ABSTRACT
Architecture has been identified as a main tool for high quality system development. It encapsulates the earliest design decisions of the system under development. These decisions constrain heavily the overall design space; therefore it is worth emphasis on the reasons behind architectural choices. Methods for architecting process have been suggested by various researches. Unfortunately, they fail to achieve frameworks that can reason on the strong relationship between general system goals and the decisions that are made to fulfill those requirements. In this paper, we have three major contributions: We introduce a general framework for architectural design cycle. This framework describes an iterative process that assists in transforming system-wide goals into effective architectural description. We show, how the prioritization values among goals and existing legacy have major impact on the resulting architecture. In addition, we demonstrate that the ordering of design decisions greatly affects what kind of structure is created. Investigating the interplay between architecture and design we focus on how we can connect refinement of requirements to the iterative creation of the architecture. Finally, we demonstrate our framework with a small design example. Using the example we exhibit how our approach improves traceability, supports trade-off decisions and explicitly records design rationale.

Keywords
Software architecture, requirements engineering.

1 INTRODUCTION
The product line approach benefits from using a common design for all family members. Quality attributes and the major features of a computer intensive system determine if it will be successful on the market place. Software architecture defines the quality attributes that the system or family of systems possesses. The architecture should have properties that satisfy all major requirements of the members of the product line. That is, the architecture should reflect the common characteristics of the product line domain while allowing specialization of products in detailed market segments.

Product architecting process relies on a correct understanding of the desired features and the non-functional requirements of the market. In the case of product lines, architects must understand both the common requirements of the market and the special characteristics of each market segment. A clear picture of commonalities is needed to correctly guide system development effort.

Software cannot be designed to provide flexibility in every respect. Flexibility requires more processing power and therefore reduces the performance of the system. Flexibility may also make the system harder to understand, modify and maintain. However, there are no universal connections between various quality aspects. A good architect is often able to find solutions that improve the properties of the system in respect of several quality attributes. For example, it is common to add flexibility by increasing indirection and thus increasing the complexity of the system, yet there examples of cases where the designer has managed to increase system flexibility by using better abstractions and thus made the system more simple not more complex.

One of the main problems in designing a good reference architecture for a product line is to find out how to handle the numerous requirements and the needs of various stakeholders. Different stakeholders define requirements for different domains. They do not use same language and there are no easy methods to combine requirements or to order them according to their importance. Yet many successful product line development projects have demonstrated that the identification of clearly defined
architectural drivers is one of the key elements of creating correct reference architecture for a set of products. Using a carefully selected group of major goals concentrates the development effort on satisfying these key requirements. Architectural drivers help the developer in avoiding the problem of trying to satisfy too many different requirements, or gold plating the overall architecture.

In this paper we advocate goal driven and iterative architecture design process. In this process architectural drivers are used as a selection criteria for different design decisions. The impact of those decisions for the system properties are evaluated and compared against the architectural drivers and other requirements. Design becomes an interplay between what is desired and what can be achieved. This helps in justifying the need for compromises and makes it easier for stakeholders to accept them. Refinement cycle is repeated until the architects and other stakeholders are confident enough in achieving the top-level system goals. Even though many of the ideas have emerged while creating reference architectures we do not cover product line specific issues because of the limited space available for this paper.

The paper is organized as follows: Chapter two introduces architectural design cycle that forms the basis of our approach. In the chapter three, we describe the essential difference between requirements and design. We also show how the definition hierarchy approach can solve many of the common system development problems. Design framework is described in the chapter four and demonstrated in the chapter five.

2 ARCHITECTURAL DESIGN CYCLE

Traditionally, architectural design has been viewed as an art form without established processes, descriptions or methods. Recently a number of approaches have been proposed to fill this void [3,5,10,12]. However, most of these methods are either domain dependent and cannot therefore be generalized enough as a general guideline to multitude of problems, or too general and as such fail to adequately address the correct refinement of the qualities of the architecture in a clear stepwise fashion. The stepwise refinement of the architectural qualities is critical in order to achieve an industrially proven method with sufficient control during architectural design. We address these problems by providing a general framework with a design cycle that easily adapts into multiple domains and problems. The correct refinement of architectural qualities is managed by our design model.

We suggest an architectural design cycle that can be easily adapted to multiple environments and which, to our experience, provides an abstract framework for a family of processes. This general framework can be easily specialized to the particular problem domain.

