The Journal of Systems and Software 85 (2012) 2034–2047

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

# Industrial architectural assessment using TARA

Eoin Woods*

*Artechra, Hemel Hempstead, Hertfordshire, UK*

ARTICLE INFO

*Article history:*
Received 15 December 2011
Received in revised form 14 April 2012
Accepted 23 April 2012
Available online 2 May 2012

*Keywords:*
Software architecture
Software architecture assessment
Case study
Expert-judgement

ABSTRACT

Scenario based architectural assessment is a well-established approach for assessing architectural designs. However scenario-based methods are not always usable in an industrial context, where in our experience, they can be perceived as complicated and expensive to use. In this paper we explore why this may be the case and define a simpler technique called TARA, which has been designed for use in situations where scenario based methods are unlikely to be successful. The method is illustrated through an experience report that explains how it was applied to the assessment of two quantitative financial analysis systems, and its strengths, weaknesses and relationship to other methods are briefly discussed.



## 1. Introduction

Scenario-based architectural assessment techniques are a well-established approach for performing structured evaluations of architectural designs, with the aim of validating that they meet certain objectives and analysing the decisions that have been made in order to achieve them. New research is published in this area regularly (Kazman et al., 1996, 1998; Barbacci et al., 2003; Olumofin and Misic, 2005; Pooley and Abdullatif, 2010) and there is evidence of some industrial adoption of the techniques too (Zalewski, 2007; Kettu et al., 2008).

However we have observed that scenario based architectural assessment techniques are not very widely used in industry, with informal approaches or "assessment by committee debate" being more common. Experience of trying to use scenario based techniques in industry has led us to conclude that this is for a number of reasons including a perception that these techniques are involved and expensive to apply, a lack of confidence about the benefits of such assessments and the fact that most of the methods focus on assessing architectural proposals rather than considering a system's implementation as part of the process (where it is available).

Our experience also suggests that the context of an industrial assessment is often at variance with the context that is assumed by scenario-based methods. For example, where ATAM assumes early stakeholder involvement and the assessment of competing architectural options, the industrial context is often one where some key architectural decisions have already been made and a system is partially or completely implemented. In these situations, the assessment needs to focus on whether the architectural decisions that have already been made can support the key requirements of the system, rather than debating options prior to implementation.

This experience with industrial architectural assessment led us to create a simple architectural review method called the Tiny Architectural Review Approach (TARA) that is quick and inexpensive to apply. It aims to be less prescriptive than most of the scenario based methods, does not assume that all of the system stakeholders can dedicate much time to the process and where the implementation of the system is available, uses this as one of its major inputs.

The remainder of this paper explains why an alternative to formal scenario based architectural assessment methods is sometimes needed, defines the steps of the TARA approach, presents a case study that explains how it was used for the assessment of two systems, and discusses the strengths and weaknesses of the method and how it compares to other architectural assessment approaches.

## 2. Using scenario based assessment methods

Most scenario-based assessment methods, such as ATAM (Kazman et al., 1998) and CPASA (Pooley and Abdullatif, 2010) are thorough and comprehensive approaches that gather the stakeholders of a system and lead them through a structured process that explores the trade offs between conflicting architectural options and the resulting implications for the effectiveness of the system. They result in a deep understanding of the architectural options for the system under consideration and the strengths and weaknesses that they are likely to embody. Such methods are valuable additions to the software architect's range of techniques, and can produce very valuable results when thoughtfully applied.

* Correspondence address. Tel.: +44 1442 254761.
*E-mail addresses:* eoin.woods@artechra.com, eoinwoods@gmail.com

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2012.04.055

However, our experience suggests that the use of formal scenario-based assessment methods in industry is quite rare and our experience of trying to introduce them has led us to conclude that there are a number of reasons for this.

Firstly, there is a common perception that applying a method like ATAM is complicated and costly, coupled with a lack of conviction that the results of the exercise will be useful (or at least useful enough to provide a return on investment). Applying a method like ATAM, having just read a book or technical report, is quite a daunting prospect with many unanswered questions, involving a number of probably unfamiliar concepts such as scenarios, architectural styles and utility trees. For a complicated system, just the difficulty of persuading the relevant stakeholders to participate is enough to deter many people from embarking on such an exercise.

A secondary reason that many people do not use methods like ATAM is that they focus on the design options for the system and do not explicitly suggest using implementation artefacts as inputs. This reflects the focus of these methods, which is on performing pre-implementation assessments, and it allows the methods to be used in order to choose between competing design options before system implementation. But many industrial assessments are initiated because of dissatisfaction with a system that is at least partially implemented already. In these situations the implementation of the system is an invaluable input into the assessment exercise. While this is not the only context in which TARA is a useful approach, it is one that most other evaluation methods do not address explicitly.

Industrial systems development is also frequently undertaken in the context of evolving and extending existing systems, rather than creating entirely new ones (so called "brown-field" development rather than "green-field" development). In these cases, the existing implementation of the system is an important part of the design context that needs to be taken into account.

A third reason that that can act as a barrier to the adoption of scenario-based methods, which has been noted by others too (Bashroush et al., 2004), is the need for significant time and commitment from a range of stakeholders in order to identify, define and validate a good set of scenarios. This can be very difficult to achieve in an industrial context when there is not a general understanding and acceptance of the benefits of architectural assessment.

In order to provide an alternative approach, that offers the option of focusing on an implemented architecture, as much as a design proposal, the Tiny Architectural Review Approach was defined to provide a simple approach to performing a basic architectural review that would be structured and repeatable as well as easy to apply with limited resources and commitment. The term "tiny" is used deliberately in the name to stress that the method is the simplest approach possible, rather than a comprehensive method.

The aims of TARA are twofold. Firstly it aims to provide some structure and guidance as to how to run a simple architectural review without extensive involvement from all of the system's stakeholders. Secondly, it aims to prove that architectural reviews are valuable and so allow discussions about the usefulness of architectural review in general and the possibility of using more sophisticated methods where the situation justifies them.

## 3. Related work

There is a large body of research literature on the subject of architectural evaluation of software intensive systems. It appears that there has been research going on in the area of architectural assessment for over 15 years, with the earliest definition of a systematic method for analysing the architecture of a system being the initial description of the scenario-based SAAM method in 1996 (Kazman et al., 1996).

Since then, methods defined by the SEI including ATAM (Kazman et al., 1998), QAW (Barbacci et al., 2003) and ARID (Clements, 2000) have been very influential in this area. Arguably ATAM has become the de-facto standard for architectural assessment where a formally defined method is used. These methods have also spawned a number of derivatives such as SAAMCS (Lassing et al., 1999) and ESAMMI (Molter, 1999) that are extensions of SAAM and HoPLAA (Olumofin and Misic, 2005) that is an extension of ATAM.

Other scenario-based architectural evaluation methods that have independently been proposed include Architecture Level Modifiability Analysis (ALMA, Bengtsson et al., 2004), Continuous Performance Assessment of Software Architecture (CPASA, Pooley and Abdullatif, 2010) and Architecture Level Prediction of Software Maintenance (ALPSM, Bengtsson and Bosch, 1999).

It is interesting to note that most of the architectural evaluation and assessment methods that have been defined in the research community are scenario based, with a consensus obviously having been reached that scenarios should underpin any effective evaluation technique. However, as Jan Bosch notes (in Bosch, 2000) there are at least four general approaches to architectural assessment: scenario-based methods, simulation-based approaches, methods using mathematical models and experience-based assessment.

An early approach aimed at making design reviews effective that did not use scenarios was Active Design Reviews (Parnas and Weiss, 1987), which uses questionnaires rather than review meetings. Much later, the SARA working group gathered the knowledge of a number of experts and created a report containing a high level approach to architectural review (Obbink et al., 2002), which does allow for the use of scenario based assessment but suggests many other techniques that can be used in conjunction with or instead of scenarios.

More recently there have been some interesting reports of people who have explored architectural assessment and analysis techniques that do not assume the use of scenarios, such as the Software Architecture Evaluation Model (SAEM, Duenas et al., 1998), an approach based on the Goal/Question/Metric framework (Zalewski, 2007), and the Independent Software Architecture Review (ISAR) approach (Tang et al., 2008) that attempts to improve architectural evaluation by defining a comprehensive standard for the documentation that is required to perform an assessment exercise.

TARA is not the only attempt to make architectural assessment more approachable in an industrial context. The Lightweight Architecture Alternative Analysis Method (LAAAM) defined by Jeromy Carriere is not yet very thoroughly defined in the literature (Carriere, 2009) but seems to have been created with similar motivations to TARA. It is an innovative method based on some of the key concepts of ATAM, which aims to simplify it, while retaining the use of key techniques such as scenarios and quality attribute trees.

Finally, Tommy Kettu and his colleagues discuss how architectural analysis is used at ABB, to support understanding and evolving existing systems (Kettu et al., 2008). In many ways, the experience reported by these authors is closest to the environment and experiences that inspired the development of TARA.

