Voyager: Software Architecture Trade-off Explorer

Jason Mashinchi and Javier Cámara

Department of Computer Science, University of York
jason.mashinchi@alumni.york.ac.uk, javier.camaramoreno@york.ac.uk

Abstract. Software engineers must ensure that systems under development are endowed with software architectures that enable them to meet their requirements. Apart from functionality, systems also have to satisfy extra-functional requirements that may include behavioural constraints that the software must adhere to, as well as qualities to optimise such as performance, availability, and energy efficiency. These qualities are often inter-dependent and heavily influenced by the structure of the system. This results in poorly understood multi-dimensional design spaces, in which trade-offs among qualities are not evident when making architectural decisions. This paper presents *Voyager*, a tool which allows engineers to visualise architectural configurations and explore the trade-offs among their quality attributes in a multi-dimensional design space. The tool produces contextual visualisations to facilitate trade-off analysis, providing engineers with a streamlined way of understanding architectural design spaces, using an approach that combines architectural structure with multi-dimensional data visualisations. A user study was conducted to evaluate the effectiveness of the tool. Results show that participants achieved a significantly higher accuracy in a shorter time span and had a better user experience when using *Voyager*, with respect to an existing comparable tool.

Keywords: Software architecture · visualisation · trade-offs · quality attributes

1 Introduction

Software is extensively used across the globe today, forming a key part of many industries with applications that range from safety-critical aviation to social networking. All software must meet its requirements – i.e. must be able to achieve its intended purpose, by performing functions required of it and meeting whatever behavioural constraints that exist [12]. For example, a piece of software may be required to calculate the speed a car is travelling, with a constraint that it must deliver a result within 10ms of receiving an input. There are many correct ways to develop software, and many possible architectural structures that allow the software to achieve its goal, each with its own benefits and trade-offs.

A key challenge for software engineers is to understand the properties of the architectural design space, to allow them to make better design decisions

with well-grounded knowledge about the trade-offs amongst concerns (e.g. cost, reliability) and the architectural constraints [4]. More often than not, the architectural design space is poorly understood, as it is not easy to represent the trade-offs that exist between desirable quality attributes and the structure of possible architectural configurations in an accessible way. Understanding this design space is useful for developing optimal software, as different configurations entail different trade-offs that software architects have to make.

Visualisation is a useful tool for developing an understanding of data. However, visualising architectural design spaces is challenging because of the multi-dimensional nature of the problem (quality metrics used for comparison often go beyond three dimensions), and the difficulty in relating explicitly structure and quality trade-offs.

This paper presents *Voyager*[1] – a tool that primarily focuses on the needs of software engineers, by combining trade-off analysis with software architectural structure visualisation. As quality metrics of architectural configurations depend on their structures, it is helpful for architects to understand and easily analyse both side-by-side. Existing tools focus on either architecture visualisation (e.g. AcmeStudio [15]), or architectural trade-off analysis (e.g. ClaferMoo [11]), unlike *Voyager* which offers a novel combination of the two, enabling engineers to narrow design spaces and find better configurations more effectively.

While *Voyager* is designed primarily for software architecture analysis, the multi-dimensional trade-off analysis features are more general purpose, so it can also be applied to other related areas with multi-dimensional data to analyse results from variable configuration spaces (e.g. software product lines [6], quantitative verification [8]). In addition, *Voyager* has extensibility features that enable integration with external tools which can act as a data source and provide additional visualisations to appear in the user interface.

2 Background & Related Work

Architecture refers to the high-level aspects of the software, such as its overall organisation, the individual components and their functionality, and the relationships and interaction between them [4]. There are many alternative software architectures that can be used to realize a software system, each with its own set of quality characteristics [10]. Selecting one of a possible set of alternative approaches to the software architecture entails carrying out a set of design decisions which have to be informed by a clear understanding the design space, including trade-offs amongst relevant quality attributes.

