risk (the more negative, the more the likelihood of the risk being induced), and a negative value in the range [-1000000, –1) indicated exacerbated risk (any existing risks’ impacts would be multiplied by the abs(value)). For example, a value of –3 means triple the impact of risks.

These extensions served their intended purpose to allow DDP studies to take into account mitigation induced/exacerbated risks. However, they opened the door to (mis?)use as a way to represent design alternatives.

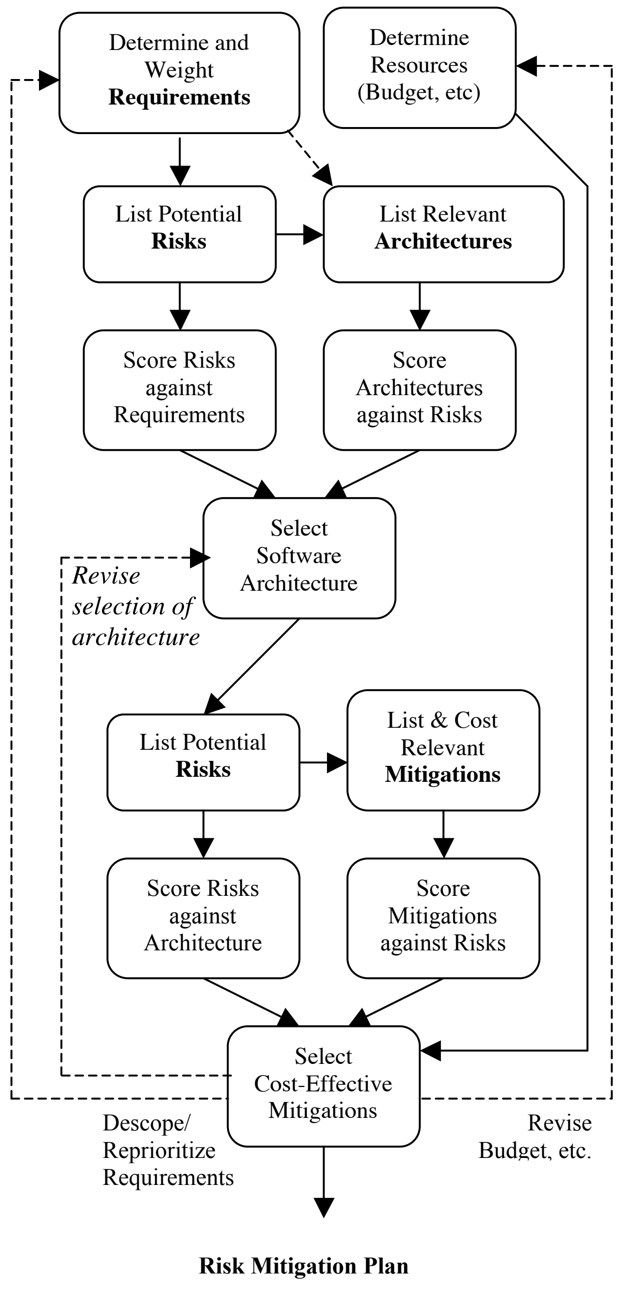
Determine and Weight **Requirements**
Determine Resources (Budget, etc)
List Potential **Risks**
List Relevant **Architectures**
Score Risks against Requirements
Score Architectures against Risks
Select Software Architecture
Revise selection of architecture
List Potential **Risks**
List & Cost Relevant **Mitigations**
Score Risks against Architecture
Score Mitigations against Risks
Select Cost-Effective Mitigations
Descope/ Reprioritize Requirements
Revise Budget, etc.
Risk Mitigation Plan

Figure 2: Revised DDP Process

To illustrate this we will first give a hypothetical and simplistic system design example. Suppose that one of the requirements for a planetary rover is to gather science data on planetary formation, using a drill to extract a core sample from rocks. Use of the drill demands a large amount of power, so lack of available power is a particularly serious risk against that science requirement. One possible mitigation to that risk is to deploy large solar panels, capable of generating sufficient power. (An alternative could be to drill more slowly but for a longer duration). The large-solar-panel mitigation has its own risks (rover is now prone to tipping; higher overall power levels lead to the risk of electrical shorts; etc.).

