Recording Software Design Decisions on the Fly

Fabian Gilson[a], Sam Annand[a] and Jack Steel[a]

[a]University of Canterbury, Private Bag 4800, Christchurch 8140, New Zealand

Abstract

A common trend in the software engineering field shows developers are not documenting their design decisions as much as they believe they should be. Part of the reason for this is that developers consider the design decision documentation process to be cumbersome. However documentation of design decisions proves to be a crucial aspect of software maintenance and enhance teams' shared understanding of a product. With the recent emergence of instant messaging in software teams (*e.g.,* Slack, Microsoft Teams), our goal is to leverage their platform coupled to chat bots and natural language processing technologies to record design decisions as they arise. In doing so we place the system for recording design decisions into an environment that is already common place for software teams, dramatically reducing the barriers involved with recording design decisions as a separate task. We therefore propose a general framework composed of a chatbot, a natural language processing engine and exporting API to wikis to record and search design decisions enabling teams to focus on their day-to-day work with minimal disruptions in their workflow, but still keeping a project documentation up to date.

Keywords

agile software development, design decisions, architecture knowledge, documentation, instant messaging, chatbot, natural language processing

1. Introduction

A software architecture is commonly depicted as a set of design decisions [1]. Design decisions have been considered by researchers as first-class citizens for decades [2] and they are subject of noticeable research interest [3, 4]. However, current evidence shows that such decisions are scarcely documented [5] despite their critical usefulness to understand a software's code base [6]. Dedicated tools have been proposed across the years [7, 8], but their practical usage in industrial context remains limited both for decision-making or traceability purposes [9, 4].

Decision-making in software architecture and development tends to be unstructured [10], the industry generally favouring ad-hoc techniques [11, 12] where tailored approaches are often preferred over universal ones [13]. Within an increasingly collaborative decision-making process in software architecture [14, 15], where the Agile Software Development (ASD) culture even empowers individuals with a decision-making ability [16, 17], it has been observed that important discussions take place on instant messaging platforms [18], strengthening a participatory culture in software development teams [19].

Joint Proceedings of SEED & NLPaSE co-located with APSEC 2020, 01-Dec, 2020, Singapore

fabian.gilson@canterbury.ac.nz (F. Gilson); sga111@uclive.ac.nz (S. Annand); mail@jacksteel.co.nz (J. Steel)

https://fabgilson.github.io/ (F. Gilson)

0000-0002-1465-3315 (F. Gilson)

Ralph and Tempero found out from interviews that software developers are taking design decisions, both consciously and unconsciously, regarding many aspects of a software, including its scope and architecture [20]. Furthermore, in an ASD context, decisions may occur at any time during a typical *"sprint"* lifecycle [21]. However, remembering those decisions post hoc is rarely possible [20] and ASD teams are neglecting to record their design decisions too [22].

Alternatively to structured decisions and rationale documentation tools, retrospective techniques to retrieve architectural knowledge from artefacts [23], code versioning [24] and issue trackers [25] have been proposed. But such tools either require a significant forensic effort or a manual screening and evaluation of automatically extracted data. In this paper, we attempt to get closer to Kruchten *et al.*'s expectation to automatically capture design decisions [26].

In current work, we propose to bring the recording of design decisions and rationale closer to when discussions happen on messaging platforms. Our contribution is composed of a combination of *(a)* a bot embedded in an instant messaging tool (i.e. *Slack* [1]), *(b)* a Natural Language Processing (NLP) engine, *(c)* a structured architecture knowledge representation and *(d)* an API to serialise decisions in textual forms into wikis. Our bot is also able to look into previously recorded decisions to help developers, as well as any involved stakeholders remember previously recorded decisions. We advocate that this lightweight setup will increase the tendency to record decisions as they occur and will serve the basis for a just-in-time management of software architecture and to some extent, domain knowledge.

In the following, we address relevant related work in Section 2. We present our technique in Section 3 where we depict our metamodel for design decisions and explain our approach. We describe the prototype in Section 4 by showing how the setup, recording and searching features work. We evaluate the usability of the proposed system in Section 5 and discuss the limitations and future development before concluding in Section 6.

2. Related Work

We grouped existing works in two categories: *(a)* (meta) models for recording architectural design decision and their tools and *(b)* analysing and mining techniques to reconstruct architectural knowledge.

