University of São Paulo
Institute of Mathematics and Statistics
Bachelor of Computer Science

# Sorting Hat

## *A Tool to Characterize the Architecture of Service-Based Systems*

Erick Rodrigues de Santana

Work Proposal

mac 499 — Capstone Project

Supervisor: Prof. Dr. Alfredo Goldman vel Lejbman
Co-supervisor: Prof. Dr. André van der Hoek (University of California, Irvine)
Co-supervisor: João Francisco Lino Daniel

During the development of this work, the author will be receiving financial support from FAPESP – grant #2020/16577-9 and #2021/14240-0.

Sao Paulo
2022

# Contents

# Chapter 1

# Introduction

Software architecture is an area that studies important concepts and practices to ensure the quality and success of a software system, hence improving its chances to succeed as project. A good architecture allows you to have a broad view on a software system and its future evolution, and to estimate the infrastructure costs and delivery times. It also influences the team organization and is the basis of system organization Bass *et al.*, 2013. However, a software developed without a good architectural plan might be hard to keep up during its evolution. A poor architecture is a major cause of impediment for the developers to understand the software Fowler, 2019.

Even with an architecture plan, it might be a challenge to architects and engineers to keep the system in accordance with it. Throughout the development process, architectural deviations can happen, that is, decisions during the implementation that hurt the planned architecture Perry and Wolf, 1992. Therefore, it is valuable to keep the backlog of the current architecture and its evolutions, because it allows the architects to identify those deviations and to make the right decisions.

There are architectural styles that can guide an architecture. The microservice architectural style has been used frequently in the context of software development. Although this style has some benefits, there are challenges faced in its implementation. Those challenges are strongly linked to the complexity of microservice-based applications Soldani *et al.*, 2018. As examples, there is the difficulty to determine the microservices' granularity and the distributed storage management Soldani *et al.*, 2018. In that scenario, the deviations can be potentiated, in a way that characterizing the current architecture of a system can be difficult and expensive to the development team.

In order to help software architects in the process of decision-making in favor of evolution of their systems, it is being developed the Sorting Hat – a tool to characterize the architecture of service-based systems. In the MVP (minimum viable prototype), it displays the system's components, their characteristics and the relationship between them. It also displays some metrics based on the characterization model called CharM, which is under development by Rosa *et al.* (2020). The data available on Sorting Hat was collected manually, which was a strenuous and a computationally automatable process.

The main goals of this work proposal for the discipline MAC0499 (Final Capstone) are

two. The first one is to provide an automated system data collection to the Sorting Hat, applying the metrics of the model under development by Rosa *et al.* (2020) to data collected. The second one is to improve the visualization of the tool and update its information to the most recent metrics in the CharM model.

This work is part of a undergraduate research sponsored by FAPESP that has been ongoing since February 2021. Since March 2022, the student has been carrying out a research internship at the Department of Informatics of the University of California, Irvine (UCI) advised by the Professor Dr. André van der Hoek, who has expertise in Software Engineer. This research internship ends on July 31st. Therefore, the first half of the work on this proposal will be developed as part of the research internship.

The rest of this proposal is structured in the following way: the Chapter 2 presents a conceptual base for this project. The Chapter 3 shows the tools' current state, presenting its structural aspects. The Chapter 4 presents the goals of this project, the methodology adopted to achieve them, and the schedule that will be adopted. Finally, the Chapter 5 describes the university where the student will do some parts of this work, and the foreign advisor.

# Chapter 2

# Background

Software architecture is a set of needed structures to the comprehension of a system, covering software elements, the relationship among them and their properties Bass *et al.*, 2013. It is also a discipline within software engineering that serves as a bridge between software requirements – functional and non-functional – and the software implementation. The software development process guided by a planned architecture facilitates aspects of comprehension, development, maintenance, and system evolution, as well as assists in the team's decision-making.

An important role to be played by development team members is that of software architect. They are responsible for knowing the business domain, understanding the development process, knowing technologies and methodologies, as well as having good abilities with programming. That knowledge is essential, because the software architect is also responsible for making decisions in uncertain contexts that affects the structure and quality of a system.