A comparable software-domain example is the design of a software subsystem with the requirement that the software be able to respond to the position of the cursor by displaying context-sensitive information that the user needs. One risk to this requirement is that this information will be displayed after an unreasonably long delay. One possible mitigation is to design this system as an event-driven system with an event loop that is designed to catch and respond to mouse movements that affect cursor position.

Our first inclination was to use mitigation-induced risk as the means to represent these design options. For example, the large solar panels mitigation for the risk of lack of power we encoded as a DDP mitigation that induces the rover tipping risk, the electrical shorts risk, etc. Each of these induced risks were added into the same list of potential risks, but with the unusual characteristic that their a-priori likelihoods were set at zero (i.e., the only way those risks could occur is through being induced when the solar panel mitigation is selected). This enabled us to avoid the need for further extension to the DDP tool.

From these latter examples, it is clear that the activities that we have encoded as mitigations are, in fact, design choices. In the software arena, these choices are **software architectural decisions**. We are dissatisfied with encoding of these as just more "mitigation" choices, albeit with some unusual characteristics. At the very least, we should call these out as architectural decisions, and so be poised to take advantage of detailed methods for architectural evaluation. We would also like to avoid the need to start DDP from a "blank slate", where all the information must be supplied anew. Clearly, the body of knowledge that pertains to architectures should be used to pre-populate DDP. Finally, and most importantly, we observe many of the risks and mitigations that derive from an architectural choice affect how well that architecture mitigates the original risks it was selected to address. For example, suppose a pipes-and-filters architecture was selected to mitigate the risk of system ossification (inability to easily make system modifications). The more the development of the system strays from strict adherence to that architecture, the more it diminishes the effectiveness of that architecture at mitigating the ossification risk. In DDP-speak, the architecture itself can be attained in whole or only in part (the latter due to the cumulative impact of risks on the realization of that architecture). Its effectiveness at mitigating risks is determined by how successfully its own risks are mitigated. We will see further examples of this in the next section.

With this observation, we propose the capture of architecture decisions *explicitly* in the DDP tool.

Before explaining this idea further, it is important to remind the reader that not all mitigations are design decisions. For example, one risk that may pertain to a piece of software is that requirements are inconsistent. One mitigation to this risk is a formal inspection process, a form of analysis. The use of formal inspections is clearly not a design decision. Indeed, in typical DDP applications, a significant proportion of mitigations fall into this testing/analysis category.

4. Incorporating software architectures

To incorporate software architectures into the DDP tool without radically changing the tool, we have proposed a two-phase process as depicted in figure 2.