Design Decisions Models and Tools: According to Bhat *et al.* [4], techniques to capture design decisions and their rationale trace back to 1980 where an existing solution (*i.e.* the antecedent) was replaced by an updated desired effect (*i.e.* consequent) [27]. Many formalisation or metamodels have been proposed since, including Kruchten's seminal 4+1 view model [28] and taxonomy [26], and Tyree and Akerman description template [29]. Many other formal models have appeared since (*e.g.,* [1, 30, 31, 32]. **In this research, we reuse Zdun *et al.* *"(wh)y"* template** [33] for its simple, yet comprehensive template and suitability in a conversational environment (*i.e.* chat-oriented bot).

Design patterns have also been studies under the lens of design decisions. Harrison *et al.* propose to use patterns as documentation mean for design decisions [34]. That *et al.* embed decisions into their architecture description language [35]. Farshidi *et al.* intend to create a

[1] See https://slack.com.

knowledge base of architectural patterns to help in decision-making process of practitioners [36]. **In our work, we do not specifically emphasise on patterns** but are more flexible on what needs to be recorded with a looser semantics (*e.g.,* decision, alternatives, effects) that can be applied to a broader range of decisions (*e.g.,* technologies, APIs, user interfaces).

More formal approaches [37, 38, 39] proposed to consider incremental changes in software models as design decisions where each *refinement* contains their design rationale, creating a graph of models with their decisions. **In this work, we target a lightweight framework** that does not require formal models or transformations for easier adoption by practitioners.

Reconstruction and Mining Techniques: Some effort has been put to reconstruct architectural knowledge and decisions from tangible artefacts. Rooted in Basili and Mills [40] retro-engineering and correctness evaluation of legacy code, Rugater *et al.* describe a manual approach to document design decisions from code [41]. Jansen *et al.* propose a technique based on the (mostly manual) analysis of architecture deltas between subsequent releases [23]. **We consider after-the-fact manual retrieval of architecture knowledge is impractical in the long run**, or at least put too much burden on the development team to be effective.

Astudillo *et al.* compare a manual and automated retrieval of decisions and rationale from project textual documentation [42]. Van der Ven and Bosch propose an automated mining process to classify decisions, rationale and alternatives from version control history [43]. Bhat *et al.* trained machine learning models to retrieve design decisions from issue trackers [25]. Shahbazian *et al.* propose a tool combining data from issue trackers, source code and version control history to map low-level changes in the code to items in the issue tracker [24]. However these works still require lots of training data and a validation phase that must be carried manually, often dealing with too many, too detailed or false positive output **where we target a just-in-time solution that can be used straightaway.**

3. Recording Decisions on-the-Fly

In present work, we advocate for a lightweight just-in-time recording of design decisions and rationale. As discussed above, dedicated complex tools are scarcely used despite their advantages and extraction methods from various sources (*e.g.,* source code, version control, textual documents) put a significant manual burden on the development team to make sense of the mined data. Therefore, leveraging recent advances in software bots [44], we propose a new documentation framework composed of a design decision metamodel, a bot integrated into an instant messaging tool and an exporting API.

3.1. Design Decision model

Following Zdun *et al. "(wh)y"* decision template [33], we propose to document design decisions with the same concepts (see **bold-faced** terms in Figure 1):

`DesignDecision` A design decision is the focal point of our recording framework and is composed by all elements of the *(wh)y* template. All parts of the template are optional

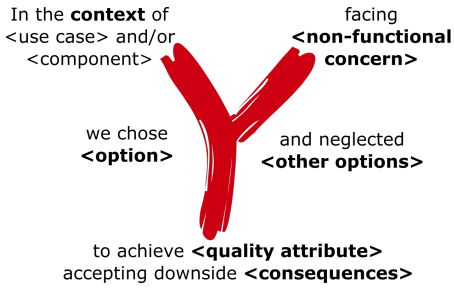

Figure 1: Zdun *et al.* "(wh)y" template of a design decision, schematised from [33].