## 4. The TARA method

### 4.1. Origins of the approach

The TARA method was initially developed in response to a request to provide an assessment of a quantitative financial analysis system that had been developed in-house by a major fund manager. The system had been developed within a specific business

unit (largely outside the purview of the people who viewed themselves as responsible for such systems). The system appeared to be successful and the question being asked was whether the system should be adopted more widely in the organisation. The system was new, and so somewhat unproven, but it was largely finished and appeared to have strong user acceptance.

A senior business manager had inherited ownership of the system due to a reorganisation and needed to know how "good" the system was in order to decide whether he was going to sponsor its on-going development (in the face of some opposition). Open and frank discussions with the manager concerned revealed that by "good" he meant whether the system's architecture was fit for purpose, rather than whether the code modules were written well at the most detailed level. The motivation for the question was to establish whether investment should continue in this system, because its fundamental design decisions were sound, or whether investment should be directed elsewhere because however well written the individual classes were, the system architecture could not support the key requirements of the system.

The sponsoring manager needed answers quite quickly and the timing and organisational and political context of the request meant that there would not have been much enthusiasm for employing a more thorough "high ceremony" method like ATAM.

At this point, the TARA method had not been defined and the options open to the assessor were to attempt the use of a standard scenario based assessment method or to try to perform the assessment in an ad-hoc manner. However the idea of a lightweight assessment approach for situations like this emerged and it was decided to try to define the method (which is now called TARA) and test it on the system in question.

At this stage the method was defined very informally, by creating a document template containing the headings for the outputs that the review would need to produce. As the headings formed, the need for other sections emerged (such as the code analysis section to balance the more abstract and subjective sections) and the first TARA review simply involved the completion of the activities needed to finish the document.

Some months later a similar situation arose, by coincidence with a similar system, another quantitative analytics system. Again, a senior manager had inherited a system by virtue of a reorganisation and needed to understand what he had become responsible for. In this case, it was assumed that the system in question was going to be used as the global strategic system for the type of processing that it was responsible for, but no architectural assessment had been performed to support this decision. The manager in question, having seen the earlier assessment's outputs, asked for a similar assessment to be performed for this second system, in order to assess its "fitness for purpose" in its proposed role.

This second architectural review was performed using the same process as the first one and as part of the exercise the first written description of the approach was produced, in order to allow it to be explained to both review participants and to other assessors who might perform future reviews. Following this, another assessor undertook a couple of reviews based on the approach in the same organisation.

In summary, at the time of writing, the method has been used to provide some insight into the suitability of the architecture of a handful of systems, by a handful of assessors, all within one organisation. This is not a strong claim to repeatability, however it has been explained to a couple of assessors not involved in its development and used in a couple of assessments separate from those described in this paper. While this is clearly small-scale use, it does suggest that there are not major barriers to wider scale application and its use is not just limited to those involved in its creation.

### 4.2. Overview of TARA

The Tiny Architectural Review Approach (TARA) is based on industrial experience in situations where full blown architectural assessment methods are not suitable for reasons such as organisational culture, lack of familiarity with architectural assessment, insufficient budget or time for a larger exercise or an inability to involve a representative stakeholder group in the process. These experiences led to the conclusion that a structured and repeatable method would be useful, provided that it could be made quick, flexible and simple to use, and require a modest initial investment of time and resources.

TARA differs from more formal scenario-based methods in a number of important ways:

- The approach does not mandate the use of scenarios because, in our experience, creating valid and meaningful scenarios requires significant time and effort from a range of system stakeholders. As already explained, TARA aims to be useful in situations where little focus and time is available from many of the important stakeholders. Hence the TARA approach is to involve stakeholders wherever possible (and certainly to validate all assumptions with them) but not to assume significant commitment and engagement on their part. We tried using scenarios without direct stakeholder input but found that an assessor creating formal scenarios themselves was a rather artificial and time-consuming activity. Instead, as we will show later, we decided to base TARA more on expert judgement than scenarios, although scenarios may well be used as part of the process.
- Most scenario-based methods (ATAM being the classical example) are at their most effective when helping an engaged stakeholder community choose between a number of architectural options. TARA focuses on the simpler situation of trying to establish how well suited a particular architecture (which has often been partially implemented) is to supporting a set of key requirements
- The method explicitly allows for the situation where the system has already been implemented. The method can be used when a system does not yet exist, but where an implementation is available it forms an important input to the process.
- TARA deliberately does not mandate specific analysis techniques (such as ATAM's use of quality attribute trees). Such techniques can all be used if appropriate, but one of the key characteristics of TARA is its simplicity and mandating additional techniques can be off-putting when a simple approach is needed.
- TARA is intended for use by a single assessor, or a small group of assessors, rather than assuming that a large group of stakeholders will be prepared to dedicate significant time to the assessment process.

The trade-off inherent in the approach is that using TARA results in an architectural assessment that is less thorough, insightful and reliable than one performed with a more formal and comprehensive review technique such as ATAM. Given the approach of stakeholder consultation rather than mandatory participation, it is also important that the assessor using the technique has good organisational and domain knowledge to compensate for this (and so TARA is probably more suitable for in-house use than use by visiting consultants).

However the great strength of the method is that it can often be used in situations where it would not be possible to use a more involved scenario based technique. TARA can also be used as a first step in architectural evaluation for an organisation that needs to be convinced of its benefits. Once benefits are forthcoming from TARA's simple approach, this may help significantly with the introduction of more sophisticated techniques.

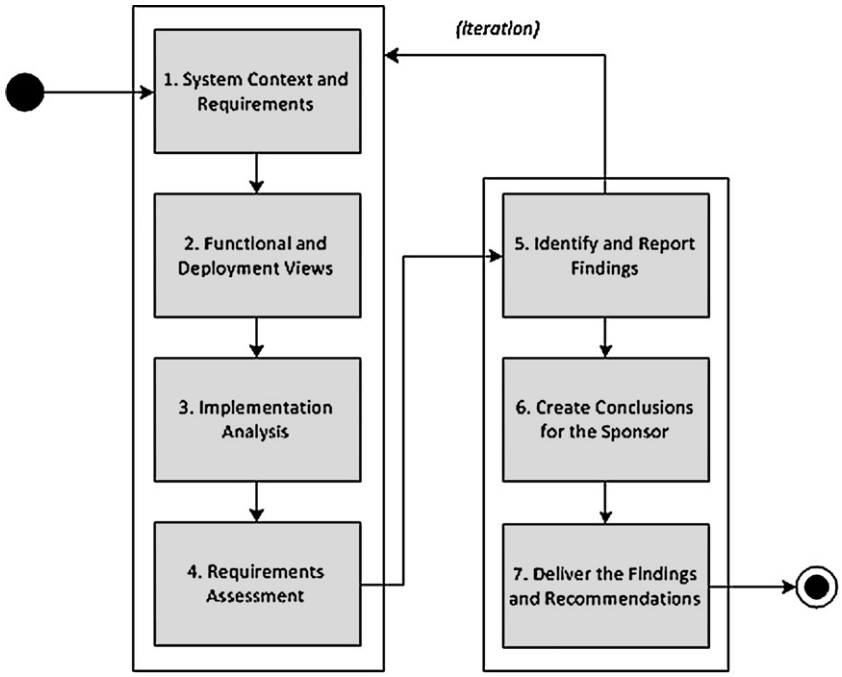


**Fig. 1.** The steps in the TARA method.

The important preconditions for a successful TARA assessment are:

- A sponsor who is responsible for the system concerned, wants a number of valid questions answered and is prepared to trust the judgement of the assessor. This precondition is important to provide a purpose and focus for the review and to ensure that the assessor is likely to receive the cooperation required to perform the assessment.
- An assessor with sufficient organisational and domain knowledge to understand the role of the system and its key functions and to understand the organisational context of the system and the review being performed. This knowledge is important because, when using this approach, the assessor cannot rely on a formal process of stakeholder consensus building to help him understand the system.
- Access to the implementation and production instance of the system being assessed (assuming it exists) and ideally access to the development and support team who are responsible for it. This access allows the assessor to gain a detailed understanding of the system based on its real implementation and production characteristics, rather than relying on the possibly biased opinions of the limited stakeholder group who are likely to be involved in the review.

These preconditions help to compensate for the inevitable limitations that result from the simplicity of the approach.

The TARA assessment process is structured into seven steps as shown in Fig. 1 and described in the sections below.

### 4.3. Step 1: system context and requirements

The first step in the process is to understand the context in which the system exists and the key functional and quality-property requirements that the system must meet. Occasionally this information will be readily to hand, in user stories, requirements documents, work tracking systems (such as Jira), requirements list spreadsheets and so on, but usually gathering this information is part of the assessment exercise. This step in the process is roughly equivalent to the step in the ATAM process called “Presenting the Business Drivers” (Kazman et al., 1998).

