

Global Analysis: moving from software requirements specification to structural views of the software architecture

C. Hofmeister, R.L. Nord and D. Soni

Abstract: Software architecture design approaches typically treat architecture as an abstraction of the implemented system. However, doing so means that the concepts, languages, notations, and tools for architecture are much more closely related to those of detailed design and implementation than to those of software requirements. Thus the gap between requirements and architecture represents a paradigm shift, while that between architecture and detailed design does not. Global Analysis, which is part of the Siemens Four Views architecture design approach, is a set of activities that serves to reduce the magnitude of this gap by guiding the architecture design process, capturing design rationale, and supporting traceability between requirements and architecture. In this paper Global Analysis is re-examined in light of five years of teaching it, reflecting on it, comparing it to other approaches, and examining how it was applied in four new systems. This experience confirms the value of the Global Analysis activities and the importance of capturing its results. In some cases the benefit went beyond that envisioned, and in other cases Global Analysis was not applied as expected. Because the templates that are provided for Global Analysis results have such a strong influence on how the activities were performed, this will be the focus of future changes.

1 Introduction

Software architecture emerged as a field of study upon recognition of distinct software development activities that are different from those of detailed design and implementation. The body of software architecture research has been both prescriptive and descriptive, always with the goal of improving the practice of software development and architecture design in particular. However, as noted by Bosch [1] the emphasis of software architecture design approaches has been to treat architecture as an abstraction of the implemented system. Thus although software architecture functions as a bridge between software requirements and detailed design, its concepts, languages, notations, and tools are much more closely related to those of detailed design and implementation than to those of requirements. The result is that the gap between requirements and architecture represents a paradigm shift, while that between architecture and detailed design does not.

We have developed an approach to designing software architecture, the *Siemens Four Views* approach [2], and one important part of this approach is *Global Analysis*, a set of

activities and artefacts that serves to reduce the magnitude of the gap between requirements and architecture design. Global Analysis activities begin with analysing the factors that influence software architecture. The factor analysis yields a small number of key issues that drive the design of the architecture. These issues arise from sets of factors that, taken together, create difficult design problems, for example when factors conflict, have little flexibility, a high degree of changeability, or a global impact on the system. The initial result of Global Analysis is a set of strategies that guide the architecture design. As design progresses, additional factors, issues, and strategies may arise, and the Global Analysis artefacts are updated to reflect the specific decisions made during design. Thus Global Analysis serves to guide the design process, to capture design rationale, and to support traceability between requirements and architecture.

The development of Global Analysis began informally, during the architecture design of an image acquisition and processing system. Our goal was to make the architecture more flexible and evolvable, describe architecture design decisions along with their rationale, and communicate and manage their use. After the project was complete, we generalised our experience and began a more formal and rigorous description of the approach. Next we applied Global Analysis retrospectively to four large systems. The variety of the application domains, characteristics, size, and complexity of the case study systems enabled us to gain valuable insights and evolve the Global Analysis approach to make it suitable for a wide variety of systems. This version of Global Analysis, described by Hofmeister, Nord, and Soni [2], has since been taught in courses and we have observed its application to four new systems.

In this paper we re-examine Global Analysis in light of five more years of teaching it, reflecting on it, comparing it to other approaches, and most importantly, closely examining how it was applied in the four new systems.

© IEE, 2005

IEE Proceedings online no. 20045052

doi: 10.1049/ip-sen:20045052

Paper first received 15th October 2004 and in final revised form 18th May 2005

C. Hofmeister is with the Lehigh University, Bethlehem, Pennsylvania, 18015, USA

R.L. Nord is with the Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 15213, USA

D. Soni is with the Dhirubhai Ambani Institute of Information and Communication Technology, Gandhinagar-382 009, Gujarat, India

E-mail: rn@sei.cmu.edu

2 Bridging the gap between requirements and software architecture

We are concerned with bridging the gap between requirements and software architecture. On one side, the Software Requirements Specification (SRS) describes the problem, not the solution. It rightly focuses on the behaviour of the system [3], including both functional and non-functional requirements. Even when the SRS is unambiguous, complete, and consistent, experience shows that requirements usually change during development [4].

On the other side, the software architecture describes the solution. It factors in many things that do not and should not appear in the SRS. Software architecture focuses on the structure or structures of the system and represents a system's earliest set of design decisions [5]. These decisions are critical to the system's ability to exhibit the behaviour specified by the requirements. In addition, as we said earlier, these decisions are typically represented as an abstraction of the implemented system, making the gap not just between problem and solution space but between two very different kinds of models.

What is needed to bridge this gap is an intermediate step. On the problem (requirements) side, the architect must identify those requirements that affect the architecture design, and identify any additional quality attributes, constraints (e.g. pre-imposed design decisions and limitations, use of COTS software), or other factors that affect the architecture (e.g. culture of the organisation, state of technology, existence of a related product line). We call these the Architecture Design Requirements (ADR), and this is one of the critical inputs to architecture design.

Next the architect should receive guidance as to how to apply his or her experience to the ADR and come up with an architecture design. When this guidance is simply in the form of activities to follow, there are no intermediate artefacts between ADRs and the structural models of the architecture. This leaves documenting rationale and traceability a free-form task left to the architect, and the task is often overlooked. If instead the architect receives guidance and a model for capturing the results, this will improve his or her ability to document rationale and traceability.

Thus we propose that a solution for narrowing the gap should address the following criteria.

Focus attention on the architecture design requirements (ADR): ADRs are the requirements, quality attributes, constraints, and other factors that affect the architecture design. These should be documented along with their key characteristics and relationships, namely: whether the ADR is negotiable, and if so, how and to what extent; how stable each ADR is, both during development and over the lifetime of the system; conflicts among ADR; and relative priorities of ADR.