To inform this selection, some tools such as Prism-MW [7], ArchJava [1] and Aura [16] facilitate modelling a set of correct architectural configurations to analyse, without much support for optimisation. Work in multi-dimensional architecture optimisation approaches is plentiful and varied in classes of techniques employed [2], with some recent approaches enabling automated synthesis

[1] The source code, user study data and a video demonstration of the *Voyager* tool is located online at: https://github.com/jasonmash/voyager

of sets of correct configurations with associated quality metrics [5]. However, these tools are not designed to facilitate systematic and interactive exploration of their output, which is often difficult to understand and cumbersome to explore. The output of these tools can be used as input for *Voyager*, which offers trade-off and architectural structure visualisations that help software engineers understand and analyse these sets of multidimensional architectural data.

As each architectural configuration consists of a structured set of components, individual architectures can be visualised and analysed using tools such as AcmeStudio [15] and SoftArchVis [13]. These tools allow visualising the software architectures to give a better understanding of how each configuration is composed and their attributes. However, these tools are limited to visualising one architectural configuration at a time, reducing their effectiveness for understanding and analysing the design space and quality trade-offs. *Voyager* incorporates basic structural diagramming tools alongside its trade-off analysis functionality, and also provides an extensions interface that allows external tools to add new static or dynamic architectural visualisations.

Other existing tools such as ClaferMoo [11] and TradeMaker [3] are good for comparing amongst many configurations, providing charts such as 2D bubble plots and matrices representing the distribution of configurations in relation to their quality attributes. However, these do not include architectural structure information, making it difficult to understand how the quality attributes relate to the architecture itself. *Voyager* incorporates the design space visualisations and trade-off analysis tools, alongside architectural structure visualisations, showing each when contextually appropriate, without requiring any user configuration.

Finding an architecture design that meets all quality requirements while balancing the trade-offs from dependent quality attributes requires multi-objective optimisation, a process that generates sets of Pareto optimal solutions (i.e. solutions for which no alternative solution that is better in one property and equally as good with respect to all others, exists [14]). Although some of the existing architecture optimisation approaches can generate Pareto-optimal solutions, none of the surveyed tools incorporates algorithms to calculate the Pareto frontier for user-selected quality attributes from a raw data set. Our tool is able to do that, making clear which configurations are Pareto-optimal for the selected attributes.

In summary, existing tools are effective for analysing either a single architecture at a time, or multiple correct architectures but without any explicit link to architectural structure. In contrast, our tool combines the benefits of trade-off analysis with that of software architectural structure visualisation to enable better understanding of architectural design spaces.

3 Voyager

Voyager is designed to support architects during evaluation of architecture design quality and satisfaction of stopping criteria when optimising architectures, helping them in understanding the design space and potentially providing feedback for the generation of new design alternatives (Figure 1, right).

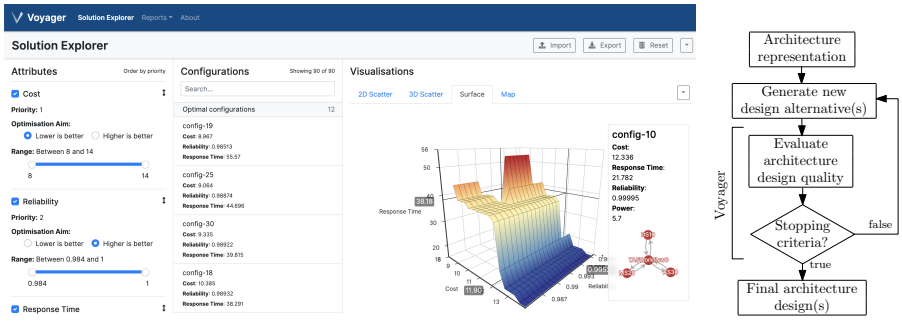

Fig. 1: (left) screenshot of Solution Explorer in *Voyager* and (right) architecture optimisation workflow (adapted from [2])

