Industrial Architectural Assessment using TARA

Eoin Woods
Artechra
Hemel Hempstead, Hertfordshire, UK
e-mail: eoin.woods@artechra.com

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 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 a case study that explains how it was applied to the assessment of two quantitative analysis systems.**

Keywords- software architecture, assessment methods, case study

I. 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 ([1], [2], [3], [7], [9]) and there is evidence of some industrial adoption of the techniques too ([13], [15]).

However, 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 me to conclude that this is for a number of reasons including a perception that these techniques are complicated 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 an abstract architecture rather than examining the system implementation as part of the process.

These experiences have led me to create a simple architectural review method called the Tiny Architectural Review Approach (TARA) that is quick and inexpensive to apply, is less prescriptive than most of the scenario based methods, does not assume that all of the system stakeholders can dedicate time to the process and uses the implementation of the system as one of its major inputs.

This paper explains why an alternative to formal scenario based architectural assessment methods is sometimes needed, defines the steps of the TARA approach, and presents a short case study that illustrates the approach by explaining how it was used for the assessment of two systems.

II. Using Scenario Based Assessment Methods

Most scenario-based assessment methods, such as ATAM [2] and CPASA [9] are thorough and comprehensive approaches that gather the stakeholders of a system and lead them through a structured process of exploration of the architectural decisions that have been made and their implications. They result in a deep understanding of the architecture (or system) under consideration and the strengths and weaknesses that it is 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.

However, the use of formal scenario-based assessment methods in industry is quite rare and my experience of trying to introduce them has led me 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 in gathering the relevant stakeholders to participate is enough to deter many people from embarking on such an exercise.

A secondary reason that people don’t choose methods like ATAM is that they focus on the design of the system and don’t explicitly suggest using implementation artefacts as inputs. This is a reflection of the focus of these methods and is understandable as it allows the methods to be used before system implementation. But many industrial assessments are initiated because of dissatisfaction with a system that is already implemented and in these situations the implementation of the system is an invaluable input into the assessment exercise.

A third reason that is specific to scenario based methods 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 if there isn’t a general understanding and acceptance of the benefits of architectural assessment.

In order to address concerns like these, 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.

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

III. Related Work

There is a large body of research literature on the subject of the 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 1994 [1].

Since then, methods defined by the SEI including ATAM [2], QAW [3] and ARID [4] 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 [5] and ESAMMI [6] that are extensions of SAAM and HoPLAA that is an extension of ATAM.

Other architectural evaluation methods that have independently been proposed include Architecture Level Modifiability Analysis (ALMA) [8], Continuous Performance Assessment of Software Architecture (CPASA) [9] and Architecture Level Prediction of Software Maintenance (ALPSM) [10], these also being scenario-based methods.

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 under pin any effective evaluation technique. However, as Jan Bosch notes in [11] 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 didn't use scenarios was Active Design Reviews [17], 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, 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, however there have been some interesting reports of people who have explored architectural assessment and analysis techniques that are not primarily based on scenarios such as the Software Architecture Evaluation Model (SAEM) [12], an approach based on the Goal/Question/Metric framework [13], and the Independent Software Architecture Review (ISAR) approach [18] 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 [14] but is an effort with similar motivations to ours, though based on a direct tailoring of ATAM and is scenario based and uses quality attribute trees.

Finally, Tommy Kettu and his colleagues discuss how architectural analysis is used at ABB, to support understanding and evolving existing systems [15]. In many ways, the experience reported by these authors is closest to the environment and experiences that inspired the development of TARA.

IV. The TARA Method

The Tiny Architectural Review Approach (TARA) is based on industrial experience in situations where full blown architectural assessment methods aren't suitable for some reason, such as those situations outlined above. These experiences led to the conclusion that a structured and repeatable method was required, which was also quick, flexible and simple to use, requiring a modest investment of time and resources.

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

- The approach isn't based on scenarios because creating valid and meaningful scenarios requires a lot of 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. We 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.
- The method assumes that the system has already been implemented. The method can be used when a system doesn't yet exist, by skipping a step, but where an implementation is available it forms an important input to the process.
- TARA deliberately doesn't mandate specific sub-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.