Relate the ADR and the structural models of the software architecture with a model of system solution strategies (SSS): The SSS should give solution information, rather than articulating the problem as the ADR do. The SSS should include both structural and non-structural strategies. An example of a structural strategy might be to use a virtual machine layer, and a non-structural strategy might be to make it easy to add new customer modes. These should be documented, including their context, rationale, and how they are related to the ADR. Alternative strategies should also be documented, eventually including the reasons they were discarded.

Guide the architect in using the SSS to develop the software architecture: The architect next needs support in using these solution strategies to design the structural aspects of the architecture. This could be in the form of guidance for comparing and choosing solution strategies or guidelines. The results of this activity should be documented, namely by capturing where SSS are used in the architecture models and why. Alternative strategies could also be documented here, rather than along with the SSS.

Support other parts of the software development process: The activities centred around the ADR and SSS can and should have an impact elsewhere in the software development process, by:

- Providing feedback about the feasibility, completeness, and consistency of the SRS.
- Ensuring that architecturally significant requirements are identified.
- Identifying risky aspects of the development and communicating them to stakeholders.
- Communicating important architectural decisions to stakeholders.
- Establishing traceability from architecture to requirements and vice versa.

These potential uses and effects should be made explicit. The format and underlying model of the ADR and SSS and the approach used to develop them can either facilitate or hinder their use in the software development process.

3 Global analysis activities and artefacts

The Siemens Four Views approach arose out of a study of the software architecture of industrial systems, and was based on the best practices we observed [6]. We found that the documentation of software architectures typically consists of diagrams conveying structural information and accompanying text explaining these diagrams. What was missing was information about the context and rationale for the decisions embodied in these diagrams.

Thus our design approach has two basic parts: one is to design and describe the structure of the system. Here we use four complementary views to reduce the complexity of designing and understanding the architecture. Different views have different engineering concerns, so not all aspects of the system must be considered at once. Our four views are the conceptual, module, execution, and code architecture views.

The second part of our approach is Global Analysis. Global Analysis produces documentation artefacts that describe the context and rationale for design decisions. But in addition to providing a format for capturing this important information, Global Analysis gives the architect guidance in how to get from the SRS to the structural views of the architecture.

The bulk of Global Analysis is done before the architecture view design but there is also feedback, so Global Analysis may be revisited when new information comes to light during view design. Equally important is that Global Analysis occurs before the design of each of the views, so there will be a number of iterations between Global Analysis and view design. Finally, the Global Analysis artefacts support traceability between the views and the SRS, and this traceability information is captured as the views are designed.

Figure 1 presents the model of the information that should be captured as Global Analysis proceeds. In [2] we describe specific templates for capturing this information, where

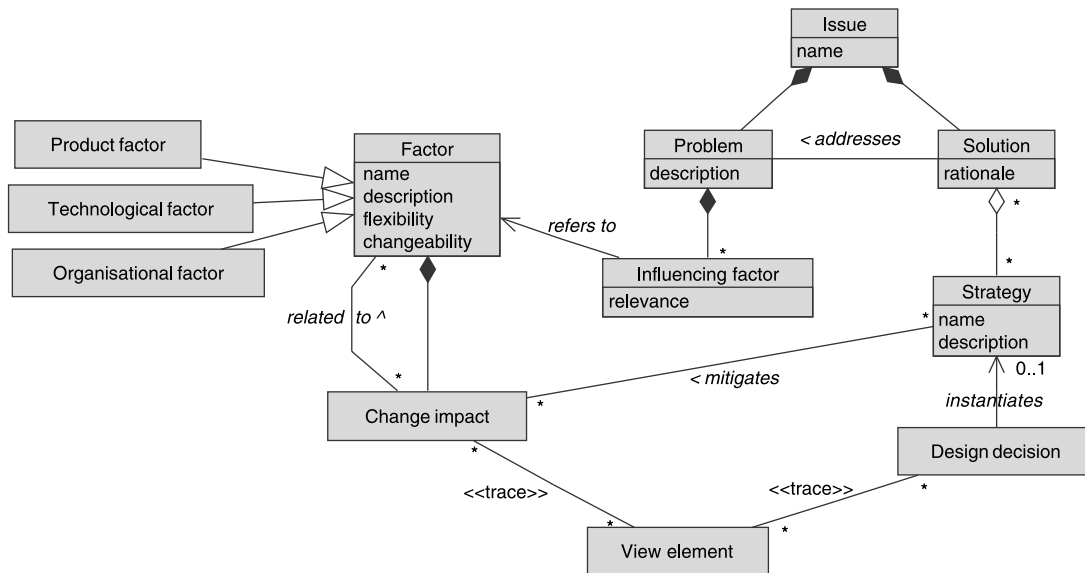


Fig. 1 UML model of Global Analysis artefacts

some of the relationships shown in the model are instead described in text fields when using the templates. The forces influencing the design are captured as a set of *Factors*. Because factors are not necessarily independent, architects rarely consider a single factor at a time but instead look at sets of related factors. The architect may also group a set of factors together because they contribute to some larger design problem, which we call an *Issue*.

3.1 Factor Tables

Although there could be hundreds or thousands of factors that influence the software design, in practice an architect would not describe and analyse them all. An architect might work backwards from trying to solve a particular problem; this would set the context and the scope of the factors to consider. Or an architect might be exploring factors with which he or she has little experience or anticipates difficulty. The architect would consider and filter a large number of factors to quickly identify those worth analysing in more detail.

