A Concept and a Multitenant Web Application for Interactive Software Architecture Analysis

Stefan Gudenkauf[1][0000-0002-1813-3448], Uwe Bachmann[1] and Niklas Hartmann[1]

[1] Jade University of Applied Sciences, 26389 Wilhelmshaven, Germany
stefan.gudenkauf@jade-hs.de, uwe.bachmann@jade-hs.de

Abstract. The Architecture Tradeoff Analysis Method (ATAM) is a well-known method for the early evaluation of software architecture decisions based on the formulation of quality scenarios. Although widely recognized as beneficial, several limitations of ATAM have become apparent over time that limit its applicability. In this paper, we propose the concept of an interactive web-based application that mitigates some of these limitations while opening the possibility for further analysis based on collected ATAM projects, e.g. based on artificial intelligence. Thereby, we address the following limitations of ATAM: a general need for tool support, the need for an overview of ATAM results, the need for a consistent documentation of an ATAM process, the need for interactive and collaborative process execution among stakeholders, and the benefits of supporting multitenancy for architecture analysis. This applies in particular to systems with microservice architecture in order to document a large number of individual components and to make the documentation accessible to everyone involved in the system. The proposed concept, *Interactive Software Architecture Analysis* (ISAA), consists of a requirements analysis, an analysis model, a concept for interactive visualization, and a set of use cases. We demonstrate the feasibility of our approach by a software prototype in the form of portable modules for the Drupal Content Management System.

Keywords: Software architecture, architecture evaluation, tools to support architecture.

1 Introduction

The Architecture Tradeoff Analysis Method (ATAM) is a well-known method for the early evaluation of software architectures [1, 2]. Although widely recognized as beneficial [9], several limitations of how the ATAM is applied have become apparent over time that limit its applicability. For example, a general need for tool support [3, 4 ,5], the need for an overview of ATAM results [3], the need for a consistent documentation of an ATAM process [5], the need for interactive and collaborative process execution among stakeholders [4, 6, 7, 8], and the benefits of supporting multitenancy for architecture analysis, as we discuss later.

In this paper, we argue that adequate tool support for software architecture analysis can mitigate these limitations. To do so, we analyze top-level requirements and propose a concept for *interactive software architecture analysis* (ISSA). We discuss ISAA in terms of an analysis model, a concept for interactive visualization, and a set of use cases that tool implementers can consider as functional requirements. While the focus of the paper lies on this concept, we also demonstrate the feasibility of our approach by software prototyping.

In Section 2, we present an overview of the ATAM, followed by an overview of related work in Section 3. Section 4 then looks at the limitations of the ATAM and distills top-level requirements. To meet these, Section 5 presents a concept dubbed *Interactive Software Architecture Analysis* (ISAA). In Section 6, we demonstrate the feasibility of our approach by developing a software prototype based on the Drupal Content Management System (CMS). Finally, we discuss future work and draw conclusions in Section 7.

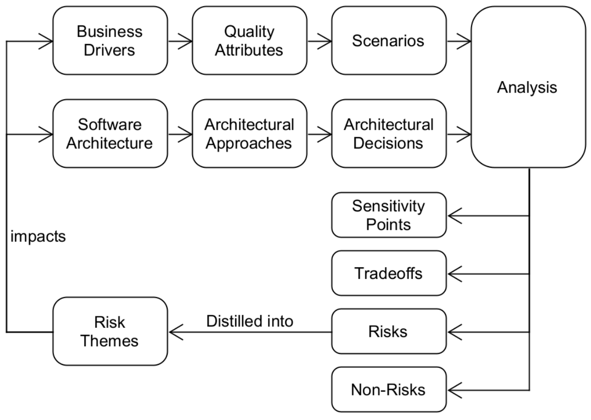


Fig. 1. Overview of the elements of the ATAM process.

2 The Architecture Tradeoff Analysis Method

The Architecture Tradeoff Analysis Method (ATAM) is a method for the early evaluation of software architectures developed at the Software Engineering Institute (SEI) of Carnegie Mellon University [1, 2]. The method relies on the formulation of quality scenarios, which stakeholders shall classify according to their domain importance and technical difficulty. This is supplemented by the presentation of the software architecture, the architectural approaches and decisions made on it. Finally, the stakeholders shall classify the effects of the architecture decisions according to a fixed scheme and form so-called risk themes, which are said to affect the software architecture as well as the original business drivers that led to the development of the software in the first place. Figure 1 shows an overview of the elements of the ATAM process and Table 1 subsumes the individual process steps of ATAM, which are executed in two phases: In the first phase, steps 1-6 are performed in a small meeting, then after a break, steps 1-9 are performed in a larger setting in phase 2.

The application of ATAM promises several benefits for software development [9]. The method focuses on identifying risks early in the software development life cycle. Quality Attributes are made explicit, and architecture documentation is promoted. This fosters the communication among stakeholders, and forms a documented basis for stakeholders to make well-founded architectural decisions. The main benefit therefore is improved software architectures.

Table 1. Steps of the ATAM process according to [1].

Step	Activity	Stakeholders
1	Present the ATAM	Evaluation team, customer representatives, architecture team
2	Present business drivers	See above
3	Present architecture	See above
4	Identify architectural approaches	See above
5	Generate Quality Attribute Utility Tree	See above
6	Analyze architectural approaches	See above
7	Brainstorm and prioritize scenarios	All stakeholders
8	Analyze architectural approaches	Evaluation team, customer representatives, architecture team
9	Present results	All stakeholders

3 Related Work

Over time, researchers published various articles on ATAM. We consider the following categories: (1) articles on the application of ATAM, (2) articles on improving or adapting the ATAM process, and (3) articles on ATAM tool support. In the following, we limit ourselves to selected work from the last two categories.