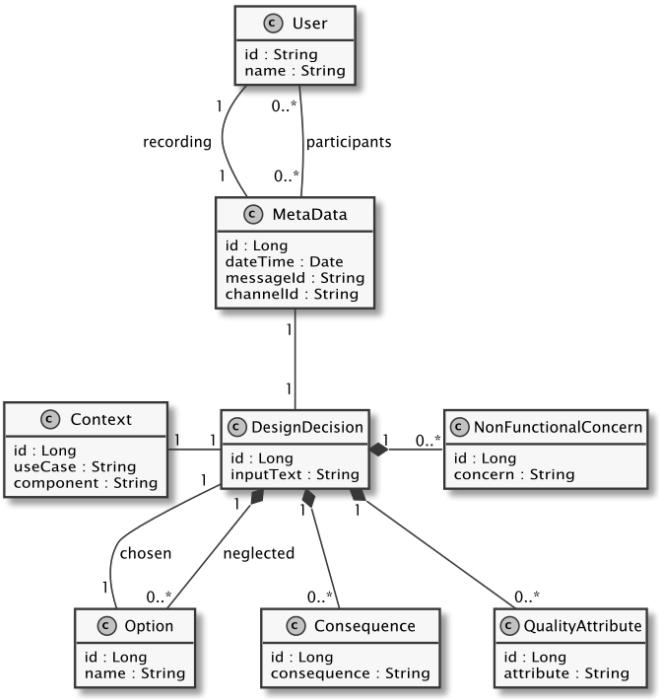

Figure 2: Metamodel of a design decision, including its metadata and links to users.

except the `Context` and the selected `Option`. An `inputText` field is used to recompose the full sentence using Zdun's template for exporting and search performance purposes.

`Context` A decision applies into the context of a use case (or user story) and/or software component. This allows involved parties to remember what part of a software system the decision refers to or what functional purpose the decision serves.

`NonFunctionalConcern` Often, architectural decisions are addressing non functional requirements (*e.g.*, response time under certain threshold). These concerns are actual needs from the system expressed within its `Context`.

`Option` Options are of two types: *neglected* and *chosen*. They are free text fields expressing what solution has been selected in the current `Context`, possibly to answer the `NonFunctionalConcern`. Only the *chosen* option is compulsory as it is a crucial part of a design decision to trace back what concrete strategy has been applied.

`QualityAttribute` Attributes are non-functional requirements that are not context-specific. Quality attributes are well established characteristics a software system should satisfy. Example of such qualities can be found in the ISO/IEC 25010:2011 quality models [45] and are often referred to as the *"ilities"* of a system.

`Consequence` This item expresses any drawbacks a particular `Option` would cause to the architecture (*e.g.*, increased development time, increased technical debt). This part of the decision may be invaluable to make sure involved parties are evaluating all aspects of their decisions thoroughly so that known shortcomings are made explicit.

The metamodel for above concepts is given in Figure 2 where additional data are collected in order to enhance design decisions with situational details:

`MetaData` Since the decision is recorded using an instant messaging platform, additional details can be collected to improve the traceability of design decisions over time. We therefore keep a reference to the original `messageId` (together with the `channelId`) that triggered the recording of the decision to allow later references to the conversation and keep decisions channel-specific.

`User` Even with the best recording mechanism, lots of information may remain in the head of involved parties. Therefore, we keep a link to the `User` that recorded the decision as well as all other persons belonging to the chat channel at the time of that decision.

We persist a structured representation of a design decision together with the details of the persons involved in that decision to allow both technical and organisational aspects to be recorded for later reference.

3.2. Bot Integrated in Instant Messaging

The first technical feature of our solution is a bot integrated into an instant messaging tool. As mentioned in Sections 1 and 2, many discussions take place on messaging platforms and our objective is to get closer to where decisions are taken. We therefore propose to record decisions in a conversation-oriented chat with a bot activated on users' demand and that will request the different parts of a decision, complying to the metamodel discussed in Section 3.1. Alternatively, a full decision will be passed in one-shot, strictly following the template in that case. Figure 3 depicts the interaction flow between the user and the system and is composed of the following steps:

1. A user issues a dedicated command within the instant messaging chat room to initiate a transaction.
2. The bot will ask for at least the `Context` and selected `Option` elements.

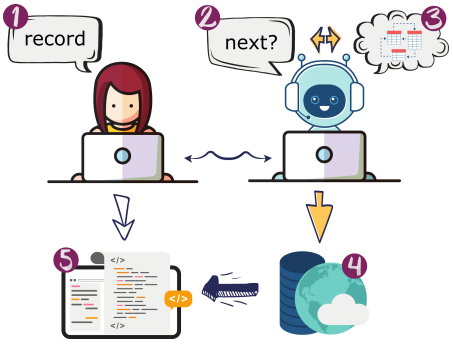

Figure 3: Interaction between a user and the system (numbers correspond to the five steps described in Section 3.2).