We developed the Factor Table and related process activities to assist the architect in understanding the important influencing factors. Trying to balance trade-offs between providing simple, flexible guidelines against more comprehensive guidelines, we ended up with a table format listing the name, description, flexibility and changeability,

and impact of the factor. An example of a Factor Table is given in Table 1. In terms of the model shown in Fig. 1, the Factor Table captures the information of Factor, Change Impact, and their associations. Rather than add more columns to the table (or subdivide the categories) for the other properties, we relied on the analysis questions that were formulated for the process activities that guide the architect in filling in the table.

Part of the process of describing the factors is determining those significant enough to include in the table in the first place. These include those with a significant global influence, those that could change during development, those that are difficult to satisfy, and those with which the architect has little experience. Associated screening questions to help make this determination include: can the factor's influence be localised to one component in the design, or must it be distributed across several components? During which stages of development is the factor important? Does the factor require new expertise? The analysis during these questions helps determine the description of the factor.

For those factors listed, additional analysis is done to assess their architectural significance in terms of change and impact. We identify two types of change: flexibility, where the architect could negotiate a change to the factor, and changeability, where the change is not under the architect's control and could change during development, as the system

Table 1: Example of Factor table

Factor	Flexibility/Changeability	Impact
O4.2 Schedule Feature Delivery		
Features are prioritised	Negotiable	Moderate impact on the schedule
T2.1 Domain-specific Hardware Probe Hardware		
Hardware to detect and process signals	Upgraded every 3 years as technology improves	Large impact on image acquisition and processing components
P1.1 Features Acquisition Types		
Acquire raw signal data and convert into images	New types of acquisitions may be added every 3 years	Affects UI, acquisition performance, and image processing

evolves, or across variations of products within a product family [7].

The impact (Change Impact) of a factor captures the potential effect of a change in this factor, so impact can refer to other factors, and/or can eventually trace to particular elements in the design (to particular view elements). Organisational factors tend to be analysed with respect to flexibility and impact on other factors, whereas technological and product factors tend to be analysed with respect to changeability and impact on design elements. In addition, impact can describe how a particular design strategy could mitigate the effect of a change to the factor. Thus the impact is used to describe three different kinds of relationships: how a change in this factor could affect another factor, how it could be minimised by the use of a particular design strategy, and/or how it could affect a particular element in the design. Note that these three kinds of relationships would not all be known at the start of Global Analysis; in particular the relationship to a design element could not be known until the views were at least partially completed. We represent these in a single text field to keep things simple and flexible, although this may not have been the best choice.

The relationships among the factors themselves include: conflicts, relative priorities, and categories. The factors in the table are organised into a three-level hierarchy according to category: the first level divides them into organisational, technological, and product factors. The second level groups related factors by type, which could come from a company-specific taxonomy or a general-purpose taxonomy such as we used. The third level is the factor itself.

Another way of grouping factors is according to their influence on the architecture, and this grouping occurs in the *Issue Cards*. The Issue Cards are also where conflicts among factors and relative priorities of factors are recorded.

3.2 Issue Cards

The purpose of the Issue Cards and their associated process activities is to assist the architect in identifying key design problems that arise from sets of factors, and in identifying a feasible solution. The solution consists of a set of design strategies, some of which could be alternate approaches, but which taken together solve the design problem. An example of an Issue Card is given in Fig. 2. In terms of the model from Fig. 1, the Issue Card captures the information of Issue, Problem, Influencing Factor, Solution, Strategy, and their relationships. As we said above, an architect might

arrive at the architecturally significant issues first and then identify relevant factors, or start with factors and use them to identify the key design issues.

The attributes of an Issue include its name, a description of the problem to be solved, the influencing factors and their relevance to the problem. Conflicts among and relative priorities of the factors should be included in the relevance field. The other attributes of Issues are a solution with accompanying rationale, and a set of strategies, each with a name and description.

Issue Cards are a way of relating design problems, their influencing factors, and strategies. Thus they connect the problem space to the solution space. The description of the issue or the design problem, together with an analysis of the influencing factors, provides the context for the strategies.

In addition to the Issue Card, we found it useful to summarise the information in a separate table that cross-references issues, influencing factors, and applicable strategies. We call this the Issue Summary Table.

3.3 Strategies

Strategies give specific guidelines about the architecture design. The strategies in an Issue Card are part of an overall solution, and the explanation of this solution provides the rationale for the strategies. As described in Section 2, strategies can be either structural or non-structural: Table 2 gives examples. The structural strategies are mainly driven by technological and product factors. Non-structural strategies generally arise from organisational factors.

3.4 Design

Issue Cards are used by the architect during design of the architecture views. The strategies are instantiated as design decisions that determine the types and number of design elements. Global analysis is a part of architectural design and is an iterative process. As the architecture is being formulated and decisions are made, this gives rise to additional constraints that could lead to new factors or the refinement of strategies into new issues. New requirements could also be formulated.

When used in the view design, strategies are translated into specific design decisions. The strategy describes a general rule, and the design decision shows how this is reflected in the view, naming specific components. An example is shown in Table 3.

Issue: Easy Addition and Removal of Acquisition Procedures
There are many acquisition procedures. Implementation of each feature is quite complex and time consuming. There is a need to reduce complexity and effort in implementing such features.
Influencing Factors O4.1: Time to market is short. O4.2: Delivery of features is negotiable. P1.1: New acquisition procedures can be added every three years. P1.2: New image-processing algorithms can be added on a regular basis. ...
Solution Define domain-specific abstractions to facilitate the task of implementing acquisition and processing applications. Strategy: Use a flexible pipeline model for image processing. Develop a flexible pipeline model for implementing image processing. Use processing components as stages in the pipeline. This allows the ability to introduce new acquisition procedures quickly by constructing pipelines using both old and new components. Strategy: Introduce components for acquisition and image processing. ... Strategy: Encapsulate domain-specific image data. ...
Related Strategies See also Encapsulate domain-specific hardware.

