A Constraint-Oriented Approach to Software Architecture Design

Mark van den Berg
Tilburg University,
Tilburg, the Netherlands
m.j.p.vdnberg@uvt.nl

Antony Tang
Swinburne University of Technology, Melbourne,
Australia
atang@swin.edu.au

Rik Farenhorst
VU University,
Amsterdam, the Netherlands
rik@cs.vu.nl

Abstract

Software architecture design constraints exist and they bound the solution space in some ways. However, in research and practice little is known about the characteristics of these constraints and how they influence decision making. In this paper we report our findings on design constraint characteristics based on case studies in two countries. We discovered how constraints typically manifest themselves in the architecture design process, and how they impact the architectural decisions taken. Based on these insights we suggest a number of implications and strategies to support architectural design.

1. Introduction

Software architecture has received an increasing amount of attention in research literature and is considered one of the most important developments in software engineering over the last decades [1]. By explicitly documenting software architecture design, it improves stakeholder communication, assists system analysis and evolution and facilitates architectural knowledge reuse. Furthermore, the architecture is a primary tool for addressing cross-cutting concerns, safeguarding quality attributes or non-functional requirements, and to enhance stakeholder communication.

However, merely documenting the result of the architecture effort in terms of components and connectors leads to a loss of knowledge about the architectural decisions made. Practicing architects usually have to make numerous design decisions during the architecting process, and must carefully consider all forces in the system environment. By definition, these decisions are critically important to all parts and subsequent design (or even lifecycle) phases of the system. When architectural decisions and their associated rationale are not captured explicitly, this inevitably leads to knowledge vaporization [2], causing high costs for change, unnecessary complexity and architectural erosion and drift [3]. Design rules and constraints are violated, previously made assumptions are overlooked and obsolete design decisions are not removed.

In a 2005 survey of practicing architects, architecture design constraints were recognized as being the most important factor in design rationale [4]. 88% of the respondents perceived design constraints as highly important and reported that they are used very often in architecture projects. 63% of the respondents said they always document constraints, and an additional 20% do so very often. This suggests that constraints hold an essential and well-understood position in any architecture endeavor. Unfortunately, this important aspect is not well reflected in literature yet; to our knowledge few researchers propose methods, tools, or frameworks that treat design constraints as primary concept in the software architecting process.

In the software architecture research community, there is a consensus that external forces exist that restricts the architect’s choice of solution space and we call them constraints. Although constraint is mentioned as a factor in architectural decision making [5-10], a clear and agreed-upon definition of the concept is missing. Requirements (functional, non-functional, system, user, etc.) are widely accepted as critical inputs to architecture design, but there has been little study on how they bound the solution space. Design constraint is a property of requirements and other factors, there is a difference between them. Design constraint represents how requirements and other factors bound the choice of design in the solution space. The more constraining these factors are the less choice in the solution space.

We argue that studying the concept of design constraints enhances the understanding of design decision making if we know how requirements and various factors relate to the design solution. As such,
we investigate in this paper the fundamental properties of design constraints and what impacts they have on architecture design.

To improve our understanding of design constraints in practice, we have conducted retrospective case study research on five completed architecture projects in the Netherlands and Australia. From this study, we have discovered ten design constraint properties and how they impact the architecture design process. This gathered understanding is culminated in a number of implications about how to utilize constraints in architecture design reasoning. These lessons support architects by creating awareness for design constraints, and also enable them to better understand, verify and maintain architecture design.

The remainder of this article is organized as follows. In Section 2, we discuss related work and develop a definition of design constraints. In Section 3 we draw on our case study experiences to elaborate on how design constraints manifest themselves in the design process and what their impacts are. In Section 4 we provide a number of techniques to be used by practicing architects to effectively deal with constraints in their projects. Section 5 concludes this paper.

2. Design constraints

A growing number of researchers have recognized that an information-intensive system architecture should include, or even be defined as, the collection of design decisions that were taken during the architecture process, as well as their rationale [2, 5-7]. Treating architectural decisions as central constructs of the architecture allows for guarding its conceptual integrity, improved design space exploration, improved analysis and improved traceability [7]. A simple description of what has been decided will not, by itself, produce all these benefits. Equally important is to document the rationale behind these decisions – all pros and cons, alternatives considered, et cetera. Decisions, rationale and related knowledge concepts are commonly known as architectural knowledge [11].