To guide the software architects and engineers in the conception of software elements and their interactions, it is common the adoption of architectural styles.

> *"An architectural style, then, defines a family of such systems in terms of a pattern of structural organization. More specifically, an architectural style determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of constraints on how they can be combined."* Garlan and Shaw, 1994.

The microservice architectural style is described as an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, preferably asynchronous Fowler and Lewis, 2014. According to Newman, 2015, some main characteristics of that architectural style are the following:

- Small services and limited by business domain;
- Low coupling among services;
- Fault tolerant services;

- Technological heterogeneity;
- Automated deploy infrastructure.

However, developing systems following that architectural style is not trivial. Some questions raised by Newman, 2015 are:

- How to define the size and the responsibilities of each service?
- How to ensure the data consistency among services?
- How to keep a low level of coupling among services?

Those questions can lead to uncertainties when implementing the system. In this sense, architectural deviation can occur, that is, inconsistencies caused when implementing the planned architecture Perry and Wolf, 1992. To restore the planned architecture, there is the concept of architectural recovery, which consists in using reverse engineer techniques to extract the implemented architecture from code artifacts Silva and Balasubramaniam, 2012.

To evaluate the current architecture of a microservice system, a model to characterize service-based architecture has been developed by Rosa *et al.*, 2020 and it is called CharM. That model, which is in constant evolution, currently has as its central point the following dimensions:

- **Modules' size**: the model considers a module as a service deploy unit, that is, the size of a module is the number of services that comprise it;
- **Data sources shared by modules**: the model considers the data source distribution among modules, indicating whether a data source is used by one or more modules;
- **Synchronous coupling among services**: it measures the synchronous dependency level among services of a system, counting the number of synchronous communications among services;
- **Asynchronous coupling among services**: it measures the asynchronous dependency level among services of a system, counting the number of asynchronous communications among services.

# Chapter 3

# Current State

The Sorting Hat's development is guided by evolutionary prototypes. The first MVP is the frontend focused on displaying a system's metrics, which can be accessed at https://the-sortinghat-front.herokuapp.com/.

Sorting Hat currently only data from InterSCity – a platform following the microservice architectural style that assists smart-cities applications. The data were collected in a case study to validate the characterization model of Rosa *et al.*, 2020. In this case study, the data were manually collected by inspecting the source-code. This enabled to develop the frontend without having the automated data collector.

The Sorting Hat's frontend is composed by 3 levels of visualization: a system, a module of the system, and a service of the module. There is a page for each one of these levels. The pages of a system and a module have two views: the graph-based view, which shows the synchronous and asynchronous communications between the modules or services; and the metrics list, which shows all the metrics that belong to that level. Figures 3.1 and 3.2 present these views.

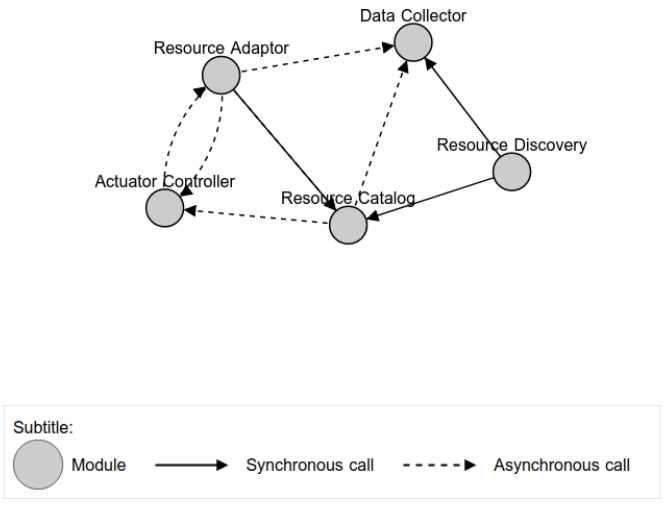


**Figure 3.1:** *Graph-based view of InterSCity's modules.*

It is important to note that, although it does not automatically collect data from systems, the views generated by the frontend may already be pertinent to the architects in future decisions.

