WEB & DATA MANAGEMENT SYSTEM FOR THE TRACKING PROCESS OF PRODUCT VALIDATION TEAM IN SYNOPSYS CHILE

ANTINOPAI ARAYA, CARLOS ANDRÉS

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO- CHILE

WEB & DATA MANAGEMENT SYSTEM FOR THE TRACKING PROCESS OF PRODUCT VALIDATION TEAM IN SYNOPSYS CHILE

CARLOS ANDRÉS ANTINOPAI ARAYA

DEGREE PROJECT TO QUALIFY TO THE DEGREE OF
INGENIERO CIVIL ELECTRÓNICO

Supervisor	:	GONZALO CARVAJAL B.
Co-supervisor	:	ESTEBAN VIVEROS S.
Co-supervisor	:	ROBERTO MORALES M.

JUNE 2021

ACKOWLEDGEMENTS

I am grateful to all the people involved in this project.

Support provided by the supervisor and co-workers was essential to develop this project.

Contributions to this document:

- Gonzalo Carvajal. (gonzalo.carvajalb@usm.cl)
- Roberto Morales. (rmorale@synopsys.com)
- Esteban Viveros. (estebanv@synopsys.com)
- Antonio Vasquez. (vasquez@synopsys.com)

Thank you!

ABSTRACT

Synopsys is a company focused on the development of new technologies in the area of Electronic Design Automation (EDA), Intellectual Property (IP) and Software Integrity (SIG). In the EDA area, Synopsys' flagship products are software tools for the the design and optimization of digital electronics schematics. These automation tools include: Design Compiler (DC), Descartes (DCRT), Fusion Compiler (FC), among others.

Among the teams in charge of EDA in the company, there is the Product Validation (PV) team. One of the focuses of the PV team is to ensure the quality of the company's tools, so other Research and Development (R&D) teams in the company can be more productive. To do so, the PV team has at its disposal some monitoring tools, how-to knowledge regarding the tracking of found issues and means to report the activity of PV employees, while other teams such as the aforementioned R&D teams create improvements or new features to the previously mentioned automation tools. More specifically, the PV team creates specific Testcases for each of Synopsys' clients designed to simulate their typical usage of the automation tools, so a better quality control can be made for them. These Testcases and their infrastructure are known as a *Suite*. An employee member of the PV team can be assigned as a *Suite maintainer*, which means he is responsible of identifying, reporting and monitoring issues found during the quality control process of the Suite until those issues are fully resolved, dismissed or at least acknowledged. This process is known as *Tracking*.

The Tracking process is currently associated to various tasks that are performed manually. Automated reports from every Testcase require repetitive and iterative interpretation from the Suite maintainer. Reporting issues requires to manually copy important metrics for debugging to a report system, and at the same time sending those metrics through email to multiple people. On top of that, management must compile these reports in order to have a top level picture of the overall status of the Suites. All these processes are recognized to be inefficient and prone to errors, but PV team focus all its resources to handle the testing and validation of the always growing list of new features developed by R&D teams, so no time is left to add improvements to the Tracking process.

This project aims to mitigate these aforementioned issues by creating a web interface where the PV team can clearly observe the current and past status of the Suites. This web interface requires a system to gather, process and store the data related to Suites, so a data management system is also part of the project. By developing this web and data management system, PV team will hopefully spent less time doing repetitive work like reporting and transcribing data, and more time focusing in finding the root cause of issues in each Suite.

Contents

List of Tables

List of Figures

1 | Introduction

This report summarizes the process to prepare a web and data management system for the tracking process of the PV team in Synopsys Chile. This chapter will first contextualize the issues of the process with a mock-up example of the current workflow experience by most PV employees. Then, a more clear definition of the problems that this project aims to reduce and the solution designed to those problems will be presented. Later, the requirements discussed with the representatives of the PV team will be presented along the restrictions to development that are involved when working with the company. Finally, the organization of the report is described to aid the reader of this document understand the workflow of the project.

1.1 Context and motivation

This project started as an internal project for the company Synopsys Chile. The author of this project worked as an intern in the company for the PV team during the summers of 2019 and 2020. During the internship of 2020, conversations were held with PV team related to the problems in the process of tracking the appearance, investigation, and resolution of issues caused by code changes in the tools created and licensed by Synopsys. The tools needed to execute this tracking process already exists, but they vary between specific methodologies and software tools to detect the occurrence of code issues, the investigation to find the cause of these issues, and the proper reporting of the status of this process from the beginning up to the resolution of said issues. Also, as the PV team is in charge of the testing and validation of code changes, their pending tasks increment along the growing number of changes added to the Synopsys software tools, so there has never been time and human resources to allocate for developing internal tools to ease the workflow of the team. Therefore, the internship was extended by the company to until April 30th 2020, as an opportunity to allocate resources for this missing internal development, given that the intern has a background both in digital electronics and software engineering.

