POMSA: Process-Oriented Metrics for Software Architecture

Lawrence Chung
Dept. of Computer Science
University of Texas at Dallas
chung@utdallas.edu

Nary Subramanian
Dept. of Computer Engineering
Hofstra University
eggnzs@mail1.hofstra.edu

Abstract
One of the important measurements in software is the extent to which the software fulfills non-functional requirements (or NFRs) such as adaptability, evolvability, reliability, testability, and the like. Inevitably these NFRs should be addressed at the time the software architecture is developed, and the measurement at the architectural level is usually a good indicator of the degree to which NFRs will be met by the final software system. The knowledge of the architectural NFR properties will enable the software organization to either proceed further with the development or to change the architecture to improve its characteristics or even the NFRs themselves. This paper proposes a framework, POMSA (Process-Oriented Metrics for Software Architecture), which aims to provide numeric scores representing the degree to which the “ilities” of a software architecture are met, as well as the intuitions behind these scores. In this framework, the intuitions behind the architectural NFR scores are traced back to the "whys" of the architecture, namely, the requirements for which the architecture exists in the first place. POMSA achieves the needed tracing by representing and reasoning about NFRs during the process of architecture development. Ultimately POMSA is intended to help detect weaknesses in the architecture, understand the reasons for the weaknesses and make improvements (that will help improve the metrics), and recalculate the metrics for the new architecture fast and intuitively. We illustrate the use of POMSA in calculating adaptability metrics for software architectures of two commercial telecom systems, each with more than 50000 lines of code.

Keywords
Non-functional requirements, metrics, software architecture

1. Introduction
"To measure is to know" - Lord Kelvin.

The knowledge of the extent to which a software system satisfies its various non-functional requirements (NFRs) such as adaptability, evolvability, reliability, testability, and the like will be useful to the software engineering community. Software architecture is the first step in the development of a software solution. Inevitably the NFRs should be addressed at the architectural level itself. Thus the measurement of the NFRs at the architectural level is usually a good indicator of the degree to which the final software system will meet the NFRs. This knowledge will help the software organization to either proceed with further design or to change the architecture to improve its characteristics or to even change the NFRs themselves. This paper proposes a framework – POMSA (Process-Oriented Metrics for Software Architecture) - which aims to provide numeric scores representing the degree to which the NFRs of a software architecture are met as well as the intuitions behind the scores. In this framework, the justifications for the architectural NFR scores are traced back to the “whys” of the architecture, namely, the requirements for which the architecture exists in the first place. Ultimately POMSA is intended to help detect weaknesses in the architecture, understand the reasons for the weaknesses and make improvements (that will help improve the metrics), and recalculate the metrics for the new architecture fast and intuitively.

One of the general problems with NFRs is a lack of their precise definition. In order to make POMSA widely usable we have adopted the NFR Framework that provides a qualitative framework to compare architectures and choose the better one, in this paper we will show the use of this qualitative framework to develop quantitative numbers for the NFRs. The NFR Framework [1, 2] requires the following interleaving tasks, which are iterative:

- Develop the NFR goals and their refinements
- Develop architectural alternatives
- Develop goal criticalities
- Develop design tradeoffs and rationale
- Evaluation and selection.

The graph that results from the application of above steps is called the Softgoal Interdependency Graph (SIG). The SIG will be used to develop the metrics for software architectures developed at step 2 above. The NFR Framework uses the concept of satisficing. Satisficing means “good enough”, that is satisfaction within limits and not absolutely. There are different degrees of satisficing and the degrees are indicated by color codes (given in Figure 1) in this paper. While the NFR Framework gives a qualitative framework to compare architectures and choose the better one, in this paper we will show the use of this qualitative framework to develop quantitative numbers for the NFRs.

Software architecture often times involves six constituents: components, connections, patterns,
constraints, styles and rationales [3, 4]. Components are the elements from which systems are built; connections are the interactions between the elements; patterns describe the layout of the components and connections; constraints are on the components, connections and patterns; styles are an abstraction of architectural components from various architectures; and rationales are the reasons for selecting architectures with NFRs as the criteria. A main goal of this paper is to provide a comprehensive framework for dealing with the rationales. In this paper we show how POMSA can be used to calculate NFR metrics for an architecture, detect weaknesses in the architecture and its reasons, compare architectures, and how to improve NFR metrics for the architecture by making modifications. A preliminary version of this paper appeared in [6].