Various models of architectural knowledge have been proposed in literature recently [5-8, 12]. Naturally, design decisions are central to these models, with associated rationale expressing reasoning and intent. Researchers often mention constraints as a factor in architectural decision making, although they employ various interpretations of the term. Tyree and Akerman, who propose a template for describing architecture decision rationale, consider constraints to be possible outputs of a decision, which may be an input to subsequent decisions [6]. Kruchten proposes an ontology of architectural design decisions [5]. In this ontology, design constraints are a negative type of ‘property’ decision, stating some trait the system will not exhibit. A constraints-dependency between two decisions is also suggested. Jansen and Bosch provide a model of architectural design decisions in which they relate problems to a set of possible solutions, one of which is selected in a decision that makes trade-offs and results in an architectural modification of a context [7]. In their model, constraints are decision outputs prescribing what is not allowed in further architectural design.

In their paper introducing Architecture Rationale and Elements Linkage (AREL) for architecture design rationale modeling, Tang et al. provide a simple conceptual model of design reasoning [8]. They argue that design reasoning comes in two forms: motivational reasons motivate or constrain a decision, and design rationale encapsulates the justification of a decision. A decision results in a design outcome, which may motivate or constrain other decisions. In AREL, motivational reasons and design outcomes are considered architecture elements – artifacts that form part of the architecture design. Architecture elements are thus inputs as well as outputs of design decisions, which are represented by the architecture rationale. By specifying the causal relationships between architecture elements and rationale, one can express the design graphically by a decision network. Tang et al. suggest four types of motivational reasons: requirements, assumptions, constraints and previous design outcomes. Here, constraints are limitations of technical, business, organizational or other nature to what can be achieved. This view is roughly adopted from Roelder et al., who – while stating that it is tricky to draw the line between assumptions, requirements and constraints – describe constraints as a kind of system-limiting requirement [9].

Many researchers have considered both requirements and constraints without distinguishing between the two. Whilst the presence of contextual forces and limiting factors influencing design decision-making is often acknowledged, there is no uniform interpretation or formal representation of the concept.

From the perspective of constraint programming, a problem-solving paradigm from artificial intelligence, and operations research which is typically used in scheduling and vehicle routing [13], many problems can be expressed in the form of variables with domains, and constraints restricting the feasible combinations of the variables’ values [14]. A constraint on a sequence of variables is a logical relation on their domains – a statement saying which combinations of values from
the variable domains are admitted [15]. A constraint satisfaction problem (CSP) is a finite set of constraints, each on a subset of the given sequence of variables.

In classical CSPs, the most basic type of CSP, constraints are either completely satisfied or completely violated. To solve a classical CSP one needs to find a value for each variable (a solution) in such a way that all constraints are satisfied [14]. We may be interested in finding all allowed solutions, or just one, with no preference as to which one. Often in reality, not all feasible solutions are equally desirable.

A simple technique for determining an optimal solution is the use of an objective function which computes the quality of any solution [15].

If a CSP contains many constraints, it might be impossible to find a solution that satisfies all of them fully. These decision points are said to be over-constrained [16]. Soft constraints provide computational techniques to arrive at the best solution that is possible if one was to allow incomplete constraint satisfaction. Each of these approaches tries to automate, in its own way, constraint relaxation; it decides which constraint must be weakened, and how, in order to optimally solve a particular problem.

2.1. So, what are design constraints?

A constraint on a sequence of variables is a logical relation on their domains [15]. For design decision-making, this translates to the value of an input to a decision determining the possible values of the outcome of that decision. The input – a so-called design concern – having a fixed value from this decision’s point of view, constrains the domain of the design outcome, which comprises all proposed design options.

As a simple example, consider a design decision of choosing a XML-based messaging platform for communicating between information systems. The organization’s current messaging platform does not support modern XML, so the design options are to (a) modify the existing messaging platform, (b) buy a packaged messaging platform from a technology vendor, or (c) tailor-make an entirely new one. One key concern constraining the decision would be the available budget. The value of this variable (measured in currency units) can dictate the feasibility of the design options in the following way: “Cost of <design option> must be below <budget>.” This definition differs from an objective function in that it disregards the degree of satisfaction or violation. This means that all options satisfying the constraint are equally feasible as far as this constraint is concerned.