3.1 Implementation

The tool is implemented using open source web technologies, delivering a cross-platform application that runs in a browser. The tool is implemented using TypeScript (typescriptlang.org), a superset of JavaScript, which utilises extra compilation steps to add features such as type checking that improve developer productivity. Additionally, open source libraries such as Vue.js (vuejs.org), Bootstrap (getbootstrap.com) and ECharts [9] are used to construct the user interface, alongside several other libraries listed in the code. While the tool is designed to run entirely within a browser, all computation and data processing takes place locally, and data is persisted across browser sessions to ensure *Voyager* behaves like any other locally installed application.

To ensure the quality of the *Voyager* tool, a suite of end-to-end and unit tests has been developed, using the Cypress (cypress.io) and Mocha (mochajs.org) libraries respectively.

3.2 Solution Explorer

The core functionality of *Voyager* is found in the "Solution Explorer", enabling users to study a set of architectural configurations and their trade-offs using context-appropriate visualisations, alongside relevant sorting and filtering tools. The Solution Explorer page is split into three columns, each showing attributes, configurations and visualisations respectively for the current data set. An example is shown in Figure 1 (left), for a set of configurations that have cost, reliability, and response time quality attributes.

3.3 Quality Attributes

The left-hand most "Attributes" pane includes the set of quality attributes inferred from the imported architecture configurations, and it has been designed to allow software architects to straightforwardly reduce the architectural design

space. This is accomplished by allowing users to: select, filter and sort configurations based on the values of their quality attributes; to set an optimisation aim (i.e. whether higher or lower values are better for the attribute); and to narrow down the range of acceptable values for a given attribute. Changing these properties updates the list of configurations and any currently visible visualisations in real-time, ensuring users get instant feedback on how changes to the design space affect the possible architectural solutions for the given data set.

3.4 Architectural Configurations

The centre of the screen contains a list of configurations that meet the requirements specified in the attributes pane. The configurations are grouped together based on Pareto optimality, where optimal architectures on the Pareto frontier are placed at the top, followed by non-dominant solutions below. Configurations are sorted according to their attribute values, the order of which is determined by the attribute optimisation aim (e.g. when higher values are better, those configurations are placed first). This effectively shows architects which of the possible structures are best suited for further consideration.

A single configuration can be selected, showing the "Selected Configuration" panel to the right. This presents each quality attribute value for the selected configuration, and a radar chart of these values relative to those of other configurations, alongside any visualisations of its software architecture.

By default, *Voyager* shows an architectural structure graph, representing each component within the architecture and the connections between them. This chart allows the user to hover over individual elements for further details and can be panned and zoomed. Additional architectural visualisations from external tools can be shown in the selected configuration panel, by providing these in an image or html-based format using the *Voyager* extensions interface. These architectural visualisations help users understand the design space, by enabling users to quickly compare possible configurations and identify which style of architectures are better or worse, which components have which trade-offs, etc.

3.5 Design Space Visualisation

Using visualisations is an effective way of understanding data sets, as it allows humans to intuitively identify patterns and trends, and spot outliers. *Voyager* shows visualisations of the design space in the rightmost panel, including data points for each visible configuration after the attribute filters have been applied. This allows users to gain an understanding of the relationships between quality attributes and therefore whether trade-offs exist.

Software architecture quality attributes, like any other data, are easy to visualise when there are one, two or three attributes to analyse, making use of graphs such as scatter plots, bar charts and 3D surface plots. However, as it is common with software architectures, there are often more than three dimensions of data to process, presenting a challenge as we cannot simply add additional dimensions to graphical visualisations, being fundamentally limited to 3D space. Therefore,

Voyager makes use complex visualisations that encode additional data into the space we are able to perceive, applying projections onto the data where necessary, and utilising additional properties such as colour, size and position where appropriate.