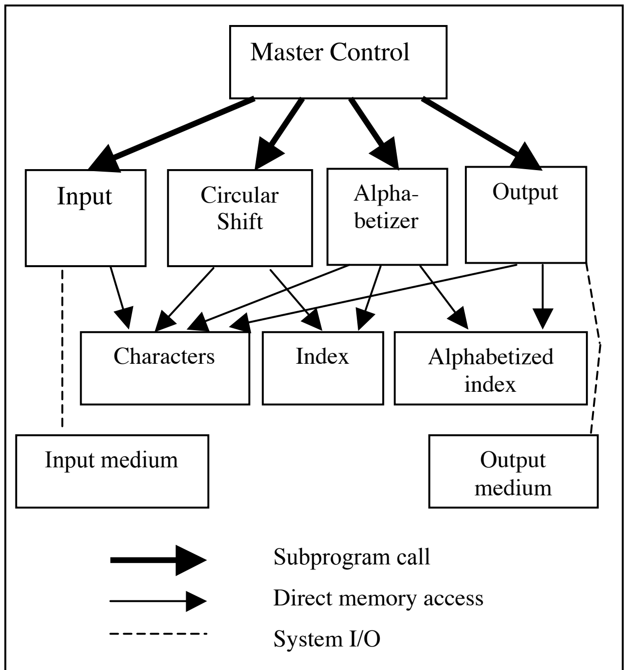

Figure 3: Shared data architecture

First we go through the original DDP process with requirements-risks-*architecture* rather than requirements-risks-*mitigations*. Thus, we are explicitly capturing alternative design architectures that will reduce or eliminate certain risks. Note that there may be a choice among several architectures that reduce a particular risk to acceptable levels.

To make this step easier, we propose seeding the DDP tool with possible classic software architectures. [1, 11] These architecture styles, e.g. pipes-and-filters, repository, object-oriented, serve as a starting point for the architecture-selection process. Designers may, of course, add their own hybrid designs.

The architectures that result from this first step become the starting point for another iteration of the original process, one that deals with *architectures*-risks-mitigations. Thus, the architecture serves both as a mitigation of risks in the first phase, and as an *induced* requirement in the second phase. Note that the selection of architecture is an important outcome of the DDP process. Although we argued that risks themselves are merely intermediaries, we do not make the argument that architectures have a similarly nebulous status.

4.1 Examples

As a small but illustrative example, consider the classic key word in context problem [10] proposed by Parnas in 1972 (and discussed by many other researchers since.)

> The KWIC [Key Word in Context] index system accepts an ordered set of lines; each line is an ordered set of words, and each word is an ordered set of characters. Any line may be "circularly shifted" by repeatedly removing the first word and appending it at the end of the line. The KWIC index system outputs a listing of all circular shifts of all lines in alphabetical order.

We will treat this paragraph as a first-order approximation to a set of requirements. In the DDP process and tool, this set is represented in a structured form, and the importance of each is evaluated and scored. For example, we might prioritize the generation of the list of all circular shifts as the most important, with the alphabetizing of this list as being important, but having a lower priority.

Now, let us consider some of the risks that might be associated with these requirements. Parnas suggests two potential risks (although he labels these as potential design changes rather than risks.)

1. Changes to the processing algorithm
2. Changes in data representation

Garlan, et al [8] add three other risks to those of Parnas.

1. Enhancement to system function
2. Performance
3. Reuse

(A nice discussion of this example and possible architectures is provided by Shaw and Garlan. [11])

These risks are scored against requirements to see, if they occur, how they would affect each requirement.

Now, we consider possible architectures for a solution to this problem. First, consider two architectures suggested by Parnas. [10] Figure 3 illustrates shared memory architecture. Figure 4 gives an abstract data type solution. Another possible architecture is the pipes-and-filters style as inspired by the Unix index utility and described by Shaw and Garlan. [11] This is depicted in figure 5.