Realistically, other inputs are likely to exist as well, which would all impose individual constraints on the design outcome; for the moment we ignore all but the budget constraint. Assuming that modification of the current platform is the least expensive alternative, a low budget could force the architect to opt for this design option. In other words, the other options are unfeasible because they do not satisfy this constraint. If the budget would increase with a sufficient amount, the second least expensive alternative could become feasible as well. Thus, an increase in the value of the ‘budget’ variable enlarges the solution domain. Following this line of argument, we define design constraint as follows.

**Design constraint:** A limiting condition that a design concern imposes upon the outcome of a design decision.

In this definition, the term design decision refers to a decision point or topic. A design constraint can come from a requirement but it should be differentiated from a requirement in two ways: (a) design constraint focuses on how it delimits solutions in the solution space; (b) a requirement can have multiple constraints depending on context of the design decision.

Figure 1 depicts the conceptual framework of this research. Crow’s foot notation is used to express the many-to-many relation between design concerns and design decisions. A design concern can be a requirement, a technical influence or other contextual factors such as budget or development capabilities. A design concern can influence multiple decisions, and each of these relationships constrains a decision in some ways.

![Figure 1: A Conceptual Framework of Design Constraints](image-url)
For each design decision a number of design options are proposed and evaluated against the design constraints. Two special types of design options are denoted in Figure 1. Infeasible design alternatives violate one or more design constraints and are therefore rejected. Design decisions result in design outcomes that must satisfy all design constraints. All other design options (not separately denoted in Figure 1) are feasible – satisfying all constraints – but found to be inferior to the design outcome after further trade-offs and argumentation. A design outcome may turn into a design concern if it motivates or constrains subsequent design decisions.

Design constraints can come from functional requirements, quality requirements, business and IT strategy, system goals and industry standards etc. Other constraints are project resources (budget, schedule, team size, available knowledge, etc.), contractual obligations, bargaining power, resistance to change and political pressure. Furthermore, many design constraints come from the technical system environment, from business practices and legislation, which can profoundly impact the architecture design solution. To emphasize, design constraints are a level of abstraction that allows the uniform handling of various forces restricting the solution space.

3. Analysis of impacts on architectural design

In the next subsections, we present our research methodology and use the case study findings to show the properties related to design constraints. The case studies provide supporting evidence to demonstrate the nature of design constraints and explain the conceptual model shown in Figure 1.

3.1. Case study methodology

A total of five case studies were conducted in the Netherlands and Australia. In the Netherlands, completed architecture projects were investigated in three organizations:
1. A new registration database system for a social security agency (SSA);
2. An information systems architecture of the asset management division of a financial service provider (AMC);

In Australia, we have studied systems from two organizations:
4. A vehicle monitoring system developed for a car manufacturer (CM);
5. A knowledge management system for an engineering firm (EF).

For each project we have gathered documentation with references to design decisions, design constraints of these decisions, and details of the decision problem and the solutions. We studied architecture descriptions, final as well as draft versions, baseline as well as target requirements specifications, reference architectures and policy documents; records of stakeholder communication such as e-mails; and project planning documents. All design decisions implicitly or explicitly expressed in the documents were identified, listed and interconnected. For each design decision, the constraints involved as well as their relative impacts were assessed. In all case studies, multiple semi-structured interviews with architects and other stakeholders were held, to allow relevant constraints to be discovered, as well as to gather all perspectives on and opinions of their nature, impact and priority. The discovered design constraints and design decisions were presented back to the architects who were involved in the study for validation.

Schematic representation of recovered design decisions and constraints was done using AREL (Architecture Rationale and Elements Linkage) [8]. In AREL, design concerns and design outcomes are called architectural elements (AEs) – artifacts that influence and/or form part of the architecture - and architecture rationale (ARs) represent design decisions. These diagrams allowed us to depict the causal relationships of design concerns with design decision and the design outcomes of those decisions, thus enabling the drawing of a decision network.