Zalewski argues that ATAM is difficult to apply to the early evaluation of large-scale software architectures [10]. This is because top-level architectural decisions, so called *system organization patterns*, are difficult to relate to the final characteristics of a software system, and because the evaluation of stakeholder expectations would require a sufficiently detailed design. As a framework for evaluating system organization patterns, Zalewski proposes the use of the Goal Question Metric (GQM) approach [11], [12] instead of ATAM. We find the notion of system organization patterns and their aspects as potentially helpful for the documentation of architectural decisions of large-scale software systems within an ATAM process. We also find the goal definition of the GQM as particularly helpful for the definition of the overall business goal related to the business drivers in the ATAM process.

Lionberger and Zhang present ATAM Assistant [5], an application that supports the ATAM workflow focusing on data collection and organization at each phase of the ATAM process. Data collection is realized by windows form based interaction (dialogs and a multiple document interface). Result artifacts, such as the quality attribute utility tree, are presented in tree view structures. At the end of the process, the user can generate a report in HTML format. The author denote the goal of the tool to reduce software architecture evaluation to one person.

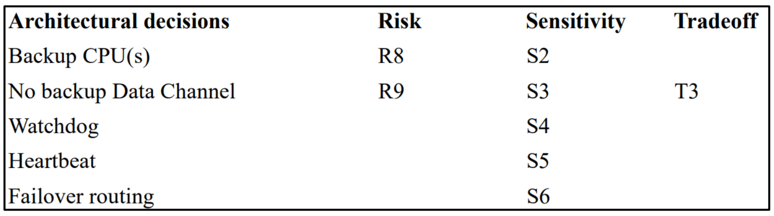
In [3], Gabel notes that at the time of writing, there is a lack of tool support for documenting the results of an ATAM process. He notes that literature often shows the application of ATAM only with spreadsheet-based reports, and that the mapping of risks, tradeoffs, and sensitivity points to the appropriate architectural decisions is often not directly apparent to the stakeholders. As a solution, Gabel proposes the application of software cartography [19], and implemented a single user desktop application to document and report the application of ATAM as a proof of feasibility.

Maheshwari and Teoh analyze the social and environmental aspects of the ATAM process [4]. To address these aspects, they envision a collaborative web-based tool to allow stakeholders to participate in ATAM processes without having to be physically collocated.

The authors Ågren et al. investigate how architecture evaluation can provide useful feedback during the development of continuously evolving systems and show the importance of dedicated feedback loops during the development of such systems [13].

4 Requirements Analysis

As part of our research for related work, we found that tool support for performing ATAM processes is still scarce. The first requirement is therefore a basic tool support to simplify the execution of the ATAM process for stakeholders (req. 1). This tool support should uniformly document the execution of ATAM processes, so that stakeholders can compare and computers can process individual ATAM processes (req. 2). This opens up the possibility of analyzing architecture variants, and machine learning from process executions that have already taken place, for example, to make suggestions for architectural decisions based on existing business drivers and quality scenarios. Further requirements lie in the documentation of ATAM results. We agree with Gabel [3] that the relationships between risks, tradeoffs and sensitivity points should be presented to the stakeholders in a way that makes them as obvious as possible (req. 3). In this way, stakeholders do not have to resolve these references between tables within reports "in their minds" (see Figure 2), but can draw direct conclusions from the relationships and can focus on forming risk themes.



Architectural decisions	**Risk**	**Sensitivity**	**Tradeoff**
Backup CPU(s)	R8	S2	
No backup Data Channel	R9	S3	T3
Watchdog		S4	
Heartbeat		S5	
Failover routing		S6	

Fig. 2. Screenshot from Kazman et al [1] showing a part of an architectural approach documentation. Note that the reader must mentally resolve the references of architectural decisions to risks, sensitivity points and tradeoffs.

However, in contrast to [3], we consider an interactive visualization [6, 7, 8] of the results of an ATAM process to be more useful than generating static reports (req. 4). The reason is that a continuous interactive visualization of ATAM results in different iteration stages enables direct feedback loops (cf. [13]) and agile procedures. Additionally, as in [4], stakeholders should be able to participate in ATAM processes without having to be physically collocated, requiring remote work and possibly collaborative editing (req. 5). Finally, we consider multi-tenancy to be an important property of an ATAM tool (req. 6). The following considerations are paramount for this:

1. Cost savings: We only have to provide customers ATAM services using a single tool instance.
2. Leveraging data: Data from all ATAM processes is available in a uniform digital format and will be available for data mining and machine learning in the future.

Table 2 subsumes the identified top-level requirements for interactive software architecture analysis.

Table 2. Top-level requirements for interactive software architecture analysis.

No.	Requirements
1	Tool support
2	Consistent and digital process documentation
3	Overview on analysis results
4	Interactive visualization
5	Remote work support
6	Support for multitenancy

5 Concept

As stated in the previous section, scenario-based architecture analysis with ATAM faces several limitations. To address these, we propose the concept of *interactive software architecture analysis* (ISAA) in terms of an ISAA metamodel, a concept for interactive visualization, and an overview of the main use cases for tool support.