Fig. 2 Example of Issue Card

Table 2: Examples of non-structural and structural strategies

Non-structural Strategies from Org. Factors	Structural Strategies from Tech. Factors	Structural Strategies from Prod. Factors
<ul style="list-style-type: none"> • Reuse existing components • Build rather than buy • Make it easy to add or remove features 	<ul style="list-style-type: none"> • Encapsulate hardware • Separate processing, control, and data • Use vendor-independent interfaces 	<ul style="list-style-type: none"> • Use feature-based components • Separate the user interaction model • Separate time-critical components

Table 3: Example of Design Decision table

Design Decision	Rationale
Decompose the Exporting component into ImageCollection, Comm, and Export components. ImageCollection and Comm handle the domain-specific image data	Strategy: Encapsulate domain-specific image data

4 Experience with Global Analysis

In this Section we examine how Global Analysis was used in four new systems (referred to as A, B, C, and D), and compare that to the way we intended it to be used. Tables 4–8 show the details of this comparison in terms of the criteria set out in Section 2. Before describing the Global Analysis experience system by system, we first summarise how the original version of Global Analysis relates to these criteria.

As can be seen in Table 5, in Global Analysis the Factors serve as ADRs. Global Analysis goes beyond the criteria for ADR by providing another important artefact that is part of the problem space. Issues, as recorded on Issue Cards, identify the key architecture design problems faced by the architect. This means that identifying issues brings the architect closer to the solution space, because he or she has identified what to consider first when designing the architecture.

In Global Analysis the strategies serve as SSS (Table 6). They are in the solution space because they give specific guidelines about the architecture design. Issue Cards are a way of relating design problems, their influencing factors, and strategies. Thus they connect the problem space to the solution space. The description of the issue or the design problem, together with an analysis of the influencing factors, provides the context for the strategies. The strategies in an Issue Card are part of an overall solution, and the explanation of this solution provides the rationale for the strategies.

Applying the solution strategies to design the structural aspects of the architecture is a part of the view design. Table 7 summarises how the Siemens Four Views approach fulfils the criteria for using strategies in the architecture design.

The key link between strategies and the views is in the design decisions, which show how strategies are embodied in the views. We initially created Design Decision Tables (as in Table 3) to succinctly summarise the process of getting from Global Analysis to the view design, and then learned that architects found these tables useful as documentation artefacts.

Finally, Table 8 illustrates how Global Analysis results are used in many software development activities to provide feedback, improve communication among stakeholders, identify and manage risks, identify, communicate, and manage design decisions, and project planning. The last four rows in this Table describe additional roles that Global Analysis fulfils that are not part of the requirements from Section 2.

4.1 Global Analysis experience

Global Analysis was first applied by others to system A, a software system for acquiring and processing meter data from electric, gas, and water meters [8]. For system A the Global Analysis activities were adapted somewhat from our original approach. The factor analysis proceeded as we had intended, making heavy use of the catalogue of example factors in order to identify relevant factors (Table 5).

However, Global Analysis was not done iteratively, as we expected. It was completed early in the project, at the start of architecture design, and used for identifying project risks and to guide the architecture design, but no further work was done on Global Analysis itself. One result of this was that the factor impact was never revisited to record the traceability to specific elements in the architecture design. In addition no factors, issues, or strategies were added or revised as the design progressed. We can attribute no direct negative consequences

Table 4: Number for Factors, Issues, and Strategies for example systems

	System A Data management	System B Image management	System C Business management	System D Automation management
Number of Factors:				
Organisational factors	14	9	28	28
Technological factors	8	7	22	14
Product factors	7	11	28	25
Number of issues	11	3	19	23
Number of strategies	24	21	100	64

Table 5: How Global Analysis (GA) fulfils criteria for ADR

Criteria for ADR	GA Intent	GA Experience			
		A	B	C	D
ADRs include all architecturally-significant requirements and quality attributes observable by a user	Via Prod. factors	Focus on risky ones	Focus on those likely to change over the lifetime	Focus on those important to stakeholder	Focus on risky ones and those concerned with variation
ADRs include the relevant quality attributes that affect the development process	Via Prod. and Tech. factors	Catalogue of example factors was very helpful	Catalogue of example factors was very helpful	Catalogue of example factors was a helpful starting point. Twice as many new ones were added	Catalogue of example factors plus the factors from system C were very helpful
ADRs include all relevant constraints	Via Tech. and Org. factors				
ADRs include all other factors that affect the architecture	Via Tech. and Org. factors				
Negotiability of each ADR is captured	Via factor flexibility	As intended	As intended	As intended	As intended
Stability of each ADR is captured	Via factor changeability	As intended	As intended	As intended	As intended
Conflicts among ADR are captured	Considered when identifying issues; captured in Issue Card	Not captured because Summary table was used for issues	As intended, but captured in Issue text document	Considered when analyzing the factors, captured in the factor impact	As intended
Relative priorities of ADRs are captured	Considered when identifying issues; not explicitly captured	As intended	As intended	Considered when analyzing the factors, captured in the factor impact	As intended

to the latter, but believe that it limits the usefulness of Global Analysis in guiding the architecture design and in capturing the traceability and rationale for the design.

Another departure from the original Global Analysis was that Issues were not recorded in Issue Cards; instead the Summary table, which we intended to be used for cross-referencing, was used exclusively. While the Summary table seems to have served the use of Global Analysis in this project, the lack of Issue Cards means that information about the context and rationale for strategies and capture of alternative strategies is missing (Table 6). This lack may become more noticeable in the future as the system is maintained. In all, 11 issues were identified along with 21 strategies to address them. After a series of design discussions, some of these strategies were combined to derive three additional strategies. These 24 strategies were then summarised, grouping similar strategies together.