To illustrate the workflow within the PV team, a representative example of the typical weekly routine of a Suite owner will be narrated next.

Carlos is a PV employee who oversees tracking the issues of a Suite made for the client *MAPPLE*, a computer company who designs the chips of its own products. MAPPLE bought a license to use Synopsys' DCNXT software to help develop the chip processors of its product line of smartphones. This Suite, called the MAPPLE Suite, includes the digital plans for a variety of chip designs from both Synopsys and MAPPLE, all selected to test DCNXT in situations MAPPLE may encounter while using this piece of software. To do so, MAPPLE Suite includes different setups of DCNXT that are configured to simulate the usage MAPPLE often applies to DCNXT. The execution of these simulations is made per design and may take from some hours to a couple of days to complete. A version of DCNXT is released for MAPPLE to use every three months, containing all the approved changes done during that time.

During Monday, Carlos checks in a web page known as *the gray page* the results of the simulations launched during last Friday, which were designed to test the changes added during that same day to the DCNXT software. Carlos then realizes that two designs had simulations that ended in failure and another design had a simulation that presented worst optimization results compared to the version MAPPLE is currently using, so he leaves a note in the gray page acknowledging those issues while he investigates the root causes of each one. Let us call these issues as A, B and C. Carlos then continues by launching extra simulations for issue A, with a special configuration that will reveal which employee introduced the changes that caused the failure. This simulation will test individually all the changes introduced during Friday. Changes submitted to integration with Synopsys' tools during a regular day of work may vary between 10 to 100 submission per day, so the results of Friday may take many hours to be ready. Carlos then reports that issue A was found. For this, he uses a web page called 'Jira', where all the issues are reported and tracked in a format known as *STAR*, and he must be checking periodically the issue reported in this web page so he can update the issue status.

Also during the same day, Carlos should already add what he did to his weekly report, which is requested by its manager to follow Carlos activities every Thursday, but he may leave it for tomorrow, since he has issues B and C to investigate. Carlos then proceeds to investigate issue B by doing what he previously did, but the results of an investigation started last week (related to another issue different from A, B and C) are ready, so he updates the corresponding STAR and notifies the corresponding employee who likely provoked the issue by introducing a bad change to the DCNXT software.

Carlos then continues to investigate issue B, but an urgent email from a member of another team arrives requesting an update from an issue reported two weeks ago (again, an issue different from A, B, and C), so Carlos checks the progress of the corresponding STAR and emails back. Carlos then realizes he reported the same information to another member from another team some days ago.

Carlos then proceeds the investigation of issue B, but an employee from another team emails him asking for some simulation data so that this employee can fix an issue reported in a STAR from last Tuesday, so Carlos gets the related data and communicates it to this employee.

Carlos now can finally continue investigating issue B, and later on issue C, which took him until past working hours as he kept being interrupted by email request of reports from different teams and managers.

After Carlos finishes, later during that Monday's night, he talks to his co-worker Andrés about their days. In this conversation, he learns that Andres found some very similar issues in his Suite for the client *SMARTSUNG*, which also took him a while to investigate. After speaking to other co-workers and managers, they all realize issues A and B, and the issues form his coworkers were all likely related to the same root cause, and were reported individually by many of them through many emails and STARs, so they must close those STARs an clarify some emails as all those are reporting the same issue, which is redundant and consumes valuable time from other employees. Only issue C was a new real issue found today.

The next day, Carlos checks the *gray page* for the results of the simulation launched during Monday for the changes added that day, and he again starts acknowledging and investigating the issues he finds while reporting them to the many managers, teams and through the many interfaces he is encouraged to use.

By Thursday, he must write his weekly report, so he collects all the related information from issues A, B, C and other issues from the rest of the week that are scattered through emails, STARs and web interfaces and starts summarizing it for redaction.

1.2 Problem and proposed solution

As shown in the previous example, there is a lack of standardization in the process of reporting the identification of issues, the response to them, and the daily schedule and activities made by a Suite maintainer. This lack of standardization also precludes managers to get a clear overall view of the current status of the Suites.

The lack of standardization in the tracking process is a consequence of the absence of internal development of software tools to address the specific needs of the PV team. Reporting is done manually by copying the results of the tracking process to various reports or even manually transcribing relevant results, which is time consuming and very prone to errors. Some members in charge of the tracking process of certain Suites have developed, or are developing, custom tools to automate the reporting process. However, although the intention is to escalate these solutions to the whole team, the reality is that the current time spent in the reporting of the tracking process does not allow a formal, polished nor integrated further development of these solutions. Also, the possibility of outsourcing this development is not an alternative, as there is manipulation of client data which often is protected by non-disclosure agreements.

The solution developed during this project approached these issues by introducing a centralized web interface for both Suite maintainers and managers. Using the developed management system, they can check the results of automated analysis of the Suites, while reporting findings they find useful in the same web page. Also, a data management system to store and analyze tracking related information was implemented, so that PV employees can get a detailed picture of the tracking process and new information, such as correlation between issues.