However the great strength of the method is that it can often be used in situations where it wouldn't be possible to use more involved scenario based techniques. 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.

The approach is structured into seven primary steps as shown in Figure 1 and described in the sections below.

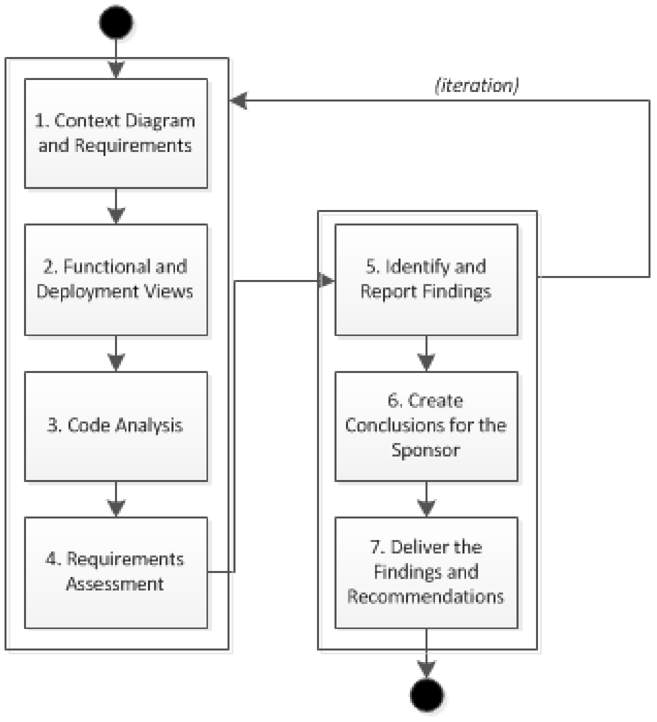
(iteration)
1. Context Diagram and Requirements
2. Functional and Deployment Views
3. Code Analysis
4. Requirements Assessment
5. Identify and Report Findings
6. Create Conclusions for the Sponsor
7. Deliver the Findings and Recommendations

Figure 1. The Steps in the TARA Method

1) Context Diagram 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, but usually gathering this information is part of the assessment exercise.

The system context and key functional requirements are usually fairly straightforward to gather from the development team, the system's key users and even the sponsor who has asked for the assessment (although the differences in the requirements focus between those groups can be illuminating in itself!)

Experience has shown that gathering a good set of system quality requirements is usually significantly more difficult and even the development team will struggle to clearly define the qualities that their system is expected to meet. The best approach in these circumstances is 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). The definition of this set of quality requirements is a useful side effect of the assessment process.

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 [16] 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) and deployment structure (the environment that the runtime elements are deployed into).

Some of this information usually exists 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 its meaning clear, rather than necessarily a completed model). Any suitable notation can be used for the architectural sketches, but we have generally used UML and found it to work well.

Having completed the process of creating the functional and deployment views, the fundamental element structure and mechanisms of the system should now be clear and provide a good basis for the rest of the assessment process.

3) Code Analysis

The creation of the context diagram, identification of requirements and the creation of the functional and deployment views are all relatively subjective activities, relying on expert judgment rather than simply recovering facts. The next step in the process analyses the system code in order to provide some more objective knowledge into the exercise. As the old saying goes "the code doesn't lie".

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 system implemented entirely in a byte-code compiled language (like Java), which separates tests from production code, is well structured, follows conventions and where some powerful analysis tools are available will be much easier to analyse than a situation where a system is written in Perl, that has followed few conventions and where a good analysis tool isn't 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 and open source tools, are easily explained and in my experience 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.

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.

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 built. The next stage is to perform an assessment of the ability of the system to meets its functional and system quality requirements.

Given the deliberate simplicity of the TARA method, this step in the process is inevitably one of judgment rather than quantifiable assessment. The ability of the system to meet its requirements can't be tested during an exercise such as this but must be assessed by expert judgment. That said, where the system has been implemented and metrics (e.g. for throughput or outages) are available then these should be used as an input to the process.