Results of Global Analysis were used in two different ways that were not envisioned by us. First, the project manager used the results to identify technical risks, which

were then managed as project risks. Second, six development strategies were concluded from the 24 design strategies and followed in the development of the product (Table 8).

System B is a modest product family consisting of four products with common core assets. The products are interactive tools for viewing, processing, visualising, annotating, and analysing images, and for report generation. System B applied Global Analysis after the architecture design had begun, using it primarily to analyse problems that were anticipated and to uncover additional ones. The focus of factor analysis was on factors likely to change over the lifetimes of the products (Table 5). As with System A, the Global Analysis was not done iteratively, giving it a more limited role than we had envisioned.

System B also departed from the original Global Analysis in its capture of the factors and the issues. Instead of Factor Tables, equivalent information was recorded in sub-sections in a text document. This was done to make the description more compact and to facilitate change-tracking. Instead of Issue Cards, System B documented

Table 6: How Global Analysis (GA) fulfils criteria for SSS

Criteria for SSS	GA Intent	GA Experience			
		A	B	C	D
SSS give solution information rather than articulating the problem (as the ADR do)	Via Strategies	As intended	As intended	As intended	As intended
SSS include both structural and non-structural strategies	Via Strategies	As intended	As intended	As intended	As intended
Context for SSS are captured	Via contributing factors for Issues	Partially captured	As intended	Partially captured	As intended
Rationale for SSS are captured	Via Issue solution	Not captured	As intended	Not captured	As intended
Relationships between ADR and SSS are captured	Via Issues: Factors contribute to it and Strategies address it	As intended	As intended	As intended	As intended
Alternative SSS are captured	Via Issue solution. Issue Card can contain alternative Strategies	Not captured	As intended	Not captured	As intended

Table 7: Siemens Four Views and use of SSS in design

Criteria for using SSS in Design	Siemens Four Views Intent	Siemens Four Views Experience	
		A - C	D
Support architect in using SSS to design the structural aspects of the architecture	Design tasks for views provide activities and guidelines for creating an initial model according to constraints, functionality, and vocabulary of the view. Strategies provide guidance in refining the design to ensure that the Issues are solved	As intended	Not known
Record where SSS are used in the architecture models and why	Use Design Decision Table to show strategies that provide the rationale for the design decision	Not used	Not known
Alternative SSS are captured	Strategies can be recorded in Design Decision Table	Not used	Not known

Issues and Strategies in a text document that also contained context and rationale, and again the format appears to have worked at least as well as the original. It has the advantage of encouraging a more extensive discussion of the design rationale [9, 10].

For System B the Global Analysis was seen as contributing to a number of benefits of the design, some foreseen and some unforeseen. The analysis helped them make good decisions about what belonged together (cohesion), thus reducing dependencies (coupling), and accommodating product variation, product evolution, and changes during development. The decisions that caused difficulty during development were all related to factors that were identified and analysed in Global Analysis, so either the analysis was lacking in detail or the impact of the factor was misjudged.

For System C the Global Analysis was applied in the very early concept phase of development, before requirements analysis had been completed. Thus it played a larger role than usual in shaping the requirements and market positioning of the system (Table 8).

System C did use Factor Tables, and this system had far more Factors, Issues, and Strategies than Systems A or B (Table 4). While Systems A and B relied heavily on the original catalogue of typical Factors when identifying factors, for System C many new kinds of factors were identified. These could be useful in extending the original catalogue. Another variation that came up in the Factor Tables of System C was the information included under Factor impact. We had intended factor impact to be used in at least two different ways. In the early design stages, the

Table 8: How Global Analysis (GA) fulfils roles in the software development process

Role in Development Process	GA Intent	GA Experience			
		A	B	C	D
Providing feedback about the feasibility, completeness, and consistency of the SRS	Use Factor Tables; flexibility indicates potential for negotiating changes in SRS	Focused on feasibility	Not known	Focused on feasibility and completeness	Focused on feasibility
Increasing interaction among stakeholders to determine which requirements are architecturally significant	Use Product Factors	Somewhat useful	Not used since factors and issues were documented after the requirements elicitation phase	Very useful. Global analysis workshop facilitated stakeholder participation	Very useful. Global analysis workshop facilitated stakeholder participation
Identifying risky aspects of the development and communicating them to stakeholders	Use Issues	Identified top 5 issues	Identified top 3 issues	Identified risks during evaluation of architecture	Identified risks during evaluation of architecture vision
Communicating important architectural decisions to team members	Use Issue Cards	Used Issue Summary Table	Used in conjunction with architecture-centric software project planning	Used Issue Summary Table	Used Issue Cards
Establishing traceability from architecture to requirements and vice versa	Use Factor Impact and Design Decisions	Not used	Deficiency noted: cannot readily trace from factors to requirements	Not known	Not known
Facilitating architecture evaluation	Use all GA and design artefacts	Not used	Not known	Very useful	Used for evaluating architecture vision
Supporting project planning	Not envisioned	Was the main role of GA	Was the main role of GA	Not known	Not known
Accommodating requirements changes during development	Strategies take factor changeability into account	As intended	Was an important benefit of GA	Not known	Not known
Managing revisions to decisions during the lifetime of the system	Artefacts can be put under revision control	No significant revisions	No significant revisions	Not known	Not known

impact helped identify strategies that could enable architecture to respond gracefully to future changes in a factor. In the later design stages, it helped trace and propagate changes to particular elements in the architecture views. System C also recorded priorities of factors and conflicts among them in the impact field (Table 5).

As with System A, the Issue Cards were not used and a Summary table was used instead, so again the context and rationale for strategies were not captured (Table 6). The large number of Factors, Issues, and Strategies used in this system suggests a need for new ways of managing this information, particularly for cross-referencing and navigation.