5.1 A Model for Interactive Software Architecture Analysis

The ISAA model consists on the following parts: (1) elements that represent the Quality Attribute Untility Tree, (2) elements that represent architectural approaches and architectural decisions, and (3) elements that represent sensitivity points, tradeoffs, non-risks and risks, and their classification into so-called risk themes. Figure 3 shows the ISAA model in form of an analysis class diagram using the Unified Modeling Language.

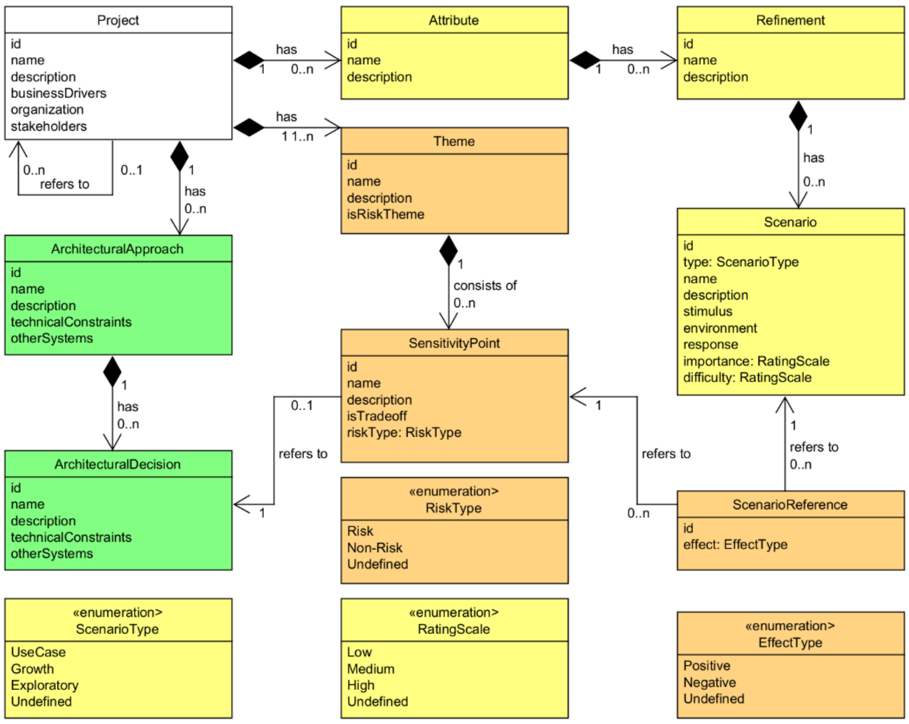


Fig. 3. The ISAA metamodel. Entities that relate to the Quality Attribute Utility Tree are colored yellow, entities that relate to software architecture are colored green, and elements that relate to sensitivity points, treadeoffs, non-risks, and risks are colored beige.

Since the individual entities refer directly to elements of the ATAM, we refrain from a detailed discussion of all entities. However, we would like to present the following rationale for the model:

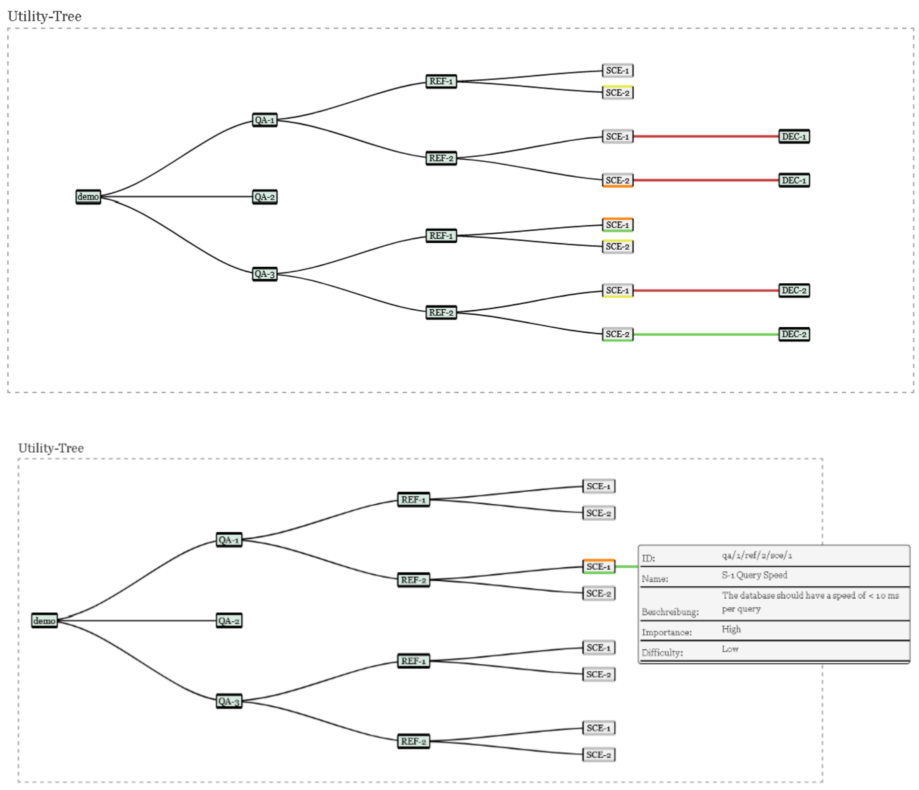
1. We consider the three parts of the ISSA model as tree structures, so that later serialization in data exchange formats should be simple. UML composition associations and one-to-many multiplicities between model entities form the respective tree structures.
2. The tree structure of sensitivity points, tradeoffs, risks and non-risks requires that they are initially classified in an anonymous risk theme. During the ATAM process, the stakeholders then assign them to dedicated risk themes.
3. We interpret the explanations of sensitivity points, tradeoffs, risks and non-risks given in [1] and [14] as follows:
   a. A sensitivity point represents the most basic relationship between an architectural decision and a quality scenario.
   b. We automatically denote a sensitivity point as a tradeoff if the architecture decision influences several quality scenarios. This is represented by the Boolean flag `isTradeoff`.