The functional requirements capabilities of the system are easier to assess than the system's qualities and the match between the capabilities and the requirements can usually be assessed using a combination of the assessor's domain knowledge and canvassing the opinions 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 and then count the number of those functions that are provided by the system. This is a point in the process where that some use of scenarios can be valuable and they should be considered by the assessor, even if not discussed explicitly with the stakeholders.

Assessing the quality property requirements is more difficult as it may well not be possible to test the system's ability to meet them and 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 relative lack of precision and certainty that tends to characterise the quality property requirements that the assessor needs to work with also makes this 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 decide on the non-functional abilities of the system. But this is also the step in the process where established techniques like scenarios, quality attribute trees, queuing models and so on can be used as the assessor sees fit. The method deliberately does not mandate their use, but doesn't 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 which 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 and 1-5).

The last three steps of the process are to "Identify and Report Findings", to "Create Conclusions for the Sponsor" and to "Deliver the Findings and Recommendations" of the assessment. These steps are common to all assessment approaches so in the interests of brevity, they are just outlined below.

5) Identify and Report Findings

Throughout the assessment activities, the assessor will have been drawing conclusions about the qualities of the system under consideration and these insights are valuable outputs of the assessment activity.

The findings need to be reported tactfully, in a well-organised report that stresses positive aspects of the system as well as potential problems. We organise the findings into logical concern-oriented groups, with each finding being clearly described with a short meaningful name, an identifier, a full description and a justification or reference to further information to support the finding.

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

6) Create Conclusions for the Sponsor

This step of the process adds a "Conclusions" section to identify the explicit or implicit questions being asked by the sponsor who commissioned the assessment and present other recommendations that are required to support them. While this may be little more than restating findings reported elsewhere, this section allows the information to be stated in a way that directly addresses the concerns of the sponsor.

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 process often involves meetings and presentations as well as circulation of the written report.

V. 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.

A. System 1 Assessment

The TARA method was initially developed in response to a request to provide an assessment of a quantitative analysis system that had been developed in-house by a major financial fund manager. The system had been developed within a business unit (largely outside the purview of people who viewed themselves as responsible for such systems) 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 they were going to sponsor its ongoing development (in the face of some opposition).

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 hadn't 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 subjective sections) and the first TARA review was performed by following the activities needed to complete the document.

The result of the exercise was a completed system assessment report, containing the sections outlined earlier in this paper. The conclusions of the assessment were largely positive, although there were quite a number of technical recommendations.

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 Figure 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 I 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 (which have not been rewritten for this paper, only reworded to remove organisation specific terminology), it is interesting to note how the two functional requirements (FR1 and FR2) are stated in rather 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 less than ideal as the assessor's judgement might not have been correct, however we have found that once non-functional requirements are stated, glaring errors or invalid assumptions are often pointed out by the key stakeholders, so this wasn't 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 Figure 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 II contains an illustrative sample of the quantitative metrics which 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 III.

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 and, somewhat to our surprise, the development team 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 stemmed from the fact that it directly answered the questions he had posed (rather than being a generic architectural assessment, of the sort he had seem before) and it was organised in a way that clearly described the system and supported all of its findings with evidence (e.g. metrics) or clear reasoning (e.g. the logic behind expert judgement). This meant that the report wasn't particularly contentious, was easy to get people to read (as it contained a lot of useful information) 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 were now understood too and were perceived to be rectifiable.