One particular use of Global Analysis was not envisioned by us. The results were used to achieve consensus among various stakeholders with respect to prioritisation of business strategies, design problems, and design strategies. It helped clarify important requirements and issues that were until then only implicitly considered in detailed design. As in System A, a small number (five) of business and development strategies were concluded from the design strategies and followed in the development of the product.

Finally, in System D, Global Analysis was performed as a first step toward designing a product line architecture for an

automation management system. The system helps monitor status of heterogeneous material handling components in a transparent manner. Vendor-specific specifications of components are imported into the system so that their run-time status can be monitored using a common user interface.

Like System A, System D used Factor Tables as originally intended in Global Analysis, making heavy use of the catalogue of example factors (extended by System C) in order to identify relevant factors. An additional set of product factors were added to characterise the modifiability requirements, for example, configurability, customisability, extensibility, and upgradeability, of the product itself.

Issues and Strategies were reverse-engineered through extensive interviews with development and project engineers. These were then documented in Issue Cards. Time allocated for Global Analysis did not allow reverse-engineering of context and rationale of these strategies.

Results of Global Analysis served to create a single document that described the collective understanding of the existing architecture. It captured what was only implicitly known by development and project engineers and documented it for future use.

4.2 Lessons learned

As can be seen from the experience, Global Analysis plays an early and integral role in architecture design. Global Analysis helps the architect focus on key design problems, communicate with stakeholders, get consensus on architectural challenges, support architecture evaluation, and record the architectural rationale for use by developers. Global Analysis was not always used in the iterative way we recommend. Global Analysis was sometimes used by project managers for identifying risks to be managed (technical, schedule), planning releases, scheduling dependencies, and developing managerial strategies.

We have observed some additional things about Global Analysis based on general feedback, common misunderstandings, and the criteria from Section 2.

Global Analysis users find the categories of factors and example factors very useful. We did not attempt to provide an exhaustive list but just to show common ones. Systems A and B were similar in scope and reused many of the factors from [2]. System C used those as a starting point and added twice as many. System D was similar in scope to C and used that set of factors as a checklist. Typically Global Analysis users captured 80–150 factors, indicating the need for a tool to help manage them.

We have also been told by architects that seeing examples of typical issues and typical strategies helps them in determining issues and strategies for their system (Table 5). There are also other existing collections of strategies, for example [11–13].

In our documentation of Global Analysis results, we always presented Factor Tables before Issue Cards. One tendency we observed in people performing Global Analysis was that a lot of time was devoted to identifying and characterising factors regardless of their importance. As a consequence, sufficient time was not available to identify and address important issues and problems. We now believe that documentation of Factor Tables and Issue Cards should be done in tandem. Most common factors can be identified and characterised very quickly. Other factors can be discovered in a goal-directed way, while identifying design problems and the factors influencing them. The set of factors does not have to be complete, but must be representative of the kinds of difficulties, risks, or impact on the architecture that the architect must address.

We believe that explicit consideration of the relative priorities of issues, as Systems A and B did, is generally useful. Although relative priorities of the factors and conflicts among them are considered in Global Analysis, these relationships are considered only in Issue Cards. Factor priorities and conflicts are not explicitly recorded in the Factor Tables, as other approaches to managing dependencies among requirements [14, 15] might suggest. As we saw with Systems A and C, there is a risk that factor priorities and conflicts may not be captured or considered in sufficient detail.

The strong focus on changeability helped result in a design that worked well in the face of major changes during development (System B). We now believe that flexibility and changeability should not be combined in one field in the Factor Tables. Flexibility covers negotiability while changeability potentially covers not just the stability but also variability, and these should be separated out. Considering variability would help support product lines.

As noted earlier, the Factor Change Impact covers multiple things: early in design it is used to record relationships among factors and suggest strategies for mitigating impact of change. As the views are completed it records traceability. System C also used it for recording conflicts among factors and relative priorities of factors. We now believe these various characteristics should be separated into multiple attributes.

A better and easier way of capturing Issues is needed. The Global Analysis users tend to use a ‘Summary of Strategies’ table rather than Issue Cards. Documenting associated factors and strategies is generally done, but what is missing is the explanation of how the factors are related to the issue, how strategies are related to each other, alternative strategies, and so on. There is a need to support traceability to requirements.

Although the Design Patterns format makes it easier to generate issues and strategies, there is a need to organise and manage issues, strategies, design decisions, and relationship among them. There is also a need to support incorporation of descriptions of issues and selected strategies as text in the architecture design document.

5 Related work

Requirements engineering [16] addresses concerns similar to those that arise when analysing product factors that influence the architecture. Global Analysis broadens this specialty to include organisational and technological factors to ensure the system is buildable and uses the results of the analysis to develop strategies that guide design.

The description of requirements is often textual, but more rigorous requirements analysis methods may employ some combination of feature modelling [17], use case modelling [18, 19] or object modelling [18]. If such an approach is used, then the artefacts can provide useful input to Global Analysis. Features can be put in Global Analysis Factor Tables for further analysis. Use cases show a specific interaction between a stakeholder and the system and can provide a means to compare and evaluate alternative design decisions. Objects encapsulate system responsibilities and can guide the choice and composition of conceptual components in the Global Analysis strategies.

Global Analysis recognises the role of quality attribute requirements. Others have written on the importance of quality attributes, also known as non-functional requirements, on software architecture [20–23].

In recent years there have been several software architecture design methods centred around quality attri-

butes, including the Attribute Driven Design (ADD) method [20] and the QASAR method [24]. These methods depend on the designer to organise the requirements and to find solutions for the satisfaction of specific requirements.