The system context and key functional requirements are usually fairly straightforward to gather from the development team, the system’s key users and perhaps the sponsor who has asked for the assessment (although the differences in the requirements focus between those groups can be illuminating in itself). This stage in the process is a key point when it is valuable to consult the stakeholders who are available and prepared to be involved in the assessment. Our experience is that meetings with small numbers of stakeholders tend to be the most efficient way to involve them in the process, but if stakeholders are prepared to attend workshops, this is the time to schedule them.

The system context can often also be deduced from the system’s implementation and operational environment, which is a useful cross check. It can also be useful to crosscheck the functional requirements that are gathered from stakeholders against the functions that the system actually provides.

In contrast, experience has shown that gathering a good set of system quality requirements is usually significantly more difficult than gathering functional requirements. In many cases, even the development team will struggle to clearly define the qualities that their system is expected to meet, meaning that the assessor needs to define these requirements. (Which can be time consuming, but means that a set of reasonably accurate quality requirements is a useful side effect of the assessment process.)

The best approach we have found for identifying system quality requirements is for the assessor to use their experience in the domain to suggest a set of credible quality requirements based on domain and organisational standards and norms (for example, estimating the system’s required availability based on working hours and its recovery point objective based on industry norms for data loss). This set of candidate quality requirements can then be validated with relevant system stakeholders. Experience suggests that development teams and key users or sponsors can spot and correct an unreasonable quality property requirement much more quickly than writing one by themselves.

The output of this step should be a clear context diagram, showing how the system relates to its environment (such as systems it receives data from or supplies data to) along with a succinct set of requirement statements that capture the key functional and system quality requirements for the system.

### 4.4. Step 2: functional and deployment views

Having understood the system's context and requirements, the next step is to understand its key design elements. As has been extensively discussed (Garlan et al., 2010) the architecture of a system is made up of a number of structures (including functional elements, information elements, deployment environment, software design structures and so on). For the purposes of this exercise, experience has shown that the key architectural structures to understand for assessment are the functional structure (runtime elements, their responsibility and interactions) and deployment structure (the environment that the runtime elements are deployed into). This step in the process is roughly equivalent to the step of "Presenting the Architecture" in the ATAM method (Kazman et al., 1998).

Some of this information can usually be found in the form of Visio, PowerPoint, whiteboard sketches or more formal artefacts like UML models. However, it is usually the case that part of the assessment activity will be the creation of fairly formal "architectural sketches" to provide outline functional and deployment views of the system. (The term "sketch" in this context means a well-defined graphical representation of the architectural structure, with enough supporting text or other information to make it sufficient for the exercise in hand, as opposed to a fully completed architectural "model" intended for more general use.) Any suitable notation can be used for the architectural sketches, but we have generally used UML and found it to work well. What is important is that whoever creates the views (typically the assessor) is clear as to what they are representing and that the notation used is clearly defined (which in itself requires effort, but is well worth expending time the time required in order to achieve this).

It is also important that at this stage the assessor clearly understands the architectural structures represented in the views and is very confident that they are correct. This is the moment to find out whether the team members have been describing a "logical" view of the system (i.e. an idealised imaginary one describing what they wish they would built) or whether the views that the assessor has managed to assemble are an accurate depiction of reality. It is usually necessary to walk through some mental processing scenarios and perform some tactful and discreet crosschecking with implementation artefacts and the production environment in order to achieve this.

The output of this step should be a small number of relevant architectural views, and a thorough understanding of the system's main architectural structures on the part of the assessor, which should provide a good basis for the rest of the assessment process.

### 4.5. Step 3: implementation analysis

The creation of the context diagram, identification of requirements and the creation of the functional and deployment views are all activities that rely to some extent on expert judgement rather than simply recovering facts. When they are available, the next step in the process analyses the system code and other implementation artefacts in order to provide some objective knowledge into the exercise. As the old saying goes "the code does not lie". As well as source code, the system's production metrics, incident reports, log files, test reports and so on can all provide useful insight into the ability of the system's architecture to meet its key requirements. That said, the focus on TARA is on the *design* of the system, so the system structures recovered from code are the key input into this step, as that is where much of the design information is often hiding.

The code analysis that can be performed depends on the languages that the system has been implemented in, the quality of the code and the analysis tools available. For example, a well-structured system implemented entirely in a byte-code compiled language (like Java or C#), where tests have been separated from production code, that follows conventions and where some static analysis tools are available, will be much easier to analyse than a situation where a system is written in Perl, following few conventions and where a good analysis tool is not available.

The basic types of code analysis recommended as part of a TARA analysis are:

- Module structure and dependencies (ideally recovered using an automated tool, so showing the real structure of the system).
- Size measured in terms of lines of code, size of binaries, number of files/classes/procedures or similar, with separate measures taken for production code and test code.
- Code characterisation metrics measured using an automated tool that can derive measures such as the cyclomatic complexity, XS, code duplication, coupling, comment to code ratio, number of large methods and similar, for each module of the system, as well as weighted averages at higher levels.
- Test coverage, measured using a coverage analyser, after running all automated tests that are available.

These measures are all easy to derive using readily available commercial or open source tools, are easily explained and provide a good characterisation of a system's implementation. They provide some quantitative background to the design recovery work and often point to areas of the system that merit further investigation. They can only provide a partial picture of the system's architecture (for example they cannot assess the runtime qualities or the deployment environment) but still provide a useful insight into the system's design and implementation.

More advanced code analysis techniques which are well worth considering if the time and tools to measure them are available include static problem analysis (using commercial tools like Jtest or open source ones like FindBugs), to provide a general indication of how carefully the code has been written, and test mutation analysis (using something like Jumble or Jester) to establish whether a high code coverage measure means anything or not. If the development team has been honest enough to keep a realistic technical debt log, then this is also a useful input (although clearly, it is important not to use it to inadvertently "punish the innocent" for doing the right thing).

As already mentioned, the production environment is also a rich source of information that can be used to assess the ability of the system to meet its key requirements. Incident logs can help to highlight areas of the architecture that need improvement, release records can suggest strengths or weaknesses in continuous delivery or testing, production logs can allow automated derivation of production metrics, infrastructure metrics can help to identify resource usage and so on. Depending on its focus, some or all of these inputs can be relevant inputs to the assessment process.

### 4.6. Step 4: requirements assessment

By this stage, the assessor should have a good understanding of the capabilities of the system and how well it has been implemented. The next stage is to move up to the level of system design and to perform an assessment of the ability of the system to meet its functional and system quality requirements.

Given the deliberate simplicity of the TARA method, this step in the process is one that relies primarily on judgement rather

than quantifiable assessment. The ability of the system to meet its requirements usually cannot be tested during an exercise such as this but must be assessed by expert judgement. That said, where the system is in production operation and metrics are available (e.g. for throughput or outages) then these should be used as an input to the process. It is also worth noting that assessing the ability of a system (or potential system) to meet its requirements almost always involves expert judgement to some extent, even if more sophisticated methods (such as scenarios) are used as part of the process.

The functional capabilities of the system are easier to assess than the system's qualities and the match between the capabilities and the current requirements can usually be assessed using a combination of the assessor's domain knowledge and canvassing the opinion of domain experts such as key end-users of the system. A structured approach to assessing functional requirements fit is to split each functional requirements area into a list of fine-grained functions of roughly equivalent complexity, and then to count the percentage of those functions that are provided by the system (effectively performing a gap analysis). This is a point in the process where the use of functional scenarios can be valuable and the assessor should consider using them, even it is not possible to validate them thoroughly with the system's stakeholders. Again, this is a situation where stakeholders should certainly be consulted but may not necessarily be fully engaged in the process.

Amassing proof the architecture's ability to meet its key quality requirements can be quite difficult. Even when available, assessment methods from the research domain may not have a lot of credibility in the organisation and may not be mature enough for use in an industrial project. On the other hand, it may not be practical to test the system in order to prove that it can achieve its key qualities. In addition, the obvious sources of knowledge about the system (such as the development team or the system administrators) may well not have accurate information or sound intuition about its ability to scale, be secure, provide a certain level of throughput and so on. As noted earlier, the lack of precision and the uncertainty that tends to characterise the available quality property requirements makes this process difficult.

Realistically, in a short assessment exercise, the assessor needs to rely on expert judgement (their own and others who they can find to assist them) in order to estimate the non-functional abilities of the system. But this is also the step in the process where a number of established techniques including scenarios, quality attribute trees and modelling techniques such as queuing models are valuable and should be used where they appear to be useful. The method deliberately does not mandate their use, but does not discourage it either. The goal should be to produce some form of measure as to how well the system is likely to be able to meet its quality objectives (such as a confidence indicator). In practice we have found that the ATAM quality attribute tree technique is useful, even if used informally, to refine the requirements to simple scenarios that can be analysed further.

The result of this step should be a clear list of the system's functional and quality property requirement areas, with a clearly defined measure of the assessor's confidence in the system's ability to meet each area (we have typically used high/medium/low, red/amber/green and 1–5).