3. Each time any user belonging to the same channel submits some details (specifying what item of the decision template it is), the bot will augment its temporary decision model with the supplied details. Thus every participant in a channel may contribute elements to the design decision.
4. When the user confirms, the decision is persisted into the database for later retrieval (each decision is then channel-specific).
5. Using an exporting API, a textual representation (in *Markdown* syntax) is pushed into a wiki that has been set up by the user when configuring the bot for the first time.

The bot can also be used to search existing decisions from within the instant messaging platform. Search queries can be composed of none or multiple keywords that will be searched for in past decisions.

3.3. Exporting API to Wikis

As visible in step 5 of Figure 3, decisions are exported to a wiki in *Markdown* syntax. We decided to use wikis for the following reasons:

- wikis can be manipulated the same way as the source code with full version control. In other words, every time a new decision is pushed, a commit is generated so that a historical record of design decisions is kept with no additional effort required;
- wikis are using simple, yet visually distinguishable syntax that can be interpreted by a wide range of tools, including text-only terminals;
- modern wikis are able to interpret *PlantUML* diagrams[2] for users to model technical aspects of a system (*e.g.*, class diagrams, use cases, components) in a textual, yet readable syntax.

[2] *PlantUML* is a Domain-Specific Language to describe UML models in a concise textual syntax. See https://plantuml.com.

4. Description of Prototype

A prototype for *Slack* has been developed and its sources are openly available[3]. As a sub-optimal solution, the discussion with the bot is emulated through an internal state machine and separate commands (using dedicated "/" notation for custom commands in Slack) instead of a fully bidirectional interaction. In the following, we describe the technical aspects of the setup, the recording mechanism, the search feature and the export to wikis.

4.1. Setting up the Bot

After downloading the bot into a workspace, users will be able to register separate wiki pages for each channel where they want to track decisions from such that each channel will have its independent set of decisions. To this end, users issue a `/setup` command and specify the main url for the wiki, the relative path to the page where the decisions will be collected, a (technical) user name and the access token for that user.

4.2. Recording Decisions

A recording transaction starts with a `/record-decisions` command and can be interrupted at any time using `/abort`. All parts but `/context` and selected `/option` are optional and can be passed in any order by any user in the channel. After each command, the bot will acknowledge and let the user know what parts are not filled yet. A decision is persisted when a user `/confirm`. The available commands are:

`/context` records the use case and optionally a component if a second part is passed to the command (separated with an *"and"*);

`/option` records the selected `Option`;

`/neglected` records the neglected `Options`. As for context, we create separate `Option` instances by cutting around conjunctions and commas;

`/concern` multiple concerns are separated around commas to record multiple `NonFunctional-Requirement`;

`/quality` multiple `QualityAttributes` can be passed as comma-separated keywords;

`/consequence` as for the `/concern` command, `Consequences` are split around commas.

Additionally, a *"one-shot"* `/record-decision` (singular) command allows users to enter a decision following the *(wh)y* template where the NLP engine cuts each part to fill in the decision model. This command currently requires all parts of the decision to be passed.

[3] See https://bitbucket.org/mde4asd/ddextractionbot.

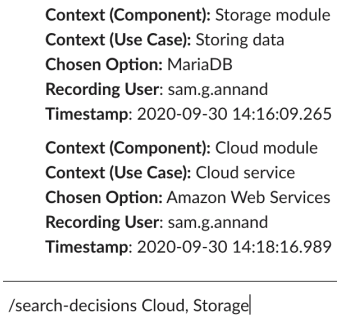

Figure 4: Screen capture of a search query and its results.

4.3. Search Decisions

Since all decisions are persisted in database, users can search for decisions that have been taken in a channel by passing a list of comma-separated keywords, similarly to common search engines. Figure 4 depicts the usage of the `/search-decisions` command and its results. As can be seen, additional metadata are given to contextualise a decision and therefore helps users remember by whom and when a decision has been taken.

4.4. Export to Wiki

After confirming a decision, it will be exported to the configured wiki page. Each channel has its own page to keep decisions organised per channels as the later may be project-specific. Simple *Markdown* syntax has been used for easy access on web-based interfaces, local graphical client editors and terminal-only machines. A simple example of a recorded decision is given in Listing 1.

```
## Context
Sign in to the system
### Chosen option
Single signon with Google
### Neglected options
* email sign in
### Quality attribute achieved
* confidentiality
* integrity
### Consequences
* depending on google SSO API
### Initial text
In the context of Sign in to the system we chose Single signon with Google and neglected email sign
      in to achieve confidentiality and integrity accepting downside depending on google SSO API.
```

Listing 1: Markdown representation of a decision.