The visibility of each visualisation in the UI is context-dependent, as their effectiveness depends on the number of configurations and the number of dimensions of attributes, determined by the selected attributes and filters. Each visualisation updates in real-time as filters are adjusted, which means users do not have to manually press refresh (or similar) like existing tools. This reduces cognitive load, by allowing users to focus their thought processes on their data, rather than on how to get the software to do what they want it to.

The visualisations shown in the visualisations panel have been selected according to their usability, clarity and function. These include bar/line charts, 2D/3D scatter plots, surface plots, configuration maps and radar charts.

Each visualisation has a dropdown menu in the top right corner, providing options such as exporting to image files, and switching between projections of 3D charts (e.g. orthographic and perspective). All visualisations include additional information in tooltips for each data point – e.g. 3D scatter plots include information about where the mouse is along each axis, and which configuration is highlighted. In addition, selecting a point provides architectural structure diagrams, acting as an effective tool for comparing architectural structure and quality attributes side-by-side (c.f. Figure 1, left).

3.6 Reports

Voyager contains reporting functionality to allow users to save any visualisation included in the application into a report for future reference. Report visualisations contain a snapshot of their source data to ensure their content is not modified by any data manipulation performed elsewhere in the application. Users can create one or more reports, each with a unique title, to group together multiple visualisations that can be labelled - this provides a straightforward mechanism for comparing between multiple architectures.

3.7 Data Sources & Extensibility

Voyager makes use of common file formats such as .csv and .json to allow users to import and export data from the application easily, enabling the use of various other tools for data collection and preparation. The state of the application can be exported directly from the user interface, and re-imported at a later date to restore the application exactly to its previous state, resulting in an output file that can be shared amongst interested parties when collaborating.

Voyager offers an extension interface, allowing third party tools to integrate with the tool by providing lists of configurations and associated customised visualisations in static (image) or dynamic (html/js) formats. Communication between *Voyager* and external tools is accomplished using HTTP requests, with the requirement for external tools to implement a REST API that returns JSON

data for specified endpoints. This technology choice was made because HTTP is a widely supported protocol, with easy implementation across many programming languages.

4 Evaluation

During development, *Voyager* has been validated with existing data sets, including the Tele Assistance System (TAS) exemplar (a service-based system) [17], showing indication of its potential to analyze trade-offs in preliminary user experiments. To further validate that *Voyager* meets the goal of providing engineers with a user-friendly tool for visualising software architectures and exploring their trade-offs, we have conducted a user study to quantify its effectiveness.

4.1 User Study Design

We constructed a user study consisting of a set of questions related to a software architectural trade-off analysis scenario. Participants are asked to make use of tools including *Voyager* and other existing comparable software to analyse the provided data for a given scenario. This allows for the collection of quantitative data that is used to compare and measure the effectiveness of our tool.

Each scenario used in the user study include sets of architectural configuration data, containing both quality attribute values and a representation of the architectural structures for each configuration. Participants are asked questions requiring them to find optimal architectural configurations for the data set, by performing tasks such as filtering, sorting, clustering and correlation to identify any trade-offs between quality attributes. Participants also must make use of individual architectural structure visualisations to compare between two or more potential configurations. The data sets used contained many configurations with cost, battery life, range and reliability quality attributes, each with representative trade-offs between each.

To establish a baseline prior ability of each participant, and to ensure they have a chance to familiarise themselves with the type of problem they are being asked to solve, the first section of the user study consists of a background task which all participants complete. This background task contains a scenario and set of questions, alongside a basic spreadsheet tool that presents the data and only offers basic data analysis tools including sorting and filtering.

Following the completion of the background task, the participants are randomly allocated one of two possible tools for use on further, more difficult questions. One of these tools is *Voyager*, and the other is ClaferMoo Visualizer [11] - a directly comparable tool with similar aims. This tool was selected for use in this study because: it provides a user interface that can be used to solve the same class of problems as *Voyager*, it is easily available and widely used, and it can be populated with fundamentally the same data set as *Voyager*.