1.3 Scope and contributions

This software project aims to simplify the process of tracking issues related to the development of new features and enhancements made by R&D teams inside the company. As the project implies the development of software tools that manage sensitive data inside the company, there are restrictions associated to the design and implementation details included in this report.

In the following, we summarize the restrictions and the scope of this report, and summarize the main contributions of this project to the company.

1.3.1 Restrictions

Synopsys provides software tools for digital design to clients such as *Samsung, Qualcomm, Intel, Apple*, among others. The tools involve the storage and processing of sensitive data associated to its clients, and as such, there are various non-disclosure agreements (NDAs) to protect his information. For similar reasons, there is also an interest from Synopsys to not reveal information about most of the tools used internally by the company.

As a consequence of the above, this project was developed following the data protection and security protocols established by Synopsys, and there was a previous agreement between the student, Synopsys staff, and professor to not reveal sensitive information about the final tool. The practical work associated to the design and implementation of the framework was performed at Synopsys' facilities, following their protocols and guidelines. Final validation and testing of the implemented framework was performed by specialized Synopsys' staff and potential final users of the developed tools, and also discussed with the supervising professor following Synopsys' guidelines. This report only summarizes part of the process to give some context and provide details on general steps and parts that can be publicly shared.

Working for Synopsys implied sticking to the restrictions of any employee. That is, the use of software in general is limited to the ones already available in the Synopsys environment. Also, open-source tools usage required to first discuss with the PV representatives if they may allow them, and always making sure said tools do not leak any data form the company or its clients. These restrictions restricted the alternatives to software already available in the environment of the company.

1.3.2 Requirements

Regarding the requirements for the developed framework, they come from specific members of PV team to clarify objectives and resource utilization. The project requires to understand concepts of digital electronics, as it involves gathering, processing and displaying metrics related to the digital design of circuits that the EDA tools developed by Synopsys are supporting. Such metrics involve data from digital cells interconnected in a circuit, which includes but is not limited to negative slack, power consumption, surface area and critical path. During the gathering of this data is not enough to just save it, but to add value to the project this data is processed to find relation between the individual issues revealed by the metrics from each experiment, which are related directly to specific digital design used for testing, and the overall issues of a Suite or even a *Tool*. Automating the process of noticing that similar metrics are showing degradation or suspicious improvements across the same Suite, version of the Tool or setup, represents an improvement to the workflow of PV and R&D teams investigating the issues.

Objectives for development are defined using a software engineering approach, which in the end meant dedicating six weeks to gather requirements by talking to PV members and defining those requirements as clear goals that could be programmed and reviewed one by one while integrating them to the final software, which consist of a web interface and data management system.

1.3.3 Contributions

After finishing the project, the tools developed for this project were delivered to the company in a working state. On the one hand, a working database that translated the concepts used in the Tracking process to inter-related tables of data, with also the means to gather the necessary data and populate the database. On the other hand, a working web page interface with filtering capabilities of the stored data, summaries and some data processing. Both services, the data base and the web interface, were left available for usage inside the team, with only requiring periodic maintenance to keep the servers online and solve any corrupt data caused by wrong input or bugs.

The software tool designed to gather data from computer farms and Suites was later recycled to feed new data to *Scheduler*, another internal project of Synopsys that allows R&D teams to execute experimental test of code changes with the tools usually used by PV teams, but with a web interface that facilitates the setup. Also, as the database was kept running and being feed with data during months, that data was then used to generate reports of the historical usage of the computing farms by PV team experiments and other experiments by R&D teams launched through the Scheduler service.

1.4 Organization of the report

This report is organized in five chapters as follows:

- **Chapter 1** presents the problems identified in the Tracking process of PV team, while also establishing scope and restrictions determined by the nature of working in a company like Synopsys. Also, a motivational example is narrated to better understand the Tracking process without technical background in the matter.
- **Chapter 2** provides an explanation of the proven methods of software engineering, where the concepts of requirement engineering and the life cycle of a software will be explored. Then, a comparison will be made between the available software tools for web development and database design as a background for the development process that was done. This is important as any software used for this project, and even further inside the company, needs to be approved by upper management of Synopsys. Even if there are better software tools to develop a project like this, only approved software that can operate inside the Synopsys environment must be used. To measure the value of one alternative over another, a score system is introduced with a focus on compatibility with Synopsys environment and usability for the developer.
- **Chapter 3** describes the development process of the project, with a focus on how the needs of the PV team regarding the Tracking process were gathered and transformed into specific goals that could be worked on as smaller sub-projects. Also, emphasis will be made on the data base designed for the project, as this data base model translated the common concepts used during Tracking to a group of inter-related data structures that could be queried, analyzed and consumed by the web interface system and PV team.
- **Chapter 4** describes the challenges and results of implementation of the project. The project is finished and implemented to be used by PV team, but other pending tasks inside the team required that the developer of the project focused on these new tasks. However, parts of the project became the base for upgrades on currently used software tools by the team, so a description of the influence of this project into these other projects is done.