More recent work by the authors of the ADD method is leading to providing the designer guidance in finding solutions. It provides a format for capturing quality attribute requirements and uses architectural tactics to transform an architecture to more closely meet its quality attribute requirements [25, 26]. It requires tool support because of the depth of the design.

Linking requirements to the architecture has parallels to documenting design rationale. Issue-based decision making [27, 28] is based on a model of issues, positions, and arguments. An issue states a problem and is responded to by a position, which in turn is supported by or objected to by various arguments. The Issue Card specialises this information for architecture design and uses the terms issues, strategies, and rationale.

Now others are also beginning to create explicit models linking architecturally significant requirements to the architecture, rather than simply treating this as a series of process steps. More recent work by the author of the QASAR method is viewing software architecture as a composition of architecture design decisions [1]. Design decisions have an explicit representation including a solution part and a requirements part, somewhat analogous to a design pattern. These could provide a more detailed way of describing the Global Analysis strategies. Kruchten [29] discusses what an ontology of architectural design decisions should look like.

The BAPO approach documents architecture in the context of business, process, and organisation; architecture is described in terms of five views [26, 30]. The first three views are in the problem space and describe the customer drivers, quality requirements, and quality properties. These are what others call the business goals and quality attribute requirements. The last two views are in the solution space and describe solution mechanisms. These views and context helps partition the problem space and the solution space but the gap between them remains.

The relationship between requirements and architecture has been the focus of two workshops [21, 31]. Some papers focus on the relationship itself. The Twin Peaks model of software development [32] suggests that problem structures and solution structures be iteratively refined. Other papers focus on bridging the gap with intermediate descriptions. For example, an architectural prescription [33] is the architecture of the system in terms of its components, the constraints on them and the interrelationships among the components (i.e. the constraints on their interactions). These work to bridge the gap by bringing the architecture side closer to the requirements side.

6 Conclusions and future work

Global Analysis provides a way of capturing what good architects are already doing and provides some insight into how to do it better. Global Analysis helps the architect make the conceptual leap from the requirements to architecture design by analysing the impact of requirements on important technical and business issues that affect design. Global Analysis records rationale and provides traceability as requirements are linked to strategies that guide design. A key contribution is the issue-oriented organisation of strategies and how Issue Cards link architecture design requirements and solution strategies.

We have gained experience with this approach in three ways. First we developed the approach informally while

designing the architecture for an image acquisition and processing system. Second, we did a retrospective analysis of four existing systems, interviewing the architects to understand the process they used to go from requirements to design, and getting their feedback on the resulting Global Analysis approach and the artefacts captured for their systems. Third, Global Analysis is being used in new software development projects. The result is the production of Global Analysis documents that are used by the architect, project manager, and other stakeholders.

This experience has given us insight into how to improve the Global Analysis approach. A catalogue of common factors, issues, and strategies is emerging. As experience grows these may be codified. It would be useful to identify a core set of factors, issues, and strategies applicable to all systems. This could form the basis of a Global Analysis checklist, used in conjunction with a template. Ideally this would be an integral part of design assets and not viewed as an extra documentation obligation.

Issue Cards were inspired by design patterns and pattern languages [34, 35]. Further study and codification of the artefacts is needed to see them effectively adopted in practice. A better articulation of the solution field in the Issue Card is needed, explaining the dependencies and trade-offs among the strategies and how they might be used separately or in conjunction with one another.

There is value in creating a Global Analysis document at the beginning of architecture design to support management functions. However, Global Analysis is not meant to be a static document but one that evolves as the architecture is designed. Ironically, having a documented artefact seems to push Global Analysis towards a one-time milestone, and it loses its iterative nature. The architect needs better support in this iterative process.

The Global Analysis results need to be presented in different ways to different stakeholders. Stakeholders see it from their individual perspectives. For example, we saw documents where strategies were grouped by issues, by project recommendations, and by architecture structure that they influence.

In conclusion, there are still gaps in the process of moving from requirements to software architecture but with Global Analysis they are smaller ones. Requirements that emerge from factor analysis are more suited to reasoning about design. Strategies are still abstract and some work is needed to get to concrete design decisions. The benefits that have been already realised in our experience with the approach include: documented factors and design strategies that guide the architecture design; inputs for developing project strategy conclusions, goals, and risks; and improved documentation of the architecture. These applications give us confidence that the approach is practical and helpful.

7 Acknowledgments

Thanks to Dan Paulish and Bob Schwanke and other Global Analysis users for sharing their experiences.

8 References

- 1 Bosch, J.: 'Software architecture: the next step'. in Oquendo, F., Warboys, B., and Morrison, R. (Eds.): *Software Architecture, First European Workshop (EWSA)*, St Andrews, UK, 21-22 May 2004, (*Lect. Notes Comput. Sci.*, **3047**, pp. 194–199)
- 2 Hofmeister, C., Nord, R., and Soni, D.: 'Applied software architecture' (Addison-Wesley, Reading, MA, 2000)
- 3 IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications - Description
- 4 Boehm, B.W.: 'Requirements that handle IKIWISI, COTS, and rapid change', *Computer*, 2000, **33**, (7), pp. 99–102