The same scenario, data set and questions are used regardless of the allocated tool, with slight terminology adjustments to account for the differences between

the tools (i.e. a *Voyager* "configuration" is called a "variant" in ClaferMoo). This second analysis task is intentionally more difficult than the background task, and contains additional quality attributes and configurations to analyse.

Each scenario consisted of four questions, each formulated to cover a comprehensive range of tasks users typically accomplish when conducting analytic activities, and also to provide quantitative data to be used to compare and measure the effectiveness of our tool. For each question, timing data was captured to understand how long it takes users to complete allocated tasks for each tool.

Question	**Rationale**
1. Identify the configuration with the lowest cost	A straightforward question to get participants familiar with the tool user interface. Requires them to use the UI to find a single configuration with the lowest value for one quality attribute.
2. Identify one (or more) configurations with the highest possible battery life and highest possible range within the same configuration	This question is designed to get participants thinking about trade-offs, as the data set contained no obvious answers, as in this case, increased range meant reduced battery life. Participants could make use of the tabular or graphical representations of all configurations, as well as sorting tools to find those configurations that were on the Pareto-front for this problem.
3. Identify one (or more) configurations that have the highest possible battery life, then the highest possible range where the reliability is greater than [threshold]	This question requires participants to make use of more advanced functionality within each tool, including filtering, to identify configurations on the Pareto-front for this problem. Participants were told that sorting/filtering/visualisation tools could be used.
4. Identify any common features present in configurations that have a battery life greater than [threshold] and a cost less than [threshold]	This question was designed to get participants to make use of architectural structure visualisation tools, to identify any components and connections that were common within a similar class of configurations.

The responses collected from each participant are validated using a numerical score for each question, representing the number of correct answers achieved out of the total set of correct answers. For the questions where optimal configurations need to be identified, the set of correct answers is the set of Pareto-optimal configurations matching the specified goal.

Following the completion of each scenario within the user study, we asked participants a series of usability questions to gather their opinion and therefore a measure of their perception and confidence of how well they performed on the task for each tool. Participants were asked **(i)** how they found the task, **(ii)** how well they thought they did, and **(iii)** how easy the tool was to use. The answer options for these questions took the form of a 5-point Likert scale, with

the results being stored as 0 being a strongly negative answer, 3 being neutral, and 4 being a strongly positive answer.

To determine whether a response we received from a participant was valid (and not filtered out), we made use of the following criteria: **(i)** the participant completed all questions, **(ii)** all answers to the question were in the expected data formats, **(iii)** timing data was present for every question, and **(iv)** the participant reported no problems completing the study.

4.2 Experiment Design

We recruited 47 participants to complete the user study of various backgrounds and abilities. Of the 47 participants who started the user study, 32 participants fully completed the study and provided results that were valid for further analysis and contained no invalid answers according to the verification criteria above.

To ensure we understood our participants background experience, they were asked to provide their current occupation, educational study level, and their level of study in STEM-related subjects. Numbers of participants at each STEM education level were as follows: Secondary: 1; Post-Secondary: 9; Bachelor's Degree: 5; Master's Degree (or higher): 17.

A web-based tool was developed to conduct the user study. This was necessary to embed a spreadsheet tool, *Voyager* and ClaferMoo Visualizer in a seamless user interface, which ensured the only technical requirement participants had to comply with was access to a modern desktop-sized web browser.

The total cohort of participants was split into two equally-sized groups, each of which was allocated either the *Voyager* or ClaferMoo Visualizer tool. In total, 16 participants (50% of the total) completed the task using *Voyager*, and 16 participants completed the task using ClaferMoo Visualizer. All 32 participants completed the background task using the embedded spreadsheet.