As can been seen in Listing 1, a full template-compliant sentence is generated from the parts passed to the bot through separate commands. These reconstructed sentences are used to speed up the search feature and may be useful to generate a summary of all decisions.

5. Discussion

5.1. Evaluation of Usability

In order to evaluate the usability of our framework, we use Nielsen's heuristics [46]. These heuristics are composed of 10 rules of thumb tailored for user interfaces and interactions. We selected Nielsen's heuristics for their popularity, flexibility and applicability to hybrid systems such as our proposal.

Simple and natural dialogue: The first heuristic is about aesthetic and minimalist design, presenting the user with what information they need when they need it. By designing our recording system in the form of a discussion, the user is asked to supply separately and in a logical order each bit of a design decision. Search results are presented in a descriptive, yet compact textual fashion. The process is not fully transparent to users (*i.e.* they still need to follow the template), but there is no need to go into a separate system any more. Still, more work should be put into organising the decisions in the wiki where all decisions are simply listed without any form of grouping.

Speak the users' language: There needs to be a match between the real world and the system. Our prototype has been designed to get closer to real world interactions between developers and stakeholders in the large, where valuable discussions and decisions are taken on instant messaging platforms. Furthermore, the required details to record a decision have been identified from the literature, often rooted in empirical studies involving the industry. Still, the bot brings a disruption in the natural discussion flow, but it is kept to its minimum.

Minimise user memory load: We should not rely on the ability of users to recall everything in order to use the system effectively. In a prior attempt, we expected users to remember the fully-descriptive template, putting a significant cognitive load on them when asking to record decisions. The recording is now handled through a bot-driven conversation where users are required to know the initiating command only, all parts of the decision being asked incrementally.

Consistency: We need to consider two types of consistency: internal and external. The wording of each of the messages displayed is standardised and consistent across commands as well as with the domain model for internal consistency (see Section 3.1). The designed system is very simple, triggered by reserved commands following *Slack* standard for user-defined actions. Interactions with chatbots are usually designed as simple question & answer discussions and our system relatively follows this convention for external consistency. Additionally, the serialised output exported from the API is also following the same naming convention for domain concepts.

Feedback: This is about giving users a visibility of the current state of the system as well as appropriate feedback. This principle drove the design of the recording command discussed in Section 4.2. By design, the bot is interactive, giving feedback after each interaction (*e.g.,* missing

elements from the decision, errors, successful recording). It also summarises what parts have been supplied and what other commands can be passed.

Clearly marked exits: Users are not free from making mistakes, so the system must offer *"emergency exits"*. Within a discussion with the bot, a user can always abort and the decision is recorded only when confirmed at the end of a transaction. Similarly, users can overwrite parts of a decision when a transaction is running. However, a full *"undo-redo"*, including the ability to amend or delete decisions will be considered in future works.

Shortcuts: This heuristic concerns the flexibility of the system and the efficiency of use where more advanced users are able to perform the same actions using *"accelerators"* (*e.g.,* keyboard shortcuts). Currently, the system has a *step-by-step* and a relatively rigid *one-shot* way of interaction with the bot.

Good error messages: Users should be able to recognise, diagnose and recover from errors. The chat-based discussion is designed to discover errors as they occur and poke users for updated answers if necessary, therefore descriptive error messages have been specified at each stage to help users diagnose their mistakes.

Prevent errors: Similarly to appropriate user feedback, preventing errors in first place is important. A decision is eventually recorded after an explicit confirmation by the user. However, more advanced NLP of some parts of answers would help increase the quality of recorded decisions (*e.g.,* detection of synonyms, duplicates). Additionally, unfinished transactions need to be picked up by the bot too.

Help and documentation: In order to make a system usable, it must be accompanied by meaningful documentation. As mentioned in Section 4, the sources of the tool are available and are accompanied with a README file where available commands and the general setup are explained. Furthermore, all commands are self-descriptive by displaying help messages to guide users during the recording process.