   c. Stakeholders can explicitly classify a sensitivity point or tradeoff as a risk or non-risk. This is represented by the enumeration `RiskType`.
4. Since the impact of sensitivity points, tradeoffs, risks and non-risks on quality scenarios can potentially be regarded as positive or negative, we added a separate entity `ScenarioReference` to represent these impacts using an enumeration `EffectType`.
5. A Quality Attribute Untility Tree is represented by instances of `Attribute`, `Refinement` and `Scenario` for a given `Project` instance. For attributes and refinements, we recommend using well-known characteristics such as these provided by the product quality model of the ISO/IEC 25010:2011(E), see [15].
6. As instances of the entity `ArchitecturalApproach` may represent the *system organization patterns* of large-scale software systems as discussed in [10], we consider applying the topics proposed in [10] when describing such systems: Subsystem decomposition, geographical and/or organizational allocation of subsystems, data input organization, data storage distribution, data processing organization, distributed data storage management, transaction management, and communication framework.
7. Business drivers that lead to the development of a software to be evaluated by ATAM should be documented in the `businessDrivers` attribute of instances of `Project`. The `description` attribute may additionally be used to document business goals formulated according to the GQM approach [11].
8. The entity `Project` represents the root node of an evaluation project. To support the analysis of the architectures of large software systems with many components, a project can refer to other projects as *related*. For example, the evaluation of a single microservices component of a microservices application [21, 22] may refer to the evaluation of the overall macro architecture of the same application.
9. We considered the individual types of quality scenarios as an enumeration `ScenarioType` used by `Scenario`.
10. Each enumeration allows `Undefined` as a value for properties not (yet) defined by the stakeholders.

5.2 Interactive Software Architecture Analysis Visualization

"Dynamic, interactive visualizations can empower people to explore the data for themselves.", Murray aptly writes in [6]. Such visualization for ATAM aims that stakeholder can track and examine the results of an ATAM process as it is carried out. In general, cognitive load should be kept as small as possible. To do so, a suitable tool support must expose the references drawn between process elements and make them explorable to the viewer. Our visualization concept consists of several views, which we discuss in terms of how they support interactivity and which benefits we see in them.

Quality Attribute Utility Tree View. This view shows an overview of the Quality Attribute Utility Tree of the ATAM, including quality attributes, refinements, scenarios, and how they are connected. The view shows a typical tree structure that was already used in [1] to statically illustrate a Quality Attribute Utility Tree. However, we consider the project itself as the root of the tree. Quality attributes, refinements and scenarios then form the rest of the nodes. Figure 4 shows an example.



ID:	qa/1/ref/2/sce/1
Name:	S-1 Query Speed
Beschreibung:	The database should have a speed of < 10 ms per query
Importance:	High
Difficulty:	Low

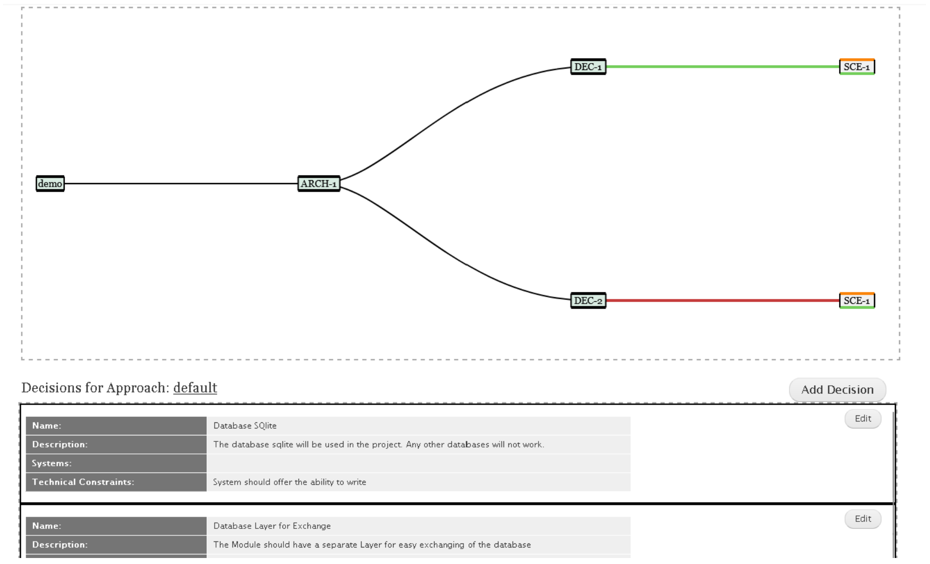
Fig. 4. Example of a Quality Attribute Utility Tree View from the ISAA prototype.

Once architectural decisions on sensitivity points, trade-offs, risks, and non-risks have been identified during the process, these architectural decisions should also be explicitly displayed in the Quality Attribute Utility Tree View, associated with the scenarios they affect.