- 5 Boehm, B.W.: 'Software architectures: critical success factors and cost drivers'. Proc. 16th Int. Conf. on Software Engineering, Sorrento, Italy, 16-21 May 1994, (ACM Press, New York, 1994), p. 365
- 6 Soni, D., Nord, R.L., and Hofmeister, C.: 'Software architecture in industrial applications'. Proc. 17th Int. Conf. on Software Engineering, (ACM Press, New York, 1995), pp. 196-207
- 7 Nord, R.L., Hofmeister, C., and Soni, D.: 'Preparing for change in the architecture design of large software systems'. Position paper, TC2 First Working IFIP Conf. on Software Architecture, 1999 (WICSA1) available at: <http://www.ece.utexas.edu/~perry/prof/wicsa1/>
- 8 Paulish, D.J.: 'Architecture-centric software project management: a practical guide' (Addison-Wesley, Boston, MA, 2002)
- 9 Schwanke, R.: 'Architectural requirements engineering: theory vs. practice', in Berry, D., Kazman, R., and Wieringa, R., (Eds.): Proc. Second Int. Workshop From Software Requirements to Architectures (STRAW'03), 2003, pp. 1-8
- 10 Schwanke, R., and Lutz, R.: 'Experience with the architectural design of a modest product family', *Softw. Pract. Exp.*, 2004, **34**, pp. 1273-1296
- 11 Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M.: 'Pattern-oriented software architecture: a system of patterns' (John Wiley & Sons, Chichester, England, 1996)
- 12 Davis, A.M.: '201 principles of software development' (McGraw Hill, New York, 1995)
- 13 Reichtin, E., and Maier, M.: 'The art of systems architecting' (CRC Press, Boca Raton, FL, 1997)
- 14 Boehm, B.W., and In, H.: 'Identifying quality-requirement conflicts', *IEEE Softw.*, 1996, **13**, (2), pp. 25-35
- 15 Chung, L., Nixon, B., Yu, E., and Mylopoulos, J.: 'Non-functional requirements in software engineering' (Kluwer Academic Publishers, Boston, MA, 2000)
- 16 Davis, A.M.: 'Software requirements: objects functions, states' (Prentice-Hall, Englewood Cliffs, NJ, 1993)
- 17 Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., and Peterson, A.S.: 'Feature-oriented domain analysis feasibility study' (CMU/SEI-90-TR-21) Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990
- 18 Jacobson, I., Griss, M., and Jonsson, P.: 'Software reuse: architecture process and organization for business success' (Addison Wesley Longman, New York, NY, 1997)
- 19 Kruchten, P.: 'The 4+1 view model of architecture', *IEEE Softw.*, 1995, **12**, (6), pp. 42-50
- 20 Bass, L., Clements, P., and Kazman, R.: 'Software architecture in practice' (Addison-Wesley, Boston, MA, 2003)
- 21 Berry, D., Kazman, R., and Wieringa, R. (Eds.): Proc. of the Second Int. Workshop From Software Requirements to Architectures (STRAW'03), Portland, OR, 9 May 2003, Available at: <http://se.uwaterloo.ca/~straw03>
- 22 Perry, D.E., and Wolf, A.L.: 'Foundations for the study of software architecture', *ACM SIGSOFT Softw. Eng. Notes*. 1992, **17**, (4), pp. 40-52
- 23 Shaw, M., and Garlan, D.: 'Software architecture: perspectives on an emerging discipline' (Prentice Hall, Upper Saddle River, NJ, 1996)
- 24 Bosch, J.: 'Design and use of software architectures' (Addison-Wesley, Harlow, England, 2000)
- 25 Bachmann, F., Bass, L., and Klein, M.: 'Illuminating the fundamental contributors to software architecture quality' (CMU/SEI-2002-TR-025, ADA407778, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2002)
- 26 America, P., Obbink, H., and Rommes, E.: 'Multi-view variation modeling for scenario analysis', in van der Linden, F. (Ed.): Proc. Fifth Int. Workshop on Product Family Engineering (PFE-5), 2004, Sienna, Italy, 4-6 Nov 2003, (*Lect. Notes Comput. Sci.* **3014**, pp. 44-65)
- 27 Conklin, J.: 'Design rationale and maintainability'. Proc. 22nd Hawaii Int. Conf. on System Science, Kailua-Kona, HI, 3-6 Jan. 1989, (IEEE Computer Society Press, Los Alamitos, CA, 1989), pp. 555-561
- 28 Conklin, J., and Begeman, M.L.: 'gIBIS: a tool for all reasons', *J. Am. Soc. Inf. Sci.*, 1989, **40**, pp. 200-213
- 29 Kruchten, P.: 'An ontology of architectural design decisions', in Bosch, J., (Eds.): Proc. 2nd Groningen Workshop on Software Variability Management, Groningen, Netherlands, 2-3 December 2004
- 30 Obbink, H., Müller, J.K., America, P., van Ommering, R., Muller, G., van der Sterren, W., and Wijnstra, J.G.: 'COPA: a component-oriented platform architecting method for families of software-intensive electronic products'. Tutorial for the First Software Product Line Conf., August 2000, Denver, CO, Available at: http://www.extra.research.philips.com/SAE/COPA/COPA_Tutorial.pdf
- 31 Castro, J., and Kramer, J. (Eds.): Proc. of the First Int. Workshop From Software Requirements to Architectures (STRAW'01), Toronto, Canada, 14 May 2001, (IEEE Computer Society Press, 2001), Available at: <http://www.cin.ufpe.br/~straw01/>
- 32 Nuseibeh, B.A.: 'Weaving together requirements and architecture', *Computer*, 2001, **34**, (3), pp. 115-117. Also, a version appears in Castro 2001
- 33 Brandozzi, M., and Perry, D.E.: 'From goal-oriented requirements to architectural prescriptions: the prescriptor process', in Berry, D., Kazman, R., and Wieringa, R. (Eds.): Proc. Second Int. Workshop From Software Requirements to Architectures (STRAW'03), 2003, pp. 107-113
- 34 Coplein, J.O.: 'A generative development-process pattern language', in Coplein, J., and Schmidt, D.C. (Eds.): 'Pattern languages of program design' (Addison-Wesley, Reading, MA, 1995), pp. 183-237
- 35 Meszaros, G., and Doble, J.: 'A pattern language for pattern writing', 1997, Available at: <http://www.hillside.net/patterns/>