The graph visualization makes explicit the synchronous connections. By observing this, the architects can make decisions to keep them or to migrate them to asynchronous, which are more consistent with the architectural style of microservices. Synchronous communications can generate dependency between services at runtime and increase coupling between them.

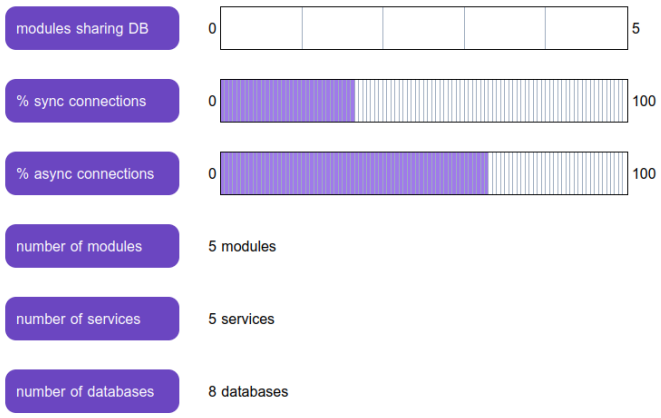


**Figure 3.2:** *Metrics list view of InterSCity.*

Viewing the list of metrics can also assist architects in making of decisions. For example, Figure 3.2 illustrates the metric of modules sharing database, which says the number of modules that share the same database. Sharing databases can generate inconsistencies, going against the adopted practices in the architectural style in question. Thus, if Figure 3.2 shows that there are modules sharing databases, architects could decide to remove these shares or to keep them if necessary.

The platform frontend architecture is represented in Figure 3.3. It illustrates an overview of the page structure, showing the user's navigation flow. Figure 3.3 also presents the components that the pages use, as well as the interaction with external elements, such as the backend, responsible for obtaining the data from the systems available on the platform.

To develop it, we used Nuxt.js, a framework for the construction of SPAs (single-page applications), a modern way of developing websites in which navigation between pages is made in a more fluid way, without the need for the browser to perform a new request to the server where the site is hosted with every page change. The construction of SPAs is also based on components, a part of the application that is common to several pages and therefore can be reused. Also, Nuxt.js allows the developer to use plugins – libraries or modules external to the application and that are made available to it.

In Figure 3.3, the three pages mentioned above are represented, which show the different viewing levels. The pages of a system and a module use the graph and metric components. For the graph visualization, we used the D3.js library as a plugin, which

allowed the construction of customized graphs from the raw data that the application intends to display.