For the sake of clarity, we suggest that the nodes should only display the name of the respective ATAM element. However, we use the following color scheme to encode further information: Scenario nodes can have colored borders above and below their respective node element's name. These indicate the domain importance (above) and technical difficulty (below) of each scenario once the stakeholders have rated it. A green border represents *low*, yellow represents *medium*, and red/orange represents *high*. In addition, associations between architectural decisions and the scenarios they affect are colored according to the type of impact, if specified. A red color indicates a negative impact and a green color indicates a positive one.

Mouse hover actions should be used to present tooltips with tabular data tailored to the respective node element. More information about each node is made available by clicking on the respective node. This approach adheres to the “Visual Information-Seeking Mantra” *overview first, zoom and filter, then details-on-demand* [7]. It provides a broad overview of how scenarios relate to quality attributes first, which the viewer can supplement with more in-depth information as required.

Architectural Decisions Tree View. The Architectural Decisions Tree View is similar to the Quality Attribute Utility Tree View. Architectural approaches and architectural decisions are presented. If stakeholders already have associated these with scenarios via sensitivity points, tradeoffs, risks and non-risks, the scenarios are also displayed in this view. As in the Quality Attribute Utility Tree View, associations between architectural decisions and the scenarios they affect are colored according to the type of impact, if specified. An example is shown in Figure 5.



Decisions for Approach: default

Name:	Database SQlite
Description:	The database sqlite will be used in the project. Any other databases will not work.
Systems:	
Technical Constraints:	System should offer the ability to write

Name:	Database Layer for Exchange
Description:	The Module should have a separate Layer for easy exchanging of the database

Fig. 5. Example of an Architectural Decisions Tree View from the ISAA prototype.

Sensitivity Impact Network View. This view focuses on Sensitivity Points. Instead of a tree structure, we considered the view to be a force-directed graph, see Figure 6. The central element of the view is the ATAM project as root. The Quality Attribute Utility Tree is placed around the project, with the position of each element being calculated by a force-based algorithm. Architectural decisions that the stakeholders have not yet associated with scenarios float as free nodes.

In the Sensitivity Impact Network View, stakeholders can interactively specify the relationships of sensitivity points to architectural decisions and scenarios (e.g. by context menu actions or through drag-and-drop mouse gestures). Stakeholders can also assess the respective impacts as positive or negative. The connections evaluated in this way are colored green or red accordingly. Each new connection then updates the graph's representation using the force-based algorithm. Stakeholders can also identify sensitivity points as risks or non-risks. Risks are then colored red and non-risks green.

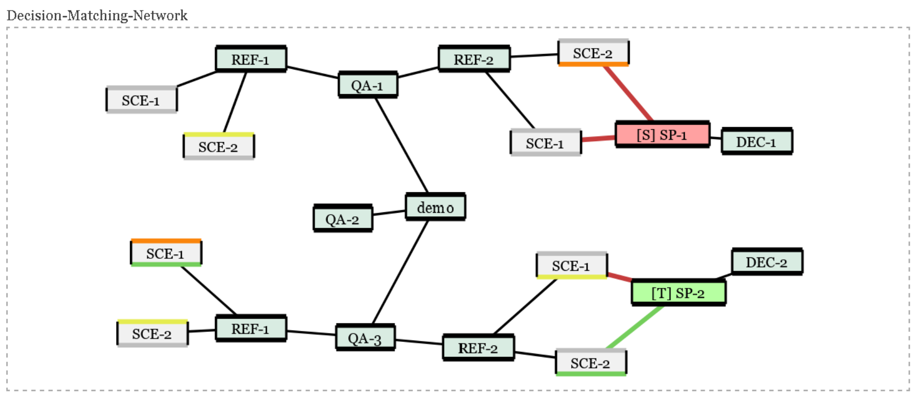


Fig. 6. Example of a Sensitivity Impact Network View from the ISAA prototype.

Whether a sensitivity point is a tradeoff is automatically determined based on the number of connections to scenarios and the effect types of the connections. With more than one connection, a sensitivity point is considered a tradeoff when different effect types come together (at least one positive and one negative effect). The sensitivity point can then be marked with a small *T* symbol. Scenarios are highlighted as in the other two views.

5.3 Use cases for tool support

We consider the following top-level use cases for *interactive software architecture analysis*, arranged by topic (Figure 7). Implementers can consider these as functional requirements for ISSA tool support.

User management. Users must be able to register themselves and to authorize themselves once registered. We currently consider the following users: *Guest* users can register themselves. Once registered, *stakeholders* have view rights to the project assigned to them. *Project owners* have full rights to their projects, and may reference other projects, in which they are involved as stakeholders, as *related* (see Section 5.1, item 8). They also can assign registered users as stakeholders to their projects. *Domain super users* have view rights to all projects, and *admins* have full rights to all projects and can assign roles to users.

Project management. Users must be able to manage their ATAM projects, which includes creating a new project, opening and updating an existing project, and deleting a project.

Requirements elicitation. Within a project, users must be able to construct a Quality Attribute Utility Tree including attributes, refinements and scenarios. They also must be able to evaluate a given scenario according to its domain importance and its technical difficulty.

Architecture analysis. Users must be able to define architectural approaches and decisions. Also, the must be able to define and classify sensitivity points, as well as to define risk themes.

Result presentation. The system must provide visual representations of ATAM results according to the interactive visualization concept described in the previous subsection. In addition, while not in the focus of the concept, report generation should also be possible (at least by using a web browser's print function).