- **Chapter 5** presents the conclusions done after the project was finished. First, a summary of the life cycle of the project is described, then focus of the conclusion is shifted upon the challenges of working with Synopsys, the benefits provided for the company, the knowledge required as an electronic engineer and the future works that can be done based on the developments done for the project.

2 | Background

This section describes the concepts of software engineering applied to this project. Solutions from PV team members to automate or ease their workflow have been limited to individual code scripting to support personal tasks, which in some instances are shared with other members. This also contributed to incentive the lack of standard procedures for the *Tracking* process. A more formal approach applying software engineering procedures is used to capture the needs of the whole PV team while also transforming them to goals that can be work upon as smaller software projects.

2.1 Design requirements and decisions taken

A common problem in the life cycle of software development is the translation of the imprecise, incomplete needs and requirements of the potential users of the software into complete, precise and formal specifications (1) . Not worrying about this early phase in development usually means the end product may not be of value to the end user or provoke either overestimation or underestimation of certain features of the software, which in turn provokes bad management of resources such as man hours or software licenses. As previously commented, some members of the PV team have made custom tools to ease their work, but the lack of any proper requirement engineering focused on a wider scope for the team has seriously limited the escalation of those solutions to a more general, standard and useful tool for the PV team. The work in (3) describes a formal way to capture these requirements by dividing the process in elicitation, specification, verification and management. These steps (2) are defined as follows:

- **Requirement elicitation** The process of obtaining knowledge about the project domain and requirements. In the context of the project it included interviews with the stakeholders, brainstorming sessions with the PV team and prototyping by discussing mock-ups of the interfaces of the project.
- **Requirement specification** The process to produce formal software requirement models. All the requirements including the functional as well as the non-functional requirements and the constraints are specified by these models in totality. In the context of this project, this translated in user stories and the domain model diagram shown in figure 3.3.
- **Requirement verification** The process of defining tasks that ensures that the software correctly implements a specific function. In the context of the project, this was done by allowing certain PV employees to test the features added to the the project once these features were developed, then acknowledging the feedback from these employees to validate or keep developing said features.
- **Requirement management** The process of analyzing, documenting, tracking, prioritizing and agreeing on the requirement and controlling the communication to relevant stakeholders. In the context of the project, this process was done by acknowledging to follow a *Gantt chart* detailing the features the be developed and the time each would take. However, this chart was modified during the development of the project to adjust to the challenges encountered while working on the features.

So, this approach was used to handle the requirement acquisition and validate them with the Stakeholders.

Regarding the project itself, work was done with some help of members of the PV team that could share their experience with Synopsys environment and their experience with conceptual challenges related to the Tracking process, so a good development framework was needed to improve time and effort efficiency, compared to the team's usual approach of just writing scripts to address specific problems.

A web framework is a software that is designed to support the development of web applications, including web services, web resources and web Application Programming Interfaces (APIs) (6). A framework aims to automate the overhead associated with common activities performed in web development, such as providing libraries for database access, session management, template design and mainly promoting code re-use. This is important for the development of the project, as the focus should be in translating the needs of PV team related to the *Tracking* process, and not in designing ways to do web or data base managing software. There are many options to choose from, so a deeper research was needed to choose one.

An important feature of the project is the data management. Approaches to designing a data management system may start with either a top-down or bottom-up analysis of the business logic (4), which is in this case the tracking process and its reporting. After specifying how the various entities and attributes of the business logic interact with each other it is necessary to define a data model. Given the proven efficacy of relational data base languages (such as SQL based ones) and Relational Data Base Management Systems (RDBMS (18)), such as MySQL (15), PostgreSQL (14) and even SQLite (16)) the solution was restricted to a relational model, so a comparison of which RDBMS fits the better for the project was made. Also, not only very volatile data was related to the project regarding events for a news feed, but also more immutable data that required deeper analysis, so the option of implementing a data warehouse(5) that fits the later need was considered. As clarification, volatile data refers to data that is created, edited, requested or deleted many times in short periods of time, and in the context of the project, issues being monitored during the Tracking process of PV is generated in large quantities each day. An example of this could be that a bad code change by R&D during a day may cause that fifty of more experiments from a Suite get crashes during execution, and this crashes require analysis from PV team to acknowledge them as issues or discard them if they are already being tracked from a previous day, in which case further crashes in the same Suite are not new information. So, fifty new events may be generated by only that Suite as long as the general issue related to them is discovered, investigated, assigned and fixed, causing a huge volume of volatile data to be created and deleted each day.

Regarding the web service, there are many design patterns (8). A proven pattern, and the main one used during development of this project, is Model-View-Controller (MVC) (8) which excels at separating the user interface/presentation from the data model. However, as shown in (8), there are other patterns that fit various other needs that are also complementary among other patterns. An example of other patterns that came up in this project is the pagination pattern (8), which was used to design the view of the news feed feature of the web service to avoid presenting a single page with all the issues stored in the database.