In order to validate POMSA we used architectures for two commercial telecom systems, each of which has more than 50000 lines of code in their implementations. These systems will be called S1 and S2: S1 is newer and is developed using a more versatile architecture as compared to S2. In order to apply POMSA, feedback from the engineers who developed the systems was used. Using POMSA we calculated the metrics for the architectures of the two systems, S1 and S2, detected weaknesses in the architectures and their reasons, and suggested measures to improve the architectures.

Section 2 introduces the POMSA framework, Section 3 applies the framework to two telecommunication systems, Section 4 briefly discusses the feedback from developers and Section 5 summarizes the papers and discusses future work.

2. The POMSA Framework

The POMSA framework consists of six major components: a set of softgoals for representing NFRs, design constituents and claims, a set of contribution types for relating softgoals to other softgoals, a set of methods for refining softgoals into other softgoals, a set of correlation rules for inferring potential interactions among softgoals, a labeling procedure which determines the degree to which a design component satisfices a softgoal, and a set of metrification schemes to map labels to numbers.

1. Softgoals can be of several types – the NFR softgoals (depicted by a cloud), the operationalizing softgoal (depicted by a dark cloud), and the claim softgoal (depicted by a dotted cloud). The operationalizing softgoal represents a design/architectural constituent, while a claim softgoal represents a supporting or denying claim (for any item of the framework).

2. Contribution types connect various softgoals – the links may connect several softgoals to one softgoal in an AND-decomposition (depicted by single arc) or in an OR-decomposition (depicted by double arc).

3. Methods are ways to refine or decompose one softgoal into offspring softgoals for purposes of clarity and achievement of better designs.

4. Correlation rules help determine the interactions between different NFRs for an architectural constituent.

5. Labels indicate the degree to which their associated softgoal (or links) are satisficed – the various satisficing degrees are mentioned in Figure 1.

6. Metrification schemes map qualitative labels into quantitative scores for a given architectural design. Labels of NFR softgoals, design softgoals, claim softgoals and links, in some combination (either only one of these, any two of these, any three of these or all of these), may be converted to numbers. There are several different metrification schemes, including: 6a) Max and Min Values: In this scheme the max and min values are computed for the labels; 6b) Single Values: Here one value is computed for the labels; 6c) Probabilistic: Here probabilities are computed for the labels. The metrification scheme guidelines are given in the next section.

The ontology used by the POMSA framework is given in Figure 1.

![Figure 1. The Ontology (partial) of the POMSA Framework](image-url)
2.1 The Softgoal Interdependency Graph

The POMSA framework requires decompositions for the NFRs of interest. Figure 2 shows the decomposition for the NFR adaptability for the telecom systems of interest. Each cloud in Figure 2 is a softgoal in the NFR Framework. Each softgoal has a name following the convention

\[ \text{Type}[\text{Topic1}, \text{Topic2}, \ldots] \]

where Type is a non-functional aspect (e.g., adaptability) and Topic is a system to which the Type applies (e.g., telecom system), and the refinement can take place along the Type or the Topic. The NFR softgoal of interest, viz.,

Adaptability[Architecture, System] is AND-decomposed into the five architecture constituents (the sixth constituent is the rationale which is justified by the SIG itself): Adaptability[Components, System], Adaptability[Connections, System], Adaptability[Patterns, System], Adaptability[Constraints, System] and Adaptability[Styles, System]. Adaptability[Components, System] is further decomposed as shown in the figure: AND-decomposed (shown by the single arc) into two softgoals – Changeability[Features] and Changeability[Design, Component]. The reason this decomposition was made is to underline the fact that changeability of design of components and changeability of features is important from the point of component adaptability. Changeability[Features] is further AND-decomposed into Decomposability[Component, SubClasses] and Cohesiveness[Component, SubClasses]. This decomposition follows from the fact that object-orientation was adopted for the two systems studied and hence changeability of features is impacted upon by the decomposability of component into subclasses and the cohesiveness of the subclasses (if any) themselves. The NFR softgoal changeability[Design, Component] is further AND-decomposed into Understandability[Design, Component] and Simplicity[Design, Component] – this decomposition follows from the fact that design for a component is changeable only when it is understandable and simple. The NFR softgoal Adaptability[Connections, System] is AND-decomposed into EasyChangeability[Processor, System] and EasyReplaceability[Components, System]. This decomposition arose from the fact one of the major concerns in system S1 was that the components should be easily changeable from one processor to another to balance the processor load; easy replaceability of components was another concern so that different components could be connected to each other easily (and in the future dynamically). The NFR softgoal Adaptability[Patterns, System] is AND-decomposed into EasyChangeability[Connections Between Components] and Isolatability[Components from Environment]. The pattern of architecture chosen should support easy changeability of connections between components and tied to this is the NFR isolatability of components from environment: the more isolated the components are from the environment, the easier it becomes to change connections between components. The NFR softgoal Adaptability[Constraints, System] is OR-decomposed (as...
shown by the double arc) into Performance[Time for Communication, Components, System] and Performance[Memory Size, System]. This OR-decomposition is due to the fact that both time and space constraint cannot usually be achieved simultaneously. Finally, Adaptability[Styles, System] is AND-decomposed into Reusability[Style] and Replaceability[Components, Connections]; reusability is an important NFR since it helps the organization in the long run and replaceability of components and connections helps change the architectural configuration easily, both in terms of software as well as the hardware on which the components execute. In Figure 2, the NFR softgoal Decomposability[Component, SubClasses] is considered a priority item (indicated by !) to avoid monolithic classes that were considered undesirable.