To balance the effects of education levels amongst participants, their allocation to groups was entirely random. This led to the unintended effect of one group having a slightly higher average education level than the other, which may have resulted in an overestimate of the difference in outcomes between the groups in the results. To account for this, a statistical T-Test was performed making use of the background task data, which did not show a statistically significant difference in the scores achieved between the two groups ($p = 0.691$, with mean values of: 5.1 for Voyager, and 4.9 for ClaferMoo participants; where the maximum score was 8). A review of each participant's occupation showed these were well balanced between groups, as similar numbers of participants with relevant occupations were present in each group (e.g. engineers, computer specialists).

The independent variable of this experiment was the tool used to complete the same scenario. The dependent variables we measured were: correctness, confidence, user perception, and time to complete each task. Correctness was measured using the scores achieved per question, while confidence and perception were measured using the usability questions.

The hypotheses for this experiment were as follows:

1. Given the same data set and questions, participants would identify more correct answers in a shorter time period using *Voyager* compared to those using an existing tool.
2. Participants would find *Voyager* subjectively easier to use and would be more confident in their results compared to using a spreadsheet or a comparable existing tool.

4.3 Analysis & Results

The results of the user study support both hypotheses of the experiment. This data is publicly accessible alongside the source code. A two-sample, one-sided statistical t-test was used to calculate a measure of whether there was a significant difference between two sets of data, making use of the output p-value, which shows a significant result if it is less than 0.05. A p-value represents the probability of observing a result at least as extreme as the observed results, assuming that the null hypothesis is true (equal means). A smaller p-value means there is a smaller probability that the null hypothesis is true, providing stronger evidence in favour of the alternative hypothesis.

Participants using *Voyager* achieved a higher average score, in less time than those who used ClaferMoo Visualizer, given the same questions and data set. The mean average scores and durations for the tool questions are shown below, in addition to the p-values obtained using a t-test as described above.

	Total Score (% correct)	**Duration** (mins)
Voyager	57	8.4
ClaferMoo Visualizer	38	10.9
T-Test (p-value)	0.0133	0.0391

Performing a statistical t-test shows there is a significant difference between both tools for the total score with mean averages of 57% for Voyager, and 38% for ClaferMoo, with $p = 0.0133$. Likewise for timing data (total time to complete tool questions), with mean averages of 8.4 minutes for Voyager and 10.9 minutes for ClaferMoo, with $p = 0.0391$. These p-values allow us to reject the null hypothesis, and conclude there is not evidence in support of equal means. This shows a statistically significant difference for both dependent variables, indicating that the hypothesis that the *Voyager* tool allows users to achieve higher accuracy answers in a shorter time period is correct. Broken down by question, in every case, participants using *Voyager* achieved a higher mean average than those who used ClaferMoo Visualizer.

The results from the usability questions that measured user perception and confidence also support the hypothesis that *Voyager* was subjectively easier to use compared to a spreadsheet and existing tools. A statistical t-test for each of the questions asked was conducted, comparing the results from *Voyager* to those from both the spreadsheet and ClaferMoo tasks. The results are as follows:

Question	Mean Average			T-test (p-value)	
	Spreadsheet	ClaferMoo	Voyager	Spreadsheet	ClaferMoo
How easy was the tool to use?	2.19	1.13	2.69	0.03627	0.00002
How did you find the task?	1.94	1.44	2.38	0.00904	0.00134
How well do you think you did?	2.38	2.25	2.81	0.01339	0.05330

For the user perception measures (how easy was the tool to use?, how did you find the task?) - it is clear that *Voyager* has a higher mean average than both a spreadsheet and ClaferMoo Visualizer, and this is statistically significant for in both cases ($p < 0.05$). This means users found *Voyager* easier to use, and found completing the same task easier using *Voyager*.

For the question quantifying how confident users felt about their answers (how well do you think you did?), *Voyager* had statistically significant difference compared to the spreadsheet ($p = 0.01339$) with a higher mean average, but there was not a significant difference compared to ClaferMoo Visualizer ($p = 0.05330$, which is greater than 0.05) despite its higher average. This is a clear contrast to the actual scores achieved on both tools, where there was a statistically significant difference in the results.