### 4.7. Step 5: identify and report findings

Throughout the assessment activities, the assessor will have been drawing conclusions about the context in which the system is being developed, the fundamental soundness of the system's design, the way in which it has been implemented, the way it is deployed and operated, and its likely ability to meet its key requirements. All of these insights should have been based on a combination of facts about the system's design and implementation and the judgement and experience of the assessor. All of these insights are valuable and should be captured as outputs of the assessment activity.

Experience suggests that it is sensible to create the assessment report in parallel with the assessment activities taking place, but this is the point at which the assessor needs to consider how best to present the information in the report, so that all of the evidence and opinions are clearly stated (and supported where necessary).

The findings need to be organised into logical groups that relate to the different aspects of the system being discussed, with each finding being clearly described with a short meaningful name, an identifier, a full description and a justification or reference to further evidence to support the finding. All findings should be reported, even if not directly relevant to the original sponsor's request, as they may well be useful to other stakeholders. Inevitably many findings tend to be critical in some way, so balancing them with some positive findings about what the system does well and expressing negative ones tactfully will help to produce a report that is perceived to be valuable and balanced and is accepted by those affected by it. This can take some thought and practice, but is an important skill along with the more obvious skills of sound judgement and technical knowledge.

As the findings are being considered and written, it is usually the case that evidence is missing or needs to be reanalysed or appraised, leading to iteration from this step back into the previous steps in the process.

### 4.8. Step 6: create conclusions for the sponsor

Ultimately the assessment exercise has been undertaken for a particular sponsor, or perhaps a specific group of sponsors. The sponsor almost certainly had particular questions in mind when they asked for the assessment to be performed and it is important to answer these questions or present other specific recommendations to address shortcomings that have been identified.

This is achieved by adding a "conclusions" section to the report (or creating a separate short report for the sponsor) that answers the explicit and implicit questions being asked by the sponsor who commissioned the assessment and presents any other recommendations that are required. Given that sponsors are often executive level managers, a separate short report that focuses on their specific questions is more likely to be read than a full assessment report.

In many cases, this part of the report may be little more than summarising, highlighting or restating findings that were already presented elsewhere, but in other cases answering the sponsor's specific questions may lead to new conclusions that were not simply findings from the study. In this situation it may be necessary to again iterate back through the earlier steps of the study to perform more analysis or gather more evidence to support a conclusion.

In cases where some conclusions are sensitive, they may need to be presented separately to the rest of the report (a good example being the answer to the question "should I continue to fund the development of this system?").

### 4.9. Step 7: deliver the findings and recommendations

The final step in the process is to deliver the findings and recommendations to all of the stakeholders affected by them and those who have provided input to the assessment exercise.

This is often a multi-step process, with a report being circulated to those who are interested in the detail and one or more presentations being prepared to deliver findings or recommendations with a particular slant or in a particular style for the audience concerned (in most cases the presentation to the development team would be quite different to the one needed for a business sponsor).

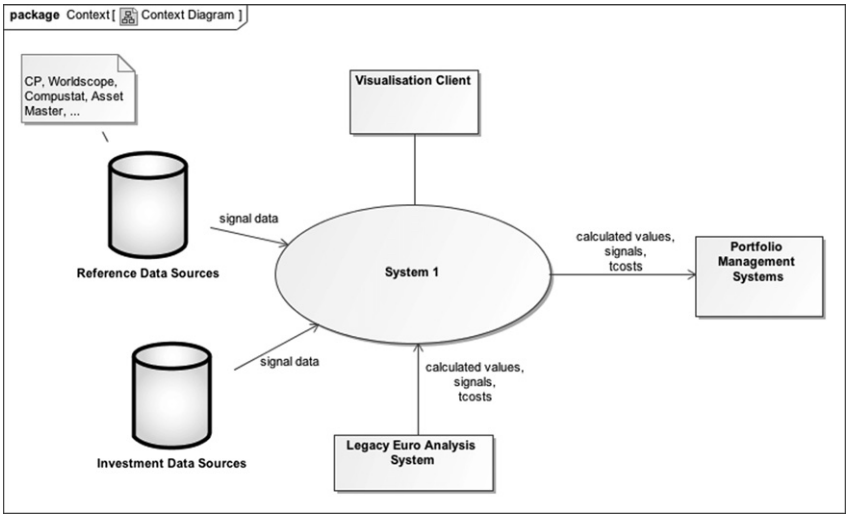


**Fig. 2.** Context diagram for system 1.

All of these documents need to make their points clearly and tactfully, in language that the audience will understand, stressing positive aspects of the findings as well as the negative ones.

We have also found that it is important for the presentation of the findings and recommendations to provide the context for the assessment (who commissioned it, why it was commissioned, who performed the assessment, why they were chosen to do so) and to explain the process followed as well as just presenting the results of the process.

## 5. Case study of TARA in use

As mentioned earlier, TARA was developed because of the need to perform industrial architectural assessments in an environment where an ATAM style assessment was unlikely to be successful. This section describes two situations where the method has been used for similar but separate system assessment exercises. In order to respect the confidentiality of the organisation involved, the output material from the reviews that is reproduced here has had some detail removed, but otherwise it is reproduced in its original form. Hence it contains the flaws, ambiguities and errors that most pieces of industrial work contain, but it is an accurate representation of the outputs of real review exercises.

### 5.1. System 1 assessment

As explained earlier, the TARA method was initially developed in response to a request to provide an assessment of a quantitative financial analysis system that had been developed in-house by a major fund manager. The system had been developed within part of the organisation that was not officially responsible for such systems. The system appeared to be successful and the question being asked was whether the system should be adopted more widely in the organisation A senior business manager had inherited ownership of the system due to a reorganisation and needed to understand whether the architecture of the system was sound, in order to decide whether he was going to sponsor its on-going development (in the face of some opposition).

No documentation really existed for this system before the review and some examples of the documentation produced as part of the assessment are shown below.

The diagram in Fig. 2 shows the context diagram that was captured early in the assessment exercise, showing that the system takes inputs from a number of data sources and a legacy system and supports a GUI client and produces outputs that are fed to portfolio management systems.

**Table 1**
Example requirements for System 1.

| | |
|---|---|
| FR1 | Quantitative model management and execution – the core responsibility of the system is to allow quantitative model to be defined and executed when required. The model defines the input data, calculation status and output data that result in the generation of the quantity and cost values, which are the system's main output. |
| FR2 | Override management – in many cases, users of the system will want to be able to override individual values or groups of values in the source data being used by the system. The system must provide the ability to create, remove and report on overrides and how they have affected the quantity value calculations. |
| NFR1 | Performance – the key performance metric is the time taken to perform a model calculation run and generate results. Currently this is assessed to take in the order of 30 min in the system, but the target time for this is about 10 min. The other important performance requirement is the implicit requirement for the user interface to be useably fast (defined by the organisation to mean never freezing, responding instantaneously to local UI events and new data being available within 10 s of a request). |
| NFR2 | Scalability – the key scalability requirement is likely to be maintaining the bound on the quantitative model execution time as the size and sophistication of the model and the input data grow. This is likely to be a key challenge in the future. A related scalability requirement is the implicit requirement for the user interface to remain usable as the amount of data in the system and in each model run grows. Finally, the system's user base will never be very large but it will probably need to support 30 or 40 users per region in the long term. |

Table 1 lists some of the requirements that were identified as part of the assessment process. Again a formal and accurate set of requirements was not available for the system, so they were identified as part of the assessment.

When reviewing these requirements, it is interesting to note how the two functional requirements (FR1 and FR2) are stated in more definite terms than the quality requirements (NFR1 and NFR2). This is because the key stakeholders, such as developers and end-users, were able to clearly state the system's functional requirements but were not able to clearly articulate the qualities that they required of the system. Hence the quality requirements are the result of the assessor's judgement and so are expressed in less definite terms. This was obviously not ideal as the assessor's judgement might not have been correct, however we have found

**Table 2**
Example quantitative measures for System 1.

| | |
|---|---|
| Implementation size | ~1150 Java classes and 20 database tables. The Java code is approximately 111,300 (raw) lines of code and is ~230,000 Java byte code instructions. |
| Test size | ~60 Java test classes which reference ~100 Java classes in the implementation. |
| Structure | Code organised into 10 modules and 8 layers, with about 15% of the leaf level packages considered to be "tangled" together. |
| Tangled code | *Engine* – package `com.abc.system. engine` (46% of the code tangled); *server* – package `com.abc.system` (42% of the code tangled) and package `com.abc. system.service` (31% of the code tangled); *Base* – package `com.abc.system` (32% of the code is tangled) and package `com.abc. system.cuboid.dimension` (30% of the code tangled). |

that once non-functional requirements are stated, glaring errors or invalid assumptions are often pointed out by the key stakeholders, so this was not a great problem in practice. Stakeholders seem to find it much easier to tell people that the stated non-functional requirements are wrong and correct them, than to write correct ones themselves!

The UML component diagram in Fig. 3 shows one of the architectural sketches created when assessing this system, illustrating its functional structure. This diagram was supported by basic textual descriptions of each of the elements in the diagram along with some descriptive text.