5.2. Limitations of Current Prototype

The prototype provides a limited set of features and it is currently not possible to amend existing decisions. Editing decisions on the wiki would result in inconsistencies between the knowledge database held by the bot and what is visible on the wiki.

Current implementation suffers from a few limitations regarding the NLP features. We are able to separate some concepts (*e.g.,* `QualityAttribute`) using conjunctions or commas, but other potentially more complex sentences are kept as-is. Additionally, we are not performing any detection of synonyms or basic typos to reconcile identical or similar elements in the database (*e.g.,* components). Last, extraction of domain concepts is limited to dedicated parts of the template.

The organisation of decisions within the wiki is mostly flat. Similarly to the possibility to amend existing decisions, an intuitive feature should be implemented to sort and/or group decisions directly using the bot. A longer term goal would be to create advanced visualisations (*e.g.,* timeline of decisions, domain knowledge).

Last and more importantly, even if we systematically discussed the usability of the current prototype, it has been evaluated by potential end users in a limited and non-empirical manner. A more in-depth user study should be carried out to evaluate its usability from users' perspective as well as conduct a deeper analysis of the perceived usefulness of its features.

6. Conclusion and Future Work

In this paper, we introduced a design decision recording framework composed of a bot integrated into an instant messaging channel, a simple Natural Language Processing engine and an exporting API to wikis. Our goal is to remove barriers preventing software development teams and project's stakeholders to document their design decisions. By bringing the recording of decisions inside practitioners' day-to-day tools, we postulate that more decisions will be recorded and the quality of documentation will increase. We evaluated the usability of the prototype using Nielsen's state-of-the-art heuristics.

To tackle the aforementioned shortcomings, we plan to extend the framework with a monitoring bot looking after discussions taking place on the chat channels. The current bot is dedicated to create a knowledge base of domain and design decisions taken during the course of a (software development) project. Based on the accumulated knowledge, the monitoring bot will notify users when it identifies changes in recorded decisions, potentially in real time with appropriate NLP features. Following this *step-by-step* process, we intend to explore how we can turn the bot into a fully transparent tool that will silently record decisions.

References

[1] A. Jansen, J. Bosch, Software architecture as a set of architectural design decisions, in: Proc. of the 5th Working Conference on Software Architecture, WICSA, IEEE, 2005, pp. 109–120. doi:10.1109/WICSA.2005.61.

[2] D. E. Perry, A. L. Wolf, Foundations for the study of software architecture, SIGSOFT Software Engineering Notes 17 (1992) 40–52. doi:10.1145/141874.141884.

[3] D. Tofan, M. Galster, P. Avgeriou, W. Schuitema, Past and future of software architectural decisions – a systematic mapping study, Information and Software Technology 56 (2014) 850 – 872. doi:10.1016/j.infsof.2014.03.009.

[4] M. Bhat, K. Shumaiev, U. Hohenstein, A. Biesdorf, F. Matthes, The evolution of architectural decision making as a key focus area of software architecture research: A semi-systematic literature study, in: 2020 IEEE International Conference on Software Architecture (ICSA), 2020, pp. 69–80. doi:10.1109/ICSA47634.2020.00015.

[5] R. Weinreich, I. Groher, C. Miesbauer, An expert survey on kinds, influence factors and documentation of design decisions in practice, Future Generation Computer Systems 47 (2015) 145 – 160. doi:10.1016/j.future.2014.12.002.

[6] P. Kruchten, P. Lago, H. Vliet, Building up and reasoning about architectural knowledge, in: C. Hofmeister, I. Crnkovic, R. Reussner (Eds.), Quality of Software Architectures, volume 4214 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2006, pp. 43–58. doi:10.1007/11921998_8.

[7] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, A. Babar, A comparative study of architecture knowledge management tools, Journal of Systems and Software 83 (2010) 352 – 370. doi:10.1016/j.jss.2009.08.032.

[8] R. Capilla, A. Jansen, A. Tang, P. Avgeriou, M. A. Babar, 10 years of software architecture knowledge management: Practice and future, Journal of Systems and Software 116 (2016) 191 – 205. doi:10.1016/j.jss.2015.08.054.

[9] Z. Alexeeva, D. Perez-Palacin, R. Mirandola, Design decision documentation: A literature overview, in: B. Tekinerdogan, U. Zdun, A. Babar (Eds.), Software Architecture, Springer, 2016, pp. 84–101. doi:10.1007/978-3-319-48992-6_6.