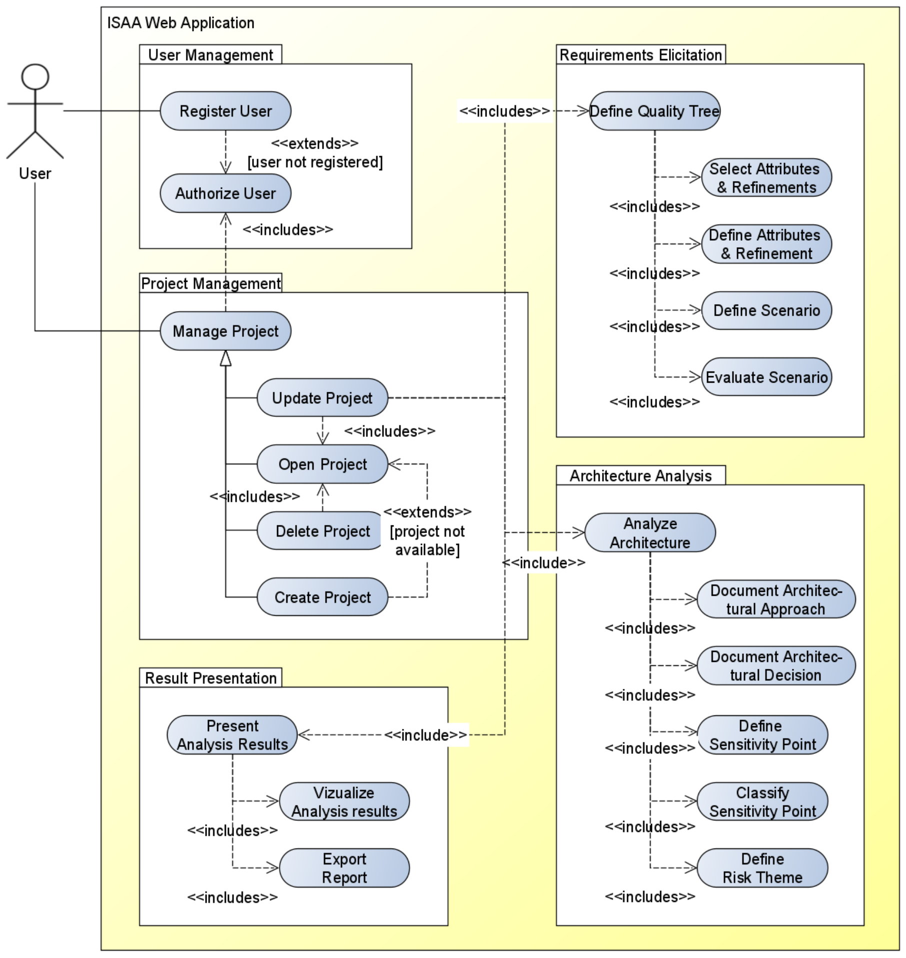


Fig. 7. Use cases of the Application.

6 Implementation

As a proof of feasibility, we implemented a software prototype for *interactive software architecture analysis* (ISSA) realized as web application based on the widely-used Linux, Aache, MySQL, and PHP (LAMP) application stack. In addition, we chose the Drupal content management system [16] as a basis for development for the following reasons [17]: Drupal is built upon the PHP web development language. It can use several database systems for persistent data storage, such as MySQL. Drupal also has a large developer community. Finally, as a distributed application framework, it is regarded highly scalable, provides several basic needs of distributed business applications such as user and role management, and is expandable with additional functionalities via a well-established module concept. Therefore, we realized the ISAA web application as a set of custom Drupal modules that handle ATAM process data input and result visualization according to the concept discussed in the previous section. The module architecture of the ISAA web application is presented in Figure 8. The focal point of the modules is the notion of an evaluation project that the underlying database system persists and that is serialized as a JSON file. The custom ISAA Drupal modules consists of the following components:

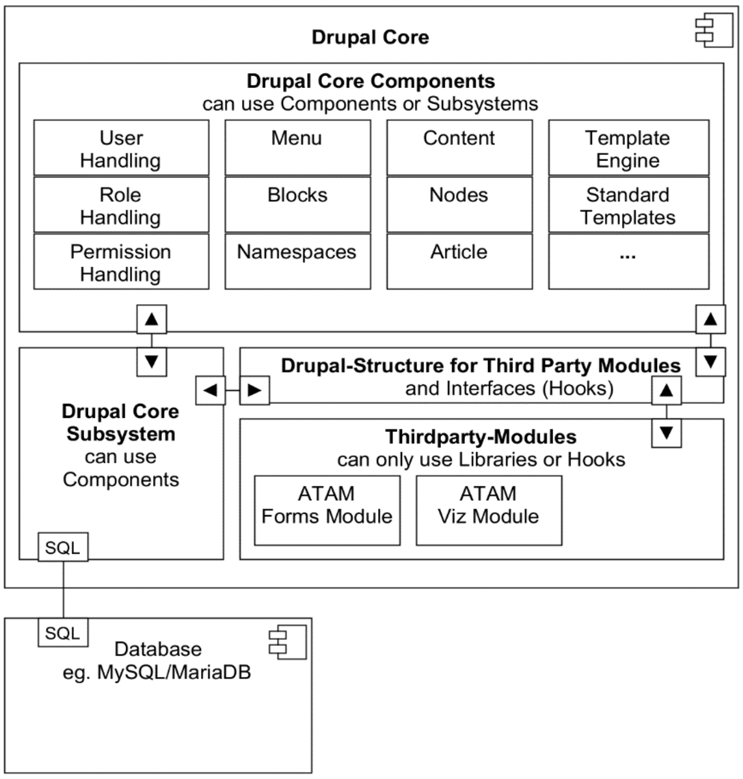


Fig. 8. Module Structure of the ISAA web application.