5 Discussion & Future Work

Voyager is a trade-off exploration tool designed for supporting effective analysis and understanding of multi-dimensional design spaces. The tool delivers a user-friendly, flexible and robust interface, offering a novel solution that combines multi-dimensional quality attribute analysis with architectural structure visualisation – neither of which appear to have been combined into a single tool before. It makes use of modern web technology to deliver clear 2D and 3D data visualisations, offering a maintainable and reliable codebase fit for future use and expansion. *Voyager*'s extensibility features enable flexible integration with other tools, opening up the potential to serve a much larger set of use-cases (e.g., software product lines).

Our user study shows *Voyager* is effective for use with multi-dimensional architecture trade-off problems, having obtained results that show it had a significantly better user experience compared with existing comparable tools, allowing participants to achieve higher accuracy of answers in a shorter time span.

There is scope for future work - including implementing new visualisation ideas (e.g. hierarchical structure exploration, parallel coordinate charts, conditional formatting etc), encouraging open source contributions, and offering enhanced support for external tool integration.

Acknowledgements. The authors would like to thank everyone who kindly volunteered to participate in the user study.

References

1. Aldrich, J., Chambers, C., Notkin, D.: Archjava: connecting software architecture to implementation. In: Proceedings of the 24th International Conference on Software Engineering. ICSE 2002. pp. 187–197 (May 2002)
2. Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., Meedeniya, I.: Software architecture optimization methods: A systematic literature review. IEEE Trans. Softw. Eng. **39**(5), 658–683 (2013)
3. Bagheri, H., Tang, C., Sullivan, K.: Trademaker: Automated dynamic analysis of synthesized tradespaces. In: Proceedings of the 36th International Conference on Software Engineering. pp. 106–116. ICSE 2014, ACM, New York, NY, USA (2014)
4. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley Professional, 3rd edn. (2012)
5. Cámara, J., Garlan, D., Schmerl, B.R.: Synthesizing tradeoff spaces with quantitative guarantees for families of software systems. J. Syst. Softw. **152**, 33–49 (2019)
6. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley Professional (2001)
7. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. LNCS, vol. 3920, pp. 441–444. Springer (2006)
8. Kwiatkowska, M.: Quantitative verification: Models, techniques and tools. In: 6th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers. pp. 449–458. ACM (2007)
9. Li, D., et al.: Echarts: A declarative framework for rapid construction of web-based visualization. Visual Informatics **2**(2), 136 – 146 (2018)
10. Mahdavi-Hezavehi, S., Galster, M., Avgeriou, P.: Variability in quality attributes of service-based software systems: A systematic literature review. Inf Softw Technol **55**(2), 320–343 (2013)
11. Murashkin, A., Antkiewicz, M., Rayside, D., Czarnecki, K.: Visualization and exploration of optimal variants in product line engineering. In: Proceedings of the 17th International Software Product Line Conference. pp. 111–115. ACM (2013)
12. Mylopoulos, J., Chung, L., Nixon, B.: Representing and using nonfunctional requirements: a process-oriented approach. IEEE Trans. Softw. Eng. **18**(6), 483–497 (1992)
13. Sawant, A.P., Bali, N.: Softarchviz: A software architecture visualization tool. 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis pp. 154–155 (06 2007)
14. Sayyad, A.S., Ammar, H.: Pareto-optimal search-based software engineering: A literature survey. In: 2013 2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering. pp. 21–27 (May 2013)
15. Schmerl, B., Garlan, D.: Acmestudio: Supporting style-centered architecture development. In: Proceedings of the 26th International Conference on Software Engineering. pp. 704–705. ICSE '04 (2004)
16. Sousa, J.P., Garlan, D.: The aura software architecture: an infrastructure for ubiquitous computing (2003)
17. Weyns, D., Calinescu, R.: Tele assistance: A self-adaptive service-based system exemplar. In: 2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 88–92 (2015)