[10] H. van Vliet, A. Tang, Decision making in software architecture, Journal of Systems and Software 117 (2016) 638 – 644. doi:10.1016/j.jss.2016.01.017.

[11] M. Anvaari, R. Conradi, L. Jaccheri, Architectural decision-making in enterprises: Preliminary findings from an exploratory study in norwegian electricity industry, in: K. Drira (Ed.), Software Architecture, Springer Berlin Heidelberg, 2013, pp. 162–175. doi:10.1007/978-3-642-39031-9_14.

[12] S. Dasanayake, J. Markkula, S. Aaramaa, M. Oivo, Software architecture decision-making practices and challenges: An industrial case study, in: Proc. of the 24th Australasian Software Engineering Conference, 2015, pp. 88–97. doi:10.1109/ASWEC.2015.20.

[13] D. Falessi, G. Cantone, R. Kazman, P. Kruchten, Decision-making techniques for software architecture design: A comparative survey, ACM Comput. Surv. 43 (2011). doi:10.1145/1978802.1978812.

[14] D. Tofan, M. Galster, P. Avgeriou, Difficulty of architectural decisions – a survey with professional architects, in: K. Drira (Ed.), Software Architecture, Springer, 2013, pp. 192–199. doi:10.1007/978-3-642-39031-9_17.

[15] S. R. V, H. Muccini, Group decision-making in software architecture: A study on industrial practices, Information and Software Technology 101 (2018) 51 – 63. doi:10.1016/j.infsof.2018.04.009.

[16] C. Zannier, F. Maurer, Comparing decision making in agile and non-agile software organizations, in: G. Concas, E. Damiani, M. Scotto, G. Succi (Eds.), Agile Processes in Software Engineering and Extreme Programming, Springer, 2007, pp. 1–8. doi:10.1007/978-3-540-73101-6_1.

[17] N. B. Moe, A. Aurum, T. Dybå, Challenges of shared decision-making: A multiple case study of agile software development, Information and Software Technology 54 (2012) 853 – 865. doi:10.1016/j.infsof.2011.11.006.

[18] B. Lin, A. Zagalsky, M. Storey, A. Serebrenik, Why developers are slacking off: Understanding how software teams use slack, in: Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion, CSCW '16 Companion, ACM, 2016, pp. 333–336. doi:10.1145/2818052.2869117.

[19] M. Storey, A. Zagalsky, F. F. Filho, L. Singer, D. M. German, How social and communication channels shape and challenge a participatory culture in software development, IEEE Trans-

actions on Software Engineering 43 (2017) 185–204. doi:10.1109/TSE.2016.2584053.

[20] P. Ralph, E. Tempero, Characteristics of decision-making during coding, in: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, EASE '16, ACM, 2016, pp. 1–10. doi:10.1145/2915970.2915990.

[21] M. Drury, K. Conboy, K. Power, Obstacles to decision making in agile software development teams, Journal of Systems and Software 85 (2012) 1239 – 1254. doi:10.1016/j.jss.2012.01.058.

[22] C. J. Stettina, W. Heijstek, Necessary and neglected? an empirical study of internal documentation in agile software development teams, in: Proceedings of the 29th ACM International Conference on Design of Communication, SIGDOC '11, ACM, 2011, p. 159–166. doi:10.1145/2038476.2038509.

[23] A. Jansen, J. Bosch, P. Avgeriou, Documenting after the fact: Recovering architectural design decisions, Journal of Systems and Software 81 (2008) 536 – 557. doi:10.1016/j.jss.2007.08.025.

[24] A. Shahbazian, Y. K. Lee, D. Le, Y. Brun, N. Medvidovic, Recovering architectural design decisions, in: IEEE International Conference on Software Architecture (ICSA), 2018, pp. 95–9509. doi:10.1109/ICSA.2018.00019.

[25] M. Bhat, K. Shumaiev, A. Biesdorf, U. Hohenstein, F. Matthes, Automatic extraction of design decisions from issue management systems: A machine learning based approach, in: Software Architecture, Springer, 2017, pp. 138–154. doi:10.1007/978-3-319-65831-5_10.

[26] P. Kruchten, P. Lago, H. Vliet, Building up and reasoning about architectural knowledge, in: C. Hofmeister, I. Crnkovic, R. Reussner (Eds.), Quality of Software Architectures, volume 4214 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2006, pp. 43–58. doi:10.1007/11921998_8.