The first phase of our design cycle is to determine the main goals of the planned system. In a large software project, a number of different goals exist. A clearly defined prioritization scheme for grouping requirements based on their importance is clearly needed. In the context of architectural design only a subset of requirements are of great concern. Architectural design concentrates on achieving high-level goals that describe the wanted quality characteristics of the system. Those architecturally significant requirements that have a high prioritization value define the main architectural drivers of the system. In practice, many functional requirements have only very limited architectural impact. More information on identifying architecturally significant requirements can be found in [6].
architect. The order in which the design decisions are made greatly affects what kind of decisions can be done later in the design cycle. This creates great emphasis on the first decisions of the architectural design. We suggest, that a careful tradeoff analysis of the cluster of the most important requirements should be performed when the first decisions are chosen.

Almost every decision constrains the system in some way. This restricts the possible design space that is available for the developer. When the overall structure of the system gets more detailed it at the same time consumes the inherent flexibility of the software intensive system. Each architectural entity that is introduced during the design cycle has some constraints and implications associated with it. These may have an impact on the properties of the system also outside the scope of an individual decision.

These constraints play a key role in the architectural design cycle, since they, together with the properties of the system, affect the prioritization of the goals of the system. The properties of the architectural elements have some positive impact on important requirements. Thus, the design cycle in each of its iteration tries to progressively get closer to satisfying the true goals of the system.

The emphasis in the cycle is really on the business goals that we want the new system to satisfy. They drive the structure of the system and the structure drives the way we see requirements. More often than not the goals are based on existing assets and preferred customer segment. This limits the choices for the top level architecture of the system and the architecture in turn determines how we see end-user needs. The whole vocabulary of the requirements specification changes if the change in starting point is big enough.

3 THE DESIGN MODEL FRAMEWORK

This section defines a novel framework, which supports the general architectural design cycle by defining refinement relationships among architectural decisions and requirement specifications. Jan Bosch defines architectural design as the following:

Architectural design is the process of converting a set of requirements into a software architecture that fulfills, or at least facilitates the fulfillment of, the requirements. [3]

In this definition architectural decisions are enablers. They rarely completely fulfill requirements as such; rather they set a context in which the implementation satisfies these requirements. In our framework, we extend this thinking even further by claiming that architectural decisions set the context in which requirements are interpreted. The meta model of our method is shown in the figure 2.

The requirements have two different roles in the system design. Requirements guide the design by expressing what is needed by the system and they constrain the solution space by excluding some design decisions.

Architecture consists of design elements. Typically design elements are discovered while making the key design decisions for the system. Analyzing architectural decisions allow estimating the overall properties of the system and guiding future choices to achieve those requirements that are ignored by the current set of design choices. Our picture of the overall system properties gets more precise as design is refined and its properties are expressed using continuously more detailed constructs.

Figure 2: Conceptual model for design information

The basis for high quality architectural design is to consider multiple ways to achieve desired qualities. Many errors in the final system design are caused by optionless architecting and premature design. Inexperienced designers tend to use same solutions and structures for any problem they face, often making the solution unfeasibly for the current problem. This problem can be tackled using explicit design candidate selection process.

In goal driven process, a system architecture is created by making decisions that address the quality requirements of the system. These decisions can be made incrementally in order that is guided in a large degree by previous choices and they success in satisfying architecturally significant requirements. These design increments can be connected to corresponding increments of requirements; making the refinement in design to correspond to the refinement in the requirements.

Only in rare cases a single design choice completely satisfies a high-level design goal. More often design
decision contributes positively into multiple relatively detailed requirements. In addition to the benefits, decisions may have implications and constraints that hinder our ability to achieve other qualities. Managing these different influences on the overall properties is one of the main activities of software architects.

Design element
A design element can be any documented piece of information that describes a solution to an architectural problem. It shows a mechanism how the wanted properties of the architecture can be achieved, acting within the current scope. Depending on the current abstraction level, a design element can be an architectural style or a single component of the overall architecture.

Patterns are particularly suitable for the design documentation purpose, since they specify a solution in the context of the problem at hand. Additionally they provide information on the benefits and the consequences of the solution, which assist in determining their impact on the overall properties of the architecture. Unfortunately, the current pattern languages do not well assist designing large-scale systems. For example, there is no guidance for estimating properties of the combinations of patterns.

Design alternatives are different means to solve the same problem. Design can be presented in any level of the abstraction from the high-level architectural patterns to the very detailed choices of algorithms. The design alternatives must be presented in a way that allows performing trade-off analysis based on the properties of the suggested design choices. Additionally, the design rationale must provide information on the implications and consequences of the use of the design element in the current context.

Different design candidates are presented in the same level of abstraction and a selection criterion is used to select one design candidate as our architectural solution. Both the alternative design candidates and the chosen architectural solution are design elements with its own set of design properties. The architectural solution must also be connected to other design decisions in a meaningful way.