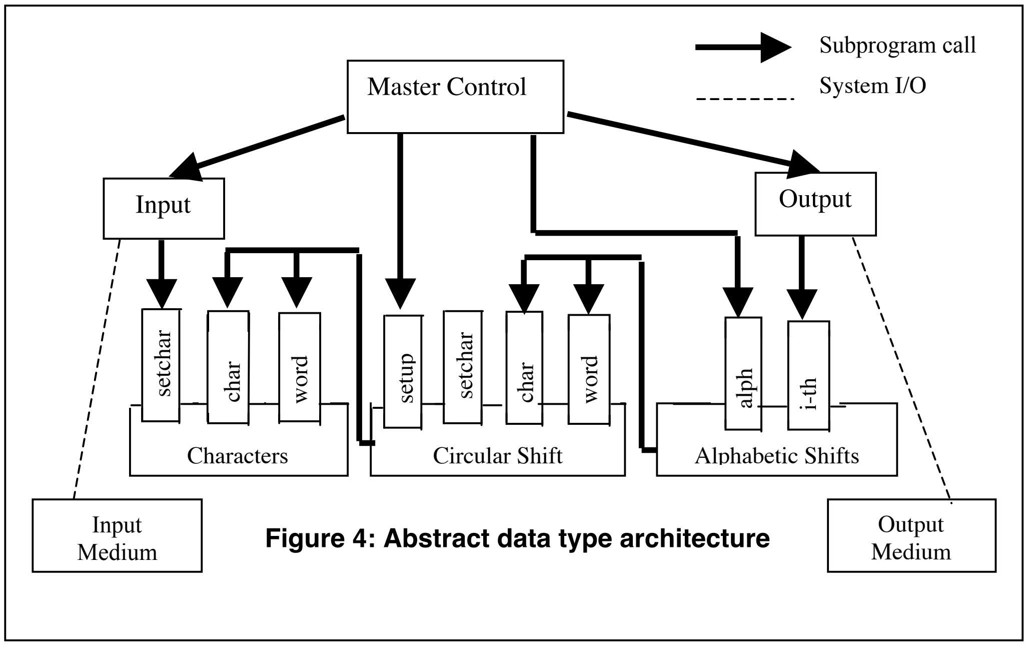

Figure 4: Abstract data type architecture

The mechanism that we use to evaluate the strengths and weaknesses of each potential architecture is to score each architecture against risks that we have identified. For example, we may determine that a pipes-and-filters architecture may have performance (i.e. speed) issues although the other two possibilities are likely to perform more adequately. Conversely, the shared data and the abstract data type architectures are likely to have trouble if the algorithm for generating the index is changed. The pipes-and-filters can more easily adapt its algorithm (by merely changing or adding a filter.) However, the abstract data type obviously can change its data representation more easily; the other two would find this type of change much more difficult. (This analysis is that of Shaw and Garlan.[11])

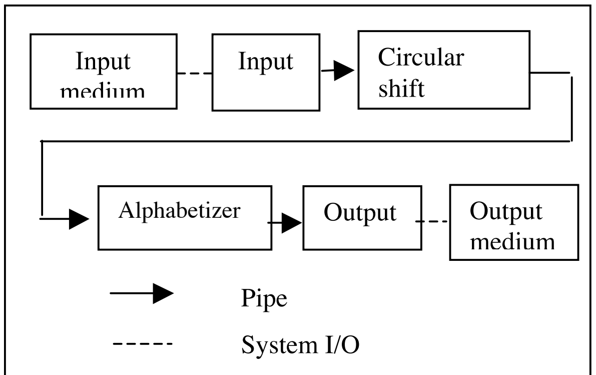

Figure 5: Pipes-and-filters

In the DDP process we would push the software engineers to *quantitatively* value these linkages between risks and architectures. For example, suppose that the engineers estimate the abstract data type design has a very small likelihood of being impacted by the risk of a change in data representation, while they estimate that performance risk of a pipes-and-filters architecture is relatively problematic. Table 1 illustrates the linkage data that engineers might produce in analyzing these architectures in light of particular risks. The numeric entries are in the range 0 to 1, where 0 means no effect, and 1 means that the architecture choice in that column completely eliminates the risk in that row. The DDP tool provides support for much larger matrices, and provides other views of this linkage data in addition to the tabular format.

In a realistic design, the number of requirements and potential risks can be large. In DDP applications at the component level (e.g., a memory device), it is typical to deal with 50 – 100 each of requirements, risks and mitigations, with hundreds of links between them. Even if the number of viable architecture choices is relatively small, the relationships between architecture and risks, and risk and requirements can make the choice of the preferred architecture quite complex. Addressing this complexity is a strength of DDP.