Table 2 contains an illustrative sample of the quantitative metrics that were collected as part of the code analysis exercise for this system.

Some of the findings and recommendations from the report are shown in Table 3.

The first recommendation in the table ("recommendation 1") is an example of a recommendation that was largely unrelated to the specific findings of the architectural assessment (and was included to answer a specific question from the sponsor of the exercise), while the second is an example of one that is directly related to a finding (the finding "finding 2" in the table).

The sponsor was pleased with the assessment report and appeared to find it very useful because it allowed him to quickly understand the strengths and weaknesses of the system and its ability to meet its key requirements, so allowing him to make important investment decisions that related to it. Somewhat to our surprise, the development team also readily accepted its findings and worked with the assessor to identify specific solutions and actions to address the recommendations. The sponsor's satisfaction with the report was primarily due to the fact that it directly answered the questions he had posed (rather than being a generic architectural assessment, of the sort he had seen before) and because it was organised in a way that clearly described the system, with specific findings, supported by evidence (e.g. metrics) and clear reasoning (e.g. the logic behind expert judgement). This meant that the report was not particularly contentious, was easy to get people to read and led to it being accepted positively by those who had to act on its recommendations.

Interestingly, the main result of the exercise was a much higher degree of organisational confidence that the strengths and weaknesses of the system were understood. In fact, although weaknesses had been identified, the credibility of the development team's (naturally) positive opinion of their system was strengthened because the weaknesses of the system were now understood and were perceived to be rectifiable, rather than being fundamental architectural problems that would be expensive to resolve.

**Table 3**
Example findings and recommendations for System 1.

| | |
|---|---|
| Finding 1 | Model implementation – the quantitative model implementation is very nicely done and a significant innovation when compared to previous such systems. The fact that the model definition is now effectively data, rather than code, means that it can be evolved much more quickly than previous systems allowed and also (in principle) understood and modified by people outside the development team. It also opens up the possibility of implementing multiple execution engines for different scales and type of workload. |
| Finding 2 | Internal dependencies – the inter-module, inter-package and inter-class dependencies in the system would benefit by a review with the development team. In particular, the number of inter-module dependencies suggests that many sorts of change could be difficult in the future. Some of dependencies within the modules also appear to be very complicated and would benefit from a review by the development team to ensure that this level of inter-package and inter-class coupling is really required. |
| Recommendation 1 | Operational documentation – when installing and running the system, people in other regions will need simple, task oriented, installation and operational documentation to guide them. This could be as simple as a Wiki page of common procedures. |
| Recommendation 2 | Simplicity supporting variation – there is going to be a need to support variation within the codeline (for example providing different override logic in one region compared to another). In order to minimise the complexity of achieving this, refactoring parts of the code to make the internal dependencies as simple as possible is likely to pay dividends later. Simplifying the dependencies will also help people to understand the code. |

### 5.2. System 2 assessment

Some months later a similar situation arose, by coincidence with a similar system, another quantitative analytics system. Again, a senior manager had inherited a system by virtue of a reorganisation and needed to understand what he had become responsible for. In this case, it was assumed that the system in question was going to be used as the global strategic system for the kind of processing that it was responsible for, but no architectural assessment had been performed to support this decision. The manager in question, having seen the earlier assessment's outputs, asked for a similar assessment to be performed for this second system, in order to assess its "fitness for purpose" in its proposed role.

The process followed for this assessment was largely the same as for System 1, although because System 2 was older and its ability to evolve was in question, the focus of the assessment placed more emphasis on assessing the architecture's ability to support change than in the previous exercise.

Predictably, this assessment produced similar deliverables to the previous assessment of System 1, but to better illustrate the process, we present a slightly different set of outputs to the ones shown in the previous section.

Fig. 4 shows the context diagram for System 2 that was created as part of the exercise. This context diagram shows that System 2 was also a data processing "pipe" taking inputs from a set of databases, with quantitative parameters specified via other interfaces, performing statistical processing on that data and writing the results to the file system.

System 2 loosely follows a "pipe and filter" architectural style and so a data flow view was very relevant for capturing some

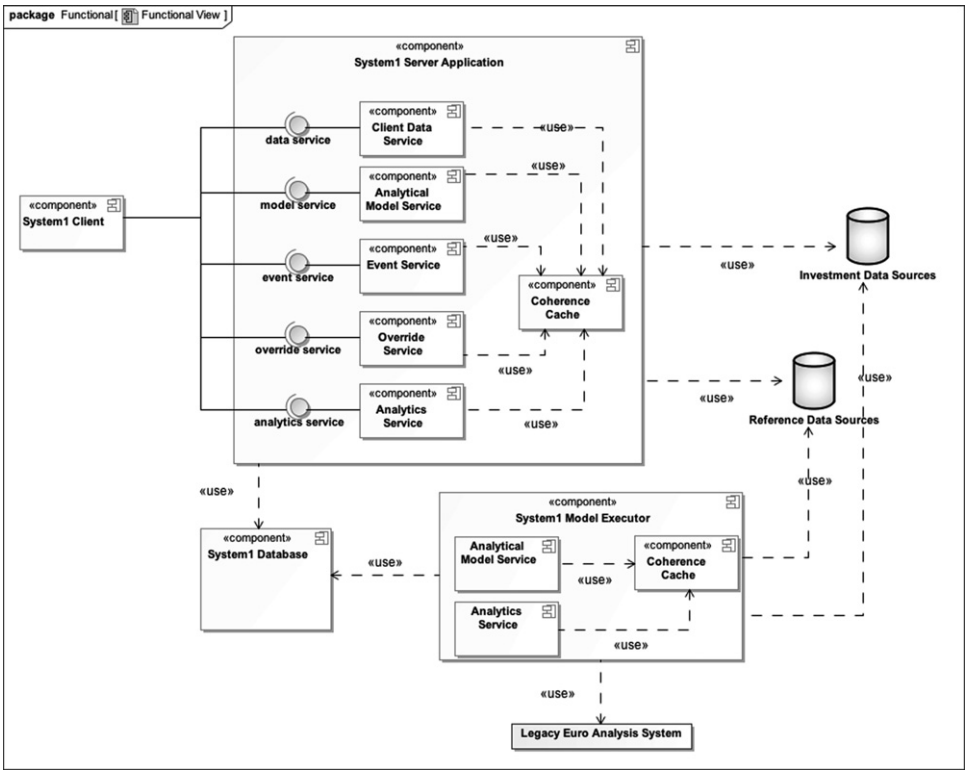


**Fig. 3.** Functional view sketch for System 1.

of the important relationships within the system and was produced as part of the assessment. The diagram from this view is reproduced in Fig. 5. This view clearly shows the classic linear processing "pipes" which System 2's functional elements are organised into.

One of the models usually produced during a TARA review is a code module structure analysis, to show the system's code modules and the dependencies between them. This provides visibility of the system's real functional element (or "component") architecture and whether it is as the designer intended. Given the age of System 2 and the concerns about evolution, this analysis was particularly relevant for this system and the result of the analysis can be seen in the dependency diagram in Fig. 6. This analysis highlighted the fact that although the module dependencies of System 2 form a recognisable structure, they do not clearly reflect the system's asserted architecture and they form a complicated structure with many cycles in the dependency graph. This finding was a valuable output of the exercise.

Some examples of the commentary as to how well System 2 met its requirements are shown in Table 4. A couple of examples of the findings that were reported for the assessment of System 2 are shown in Table 5.