ATAM Forms Module. A Drupal module for data collection and management within an evaluation project via web forms. Table 3 shows the module's structure and Table 4 presents the modules web API regarding the quality attribute utility tree. As of this writing, the module supports data collection related to the overall project and the quality attribute utility tree.

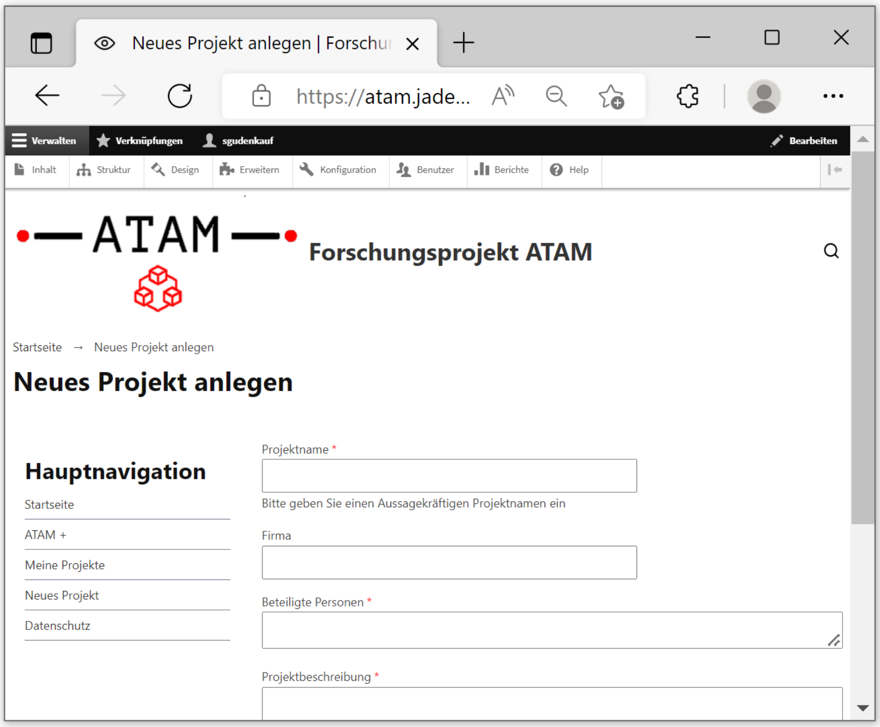
ATAM Viz Module. A Drupal module for the interactive visualization of evaluation projects. Its module structure is similar to the ATAM Forms Module. A `LocalStorageController` loads data from the database into a viewer's web browser and vice versa for fast manipulation and persistent storage across sessions. Further controller classes handle data transformation and presentation logic. The actual visualization and generation of UI elements is realized at the presentation level and almost exclusively with JavaScript. The visualization thus reacts interactively to the user and can continuously update itself.

Table 3. Structure of the ISAA prototype's ATAM Forms Module.

Module Structure	Description
atam	Main project folder; contains several files for database setup, libraries, permissions etc.
atam/src	Custom source code
atam/src/Controller	Controllers for the module
atam/templates	Custom templates used by Drupal for page generation
atam/src/Form	Custom forms conforming to the Drupal Form API
atam/css	Custom layout stylesheets
atam/img	Image files
atam/js	JavaScript files

Table 4. URL patterns for data collection in the ISAA prototype regarding the quality attribute utility tree.

URL pattern	Meaning
/project	Empty project form
/projects	List of projects
/project/123	Project with identifier 123
/project/123/json	JSON representation of project 123
/project/123/qa	Empty quality attribute form
/project/123/qas	List of quality attributes
/project/123/qa/456	Quality attribute with identifier 456
/project/123/qa/456/ref	Empty refinement form
/project/123/qa/456/refs	List of refinements
/project/123/qa/456/ref/789	Refinement with identifier 789
/project/123/qa/456/ref/789/sce	Empty scenario form
/project/123/qa/456/ref/789/sces	List of scenarios
/project/123/qa/456/ref/789/sce/012	Scenario with identifier 012



Bitte geben Sie einen Aussagekräftigen Projektnamen ein

Fig. 9. Screenshot of the ISAA prototype for data collection.

7 Conclusion

We dedicated this paper to the problem of adequate tool support for the analysis of software architectures. To do so, we researched related work and analyzed top-level requirements. Based on this we proposed a concept for *interactive software architecture analysis* (ISSA) that we presented in terms of an analysis model, a concept for interactive visualization, and a set of use cases that tool implementers can consider as functional requirements. While the focus of the paper lies on this concept, we also demonstrate the feasibility of our approach using a software prototype that we realized in form of portable modules for the Drupal content management system. In addition, we also describe the architecture of the prototype.