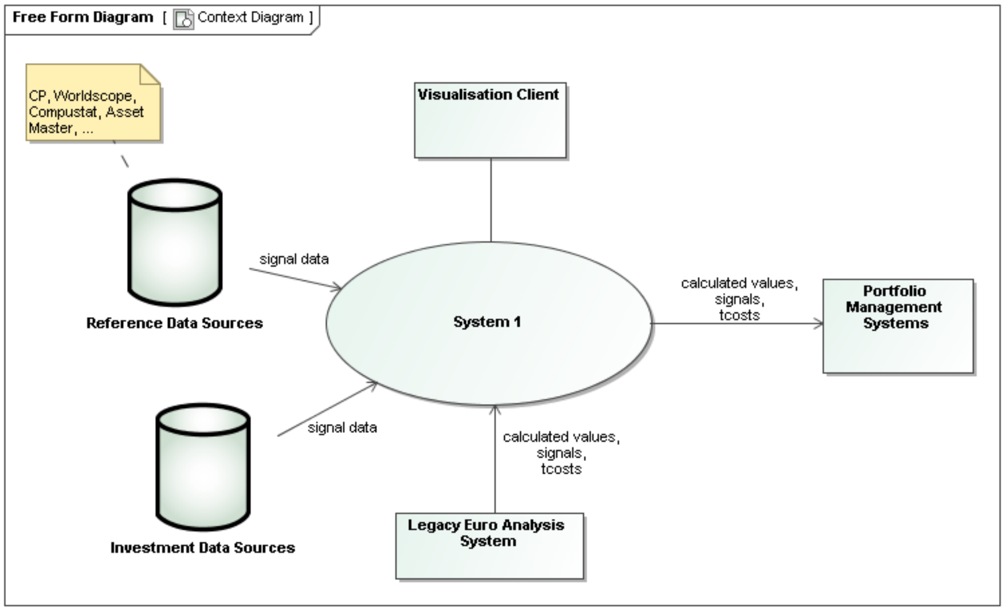
Free Form Diagram [Context Diagram]
CP, Worldscope, Compustat, Asset Master, ...
Visualisation Client
Reference Data Sources
signal data
System 1
calculated values, signals, tcosts
Portfolio Management Systems
signal data
calculated values, signals, tcosts
Investment Data Sources
Legacy Euro Analysis System

Figure 2. Context Diagram for System 1

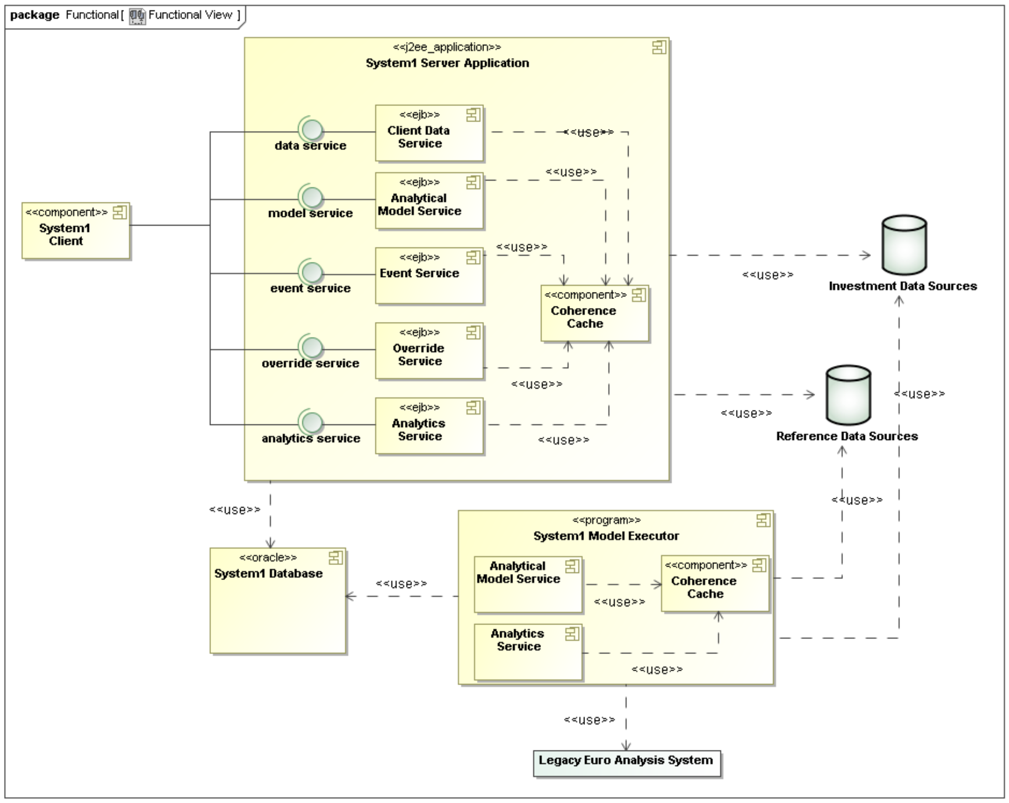
package Functional [Functional View]
<<j2ee_application>> System1 Server Application
<<ejb>> Client Data Service
<<ejb>> Analytical Model Service
<<ejb>> Event Service
<<ejb>> Override Service
<<ejb>> Analytics Service
data service
model service
event service
override service
analytics service
<<component>> System1 Client
<<component>> Coherence Cache
<<use>>
Investment Data Sources
Reference Data Sources
<<oracle>> System1 Database
<<program>> System1 Model Executor
Analytical Model Service
Analytics Service
<<component>> Coherence Cache
Legacy Euro Analysis System