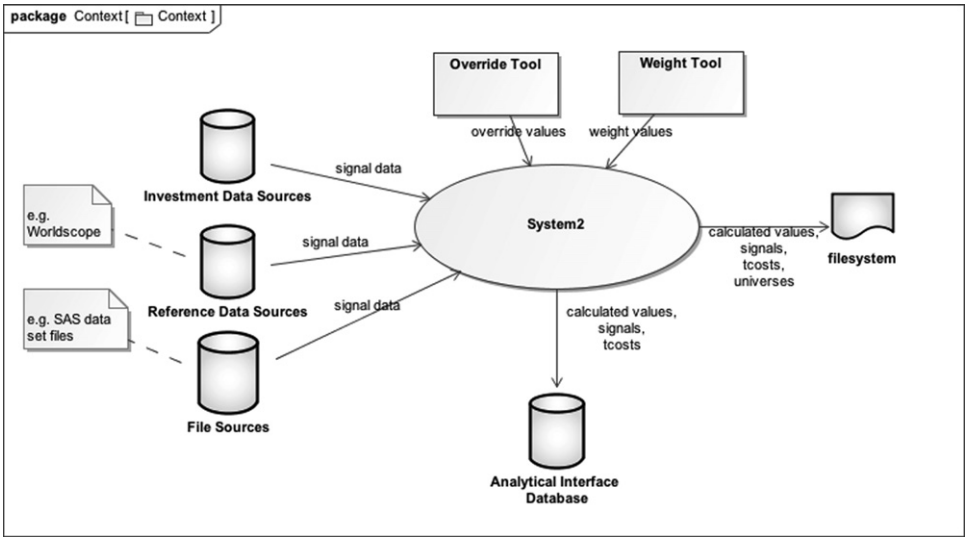


**Fig. 4.** Context diagram for System 2.

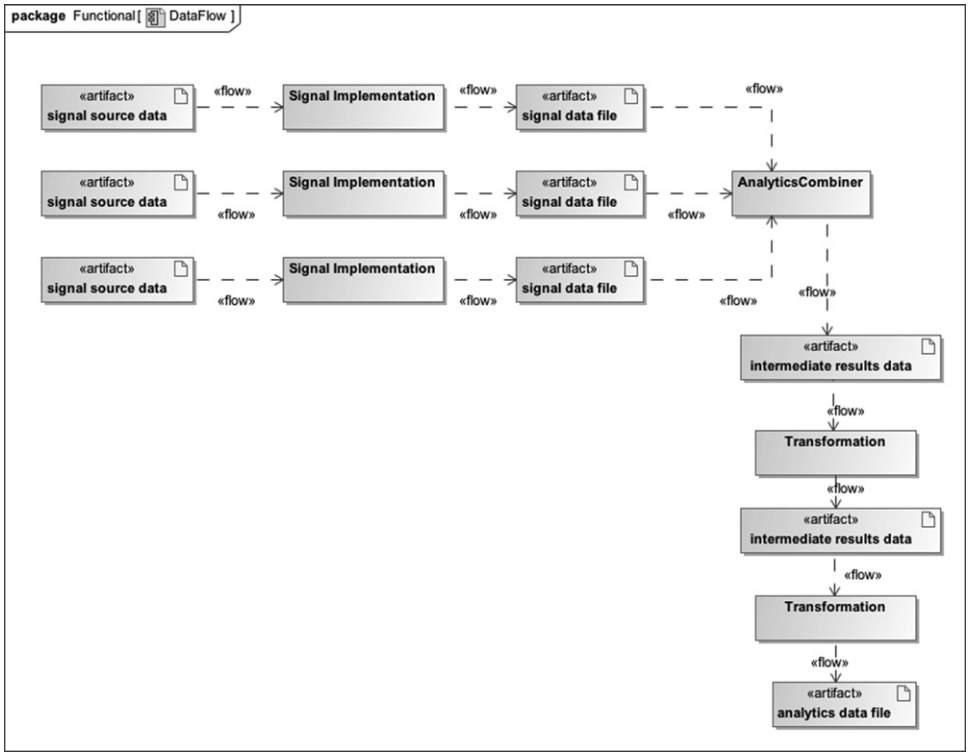


**Fig. 5.** Data flow information view for System 2.

The sponsor of this exercise also received the assessment very positively and the development team accepted most of the findings too (even though some of the findings were more critical than in the first case and had to be expressed tactfully). Again, the sponsor's satisfaction from the report stemmed from its focus on answering his specific concerns and its fact and evidence based approach. The fact that the findings were factual, fair and backed up by firm evidence (rather than simply being opinions) also helped with the acceptance of the results by the development team.

## 6. Evaluation of the approach

The TARA method has now been used successfully to assess a small number of systems and it has been successful in use, albeit

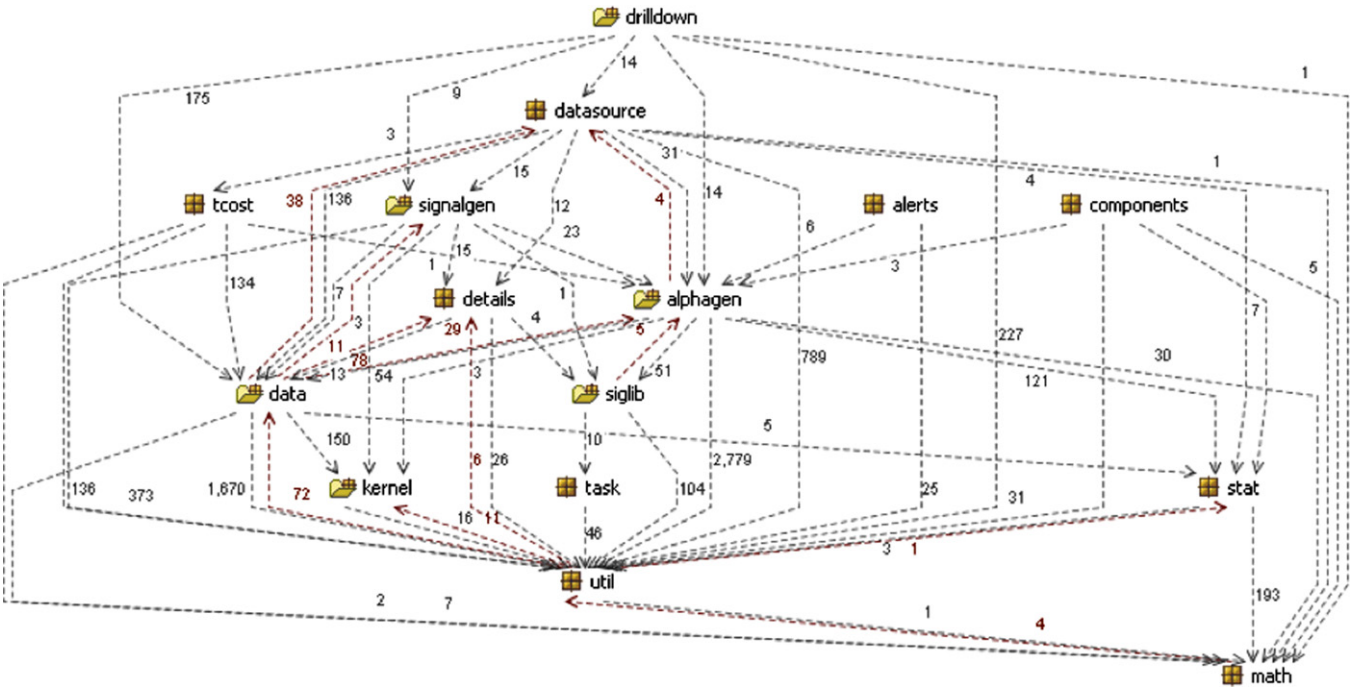


**Fig. 6.** Module dependencies for System 2.

**Table 4**
Requirements assessments for System 2.

| | |
|---|---|
| Derived value data generation | The initial part of the process (signal data processing) is performed by specific Java classes, containing code to extract the signal data from one or more data sources and to perform any initial processing required for it to be useful. The latter part of the process (merging and statistical processing) is performed by the ValueCombiner (according to configuration settings) and the Java transformation classes, running in Transformation Queues (the definition of the transforms to use also being part of the system configuration). This appears to work well and obviously provides enough flexibility for the current strategies being supported. |
| Data visualisation and analysis | System 2 does not provide data visualisation and analysis capabilities, the assumption being that portfolio managers, researchers and other interested parties will use other tools for these tasks. The lack of a server in the system's architecture means that there is no obvious way of remedying this without integrating a lot of code from other systems or a lot of development work. |
| Scalability | The simple batch based programs that System 2 uses mean that it probably exhibits quite good scalability requirements, at least to moderate scalability degrees. It is possible to split the workload up across many batch program invocations so that a lot of work can be done in parallel provided that the data dependencies allow this. The fact that the system also writes its signal data to intermediate files for the data pipelines to use means that signal data need only be generated once and is shared between compute runs, again helping with scalability. Scalability challenges are likely to emerge if very complicated calculations are defined that need to merge many large signals and then perform long pipelines of transformations on the result. |

**Table 5**
Example Findings for System 2.

| | |
|---|---|
| Overall structure | As reported earlier, the module structure of the system is very complicated and looks quite confused. This is probably the result of extensive evolution (see earlier) but it makes the system difficult to understand at the detailed level, and would make large-scale modification, extension or repurposing difficult. |
| Standardisation | The code of System 2 has obviously been developed by a number of people in a number of styles since it was originally created. It does not appear to follow any particularly strong coding or design conventions and while this obviously does not affect how the software runs, it does make it more difficult to understand, extend and maintain. |

in a specific environment. As explained earlier, sample size is small (and only relates to one organisation) but so far the method has proved to be useful when applied by a small number of architects. Given its simplicity, it is worth briefly considering what has made TARA successful and also where its weaknesses are.