For the project in general, a major challenge was to know how much was needed to understand from the business logic without committing much time in understanding its most complex details that are only useful for the suite maintainers in charge of the tracking process. Investing too much time in this may provoke a longer development cycle wasted outside of the required features of the Stakeholders. In (9) an approach to software development is presented, called DevOps, which aims to enforce a dynamic software development cycle summarized in constant feedback, periodic releases of the web service focused on specific features and fast implementation. A DevOps approach allowed the development process to focus more in the project and less in the business logic.

2.2 Framework technology

Conceptually, the project requires two systems to be complete: a data management system and a web user interface system, as shown in Figure 2.1.

For the web project, a web server must be made available, but there is also a need to serve content from the data management project in a way that allows comfortable interaction with the information by both managers and suite maintainers. The success of the web project is achieved if it can show to the user new

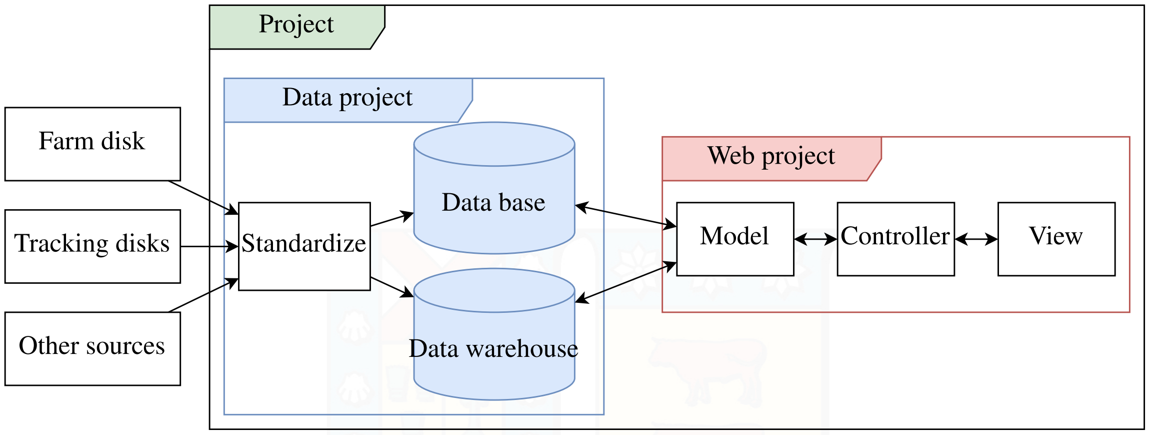


Figure 2.1: Project Structure. Sources of data external to the project (Green) are collected by the *Data project* (Blue), then this data is displayed and interacted with by using the *Web project* (Red).

information derived from the data management project, while also improving the efficiency of the tracking process.

Regarding the data management project, a relational database (18) structure was enough to map the business logic behind the tracking process, while also being suitable for data storing, analysis and interaction. Also, RDBMS are of common usage in software development, so there exists various options to implement one.

Given that there are already existing frameworks for development and implementation in the areas of both systems, a comparison between a few of them was made, where the deciding factors to include them as possible framework for this project were if they are implementable inside the Synopsys technology stack and if there is any experience among the developer and the Synopsys employees that contributed to the project.

2.2.1 Selection criteria

The criteria defined to choose the adequate framework was defined in order to fulfill the expectations of the Stakeholders. These are shaped by the necessity of an efficient schedule, independence of development and the improvement in efficiency for both managers and suite maintainers. This resulted in the following score system:

- **40% Compatibility with available assets:** Synopsys software & hardware ecosystem is already established, so choosing an alternative must consider the ease of implementation on this context.
- **15% Documentation** An organized and widely documentation makes easier to use a framework or software, improving development time.
- **15% Governance** An alternative may have features blocked behind payment, so if it is not already available in Synopsys ecosystem, then it will mean expended time in finding a workaround or another alternative while in mid-development.
- **5% Ease of usage** Alternatives that offer higher level interfaces or languages should be preferred, as this allows the developer to focus on the expectation of the Stakeholders.
- **25% Prototyping capabilities** It is very important to be able to present functional prototypes to the Stakeholders of the various features the project may include. So the score must reflect how easy is to get simple, working, fast functionalities with the alternative.