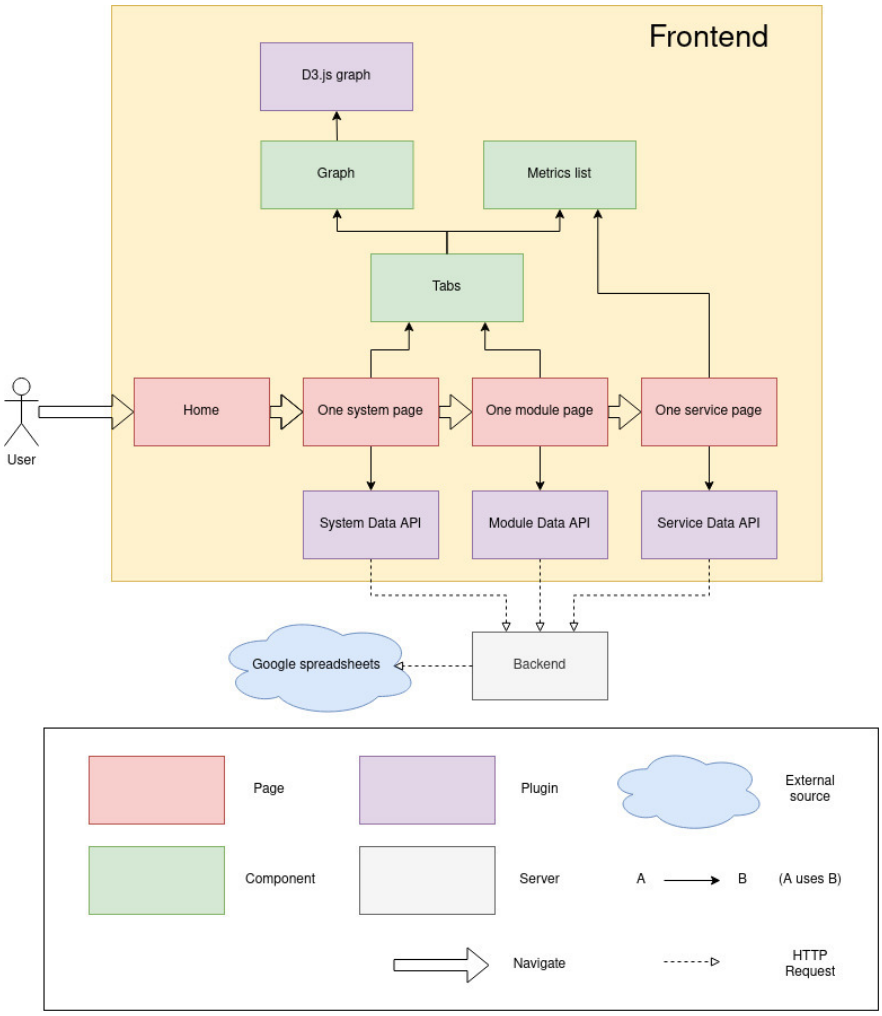


**Figure 3.3:** *Sorting Hat's frontend architecture.*

In this MVP, InterSCity data is available. These data were stored in Google spreadsheets. For the platform to get them, some abstractions were created: each of the three pages uses a plugin to request the data from the backend and transform it to the way the pages understand it. In this way, each of the plugins acts as an abstraction from the server. The backend has a similar goal: it requests data from Google spreadsheets and transforms it for objects that are understood by each of the plugins. All these abstractions were created with the objective of modularizing the application, providing testability gains, decoupling and scalability.

The platform is open source licensed and the source code is available at: https://github.com/the-sortinghat/frontend. The frontend currently is being developed by two developers and has over 2700 lines of code written. The MVP of frontend was presented in the fourth edition of the InterSCity Workshop.

# Chapter 4

# Proposal

## 4.1 Goals

The data collection for the Sorting Hat is still done manually, making this process harder. Also, there are some issues related to the frontend: its usability is a bit complicated, with many contexts switching, and the metrics of the model under development by Rosa *et al.* (2020) are not up to date. The goal of this work for MAC0499 (Capstone Project) goes in two different yet complementary directions: provide an automated data collection to the Sorting Hat; and make the frontend more intuitive, easy to use and fluid to work with, as well as update its information to the latest metrics from the CharM model.

## 4.2 Methodology

An overview of the methodology that will be adopted in this work is illustrated in the Figure 4.1.

In the Study stage, the goal is to study some tools that will guide the implementation of the metrics collector. First, we need to map each metric from CharM model to the data we need to collect, and ideate ways to collect them. Next, we will search for tools and existing researches on static and dynamic data collection, and select some of them to assist in implementing our automated data collector. This will be an important step because we need to identify which metrics from CharM model we are able to collect and which we are not.

In the implementation stage, we will develop the metrics collector. The collector will use the selected tools to get all the necessary data. After getting the system data, the collector will apply the CharM metrics and those data will be accessible through an API.

In parallel with the development of the metrics collector, we will improve the Sorting Hat's frontend. First, we will make a prototype in order to see if we improve the tool's usability, and after that, we will develop the improved parts in the Sorting Hat's frontend. Second, we will get the updated metrics from CharM model and update the information in the frontend. Also, we will develop new pages to enable the systems registration.