3.2. Observations from the case studies

The goal in this research was to explore and develop a concept of design constraints, and understand how they influence architectural design decisions and decision outcomes. We have pursued this goal by asking questions and finding evidence to explain the relation between design constraints and architectural design. After constructing and validating AREL decision diagrams for all case studies, we investigated the nature of design constraints. We did so by identifying design constraint properties and finding evidence in the case studies to support them. In the remainder of this section we elaborate on these properties and the evidence found.
Property 1 - Functional and non-functional (or quality) requirements constrain architectural design decisions. Requirements have prominent influences on architectural design. Functional as well as quality requirements delimit the choices in the solution space, singly and in combination. A proposed design option must fulfill all the constraints imposed upon it simultaneously in order to become a feasible solution.

As the basis of the constraint-based model depicted in Figure 1, this property is also self-evident. There are many examples of requirements exerting constraints on architectural design and they are illustrated in the examples below.

Property 2 - Important constraints on design decisions can come from non-requirements. Policies, business strategy, infrastructure and other factors that could be considered contextual to the system can impose constraints on an architectural design. A contextual constraint may be difficult to satisfy and difficult to negotiate when it conflicts with other constraints.

Evidence from case study EF: A security policy is a contextual constraint, it states that external users such as partners should not have access to the Intranet. However, one of the system requirements is to grant access to external users. This security policy implies that the existing authentication mechanism, i.e. Integrated Windows Authentication, cannot be used for authenticating external users, and new design must be created to cater for external user access without accessing the Intranet. (see Figure 2 – round-edge boxes are ARs, squared boxes are AEs, arrows are causal relations)

Evidence from case study SSA: SSA’s new database system requires regular data updates for which a data file is received from a separate government organization. This file contains a new, cleaned-up data set, but it cannot directly replace the old data set (i.e. a design constraint) because removed data records would cause a critical SSA system to crash. While this was understood by the architect, it was not documented as such.

Evidence from case study LAC: LAC wishes to modernize a large, central legacy information system by extracting non-core functionality and moving it into separate specialized systems. Although many such non-core functions were embedded in the system, only four major elements (including workflow management) were selected for relocation, to accommodate for limited user adaptability and to keep the project manageable in terms of technical changes.

Property 3 - Constraints are sometimes implicit and undocumented. The restrictive influences on decisions are sometimes not spelled out explicitly. Architects may have a vague notion of the existence of some design constraints, but fail to specify such constraints and their impact. Constraints can be omitted because they are undiscovered, forgotten, not recognized as important, or considered obvious.

Evidence from case study EF: In the EF case study, a key constraint is the geographical distribution – locally, national and international – of users who would need to access the system. The access locations of the users place a key constraint on the means of communications, and hence the architectural capabilities of the system. This has not been made explicit in the context of designing for distributed users (see Figure 3).

Evidence from case study SSA: A security policy implies that the existing authentication mechanism, i.e. Integrated Windows Authentication, cannot be used for authenticating external users, and new design must be created to cater for external user access without accessing the Intranet.

Figure 2: Contextual Constraints Influencing Design in Case Study EF

Figure 3: Ambiguous Constraints Leading to Unjustifiable Design in Case Study EF
Property 4 - Constraints that are not well-defined lead to uncertainties of whether an architectural design is appropriate. A lack of a clear definition of design constraints would create difficulties in considering design alternatives and selecting a design solution.

Evidence from case study EF: In the EF case study, the performance constraint is imprecise and general. For instance, the non-functional requirement requires timely access to information content. There is no quantifiable constraint on what performance target the system has to meet. “Timely access” is subject to interpretation. On this basis, the design became an ad hoc judgment by the architect and the design decision cannot be fully justified (see Figure 3).

Property 5 - Constraints can conflict with each other. Architects often have to consider several simultaneous constraints when making design decisions. Some of these constraints may be conflicting, in that their combined effects leave no feasible design options.

Evidence from case study AMC: AMC wants to improve customer service by offering client reporting facilities. They could decide to develop this capability themselves, but they do not have all necessary reporting data. Alternatively, it could be delegated to OTC, an outsourcing partner that produces the bulk of the required data. However, OTC is not able to handle ad hoc report requests, a feature that is desired by AMC. Both options are thus infeasible because these constraints are in conflict.

Evidence from case study CM: A user interface design in a CM system needs to display data from a list that consists of thousands of items. This information cannot be displayed in the limited screen space where users would also need to perform item selection. The two constraints therefore conflict with each other and cannot co-exist on the same web page.