With the assistance of DDP, the design team can now select a tentative architecture. (This is a tentative architecture because the entire process is iterative. For example, the phenomena of requirements volatility and requirement creep are well known.) This begins the second phase of the DDP process. The starting point for this phase is this tentative architecture. We list potential risks inherent in this architecture. The risks enumerated in the previous phase were those associated with requirements regardless of architecture choice. Here we are looking for design and implementation risks. What things stand in the way of successfully implementing this system with this architecture? If the system is highly interactive, a pipes-and-filters architecture style is quite risky. However, an event-driven style would have much lower risks in this area. Because of the paradigm shift needed in object-oriented design (OOD) from traditional procedural design, OOD may have a high dependency on having a trained staff.

Table 1: Risk – Architecture matrix

		Architectures		
		Shared data store	Abstract data type	Pipes and filters
Risks	Algorithm change	0.9	0.7	0.1
	Data representation change	0.7	0.1	0.9
	Performance issues	0.1	0.1	0.7

The process of listing risks and evaluating the impact of each against the tentative architecture can be a tedious one. It is clear that many software risks are common across projects. We have preloaded DDP with a set of common software risks. (We have used the risk taxonomy identified by researchers at the Software Engineering Institute. [2]) Furthermore, we have entered linkages between these risks and a set of common architecture styles. [11] Thus, a choice of architecture obtains an associated set of risks and impacts. The design team can use this as a starting point, adding additional or more specific risks, and modifying or adding linkages.

Having identified software risks associated with this architecture, we now identify those activities, i.e. mitigations, that we can perform to eliminate, avoid, or reduce the impact of risks. For example, if there is the risk that our staff is not experienced in OOD, we could give them additional training or hire some experienced OO designers. Each such mitigation has a cost – the cost of training materials and time, or salaries and benefits for experienced designers.

We evaluate each mitigation against each risk to score its effect at reducing that risk. The effect of experience designers is likely to be greater against the risk of inexperienced staff than is training. (A new design method is often not fully understood until a certain level of experience is reached that cannot be provided by even the best training.)

Table 2 illustrates this matrix. Again, the numeric entries are in the range 0 to 1, where 0 means no effect and 1 means that the mitigation in that column completely eliminates the risk in that row.

Finally, this collection of information (risks x architecture, risks x mitigations) is combined with budgeting information to make decisions about which set of mitigations will achieve the system requirements using the tentative architecture and within budget and resource constraints. This is typically a complex decision given the enormous number of interactions among requirements (with their relative weights), risks (with their likelihoods), the tentative architecture, mitigations (with their costs), and linkages among these. DDP provides graphical displays of this information that helps the design team explore this complex trade space. An optimizer is available that uses simulated annealing to find near optimal choices of mitigations within a specified cost bound.

Table 2: Risk – Mitigation matrix

		Mitigations		
		Provide OOD training	Hire experienced OOD staff	Perform formal inspections
Risks	Inexperienced staff	0.7	0.9	0.0
	Inconsistent requirements	0.0	0.1	0.9

As mentioned previously, this is an iterative process. In these activities, it is common for the design team to discover additional requirements or learn of the infeasibility of certain requirements (resulting in the need for *descoping* [6]). Additional risks of a particular architecture choice may not be apparent until very late in the process. Thus, the entire DDP process may be iterated to capture these changes. However, note that subsequent iterations are likely to be more efficient because of the leverage of information derived during previous iterations.

The reader may be struck by the length and complexity of this process. We assert that this is the nature of the task, not a side effect of our process. Design of a complex software system is difficult.

5. Conclusions, Status, and Related Work

The argument set forth in this paper is that risk can and should be used to guide architectural decisions. These include both the choice of architecture itself, and the decisions that flow from that choice. We have shown how we arrived at this position through our observations of a risk-based decision process in use in real-world design activities. The gradual evolution of that process has led to the point where we believe that architecture deserves a place as a first-class object within the process itself. These points have been illustrated using a small but familiar example, the key word in context problem introduced by Parnas.