Table 2.1: Django vs Laravel

Characteristic	**Django**	**Laravel**
Definition	Full stack web application framework written in Python	Full stack web application framework written in PHP
Maintenance	By Django Software Foundation	By developer itself and community
Architecture	Model-View-Template (MVT) model	Model-View-Template (MVT) model
Platform	Supports cross platform	Supports cross platform
Generality	It has rapid development feature and a great community	It has clean architecture and a growing community
Scalability	Supports high scalability	Supports high scalability
Compatibility	Several Python-compatible frameworks exist	Only framework to be considered for PHP
Security	Highly secure for enterprise level applications	Implements basic security features
Github Stars	43,384	34,292
Performance	Quite Fast (69k JSON responses/second)	Relatively slow (8k JSON responses/second)
Number of websites	205,106	121,173
Front-end support	Quite complex to tie up a front-end JS framework with Django	Supports Vue.JS out of the box
Database compatibility	PostgreSQL, MySQL, Oracle, SQLite	PostgreSQL, MySQL, SQLite, SQL Server

2.2.2 Framework for web development

In the context of this project there is the need to develop a web application to fulfill the requirements of the Stakeholders in a DevOps (9) approach. As mentioned earlier, DevOps is a method that encourages to continually integrate the results during development of the project with the infrastructure of the company, so that feedback can be gathered and reviewed quickly. This process results in a development flow where the project is designed so that it is divided in smaller projects focusing on a subset of features from the main project, and once one of these smaller projects is developed, then it can be immediately be integrated in the environment of the company for quick testing.

Thus, software tools are needed that support this development process by abstracting it from common back-end problems, allowing it to focus on the previously mentioned requirements. A proven solution in web development is the usage of a web framework, a concept previously defined.

There are many frameworks available for web development, but a comparison was made with only two of them: Laravel and Django. This decision was made due to having previous experience with these frameworks and also having experience with Python and Laravel. Also, both frameworks have been used by PV employees during previous projects inside the company, so some issues regarding implementation or usage of these frameworks could be discussed with experienced employees in the matter. The comparison can be seen in Table 2.1.

In general, Django offers better features for a fast development. It lacks good front-end support, which meant that web interfaces of the project had a simple look. However, no urgency from the Stakeholders regarding this topic was perceived.

Given that both options differ in their programming language, it is relevant to compare both languages as well, based on developer-focused criteria (10). Comparison of Python and PHP can be seen in Table 2.2. Here the main appeal of Python over Laravel is its ease of learning, which is good for prototyping and to quickly develop features so then said features can be integrated and tested in the environment of the

Table 2.2: Python vs PHP

Characteristic	**Python**	**PHP**
Ease of learning	Very simple	More complex
Community Support	Great	Great
Documentation	Extensive	Extensive
Pricing	Free	Free
Library Support	Well-developed support for all types of applications	Not so extensive but standardized source of libraries
Speed	Faster than PHP 5.x	PHP 7.x is 3x faster than Python
Debugging	Python Debugger (PDB) is well documented and easy to use	XDebug is well documented and easy to use

Table 2.3: Django-Python vs Laravel-PHP

Criteria	**Django-Python score**	**Laravel-PHP score**
Compatibility with available assets	30/40	10/40
Documentation	10/15	15/15
Governance	15/15	15/15
Ease of usage	5/5	2/5
Prototyping capabilities	25/25	20/25
Final score	**85%**	**62%**

company.

During the capture of requirements it was discovered that the Stakeholders needed a business intelligence feature to process historical data to be stored. Thus, some libraries for features from outside of web development were needed, and the Python community has a wider selection of solutions for this kind of applications.

Finally, as shown in Table 2.3, it was concluded that the Django framework based on Python was more suitable for the project than the framework Laravel that is based on PHP language. The reasoning to assign each score of Table 2.3 is explained next.

- **Compatibility with available assets:** There are already many projects inside the company that are running on a Django server, so its compatibility is already proven. However, all of those projects are using a deprecated version of it (1.8 vs. the most recent 3.0 release) and so are compatible with an older version of Python itself (usually 2.7 versus modern versions up to 3.8).Regarding Laravel-PHP, there is just a couple of projects using this framework inside the company, and all are relatively new in comparison to other web projects, so its compatibility is daily questioned.
- **Documentation** Django possesses a widely detailed documentation with examples of usage, small projects and separated by versions (11). However, Django itself is server-side focused, so many client-side and database-side implementations must be implemented without much help from this documentation. Laravel documentation is wide, organized and enforces development standards, so learning from other projects is usually easier than Django.
- **Governance** Both Django and Laravel are free, open-source and have a big community who offers packages with ready-to-use functionalities.
- **Ease of usage** Python is a high level language with simple syntax. In addition, Django projects come with a management script to start new applications. Laravel in the other hand required the developer to gain expertise, which may compromise time.