The dark-bordered clouds in Figure 2 are the operationalizing or design softgoals. These softgoals represent design elements – in this case the architectural constituents. The descriptions of these constituents are given later. The colored lines connecting the design softgoals to the NFR softgoals are the contributions that the design softgoals make toward the NFR softgoals and follow the legend of Figure 1. In Figure 2 there is a dotted cloud – these clouds are the claim softgoals and they provide justifications for contributions.

The SIG helps to reason about the propagation of labels. Each operationalizing softgoal can be assigned a label – S for satisfied, D for denied or U for unknown. Then by using the label propagation algorithms of the NFR Framework these labels can be propagated up the SIG to determine the satisficeability or otherwise of any of the softgoals in the SIG. POMSA provides a systematic method to convert these labels into numbers and to propagate these numbers up the SIG to obtain metrics for relevant softgoals.

2.2 Guidelines for Generic Metrification Schemes

Any metrification scheme M converts labels of the SIG into metrics. This conversion is accomplished using the generic guidelines of Figure 3. In Figure 3, guidelines M1, M2, M3 state the rules for metrification of an element of the framework: thus M1 says that the label of leaf softgoals gets converted into a metric, M2 says that the label of a contribution converts into the metric for the contribution; and M3 says that the criticality gets converted into the metric for criticality – however, since criticalities can be assigned to two elements of the framework, the softgoal and the contribution, M3 is broken into two parts: M3A applies to the criticality of the softgoal while M3B applies to the criticality of the contribution. M4 states that for any leaf softgoal, the metric of its label, the metric of its criticality, the metric of its contribution to its parent, and the metric of its contribution’s criticality together form the metric for the individual contribution of that leaf softgoal. M5 says that the metric of all individual child softgoal contributions result in the metric for the parent softgoal. M6 applies to contributions that have children (for example, in the form of claim softgoals) and states that the metric of the parent contribution is computed from the metric of the contributions of all child softgoals of that contribution. The different guidelines are explained pictorially in Figure 4.

3. Application of POMSA Framework

The metrification schemes of the POMSA framework will be used iteratively and incrementally throughout the process of software architecture development. The architectures given in Figure 5 and Figure 6 have been incrementally and iteratively developed with metrics being computed at the end of each iteration. We have assumed that the architect wants to determine metric scores incrementally. However, due to space limitation we show the metric computation only for one component. We first describe the architectures considered and then we show the metrification.

3.1 Architecture for Two Telecom Systems

System S1 consists of modules and access points (AP) as shown in Figure 5 which functions somewhat similarly to the processes and ports in [5]. Modules are the highest level of components in the system – they are classes which can in turn be composed of other classes; modules execute as a separate thread. Each module represents a protocol and there are four protocols: physical layer protocol (PLP), MACP (Medium Access Control Protocol), and two application protocols, App1 and App2. There are two application protocols since one handles data and the other handles voice. The access points are the connections provided by the services layer. The services layer provides functions for creating modules and their access points as well as connecting the access points of different modules. The data passed between access points are referred to as primitives: the sender does not need to know the identity of the receiver and neither does the receiver need to know the identity of the sender; the data is of utmost importance. The App1 and App2 applications are connected to MACP and the latter is connected to PLP. The pattern of the architecture is module-access point-module pattern, i.e., the common paradigm in the so-called MBA architecture (where MBA stands for module-based architecture) of Figure 5 is that of modules talking to each other through access points, and this pattern will be referred to as the MBA pattern. The
Figure 3. Guidelines for Generic Metrification Schemes

M1: label(leaf softgoal) → metric(leaf softgoal)
M2: label(contribution) → metric(contribution)
M3A: criticality(softgoal) → metric(criticality(softgoal))
M3B: criticality(contribution) → metric(criticality(contribution))
M4: {metric(leaf softgoal),
    metric(criticality(leaf softgoal)),
    metric(contribution),
    metric(criticality(contribution))} → metric(individual contribution of leaf softgoal)
M5: {metric_i(individual contribution of child softgoal_i)} → metric(parent softgoal)
M6: {metric_i(individual contribution of child softgoal_i)} → metric(parent contribution)

Note 1: ↑ represents the generic contribution - it could be MAKE, HELP, HURT or BREAK
Note 2: ï represents generic criticality - it could be !, !!, !!!, ...