Figure 3. Functional View Sketch for System 1

TABLE I. 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 minutes in the system, but the target time for this is about 10 minutes. The other important performance requirement is the implicit requirement for the user interface to be usably fast (defined by the organisation to mean never freezing, responding instantaneously to local UI events and new data being available within 10 seconds of a request).
NFR2	Scalability – the key scalability requirement is likely to be maintaining the bound on the quantative 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 II. 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).

TABLE III. 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 could do with some review. 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.

B. 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 maintainability than in the previous exercise.

Predictably, this assessment produced similar outputs to the assessment of System 1, but to better illustrate the process, we present some alternative outputs to the ones shown in the previous section.

Figure 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 loosely follows a "pipe and filter" architectural style and so a data flow view was very relevant for capturing some of the important relationships within the system and was produced as part of the assessment.

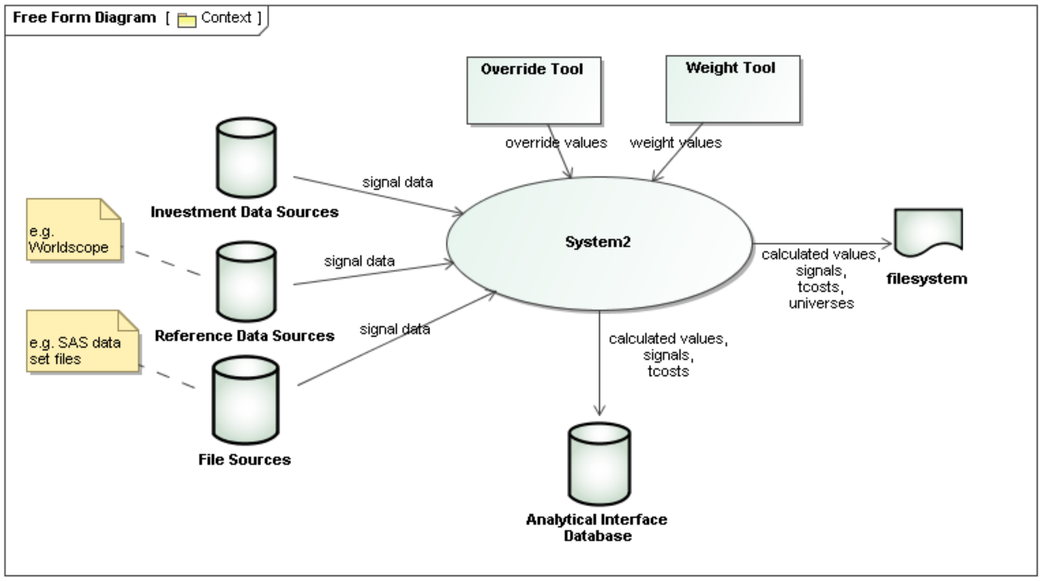

Figure 4. Context Diagram for System 2

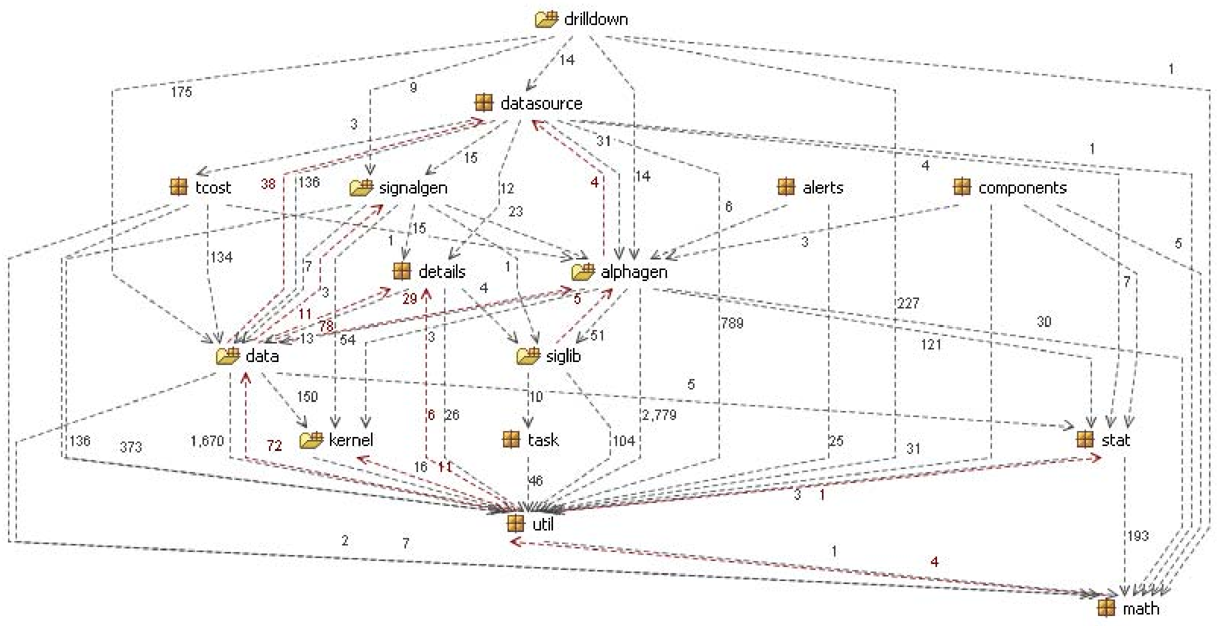

Figure 5. Module Dependencies for System 2

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. Given the age of System 2 and the concerns about maintainability, this analysis was particularly relevant for this system and the result of the analysis can be seen in the dependency diagram in Figure 5. This analysis highlighted the fact that the module structure of System 2 is complicated with many cycles in the dependency graph and this finding was a valuable output of the exercise.

Examples of the commentary as to how well System 2 met its requirements are shown in Table IV.

TABLE IV. 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 doesn't 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.

A couple of examples of the findings that were reported for the assessment of System 2 are shown in Table V.