Table 2.4: SQLs comparison

Characteristic	SQLite	PostgreSQL	MySQL
Concurrency control	No	Multiple transactions at the same time	Multiple transactions at the same time
ACID compliance	Yes	Yes	Yes
Documentation	Extensive	Extensive	Extensive + big community
Governance	Free and open- source. Paid extensions	Free and open source license	Open source license + paid versions
SQL compliance	Poorly	Largely, 160/179	Partially
Supported Platforms	All mayor mobile OS, NetBSD, FreeBSD, Solaris 10, optional in MacOS, Tizen, W10	Solaris, Windows, Linux, OS X, Unix-OS, Hp-UX OS	Solaris, Windows, Linux, OS X, FreeBDS OS
Security	Low to none. Only typical permission from the operating system. No server allows better control over queries	Secure: Native SSL support for connections and encryptions	Highly secure: Protocols based on ACL + advanced built- in features
Access methods	Whole database is contained in a single file, so it is easy to replicate.	Master-slave + third party support for others	Master-master, master-slave
Replication	Server-less, it only requires access to disk	All standards supported	All standards supported
Performance	Strong even in low memory environments	Widely used in large systems where read and write speeds are crucial and require execution of complex queries. Performance plug-ins also available.	Widely chosen for web-based projects that require a database simply for data transactions. Un-optimized for heavy load or complex queries.
Programming language support	Same or more than the other 2, except: Delphi, Earlang, .NET, Lisp	C/C++, Delphi, JS, Java, Python, R, Tcl, Go, Lisp, Erlang, .Net	C/C++, Erlang, PHP, Lisp, Go, Perl, Java, Delphi, R, Node.js

- **Prototyping capabilities** Django offers a development server configuration to quickly test web project functionalities, avoiding the configuration of server scripts, security and static content serving such as multimedia files. Laravel possesses an easy to understand quick start guide (13). However, it is not available on all the available versions, which may conflict with the available PHP version found in the company's resources.

2.2.3 Relational database management system

The project needed to accomplish two functionalities regarding the storage of data. For the daily functionalities of the web interface there is a need to constantly update registers of issues, statements and reports, so an OLTP (On-Line Transactional Processing) optimized Data Base Management System (DBMS) is the best fit. On the other hand, the system must also provide with analysis capabilities for a large and always growing set of stored data regarding the results of each suite, but with little to no volatility as this data was a daily snapshot. Thus, an On-Line Analysis Processing (OLAP) optimized DBMS is a better solution.

In addition, regardless of the chosen web framework, a Relational Data Base (RDB) seems to be the go to paradigm of data modeling due to both frameworks limitations. This implies that an Structured Query

Table 2.5: Score results of SQLs comparison

Criteria	**PostgreSQL**	**MySQL**
Compatibility with available assets	30/40	10/40
Documentation	15/15	15/15
Governance	15/15	15/15
Ease of usage	2/5	5/5
Prototyping capabilities	15/25	25/25
Final score	**77%**	**70%**

Language (SQL) variant was the querying language used for the project.

After limiting the options to RDBs, a comparison of RDBMS was done, as shown in Table 2.4. The options considered are *SQLite* (16), *PostgreSQL* (14) and *MySQL* (15), which are all available in the Synopsys environment.

Early during analysis of which RDBMS to use, SQLite was discarded. This was due to the access method used by this query language. SQLite stores the data base locally in a file, so access to the data base is managed the same as any other file in disk. This means that multiple users can not edit the data base at the same time, which is necessary for an online data base that must allow multiple PV employees to interact with Tracking process data.

As cited from (17), "MySQL will be ideal for your project if you require a high-security RDBMS for web applications or custom solutions, but not if you need a fully SQL compliant RDBMS capable of performing complex tasks swiftly. Meanwhile, PostgreSQL will be ideal for your project if your requirements revolve around complex procedures, integration, intricate designs and data integrity, and not around high-speed and ease of setting up".

Given the necessity of two different focuses, it may be a good idea to use MySQL for the data management of the news feed contents as is highly volatile and requires quick accessing. In the other hand, PostgreSQL seems better suited for the needs of data warehousing that comes from the requirement of data management of daily snapshots (PRS reports, stability and disk status) and its consequent analysis. SQLite just seems to not provide enough functionalities for the project.

- **Compatibility with available assets:** PostgreSQL 8.4 is available in the company's resources. Although is an old version, it is functional. MySQL is also available as a resource in the company. However, it requires administrative permission to allow computer's communication sockets liberation.
- **Documentation** (14) (15) There is detailed documentation for all the relevant versions for both PostgreSQL and MySQL.
- **Governance** Both are free, and open-source. However, MySQL is also available with various proprietary licenses.
- **Ease of usage** PostgreSQL aims to be SQL compliant while also implementing additional features. Also, starting a PostgreSQL server requires delicate configuration. MySQL on the other hand, offers a user-friendly, graphic interface that is available for configuring and starting an SQL server, in addition to the more direct MySQL shell option.
- **Prototyping capabilities** As specified before, the configuration of a PostgreSQL server is delicate. Also, it requires the usage of a PostgreSQL shell as interface. However, Django offers its own interface to deal with a PostgreSQL server. MySQL user friendly interface allows for quick and easy prototyping.