Experience of using TARA suggests that the main reasons that it has been successful are:

- *Simplicity* – people are often suspicious of what they perceive as "high ceremony" methods containing many techniques with strange names that they think look like "common sense". TARA addresses this by using a very low ceremony approach that is easy to explain and deliberately does not try to introduce further named techniques as part of its application.
- *Structure* – the approach brings structure and standardisation to the assessment process in a lightweight way. Both assessors and stakeholders find this useful as it helps to ensure a balanced process that does not overlook important factors. Compared to the typical ad hoc practices found in most industrial settings, this is a useful step forwards.
- *Speed* – the benefit of being able to explain what you're going to do in 10 min and do it in 2 or 3 days, write it up in another couple of days and deliver the results in a couple of hours cannot be overstated. The speed of the process overcomes many objections to architectural assessment and often allows the technique to be used to establish enough credibility to allow the idea of more comprehensive assessments to be discussed.
- *Use of implementation artefacts* – integrating architectural design representation with an analysis of the system's implementation with facts derived from the production environment help to validate the results of the exercise and provide evidence to help people accept its findings.
- *Simple and focused outputs* – the outputs of TARA are all easily comprehensible, directly answer a set of sponsor questions and contain a lot of useful information; in some cases the TARA report is the only architectural description information that exists for the system being considered.
- *Concise outputs* – the report for a system tends to be about 5000–8000 words, with 3 or 4 diagrams, so the results are easy to read and comprehend (although this is really the result of the report format rather than the method itself).

Conversely, the weaknesses that the method shows in practice are:

- *Reliance on expert judgement* – all architectural assessment methods rely on expert judgement to some degree, however TARA is very reliant on the knowledge and judgement of the assessor who is performing the assessment. The method does not mandate particular stakeholder input or work to find a consensus between different stakeholder groups (although some parts of the process may result in this, such as the requirements fit analysis). This means that an assessor with strong domain and organisation knowledge, yet perceived as independent, is required and such a person may not always be available.
- *No trade-off analysis* – the method does not explicitly lead the assessor through a consideration of the system's design decisions and the trade-offs inherent in them (although the consideration of system requirements does result in some consideration of this). An assessor can perform trade-off analysis at any point in the process, but the method does not require this.
- *Structure based* – a related point is that while more sophisticated methods like ATAM are really analysing the design process, the design decision options and the tradeoffs that they imply, TARA has a different focus. This method focuses on the architectural structures of the system and it is usually used when at least part of the system already exists, so less effort is spent considering the decisions and tradeoffs inherent in the design, and more effort assessing the architecture that is being implemented and how suitable it is for the requirements it must meet.
- *Relatively shallow* – the simple approach and low resource investment of TARA assessment means that the insight achieved is relatively shallow compared to more sophisticated approaches. The results of a TARA assessment should be treated with some caution and parts of the assessment reconsidered should they appear to lack evidence or be in contradiction with other expert opinion.

Most of these strengths and weaknesses stem from the fundamental simplicity of the method and probably cannot be addressed effectively while still keeping the characteristic simplicity of TARA that is necessary in the situations where it is to be used. Where a

more sophisticated method is needed, and the environment will allow its application, then such methods exist already and do not need to be reinvented.

## 7. Selecting an architectural assessment approach

A noted earlier, Jan Bosch observed that there are four types of architectural assessment, namely scenario based assessment, simulation, mathematical modelling and expert judgement (Bosch, 2000). When considering how to perform these assessments, we used this taxonomy to consider what options were available to help assess the systems concerned.

We quickly came to the conclusion that mathematical and simulation based approaches were not particularly attractive for the kind of assessment that was needed in our situation. This was for a number of reasons. Firstly, the systems already existed and so could be directly tested rather than creating a mathematical or simulation model and testing it. Secondly, we found that the qualities that we wanted to assess (such as functional fit to requirements, production reliability, simplicity of implementation and scalability) were not particularly amenable to mathematical and simulation based analysis at reasonable cost. Finally, even if a modelling or simulation approach had been available, interpreting the results would have still required expert judgement with organisational and domain knowledge.

Having said this, it is important to acknowledge that mathematical models can be created for investigating system reliability or scalability, and simulation models can investigate a number of aspects of behaviour of a proposed system. However, as has been noted by others (Abowd et al., 1997; Mårtensson et al., 2003), building a model of the system that is comprehensive enough to provide reliable results is expensive and experience suggests that such models require a great deal of analysis and tuning in order to produce meaningful results for a particular investigation, so they often do not have a good return on investment in practice. It is also worth noting in passing that a survey of the main computer science literature databases suggested that there are not very many mathematical or simulation based methods that have been thoroughly developed and tested in the research domain, let alone applied widely in practice.

When considering the scenario-based approach, we already knew that scenarios were practical and useful and that there was a large amount of research literature available on the subject of using scenarios for assessment (see the surveys Dobrica and Niemela, 2002; Babar and Gorton, 2004) and few industrial case studies (such as Kettu et al., 2008). However, our experience had also suggested that defining a comprehensive set of high quality scenarios was quite an expensive process, particularly when compared to a more informal, expert-judgement based approach. In this particular situation, an additional concern was that the process of defining scenarios seems quite artificial when done by one person; much of the value of defining scenarios is found in their ability to focus stakeholder attention in important areas and to encourage useful discussion about requirements and priorities. So while we knew that scenarios were a useful approach, it was not obvious that they were worth the investment in this particular case, or that the required stakeholder commitment would be forthcoming.

This left the option of expert judgement, which as explained above, was the approach taken, while structuring the process into the simple steps that became the TARA method. As already discussed, expert judgement has a number of significant limitations, most importantly its reliance on the knowledge, independence and experience of the assessor, and the relatively shallow level of assessment when compared with a more thorough process. However, it is very efficient and easy to justify and some of its weaknesses can be balanced by an awareness of its limitations. In this environment it appeared to be the approach that would be most likely to succeed.

A useful side-effect of considering a number of approaches was a fairly clear set of criteria that could be used to identify the situations in which the different approaches would work. This allows an assessor to judge when it may be prudent to select an expert-judgement based method like TARA rather than a more comprehensive and expensive approach.

Mathematical evaluation methods are likely to be the correct choice when the following conditions hold:

- An important characteristic of interest of the system being evaluated can be clearly and verifiably represented as a mathematical model (e.g. where the concurrency in a system can be analysed using a Petri-net model or the throughput of a system can be estimated as a series of algebraic equations).
- An assessor is available who can confidently create and analyse the mathematical model to draw conclusions from it and communicate them clearly to others. For realistically sized examples, this often implies the availability of a tool that allows the model to be captured and analysed.
- The mathematical model can be created and tested at a cost that is significantly lower than creating and testing a prototype or simply testing the system if it exists.
- In most cases, an existing and proven mathematical modelling formalism needs to exist already and to have been applied successfully to similar problems. (Otherwise the assessment is really a research activity rather than a routine architectural assessment.)

Simulation based evaluation methods can range from symbolic mathematical approaches to implementation prototypes and are likely to be an effective approach when the following conditions hold:

- A high fidelity simulation of the system under assessment can be created in a simulation technology, at a cost that is significantly smaller than creating a skeleton version of the system (or testing the system directly if it already exists).
- The characteristics of interest of the system under assessment can be reliably simulated using the simulation technology chosen. (For example, if memory usage under load is a key quality to be assessed, then a simple functional simulation, while possibly still useful, is unlikely to suffice.)
- The early assessment of the characteristics of interest is sufficiently important to commit significant resources to simulating the system, rather than just developing it incrementally (as simulation is likely to be significantly more expensive than the other assessment approaches).
- An assessor is available who has sufficient skill with the simulation technology to allow a simulation to be created that will allow reliable insights to be gained about the system.
- In nearly every case, simulation requires a proven simulation modelling notation and tool to be available and to be usable by the assessor(s), or for the assessors to be capable of building a high fidelity prototype using standard software development technology.

Scenario based evaluation methods are probably the best understood approach to architectural assessment, both from a research perspective and an industrial practice point-of-view. They have relatively few prerequisites and are likely to be successful when the following fairly simple conditions hold:

- The need for architectural assessment is well understood and a strong stakeholder commitment to the process exists. This will result in stakeholders being willing to be involved in the creation and validation of a set of scenarios that will allow the key qualities of the system to be explored.
- The assessors and stakeholders can commit a significant amount of time in order to achieve a thorough assessment. The amount naturally varies by stakeholder and according to the size of system, but for a 1MLOC information system, an estimate might be 1.5–2 days for non-technical stakeholders and 4–5 days for key technical stakeholders such as architects and production operations managers.
- The assessors have enough experience with the assessment method in use to allow them to apply it fluently and adapt it confidently to the situation at hand.

The success of expert-judgement based evaluation methods is more likely to be context-dependent. Although they are simple techniques, they do have some quite specific conditions that are likely to be required for them to be effective:

- Strong sponsorship and specific (and legitimate) questions are needed from an authoritative sponsor in the organisation, in order to provide the assessor with direction, legitimacy and access to the people and resources they need.
- The assessor needs enough organisational and domain knowledge and inter-personal skill, to allow them to perform the assessment in a self-guided fashion and make sound judgements, without extensive stakeholder input and direction.
- The assessor needs to have a good knowledge of software architecture evaluation techniques so that they can judge which techniques to apply to their specific situation (if any).
- The assessor needs to have a good technical knowledge of the type of system that they are assessing to allow them to interpret the information they gather and draw valid conclusions from it.
- The assessor needs to be given access to the system's implementation its production instance.