Figure 4. Pictorial Explanation of the Guidelines for Generic Metrification Schemes
constraint is the services layer itself since the architecture uses the facilities provided exclusively by the services layer. And finally the style is object oriented, since the whole system is based on classes and their objects. System S2’s architecture is given in Figure 6 and is more traditional. At the highest level, S2 consists of three layers (physical layer, MAC layer and application layer – only voice application is there in this system), each composed of several classes, and the layers talk with each other using message passing (MP) mechanisms provided by the RTOS (real-time operating system). The pattern is one of layer-message passing-layer. The constraint is the incremental nature of processing done in the system, while the style is object-oriented.

3.2 Qualitative Development

Based on feedback from engineers, the SIGs for the architectures of the two systems S1 and S2 are shown in Figure 2 and Figure 7, respectively. In Figure 2, Claim1 stands for “AP supports easy replaceability of components; simply connecting AP’s to the new component is sufficient”, and in Figure 7, Claim1 stands for “MP does not support easy replaceability of components; the new component needs to know the identity of all other components it sends messages to and all other components need to know the identity of the new component to send messages to the latter”.

In qualitative reasoning, the labels (as described in Section 2.1) are assigned to operationalizing softgoals and the labels are propagated upward using the algorithms of the NFR Framework. For example, in Figure 2 (although omitted from the figure), if the design softgoal MACP is satisficed, it is given the label ‘S’. Since it makes a HELP contribution with the NFR softgoal Decomposability, the latter can also be labeled ‘S’. Since MACP also makes a HELP contribution with the NFR softgoal Cohesiveness the latter can also be labeled ‘S’. Since the two children of the NFR softgoal Changeability[Features] are satisficed (considering the contributions only due to MACP), the latter is labeled as satisficed. Likewise the NFR softgoal Changeability[Design, Component] is also found to be satisficed, because of which the NFR softgoal Adaptability[Components, System] is also satisficed. In quantitative reasoning these labels are converted into numbers.

3.3 Example Metrification Schemes

POMSA uses a set of metrification schemes to convert a qualitative SIG into a quantitative one. Any metrification scheme satisfying the guidelines of Figure 3 can be used. We will be illustrating POMSA using two different metrification schemes: a sample single-value scheme and a probabilistic scheme. Each of these is but one possible scheme of that type. The schemes are shown in Figure 8, and Figure 9. Due to space limitations, both these schemes do not deal with unclear cases, contribution satisficing, and multiple criticality.

In Figure 8 a sample single-value metrification scheme is given. This scheme assumes that the metrics are between +1 and -1. Thus the softgoal metrics rules M1.1 and M1.2 allocate a metric of +1 for satisficed softgoals and a metric of -1 for denied softgoals. The rule M2
allocate metrics for contribution between +1 and -1 depending on the contribution type (here it is assumed that the contributions are themselves satisfied – if not the contributions are given a metric of 0). The M3 rules give the metrics for the criticalities of a softgoal and a contribution. M4 gives the formula for computing the metric propagated by a child softgoal to its parent. M5 gives the rules for combining propagated values from multiple children for AND, OR, and for leaf NFR softgoals. M6 gives the method to compute metrics for a contribution that has children.

The sample probabilistic scheme given in Figure 9 is similar to Figure 8 except that probabilities assigned have values between 0 and 1, contributions are assigned probabilities in ranges depending on their type (M2), the criticalities are assigned values differently (M3), and the propagated metrics are used differently in M5 to compute the values for AND, OR and leaf NFR softgoals.