TABLE V. 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 doesn't appear to follow any particularly strong coding or design conventions and while this obviously doesn't affect how the software runs, it does make it more difficult to understand, extend and maintain.

This assessment was also received very positively by the sponsor, and reasonably positively by the development team (even though some of the findings were more critical than in the first case and had to be expressed tactfully). The fact that the findings were factual, fair and backed up by firm evidence (rather than simply being opinions) all helped with the acceptance of the results.

VI. EVALUATION OF THE APPROACH

The TARA method has now been used successfully to assess a number of systems and it has been very successful in use. The sample size is small (and only relates to one organisation) but so far the method has proved to be useful. 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 initially look like "common sense". TARA addresses this by using a very low ceremony approach that is easy to explain and deliberately doesn't 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.
- Speed – it cannot be overstated that the ability to explain what you're going to do in 10 minutes and do it in 2 or 3 days, write it up in another day and deliver the results in a couple of hours overcomes many objections to architectural assessment and often buys enough credibility to allow the idea of more comprehensive assessments to be discussed.
- Simple and Widely Useful 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:

- Expert Subjectivity – use of the method is very reliant on the knowledge and judgement of the assessor who is performing the assessment. The method doesn't explicitly gather stakeholder input and find a consensus between different stakeholder groups (although some parts of the process may result in this, such as the requirements fit analysis).
- No Trade-Off Analysis – the method doesn't 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 doesn't require this or explain how to do it.
- Structure Based – a related point is that while more sophisticated methods like ATAM are really analysing the design process, the decisions made and their tradeoffs as much as the system itself, TARA doesn't do this. The focus of TARA is on the architectural structures of the system and it is usually used when the system already exists, so less effort is spent considering the decisions and tradeoffs

inherent in the design, and more effort assessing what is there and recommending how to change it.