Regarding future work, we plan to extend and consolidate the ISAA concept and our web application prototype, and to evaluate it according to Moody's Method Evaluation Model [18]. Students of the Software Engineering course can most likely carry out an initial evaluation. As soon as tooling is sound, we would like to use it in research projects with our industrial partners. We also consider analyzing architectural variants for ISAA tool support using ATAM, which we then like to compare to the architecture of our current ISAA prototype. As soon as we collected a sufficient number of ATAM processes, we also like to investigate how data mining and machine learning can utilize process data to improve future ATAM processes, for example by recommendations on similar business drivers, similar scenarios, and similar architectural decisions. Finally, a closer look at the evaluation of the architectures of large software systems with many components, e.g. microservices architectures, can reveal additional challenges to ATAM and ISAA. How, for example, can ATAM variants cope with the *complexity explosion syndrome* [10] observed by Zalewski? A continued use of ATAM-defined artifacts provided by ISAA throughout the entire software development lifecycle might provide part of an answer, as well as extending the method by the notion of evaluation project groups, hierarchies, or dependency references between evaluation projects (as discussed in Section 5.1).

References

1. Kazman, Rick, Mark Klein, and Paul Clements. 2000. *ATAM: Method for Architecture Evaluation.* Fort Belvoir, VA. https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13706.pdf. Accessed 2 April 2022.
2. Kazman, R., M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere. 1998. The architecture tradeoff analysis method. In *Proceedings. Fourth IEEE International Conference on Engineering of Complex Computer Systems (Cat. No.98EX193)*, 68–78. Fourth IEEE International Conference on Engineering of Complex Computer Systems. ICECCS '98, Monterey, CA, USA. 10-14 Aug. 1998. IEEE Comput. Soc. doi: 10.1109/ICECCS.1998.706657.
3. Gabel, Andreas. 2013. Softwareunterstützung für die Evaluation von Softwarearchitekturen mit der Architecture Tradeoff Analysis Method. Thesis, Carl von Ossietzky Universität Oldenburg, Oldenburg.
4. Maheshwari, Piyush, and Albert Teoh. 2005. Supporting ATAM with a collaborative Web-based software architecture evaluation tool. *Science of Computer Programming* 57 (1): 109–128. doi: 10.1016/j.scico.2004.10.008.
5. Lionberger, Brad and Zhang, Cui. ATAM Assistant: A Semi-Automated Tool for the Architecture Tradeoff Analysis Method. In Proceedings of the 11th IASTED International Conference on Software Engineering and Applications, November 19-21, 2007, Cambridge, Massachusetts, USA, 330–335.
6. Murray, Scott. 2013. *Interactive data visualization for the web: An introduction to designing with D3*. Beijing, Köln: O'Reilly.
7. Shneiderman, B. 1996. *The eyes have it: a task by data type taxonomy for information visualizations: August 14 - 16, 1996, Blue Mountain Lake, New York*. LosAlamitos, Calif.: IEEE Computer Soc. Press.
8. Yau, Nathan. 2011. *Visualize this: The FlowingData guide to design, visualization, and statistics*. Indianapolis, Ind.: Wiley.
9. Software Engineering Institute. 2018. Architecture Tradeoff Analysis Method Collection. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=513908. Accessed 24 May 2022.

10. Zalewski, Andrzej. 2007. Beyond ATAM: Architecture Analysis in the Development of Large Scale Software Systems. In *Software Architecture*, ed. Flavio Oquendo, 92–105. Berlin, Heidelberg: Springer Berlin Heidelberg.
11. V. Basili, G. Caldiera, and H.D. Rombach. 1994. Goal Question Metric Approach. In *Encyclopedia of Software Engineering*, pp. 528-532. John Wiley & Sons, Inc.
12. van Solingen, Rini, Vic Basili, Gianluigi Caldiera, and H. Dieter Rombach. 2002. Goal Question Metric (GQM) Approach. In *Encyclopedia of software engineering*, ed. John J. Marciniak. New York: Wiley.
13. Ågren, S. Magnus, Eric Knauss, Rogardt Heldal, Patrizio Pelliccione, Anders Alminger, Magnus Antonsson, Thomas Karlkvist, and Anders Lindeborg. 2022. Architecture evaluation in continuous development. *Journal of Systems and Software* 184:111111. doi: 10.1016/j.jss.2021.111111.
14. Masak, Dieter. 2010. *Der Architekturreview.* Berlin, Heidelberg: Springer Berlin Heidelberg.
15. ISO/IEC. 2011. ISO/IEC 25010: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. Accessed 22 May 2022.
16. Drupal.org. 2018. Drupal - Open Source CMS. https://www.drupal.org/. Accessed 12 June 2022.
17. Sinha, Amit U., Emily Merrill, Scott A. Armstrong, Tim W. Clark, and Sudeshna Das. 2011. eXframe: reusable framework for storage, analysis and visualization of genomics experiments. *BMC bioinformatics* 12:452. doi: 10.1186/1471-2105-12-452.
18. Daniel L. Moody. 2013. The Method Evaluation Model: A Theoretical Model for Validating Information Systems Design Methods. In *ECIS 2003 Proceedings*.
19. Wittenburg, André. 2007. Software Cartography: Models and Methods for the Systematical Visualization of Application Landscapes. Doctoral thesis, Technical University of Munich, Munich. https://d-nb.info/988065851/34. Accessed 14 June 2022.
20. Lewis, James, and Martin Fowler. 2014. Microservices: a definition of this new architectural term. https://martinfowler.com/articles/microservices.html. Accessed 14 June 2022.
21. Wolff, Eberhard. 2018. *Microservices: Grundlagen flexibler Softwarearchitekturen*, 2nd edn. Heidelberg: dpunkt.verlag.