The status of this work is that all the aspects of DDP described in section 3 exist and have seen use in actual spacecraft technology risk studies. Instances of the phenomena we described in that section, of mitigation induced or exacerbated risks, and of design decisions encoded via this mechanism, have arisen in these same actual studies. The extensions needed of the DDP tool to support the two-phase approach, with mitigations leading to derived requirements, have been incorporated in an, as yet, unreleased version. We have used this within our own experimentation, but it has not yet seen field use in real project applications. Likewise, our encoding of architectural considerations is also at the stage of internal experiments that have yet to see actual customer application. Additional information DDP can be found at the Defect Detection and Prevention website, http://ddptool.jpl.nasa.gov

A full comparison with related work is beyond the scope of this workshop paper. We do draw attention to a distinguishing characteristic of DDP, namely that it is able to accommodate both architectural design decision concerns, and other elements of project planning (analysis, testing and process, represented as mitigations in the DDP framework). Furthermore, DDP does so in a quantitative manner. The combination of these aspects sets DDP apart from many of the other approaches to architectural decision making, e.g., the influence diagrams of [3] (shown in use in [9]), or the goal graphs of [12]. A key common thread that we have with those referenced bodies of work is the reliance on computer support for decision-making. Real-world problems involve a myriad of concerns, whose number and complex interconnections warrant support.

Acknowledgements

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology and at Miami University, Oxford, Ohio, both 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, the Jet Propulsion Laboratory, California Institute of Technology, or Miami University.

Dr. Steve Cornford leads the DDP project at JPL. DDP technology has been developed with NASA's Software IV&V, Code R and Code Q support, managed in part through JPL's Assurance Technology Program Office.

6. References

[1] Bass, L., P. Clements, et al. (1998). Software Architecture in Practice. Boston, Addison-Wesley.

[2] Carr, M. J., S. L. Konda, et al. (1993). Taxonomy-Based Risk Identification. Pittsburg, PA., Software Engineering Institute.

[3] Chung, L., B.A. Nixon, B.A., E. Yu, E., and Mylopoulos, J., 2000 “Non-Functional Requirements in Software Engineering” Kluwer Academic Publishers.

[4] Cornford, S. L., M. S. Feather, et al. (2001). DDP – A tool for life-cycle risk management. IEEE Aerospace Conference, Big Sky, Montana.

[5] Feather, M.S., Cornford, S.L. Dunphy, J. & Hicks, K.A. (2002). A Quantitative Risk Model for Early Lifecycle Decision Making; Proceedings of the Conference on Integrated Design and Process Technology, Pasadena, California, June 2002. Society for Design and Process Science

[6] Feather, M.S., S.L. Cornford & K.A. Hicks (2002) Descoping; Proceedings of the 27th IEEE/NASA Software Engineering Workshop, Greenbelt, Maryland, Dec 2002. IEEE Computer Society.

[7] Gamma, E., R. Helm, et al. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Reading, Massachusetts, Addison-Wesley.

[8] Garlan, D., G. E. Kaiser, et al. (1992). "Using tool abstraction to compose systems." IEEE Computer 25(6).

[9] Mylopoulos, J., L. Chung, S. Liao, H. Wang & E. Yu. “Exploring Alternatives during Requirements Analysis”, IEEE Software 18(1), Jan-Feb 2001, pp 92-96.

[10] Parnas, D. L. (1972). "On the criteria to be used in decomposing systems into modules." Communications of the ACM 15(12): 1053-1058.

[11] Shaw, M. and D. Garlan (1996). Software Architecture: Perspectives on an Emerging Discipline. Upper Saddle River, NJ, Prentice-Hall.

[12] van Lamsweerde, A., 2001, “Goal-Oriented Requirements Engineering: A Guided Tour”, Proceedings 5th IEEE International Symposium on Requirements Engineering, Toronto, Canada, August, pp. 249-2