- 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 can't 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.

VII. 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 has also helped us to gain enough confidence from the sponsoring managers to start conversations about the role of architectural assessment and where it may be 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 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.

VIII. References

[1] R. Kazman, G. Abowd, L. Bass, and P. Clements, Scenario-Based Analysis of Software Architecture, IEEE Software, pp. 47-55, Nov. 1996.

[2] R. Kazman, M. Klein, M. Barbacci, H. Lipson, T. Longstaff, and S.J. Carriere, The Architecture Tradeoff Analysis Method, Proc. Fourth Int'l Conf. Eng. of Complex Computer Systems (ICECCS '98), Aug. 1998.

[3] M. Barbacci et al, Quality Attribute Workshops, technical report, CMU/SEI-2003-TR-01, Software Engineering Institute, August 2003.

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

[5] N. Lassing, D. Rijsenbrij, and H. van Vliet, On Software Architecture Analysis of Flexibility, Complexity of Changes: Size Isn't Everything, Proc. Second Nordic Software Architecture Workshop (NOSA '99), pp. 1103-1581, 1999.

[6] G. Molter, Integrating SAAM in Domain-Centric and Reuse- Based Development Processes,o Proc. Second Nordic Workshop Software Architecture (NOSA '99), pp. 1103-1581, 1999.

[7] F.G. Olumofin and V.B. Misic, Extending the ATAM Architecture Evaluation to Product Line Architectures, TR 05/02, Department of Computer Science, University of Manitoba, June 2005.

[8] P. Bengtsson et al., Architecture-Level Modifiability Analysis (ALMA), Journal of Systems and Software, 2004. 69(1-2).

[9] R. J. Pooley, A. A. L. Abdullatif, "CPASA: Continuous Performance Assessment of Software Architecture," Engineering of Computer-Based Systems, IEEE International Conference on the, pp. 79-87, 2010 17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, 2010.

[10] P.O. Bengtsson and J. Bosch, Architecture Level Prediction of Software Maintenance, Proc. Third European Conf. Software Maintenance and Reeng., pp. 139-147, Mar. 1999.

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

[12] J.C. Duenas, W.L. de Oliveira, and J.A. de la Puente, A Software Architecture Evaluation Model, Proc. Second Int'l ESPRIT ARES Workshop, pp. 148-157, Feb. 1998.

[13] A. Zalewski, Beyond ATAM: Architecture Analysis in the Development of Large Scale Software Systems, Lecture Notes in Computer Science, 2007, Volume 4758, Software Architecture, Pages 92-105.

[14] S.J. Carriere, Lightweight Architecture Alternative Assessment Method, no paper or technical report exists yet so the best reference is http://technogility.sjcarriere.com/2009/05/11/its-pronounced-like-lamb-not-like-lame.

[15] T. Kettu, E. Kruse, M. Larsson and G. Mustapic, Using Architecture Analysis to Evolve Complex Industrial Systems, Lecture Notes in Computer Science, 2008, Volume 5135, Architecting Dependable Systems V, Pages 326-341.

[16] D. Garlan, F. Bachmann, J. Ivers, J. Stafford, L. Bass, P. Clements, and P. Merson. Documenting Software Architectures: Views and Beyond (2nd ed.). 2010. Addison-Wesley Professional

[17] D.L. Parnas and D.M. Weiss, Active Design Reviews: Principles and Practices, Journal of Systems and Software, vol. 7, no. 4, 1987, pp 259-265.

[18] A. Tang, F.-C. Kuo and M.F. Lau Towards Independent Software Architecture Review, in 2nd European Conference on Software Architecture (ECSA 2008), 2008, pp. 306-313.

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