Selection criterion
The selection among various design alternatives is based on the selection criteria, which describes the reason why a certain decision is made. A different choice among design alternatives thus often has different selection criteria. The criterion seems to often follow a general trend that the abstraction level of the corresponding requirements reduces when design is refined. The complete satisfaction level of a high-level goal can be defined only when the whole system has been implemented. Therefore, the actual selection of the criterion is based on the current estimates on the state of the design and its properties.

The selection process is guided by the major goals or other requirements of the system. That is, the appropriate selection criteria are chosen to guide the system design towards the wanted overall properties. In many cases, this requires multiple steps during the design process. Especially, these steps may happen in very different times and abstraction levels for each concern.

Design property
Each design element has some properties. During our architecting process, the role of the properties of the architectural design elements is essential. Architectural decisions are used to satisfy the quality goals of the system. Moreover, the each main architectural decision should be defended on the basis of which kind of implications and consequences this decision has on the quality attributes of the overall system. Unfortunately, complete specification of the overall impact of one decision on the overall system is often very difficult. The decisions are made in the evolving context and their properties are dependent on the properties of the other decisions. Thus, the combination of all decisions that are made during the development of the architecture creates the overall properties of the system. Despite this fact, we can predict the qualities of the system in the necessary level of detail so that it is possible to effectively support choosing the correct alternative that leads towards the wanted properties of the resulting system.

4 AN EXAMPLE
We have constructed this example to demonstrate how the architecture design cycle and design model framework can be used in practice. The design outlined in this example should not be taken as a proven solution for this problem. Architecture design can only be demonstrated by system level examples. In order to make this description short enough we have skipped over many important details. We hope however that based on this example reader is able to understand how this approach could be used in designing a system she is more familiar with.

As an example we use ambulance dispatch system. An ambulance dispatch system is responsible for receiving the emergency calls; understanding the nature of the calls; dispatching ambulances based on the calls and available resources; and monitoring the progress. Our example is inspired by the London Ambulance Service case study [2].

In goal driven design the definition of the top-level goal is of outmost importance. We have defined ours as: “Fast and reliable allocation of resources that best match the need.” Note that any realistic ambulance dispatch service has other goals ignored in our example.

Refinement is the next step in our design cycle. Our top-level goal has three components: time, reliability and best match.

Time can be seen as consisting of four parts:

1. Time to get through to the system
2. Time spent in understanding the nature of the call.
3. Time spent in allocating the resources.
4. Time taken by the resources to reach the emergency site.

We should optimize the sum of these components.

Reliability is best understood through failures. Our incomplete refinement says that there are several ways to fail:

1. Emergency call does not get through.
2. Request is taken but lost.
3. Description of the emergency (position, type) is incorrect.
4. Allocation does not take place.
5. Ambulance crashes and does not reach the destination.

Best match is a compromise of (at least) location, availability, capacity, equipment and personnel.

System design takes always place in an environment that out of social, economical and technological reasons places several constraints. In our case for example we do not have a reliable communication method with the ambulances, we have no reliable way of knowing where they are at any given moment, and we cannot depend on the crew of each ambulance being exactly as we have planned. As our design proceeds, it will create its internal constraints. However, system design cannot remove or change the constraints placed by the environment. At best it can give them another interpretation.

Our first design decision is the overall architecture. We have too primary candidates. Either we allocate the resources centrally or we let the resources bid and pick the best match. In the case of LAS [2] the designers selected the central resource allocation model. It leads into a structure where knowledge of resource status (location, crew, availability) is kept up to date in a central database and allocation is based on that knowledge. In the presence of the environmental constraints listed above this leads into a situation where the database content does not reflect reality. Allocations based on incorrect information are often incorrect and system fails to achieve its top-level goal. Environmental constraints become our selection criteria.

Second time around in the design cycle we are wiser and select the distributed model. Straightforward implementation of this is a one where we just broadcast in the whole area of operation and let every available ambulance reply with their current location, equipment and crew. Now we can allocate enough appropriate resources for each emergency. Unfortunately when we look at the overall properties of this structure we will notice that each emergency call will create an avalanche of bids. Processing them will place a lot more load to our system that the actual emergency calls did. We will not be able to achieve our top-level goal. Time spent in allocating the resources will be too high and soon emergency calls will no longer get through. We need to take economical use of communication resources as selection criterion.