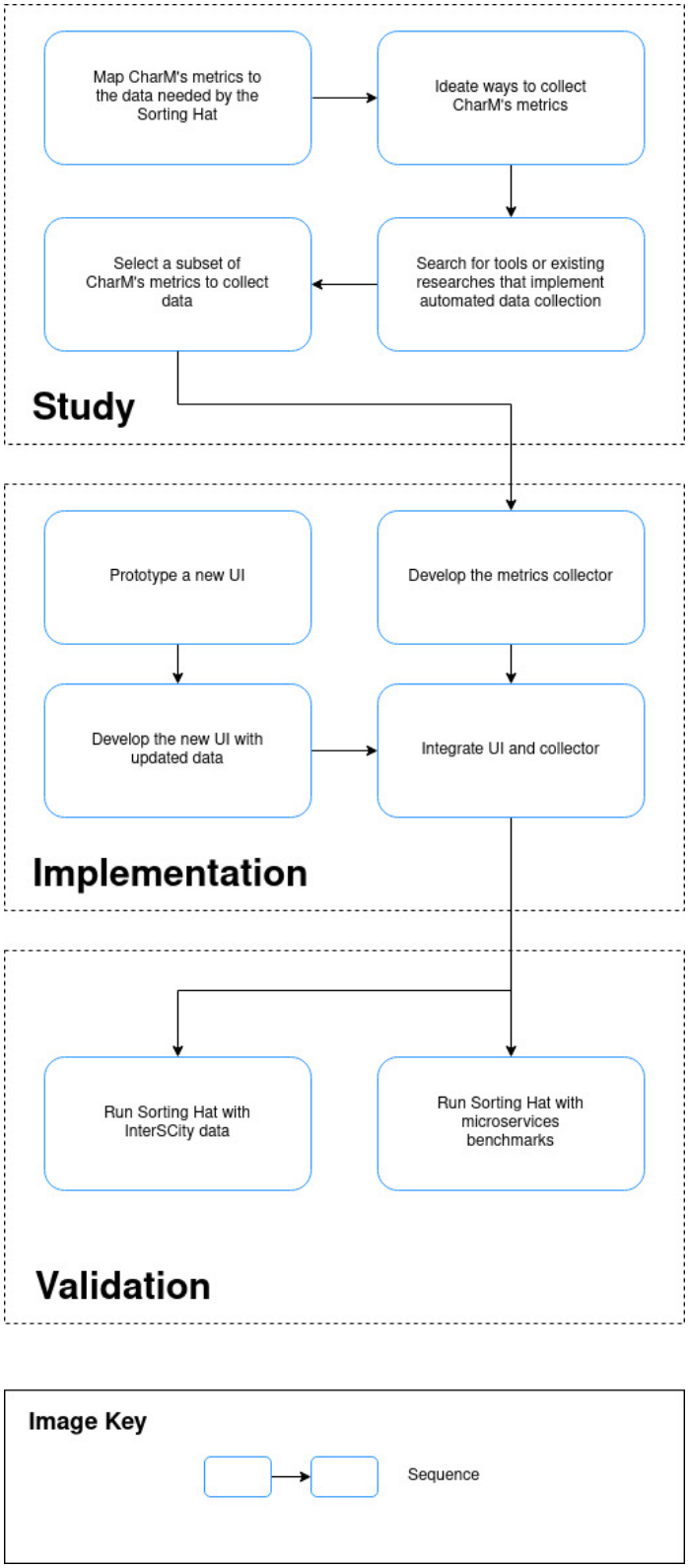


**Figure 4.1:** *Methodology of the proposal*

Having the metrics collector and the updated frontend, the next step is to connect these two parts, so that they can work together.

Finally, it is necessary to validate the Sorting Hat's efficacy and accuracy when working with real-world systems. First we will test the improved version of Sorting Hat with the InterSCity platform and compare the results with the data manually collected. Second, we will run the tool with a few microservices benchmarks to check if the results produced by the tool are relatively close to the prescriptive architecture of those benchmarks.

## 4.3 Schedule

Following the methodology proposed in the Figure 4.1, we are proposing the following tasks:

1. (Study) Map CharM's metrics to the needed by the Sorting Hat.
2. (Study) Ideate ways to collect CharM's metrics.
3. (Study) Search for tools or existing researches that implement data collection.
4. (Study) Select a subset of CharM's metrics to collect.
5. (Implementation) Develop the metrics collector.
6. (Implementation) Prototype and develop a new UI with updated data.
7. (Implementation) Integrate UI and collector.
8. (Validation) Validate results with InterSCity and microservices benchmarks.
9. Write final monograph.