### 3.4 Application-specific Metrification Schemes

The step-by-step application of the sample single-value metrification scheme is shown in Figure 10 for the design softgoal MACP. In Figure 10a, the design softgoal MACP is indicated as satisfied (shown by S inside the softgoal) – this is decided based on the feedback from the developers. By rule M1.1 of Figure 8, the design softgoal MACP is assigned a metric of 1; since the softgoal is critical (indicated by !), by rule M3A of Figure 8 and based on developer feedback, the criticality is assigned a value of 1. The contributions that MACP makes to the NFR softgoals are assigned appropriate metrics by rule M2 of Figure 8. Then by rule M4 the value propagated to the leaf NFR softgoals are calculated – the values propagated are indicated inside of the NFR softgoals. The values of the leaf NFR softgoals propagate to their parents by rule M5.1 for AND contributions and the metrics for the parents are computed. At the end of the metric propagation we have a metric for the adaptability of the design softgoal MACP. The last figure in Figure 10 (10f) shows the application of the sample single-value metrification scheme when two design softgoals MACP and PLP are considered together – here at the leaf NFR softgoals the rule M5.3 is used to determine the metrics.

Figure 11 shows the application of the sample probabilistic metrification scheme (in Figure 11 only the relevant parts of the SIGs have been shown).

### 4. Feedback from Developers

We were able to obtain feedback from four different developers involved with the systems S1 and S2 – both of which are real commercial telecom systems. The final systems had more than 50000 lines of code each. The developers were given an overview of POMSA and the SIGs/metrification schemes were presented. The following feedback was obtained:

1. Engineers prefer grading schemes to absolute scores: thus they prefer to answer as very good, good, bad and very bad, rather than providing an absolute number, such as a number between 1 and 10; the former they feel is more consistent.
2. Not all metrification guidelines of Figure 3 need be satisfied by a single metrification scheme. Thus simpler forms of a metrification scheme may be used
that helps in faster computation of the metrics starting at the leaf NFR softgoal level, for example.

3. The initial NFR decomposition should be done by a group of system engineers at the beginning of the project and should be changed as little as possible. All development should be metrified using this decomposition. This decomposition should be traceable to the system requirements. This will also allow the automation of POMSA for the project.

4. From the feedback that was received, POMSA promises to be an effective framework to determine systems’ NFRs. One of the observations that were made was the usage of Gaussian distribution as a metrification scheme – POMSA permits use of any metrification scheme including statistical ones.

5. Scalability of POMSA is another issue that needs to be further researched. As the levels of decomposition of the NFR softgoals, design softgoals and claim softgoals increase, it may not be possible to keep track manually of all the interactions, and to (re-)calculate metrics manually. We believe automation will help address this scalability issue. There are tools available [7] that can store the various elements of a

Figure 8. Sample Single-Value Metrification Scheme

Figure 9. Sample Probabilistic Metrification Scheme

that helps in faster computation of the metrics starting at the leaf NFR softgoal level, for example.

3. The initial NFR decomposition should be done by a group of system engineers at the beginning of the project and should be changed as little as possible. All development should be metrified using this decomposition. This decomposition should be traceable to the system requirements. This will also allow the automation of POMSA for the project.

4. From the feedback that was received, POMSA promises to be an effective framework to determine systems’ NFRs. One of the observations that were made was the usage of Gaussian distribution as a metrification scheme – POMSA permits use of any metrification scheme including statistical ones.

5. Scalability of POMSA is another issue that needs to be further researched. As the levels of decomposition of the NFR softgoals, design softgoals and claim softgoals increase, it may not be possible to keep track manually of all the interactions, and to (re-)calculate metrics manually. We believe automation will help address this scalability issue. There are tools available [7] that can store the various elements of a
such tools will need to be modified to accept metrification schemes so that they may calculate metrics as well.

5. Conclusion

In this paper we have introduced the POMSA framework, a framework for Process-Oriented Metrics for Software Architecture. Besides helping
calculate the metrics for NFRs at the architectural level, POMSA provides the following advantages: intuitive calculation of metrics, traceability of metrics to requirements, calculation of metrics at the architectural level, ability to analyze reasons for weaknesses in the metrics, ability to visualize the effect of architectural changes on the metrics, and historical record keeping for later reference.

In this paper we compared architectures for two commercial systems used for testing cell phones. These systems were individually more than 50000 lines of code. In order to apply POMSA we used the feedback from the engineers involved with the development of the systems. We computed the adaptability metrics for the architectures using the POMSA framework using two different metrification schemes including single value, probabilistic, analyzed reasons for their metric values which helped detect room for further improvements, and suggested possible improvements. We received positive feedback on the utility of this framework, as well as recommendations from the development engineers.

There is much work still needs to be done – exploring guidelines for developing specific metrification schemes which preserve the relative ordering of qualitative measures, studying the scalability of POMSA possibly with aid of automated tool, and applying of POMSA to other software systems for further feedback on the strengths and weaknesses of the framework. Although much work still remains to be done, it is our opinion that our preliminary studies show that the POMSA framework will be of much help to software development organizations in practice with knowledge of measurements.

**References**