Third time around we tackle the avalanche of bids. There are too ways we can diminish the flow. First we will in the request specify what kind of resources are required. Secondly we will broadcast only in the area close to the emergency. When we look at the overall properties of this design we can immediately see that it matches the refinement of our top-level goal much better than the previous ones. We get fairly fast allocation of fairly well matching resources. Our biggest problem is the possibility that we do not get enough bids.

Back to the design cycle. We need to modify our design so that it guarantees that we will allocate all available resources if needed. First modification is to select the size of the broadcast area based on the size of the emergency. This gives us a better first guess. Second modification is iteration. If we do not get enough resources we allocate what we got and broadcast the request in the neighboring areas. This guarantees that every single available ambulance will eventually get allocated.

We will stop our example here. If you go back to the refinement of the top-level goal you can see that we have tackled only a fraction of the issues. We have also left out important social and economical constraints.

5 RELATED WORK
Numerous methods for requirements and architectural design exist. In this section we describe the methods that are most closely related to our approach.

Architectural design process proposed by Jan Bosch creates the initial architecture based on the functionality of the system [3]. After this initial design step the architectural candidate is analyzed against the desired properties of the system. The analysis results are used as a basis for progressive architectural transformation, which try to change the properties of the initial design to match the specifications. This process provides a practical solution, which tries to guarantee that the architecture possesses the main quality requirements of the main stakeholders. However, this method does not provide tools for recording exact details of the progress of the design process through various iterations. Additionally, the transformation start from the initial architecture, which is based on functional design, always exhibits many implicit design decisions that are not recorded in any way. Overriding existing implicit decisions is often very difficult because many of these decisions can be undetected by developer performing transformations, this may easily result in less than optimal design. It is questionable whether it is at all possible to arrive at a good architecture if the initial guess based on the
functionality does not happen to support the wanted quality characteristics.

Chung et al. [7,8] have proposed a method to relate non-functional requirements to architectural design alternatives. This method provides guidance and documentation of rationales behind each trade-off during architectural design. Their method is similar to ours as they also refine the non-functional requirements. Attempt to model the relation among all different non-functional requirements seems to make this method hard to use in practice. The modeling of dependencies of requirements without the context of the design decisions causes, even for a simple case, the goal-graph to expand into a web of connections that is difficult to understand or manage.

Alonso et al. [9] have also proposed using DDTs for documenting design decision during product family development. Their method allows specifying different variants of software product line using variant of design decision trees. However, they do not directly address any issues of the iterative design process or the details of the requirement – design relation.

Architecture trade-off analysis method (ATAM) is designed to provide a framework analyzing different architectural candidates and thus helping a developer to make well-justified decision on different architectural elements on the wanted quality attributes [11]. This approach is well suited with our design cycle. The ATAM method concentrates on one step in our design cycle – the architectural candidate selection. A developer can easily use similar methods while in our architectural design cycle. Naturally, caution should be used when considering to what degree trade-off analysis is used in each design element selection.

Architecture based design method (ABD) imply similar way of designing architecture [12]. They also have refinements in requirements as well as in design. However, they do not provide any methods or tools to document this information in any concrete way. In addition, they fail to identify how iterations in design can be mapped to the overall properties of the system. Their approach seems to be best suited in those cases where the requirements are sufficiently independent from each other.

6 CONCLUSIONS
We described a framework for architectural design. We proposed using definition hierarchy approach for goal structuring and as a starting point for the design process. This allows us to tie the main goals of the system together with the main decision made on the structure of the system architecture, which will assist in creating an architecture that can fulfill the main requirements of the different stakeholders.

A developer armed with a clear definition of the quality requirements, specified in the definition hierarchy, drastically helps creating a high-quality architecture that possess the main concerns of the key stakeholders. Definition hierarchy specifies quality attributes in clear and unambiguous fashion. Our process, focuses on the matching the quality requirements in the definition hierarchy to the architectural decision made during the development.

We also demonstrated, how this approach could be used with a small example. The example described how goals and decisions on the system structure interact while the design progresses from the abstract goals into more detailed design descriptions. The arrival of certain architectural decisions changes the importance of goals of the system because of partial satisfaction of some requirements.

7 FUTURE WORK
Our design framework has been demonstrated being suitable for documenting the design rational trough iterations. Describing the actual steps and the elements of the architectural design promises great improvements in the traceability, impact analysis and change management activities. The traceability of individual elements and tracking the context of the design seem to significantly improve the maintainability of complex real-life systems.

We have an existing tool to support requirements engineering as well as transition to the design process. We are developing improved support for change management and impact analysis. The ability to create all-inclusive traces and using those traces for change impact analysis requires still more research to be validated further.

REFERENCES