Property 6 – Relaxing constraints is a design strategy to find a viable solution. When a design decision is over-constrained or multiple design constraints are in conflict, and no possible solutions can be found, at least one of the constraints must be relaxed to enable a design outcome.

Evidence from case study AMC: AMC, in its decision on client reporting, was faced with an over-constrained problem. The constraint they choose to take on was its lack of necessary report data; frequent data delivery from OTC to AMC would allow AMC to do client reporting themselves.

Evidence from case study EF: The system is to provide access to all knowledge content. A constraint is to support remote accessing tools such as blackberries to obtain the knowledge content through email links. Some of the documents cannot be rendered on blackberries. So there is no viable solution. As such, document viewing on blackberries through email links are not supported.

Property 7 - A design concern may exhibit different constraint behavior. A design concern can have more than one constraints, each constraint can influence a different design decision depending on the issues that a decision is addressing. Therefore, a design concern such as a requirement can have multiple constraints that influence decisions in different ways.

Evidence from case study CM: It has been decided that to preserve the transaction integrity of monitoring schedule update, a “SELECT FOR UPDATE” locking strategy is used to exclude simultaneous update to database records. This design outcome, also in itself a design concern, has created constraints for three separate decisions: (1) a lock must be maintained within a business transaction. This constrains the decision of how to deal with database connection mechanism; (2) the constraints that a lock can block any update access until it is completed. It affects the decision issue of handling deadlock and as a result subsequent decision chooses a NO WAIT approach; (3) the blocking constraint affects a third decision, which is on how to manage timeouts for web sessions that are abnormally terminated. This example demonstrates that a single design concern can exhibit different constraints depending on the decision issue.

Property 8 - Some constraints persistently influence decisions throughout an architectural design. The nature of these constraints does not change and they influence design decisions persistently. Design decisions are bound by these constraints throughout the architecture design. These constraints are early design decisions and are fundamental to the architecture design of a system.

Evidence from case study CM: A number of designs have to observe the same constraint that the system is based on thin-client or web technologies. This is a constraint that influences multiple decisions. For instances, how to secure
communications between web clients and the servers? how to maintain conversational states? how to manage transactions and database connections?

Evidence from case study LAC: Supporting LAC’s primary processes is a large, central information system taking care of nearly all business-critical functions plus workflow management, word processing and more. A broad redesign was needed to improve maintainability and to introduce flexibility. Being an important goal of the system design, the maintainability quality requirement impacted decisions throughout the architectural design.

Property 9 - A single constraint could affect more than one decision indirectly. Even though a single design constraint can technically only influence one decision directly, the effects of a constraint may live on to influence further design. This can happen when the directly resulting design outcome in turn constrains a subsequent design decision.

![Diagram of Constraint Indirectly Affecting a Later Decision in Case Study SSA](image)

Figure 4: Constraint Indirectly Affecting a Later Decision in Case Study SSA

Evidence from case study SSA: The data file SSA regularly receives to update its database system can only be delivered in full because of technical limitations in the sending organization’s information system. As mentioned earlier, if obsolete data records would be removed from the database, a major system of SSA would exhibit erroneous behavior. This means the received data file cannot be copied straight into the database; a script first needs to detect any mutations, which are then entered into the database manually. In this case, technical limitations constraining a decision on data file delivery indirectly impact the data update procedure (see Figure 4).

Property 10 - A constraint can change as architectural design is developed. As architectural decisions are made to create a partial solution, the newly derived solution can add further and new kinds of constraints to a system. In the process of architectural design, many partial solutions are chosen and each of them can introduce new technical constraints to the rest of the system. As such the nature, the impacts and the number of constraints of interrelated decisions can broaden, making the relationships between decisions and design outcome more intricate, especially how an early decision would create a ripple effect of constraints on a design.

Evidence from case study EF: The constraint of the geographic distribution, presented earlier to support Property 1, leads to the decision to use remote cache. The distributed design has led to a new reliability constraint on the system so that continuous access is maintained when web server is unavailable.

4. Implications for the design process

Some properties observed in Section 3 may seem obvious, however it is necessary to thoroughly establish their validity to create a foundation of a constraint-oriented approach to architecture design. In the previous section, we have identified ten properties of software architecture design constraints. These properties have implications for the architecture design process. In this section, we outline these implications and recommend practices that architects could adopt in their approach to design.