In summary, considering the situation in which most industrial software architects need to use architectural assessment, mathematical and simulation-based methods are usually quite expensive options. They will occasionally be useful when a modelling approach is available that neatly matches the problem and the question to be answered, but in many cases their cost and complexity are difficult to justify.

Scenarios will often be the best answer for assessing an industrial software architecture and they have been used successfully in practice. The major caveat with using a formal scenario based method is that significant stakeholder involvement is needed in order to obtain maximum benefit from them and so justify their cost. They are perhaps the most obvious choice when the need for assessment is clear and external consultants are engaged to perform the evaluation, or where an in-house team of assessors is available.

Expert judgement is arguably the weakest form of assessment, but is also the easiest to justify and execute. It requires an explicit understanding of its weaknesses and a focus on both a specific set questions to be answered and drawing conclusions based on gathered facts rather than personal opinions. It will however often be the only practical approach for many industrial architectural evaluations and so understanding how to apply it successfully is an important step in improving industrial practice.

## 8. Further work

While this paper describes an industrial case study and an assessment method that was derived from it, there are a number of possible avenues for further research-oriented work.

Firstly, a review of the literature on software architecture assessment methods suggests that lightweight industrial methods have not been included in the major studies (such as Dobrica and Niemela, 2002). Methods such as LAAAM and TARA and industrial experience such as that related in (Kettu et al., 2008) are a related but separate group of experiences, which could usefully be surveyed and studied in order to relate them to the methods that have emerged from the research environment.

Secondly, while a lot of research has been performed in the area of architectural assessment of software systems, experience suggests that industrial use of its results is relatively rare. Assessing the current state of industrial practice (particularly if performed at a deeper level than a "friends and family" survey) would be a useful input into the research domain.

Finally, a major open question relating to the TARA approach is its repeatability and applicability outside the organisations that we are familiar with. Reports of attempts to apply it by those unconnected to its creators would be a useful validation or nullification of its usefulness.

## 9. Summary and conclusions

This paper set out to do three things: (a) to explain why scenario based assessment methods are not always used in industry; (b) to explain a simple less sophisticated approach which has proved useful as an initial starting point for architectural assessment; and then (c) to illustrate the approach by showing how it had been used on two system assessment exercises and the results that it produced.

While scenario based assessment approaches can produce good results, they can be quite complicated exercises to run, requiring significant amounts of time from a large group of people. The benefits may not be immediately apparent to many of the participants and the sophistication of some methods makes them daunting in some environments.

However, a frequent situation in an industrial context is for an architect to be asked for their opinion as to the "quality" of an existing system and this implies the need for some sort of architectural assessment activity.

In order to allow us to structure architectural assessment exercises where we could not embark on full-blown scenario based methods, we defined the Tiny Architectural Review Approach (TARA) which is a simple method which can be used by a single assessor or a small group of assessors and is not predicated on gaining the attention and large amounts of time from the system's stakeholders.

We have used TARA for a number of assessment exercises, a couple of which form the case study in this paper, and have found it to be an effective approach within its limitations. It has both allowed us to assess systems and report our findings and recommendations in a structured way. It also helped us to gain enough confidence from the sponsoring managers to start conversations about the role of architectural assessment and where it was worth considering committing more effort to it.

The conclusion we have drawn from this experience is that it is beneficial to have simple, less formal options for architectural assessment to compliment the more established and sophisticated approaches. Simple methods are valuable in situations where the focus is an existing system or where the

resources and commitment for a more significant effort cannot be gathered.

## Acknowledgements

I would like to thank the reviewers of the earlier WICSA conference paper, on which this paper is based, and the attendees at the conference, in particular Len Bass, who came to the paper session to listen to the paper presentation, asked interesting questions and made a number of constructive suggestions. I would also like to thank the JSS reviewers who helped to improve the paper considerably by making many useful suggestions. Finally, I am grateful to Dr. Rabih Bashroush who also reviewed the paper and made a number of helpful recommendations on how to improve it.

## References

Abowd, G., Bass, L., Clements, P., Kazman, R., Northrop, L., Zaremsk, A., 1997. Recommended Best Industrial Practice for Software Architecture Evaluation. Technical Report CME/SEI-96-TR-025, Software Engineering Institute.

Babar, M.A., Gorton, I., 2004. Comparison of scenario-based software architecture evaluation methods. In: Asia-Pacific Software Engineering Conference, pp. 600–607.

Barbacci, M., Ellison, R.J., Lattanze, A.J., Stafford, J.A., Weinstock, C.B., Wood, W.G., 2003. Quality Attribute Workshops, Technical Report, CMU/SEI-2003-TR-01, Software Engineering Institute.

Bashroush, R., Spence, I., Kilpatrick, P., Brown, T.J., 2004. Towards an automated evaluation process for software architectures. In: 8th IASTED International Conference on Software Engineering (SE2004), pp. 54–58.

Bengtsson, P., Lassing, N., Bosch, J., van Vliet, H., 2004. Architecture-level modifiability analysis (ALMA). Journal of Systems and Software 69, 1–2.

Bengtsson, P.O., Bosch, J., 1999. Architecture level prediction of software maintenance. In: Proceedings of the Third European Conference on Software Maintenance and Reengineering, pp. 139–147.

Bosch, J., 2000. Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach. Addison-Wesley Professional, Reading.

Carriere S.J., 2009. It's Pronounced Like Lamb Not Like Lame, http://technogility.sjcarriere.com/2009/05/11/its-pronounced-like-lamb-not-like-lame (retrieved 10.04.12).

Clements P.C., 2000. "Active Reviews for Intermediate Designs," Technical Report CMU/SEI-2000-TN-009, Carnegie Mellon University.

Dobrica, L., Niemela, E., 2002. A Survey on software architecture analysis methods. IEEE Transactions on Software Engineering 28 (7), 638–653.

Duenas, J.C., de Oliveira, W.L., de la Puente, J.A., 1998. A software architecture evaluation model. In: Proceedings of the Second International ESPRIT ARES Workshop, pp. 148–157.

Garlan, D., Bachmann, F., Ivers, J., Stafford, J., Bass, L., Clements, P., Merson, P., 2010. Documenting Software Architectures: Views and Beyond, 2nd ed. Addison-Wesley Professional, Upper Saddle River, NJ.

Kazman, R., Abowd, G., Bass, L., Clements, P., 1996. Scenario-based analysis of software architecture. IEEE Software 13 (6), 47–55.

Kazman, R., Klein, M., Barbacci, M., Lipson, H., Longstaff, T., Carriere, S.J., 1998. The architecture tradeoff analysis method. In: Proc. Fourth International Conference on the Engineering of Complex Computer Systems (ICECCS 98), pp. 68–78.

Kettu, T., Kruse, E., Larsson, M., Mustapic, G., 2008. Using architecture analysis to evolve complex industrial systems. Lecture Notes in Computer Science 5135, architecting dependable systems V, pp. 326–341.

Lassing, N., Rijsenbrij, D., van Vliet, H., 1999. On software architecture analysis of flexibility, complexity of changes: size isn't everything. In: Proceedings of the Second Nordic Software Architecture Workshop (NOSA 99), pp. 1103–1581.

Mårtensson, F., Jönsson, P., Bengtsson, P.O., Grahn, H., Mattsson, M., 2003. A case against continuous simulation for software architecture evaluation. In: Applied Simulation and Modelling (ASM2003), pp. 97–105.

Molter, G., 1999. Integrating SAAM in domain-centric and reuse-based development processes. In: Proc. Second Nordic Workshop Software Architecture (NOSA 99), pp. 1103–1581.

Obbink, H., Kruchten, P., Kozaczynski, W., Postema, H., Ran, A., Dominic, L., Kazman, R., Hilliard, R., Tracz, W., Kahane, E., 2002. Software Architecture Review and Assessment (SARA) Report, Version 1.0.

Olumofin, F.G., Misic, V.B., 2005. Extending the ATAM Architecture Evaluation to Product Line Architectures, Technical Report 05/02, Department of Computer Science, University of Manitoba.

Parnas, D.L., Weiss, D.M., 1987. Active design reviews: principles and practices. Journal of Systems and Software 7 (4), 259–265.

Pooley, R.J., Abdullatif, A.A.L., 2010. CPASA: continuous performance assessment of software architecture. In: 17th IEEE International Conference on the Engineering of Computer-Based Systems, pp. 79–87.

Tang, A., Kuo, F.-C., Lau, M.F., 2008. Towards independent software architecture review. In: 2nd European Conference on Software Architecture (ECSA 2008), pp. 306–313.

Zalewski, A., 2007. Beyond ATAM: architecture analysis in the development of large scale software systems. Lecture Notes in Computer Science 4758, software architecture, pp. 92–105.

**Eoin Woods** is lead technology architect for one of the equity business lines at a major international investment bank. His main technical interests are software architecture, distributed systems, computer security, and data management; he is co-author of the well-received book "Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives", published by Addison Wesley. Eoin can be contacted via his web site at www.eoinwoods.info.