[27] M. Sintzoff, Suggestions for composing and specifying program design decisions, in: B. Robinet (Ed.), International Symposium on Programming, Springer, 1980, pp. 311–326. doi:10.1007/3-540-09981-6_22.

[28] P. Kruchten, The 4+1 view model of architecture, IEEE Software 12 (1995) 42–50. doi:10.1109/52.469759.

[29] J. Tyree, A. Akerman, Architecture decisions: Demystifying architecture, IEEE Software 22 (2005) 19–27. doi:10.1109/MS.2005.27.

[30] M. Che, D. E. Perry, Scenario-based architectural design decisions documentation and evolution, in: 2011 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, 2011, pp. 216–225. doi:10.1109/ECBS.2011.16.

[31] M. Soliman, M. Riebisch, Modeling the interactions between decisions within software architecture knowledge, in: P. Avgeriou, U. Zdun (Eds.), Software Architecture, Springer, 2014, pp. 33–40. doi:10.1007/978-3-319-09970-5_3.

[32] M. Bhat, K. Shumaiev, A. Biesdorf, U. Hohenstein, M. Hassel, F. Matthes, Meta-model based framework for architectural knowledge management, in: Proccedings of the 10th European Conference on Software Architecture Workshops, ECSAW '16, ACM, 2016, pp. 1–7. doi:10.1145/2993412.3004848.

[33] U. Zdun, R. Capilla, H. Tran, O. Zimmermann, Sustainable architectural design decisions, IEEE Software 30 (2013) 46–53. doi:10.1109/MS.2013.97.

[34] N. B. Harrison, P. Avgeriou, U. Zdun, Using patterns to capture architectural decisions, IEEE Software 24 (2007) 38–45. doi:10.1109/MS.2007.124.

[35] M. T. T. That, S. Sadou, F. Oquendo, Using architectural patterns to define architectural decisions, in: 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, 2012, pp. 196–200. doi:10.1109/WICSA-ECSA.212.28.

[36] S. Farshidi, S. Jansen, J. M. van der Werf, Capturing software architecture knowledge for pattern-driven design, Journal of Systems and Software 169 (2020) 110714. doi:10.1016/j.jss.2020.110714.

[37] A. Cimitile, F. Lanubile, G. Visaggio, Traceability based on design decisions, in: Proceedings Conference on Software Maintenance 1992, 1992, pp. 309–317. doi:10.1109/ICSM.1992.242530.

[38] D. Dermeval, J. Pimentel, C. Silva, J. Castro, E. Santos, G. Guedes, M. Lucena, A. Finkelstein, Stream-add - supporting the documentation of architectural design decisions in an architecture derivation process, in: 2012 IEEE 36th Annual Computer Software and Applications Conference, 2012, pp. 602–611. doi:10.1109/COMPSAC.2012.81.

[39] F. Gilson, V. Englebert, Transformation-wise design of software architectures, in: S. Hammoudi, L. Pires, J. Filipe, R. das Neves (Eds.), Communications in Computer and Information Science, volume 506, 2015, pp. 49–65. doi:10.1007/978-3-319-25156-1_4.

[40] V. R. Basili, H. D. Mills, Understanding and documenting programs, IEEE Transactions on Software Engineering SE-8 (1982) 270–283.

[41] S. Rugaber, S. B. Ornburn, R. J. LeBlanc, Recognizing design decisions in programs, IEEE Software 7 (1990) 46–54. doi:10.1109/52.43049.

[42] H. Astudillo, G. Valdés, C. Becerra, Empirical measurement of automated recovery of design decisions and structure, in: 2012 VI Andean Region International Conference, 2012, pp. 105–108. doi:10.1109/Andescon.2012.33.

[43] J. S. van der Ven, J. Bosch, Making the right decision: Supporting architects with design decision data, in: K. Drira (Ed.), Software Architecture, Springer, 2013, pp. 176–183. doi:10.1007/978-3-642-39031-9_15.

[44] C. Lebeuf, M. Storey, A. Zagalsky, Software bots, IEEE Software 35 (2018) 18–23. doi:10.1109/MS.2017.4541027.

[45] ISO/IEC, ISO/IEC 25010 System and software quality models, Technical Report, Internation Organization for Standardization/International Electrotechnical Commission, 2010.

[46] J. Nielsen, Usability Engineering, Morgan Kaufmann Publishers Inc., 1993.