Implication 1 - Specify design constraints unambiguously. Constraints which are not fully understood and well-defined leave much room for interpretation (from Property 1, 2, 3 and 4). If the impact of a concern on a design decision is not explicitly known, this may lead to uncertainties and errors in the evaluation and selection of design options. The selection of a design outcome that violates a wrongly interpreted design constraint can result in misinterpreted and possibly inappropriate software architecture.

For practicing architects, it would be useful to outline design constraints explicitly. This would help them to evaluate that the chosen design solution satisfies all the specified design constraints. Specifying
design constraints unambiguously is therefore an important step of architecture design reasoning.

**Implication 2 – Consider the impacts of constraint based on the scope of a design concern.** We have shown that design concerns impose constraints on design decisions. Some concerns influence a multitude of design decisions (from Property 7). Other design concerns may constrain decisions throughout the design (from Property 8) and affect design decisions indirectly at different levels of the design (from Property 9).

We have noticed from our case studies that decisions on these broad scope design concerns are typically made at the early stages of the architectural design, relating to quality requirements or project resource restrictions.

The scope of impact of design constraints influences the choice of a solution. The wider is the constraint scope of impact the more influence it has on the design. Thus it is important for architects to recognize the scope of impacts of design constraints in design reasoning.

In the architectural design process, architects must explicitly estimate the likely impacts of the design constraints when making an early design decision. Architects must look-ahead to estimate if the solution space is overly constrained before deciding. Since these constraints influence a wide range of decisions directly and indirectly, an early decision is tentative until the impacts of such constraints are considered subsequently in all related design. So the early constraint is reassessed repeatedly at different decision points to ensure that they are not violated.

**Implication 3 – Relax design constraints to resolve constraint conflicts.** Architects may encounter situations where none of the proposed design options are feasible, meaning that no design option can satisfy all design constraints (from Property 5).

In these cases, the decision is over-constrained. It implies that two or more constraints are conflicting with each other, and a feasible design outcome is impossible as long as the constraint conflict is unresolved. To do so, at least one constraint has to be relaxed so that one of the design options becomes feasible (from Property 6).

Constraint relaxation often involves negotiation with stakeholders. They can relax their requirements, increase budget, allow certain design element or combinations thereof, et cetera.

In [15], four basic methods of constraint relaxation to solve an over-constrained problem are listed. They can be used to resolve design constraint conflict as well. Consider the example design decision in Section 2.1 on messaging platform modernization. Let us assume that none of the proposed design options is feasible under the current budget. To solve this problem, the following techniques could be applied:

1. **Budget could be increased so that the least expensive option can be accepted.** This corresponds to changing the value of the budget variable to a level where at least one option is able to satisfy the monetary constraint. If one applies this relaxation technique to a concern that constrains multiple decisions, all of the constraints it imposes are equally relaxed.

2. **Although the budget cannot be increased, we find a design option that is within the constraint of the budget but it compromises a related constraint such as a requirement, i.e. reduce the scope of work.** The objective is to find a design option that minimizes the compromise to the overall related design concerns.

3. **Another relaxation technique is to declare that the constraining impact of available budget on the design outcome is to be ignored, resulting in the complete removal of the constraint from the problem.** This technique is not always possible because many constraints cannot be totally relaxed.

4. **The architect could even choose to forsake the entire decision.** While this will certainly avoid the issue of having no feasible design options, in many cases it is unlikely such a solution is a plausible step in architecture design.

For binary constraints, like the one in our example, removing the input variable from the problem is equivalent to removing the constraint it imposes. And because removing the output variable, the design outcome, implies abandoning the entire design decision, we feel we can put aside the fourth constraint relaxation technique. The remaining three techniques allow us to identify design constraint properties that indicate to what degree a constraint can be relaxed if needed:

1. **The flexibility of a design constraint reflects the architect’s freedom to positively change the ‘value’ of the underlying design concern.** A constraint is less flexible if the architect sees less opportunity to moderate the concern in such a way that a design option becomes feasible, e.g., a lower system availability requirement, a less demanding response time requirement, a higher budget, more time for system development, etc.