The Table 4.1 shows these tasks and the time that will be spent to do them.

| Tasks | Months | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Mar | Apr | May | June | July | Aug | Sep | Oct | Nov |
| 1 | | | | | | | | | |
| 2 | | | | | | | | | |
| 3 | | | | | | | | | |
| 4 | | | | | | | | | |
| 5 | | | | | | | | | |
| 6 | | | | | | | | | |
| 7 | | | | | | | | | |
| 8 | | | | | | | | | |
| 9 | | | | | | | | | |

**Table 4.1:** *Work plan's schedule.*

# Chapter 5

# About University of California (Irvine) and Prof. Dr. André van der Hoek

The University of California at Irvine - UCI, is an American public university that believes that real progress is made when different perspectives come together to promote a better understanding of the world. According to the Tomes Higher Education ranking, in a global analysis, in 2020 the UCI occupies 96th position. When considering only the American institutions, the UCI occupies the 40th position, in the same ranking. The objectives of the UCI Department of Informatics are: to create innovative information technologies that meet the diverse needs of society, and to educate its students to be entrepreneurial leaders. The main values cultivated by the department are: creativity, engagement, and the establishment of partnerships.

Professor Dr. André van der Hoek is co-author of the books “Software Design Decoded: 66 Ways How Experts Think” and “Studying Professional Software Design: a Human-Centric Look at Design Work”, the only published books that detail the practices of software design professionals. He was recognized as Outstanding Scientist by the ACM in 2013 and received the Award for Excellence in Engineering Education Courseware in 2009. Additionally, in 2005, he was honored Professor of the Year at UCI for his outstanding and innovative educational contributions. Additional information about the work of Professor André van der Hoek can be found in his profile at Google Scholar and in DBLP.

Therefore, considering the vast experience that Professor Dr. André van der Hoek has in the areas of Software Architecture and Experimental Software Engineering, we believe that he will be able to contribute to the enrichment of our project. Through his in-depth knowledge of the day-to-day life of software architects, he will be able to contribute to making the Sorting Hat platform a really practical and useful guide for software development professionals. Furthermore, through the values cultivated by the UCI Department of Informatics and the Software Design and Collaboration Laboratory, our project can be enriched by exchanging experiences and ideas with other students.

# References

[Bass *et al.* 2013] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. 3rd. Addison-Wesley, 2013 (cit. on pp. 1, 3).

[Fowler 2019] Martin Fowler. *Software Architecture Guide*. Url: https://martinfowler.com/architecture/, 2019 (cit. on p. 1).

[Fowler and Lewis 2014] Martin Fowler and James Lewis. *Microservices*. Url: https://martinfowler.com/articles/microservices.html, 2014 (cit. on p. 3).

[Garlan and Shaw 1994] David Garlan and Mary Shaw. "An introduction to software architecture". In: *Carnegie Mellon University* CMU-CS-94-166 (1994) (cit. on p. 3).

[Newman 2015] Sam Newman. *Building Microservices: Designing Fine-Grained System*. 1st. O'Reilly Media, 2015 (cit. on pp. 3, 4).

[Perry and Wolf 1992] Dewayne E. Perry and Alexander L. Wolf. *Foundations for the Study of Software Architecture*. ACM SIGSOFT Software Engineering Notes, 1992 (cit. on pp. 1, 4).

[Rosa *et al.* 2020] T. Rosa, A. Goldman, and E. Guerra. *Modelo para Caracterização e Evolução de Sistemas com Arquitetura Baseada em Serviços*. Workshop de Teses e Dissertações do CBSoft - WTDSoft 2020, 2020 (cit. on pp. 1, 2, 4, 5, 9).

[Silva and Balasubramaniam 2012] Lakshitha de Silva and Dharini Balasubramaniam. *Controlling software architecture erosion: A survey*. Journal of Systems and Software, 2012 (cit. on p. 4).

[Soldani *et al.* 2018] Jacopo Soldani, Damian Andrew Tamburri, and Willem-Jan Van Den Heuvel. "The pains and gains of microservices: a systematic grey literature review". In: *Journal of Systems and Software* 146 (2018) (cit. on p. 1).