2. **Constraint violation in a particular design decision may be justified if the architect can compensate for
that violation by making up for it elsewhere. For instance, selecting a design option that goes over the budget may be warranted if it provides the opportunities to reduce the costs in the other parts of a system. The higher the opportunity for compensation, the easier it is to relax a constraint by tolerating some constraint violation.

3. A design constraint’s priority reflects the relative importance of satisfying the constraint. High-priority constraints should not be dropped from the design decision point, as their influence on the design outcome is crucial and indisputable.

**Implication 4 – Trace interrelated design decisions to understand constraint propagation.** The impacts of a constraint can cascade down a chain of design decisions (from Property 9 and 10). This happens through a mechanism we call propagation.

Suppose that a certain design concern imposes Constraint A on Decision X, and Outcome Y imposes Constraint B on Decision Y. Constraint A propagates to Decision Y because it influences that decision not directly, but through the outcome of Decision X which it did influence directly. From the sole perspective of Decision Y, there is no Constraint A, merely Constraint B. But since Constraint B is imposed by a design outcome affected by Constraint A, the latter constraint has a propagated effect on a subsequent design decision. Constraints can propagate down a chain of design decisions of indefinite length.

If an architect encounters an over-constrained design problem, exploration of relaxation opportunities is not bound to direct decision inputs only. Suppose that in the context of Decision Y, Constraint B is in conflict with another constraint. It could be that relaxing Constraint A will allow for a different Outcome X that imposes a less restrictive Constraint B (if at all), thereby resolving the constraint conflict in Decision Y.

In order to gauge the overall impact of a series of interrelated design constraints on the architectural design, traceability of the relationships with design decisions must be established. Therefore, the relationships between constraints and decisions must be documented, and tools must be available to support it.

**Implication 5 – Assess metamorphosed constraints on design decisions.** Metamorphosis is an added effect on top of propagation (Property 10). Note that the nature of Constraint B could be quite unlike that of Constraint A. If they are not alike, then the propagating constraint has metamorphosed. This side effect means that new types of constraints have been introduced into the design process, and they may further limit the design choices in the solution space. Additionally, metamorphosed constraints may cross-cut other parts of the architectural design and creates new complexity.

It is therefore important for architects to recognize that a constraint has changed to other kinds of constraints and assess how the new constraints would influence the system. Omissions of such new constraints have the same negative implications as ambiguous constraints and failing to recognize them could create an inferior design solution. Metamorphosed constraints could broaden the scope of impact as in implication 2 or propagate further into other areas of a design as in implication 4. Therefore architects must be able to identify and trace them clearly during the design process in order to deal with them.

5. Conclusions

Software architects and researchers have some notions of design constraints, but there are limited efforts to understand their nature and the role they play in software architecture design. Design constraint has also been a factor in many architectural decision making models and techniques but not any one of them articulate what it is and what it does. We first differentiate design constraint from requirements. We suggest that design constraint can provide a uniform and systematic approach to modeling architectural decision-making.

In this work, we have studied five systems from different companies in the Netherlands and Australia. We have identified the design constraints and analyzed their impacts on architecture design. In doing so, we have found ten design constraint properties. With that we propose a constraint-oriented conceptual framework of design. We have suggested five important implications of design constraints for the architecture design process and we have indicated how they influence the practice of architectural design. This has led us to conclude that specificity, scope, relaxation, propagation and metamorphosis of design constraints can influence software architecture design.

In future study, we plan to extend the modeling technique, based on AREL, to represent design constraint as a first-class entity. This should help to enhance the software architecture design reasoning process. Additionally, we will continue to classify design constraints and how they influence architectural design, as well as investigate constraint relaxation techniques and the factors inhibiting their application. Finally, we plan to further explore key perspectives of design constraints, i.e. scope, impact, propagation and
metamorphism, and how they can be systematically used in design reasoning. We believe that achieving these steps will lead to concrete design guidelines and tool provisions for practicing architects.

Acknowledgements
This work is supported partially by the Swinburne University of Technology Research Development Scheme and has been partially sponsored by the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) program on Software Engineering Research via contract 638.001.406 GRIFFIN: a GRId For inFormatIoN about architectural knowledge. We thank Alice Yip for her work on the case studies and we express our gratitude to our industry partners in Australia and The Netherlands.

6. References