2.2.4 Selected alternative

Regarding the web framework, Django-Python is the chosen option. Not only it scored higher than Laravel in the previous segment, but there is also the fact that various co-workers that were related to the project have some degree of experience with it, allowing quicker feedback during the development process. In addition, Django's development server mode is already proven to work under certain conditions in the company's environment which are available for this project, such as certain hosts for the server and software tools. Regarding the data management system, PostgreSQL scored higher than MySQL. However, it was discussed with the Stakeholders that a mixed solution could be implemented as both systems offer functionalities and performances more suitable for certain aspects of the project. On one side, PostgreSQL is optimal for data warehouses, which are suitable for the analysis and reporting needs of the Stakeholders representing the management of Synopsys. On the other hand, MySQL is more suitable for smaller, faster volumes of transactions with the database, which are needed by the Stakeholders representing the suite maintainer employees of the company. Nevertheless, the complications arisen from a proper implementation of a MySQL server in the company's environment were blocking enough to decide to discard this option mid-development. So, all data related solutions were implemented using PostgreSQL.

3 | Development process

This chapter starts by describing how the process of gathering the requirements from the stakeholders was done. Requirements were compiled by realizing interviews with PV team, and then defining based on said interviews the aspects of the problem and solution. The, the development process of the project will be described by splitting this process into the development of the data management system and the development of the web interface system. This splitting is done for clarity, as the web interface system consumes the database of the data management system, so it is simple to separate both systems during development.

3.1 Gathering requirements

To start the project at Synopsys, a requirement engineering approach was used to gather the needs of PV team. This approach was necessary, as the were not any procedures inside the team to handle software projects for internal use. Bear in mind, PV team manages many software projects related to the testing of EDA tools, but this does not translate to expertise in designing software tools for productivity at the office.

The gathering of requirements for the project aims to define all the needs derived from the already explained problem of lack of standardization in Tracking, so then those needs can be compiled and used to define clear features to develop. In order to do so, development started with a series of interviews with 8 members of the team. In Table 3.1, a summary of reporting means as told by PV team shows the variety of means they use to report their daily activities.

Table 3.1: Means of reporting of PV team

Mean of reporting	PV members using it
Daily email	7
Sharepoint sign-off dashboard	5
Sharepoint Powerpoint presentation	2
STAR + email to R&D	6
Gray page comments	7

Conclusions from the interviews are shown next:

- **What is the problem?** A lot of time is wasted checking current status and results from each Suite, while also reporting any finding through multiple means. Also, these reports is prone to errors due to being manually done. Lastly, reports automatically generated by gray page related systems are not necessarily synchronized with the current status of the Suite experiments..
- **Where is the problem?** In two processes: During reporting process and during the maintaining process of the Suite. This is what was previously mentioned as the Tracking process.
- **What issues are prevented by solving this problem?** Wasting time from Suite maintainers which could be use in other tasks. Also, erroneous reports.

- **Whose problem is it?** Of management and Suite maintainers
- **Why it needs to be solved?** In on hand, management wants clear and high quality reports to take decisions based on them. On the other, Suite maintainers want to avoid wasting time in tedious tasks like reporting, while also having access to updated and easy to check information regarding the Suites.
- **How would a new tool of software help?**

  Centralization

  - A common dashboard where information form the Suites can be checked on, both in operative (suite owners) and in management level.
  - A dashboard where typical response actions during Tracking can be done, such as relaunching experiments, start investigations or create STARs.
  - A dashboard that allows to monitor Suite progress.

  Automatizing

  - A software capable of identifying common problematic situations and reporting them to the related Suite maintainer.
  - A software capable of auto-completing report templates and publish them with minimal human intervention.

  Valuable historical metrics

  - A database capable of keeping valuable data for management regarding a Suite behaviour through time.
  - A database over which predictions regarding Suite behaviour can be made.

3.2 Data management system

The web interface system requires reliable access to structured data worth to be displayed and interacted with. For this, a data management system is developed, using PostgreSQL as the chosen RDBMS for the project. This data management system requires a way to collect the data needed to populate a data base, but this data is scattered inside the Synopsys environment among disks storing data related to the Tracking process, disks used by the computer farms executing the experiments and other sources such as scripts designed by PV team to monitor or collect data for specific purposes. Thus, a script called Health Collector was develop to gather all this data. Then, a data model for the data base was designed to translate the concepts of the Tracking process into data structures that can provide value for analysis or monitoring. Then, the data model is implemented by setting up an online PostgreSQL server to serve the data to the web interface project, as previously shown in Figure 2.1.

3.2.1 Health Collector

Before getting a database up and running, it is necessary to design a way to gather data from Synopsys environment. That is, to gather data from disks containing log files of experiments, computer farm log files and other sources, all protected inside the Virtual Private Network (VPN) (7) of Synopsys. This introduced the challenge of not having a centralized source of all the data desired by the stakeholders to be saved and analyzed. In this data one can find the results of experiments done in each Suite tracked by PV, as well as metrics that can be used to derive information regarding current status of one or many Suites.

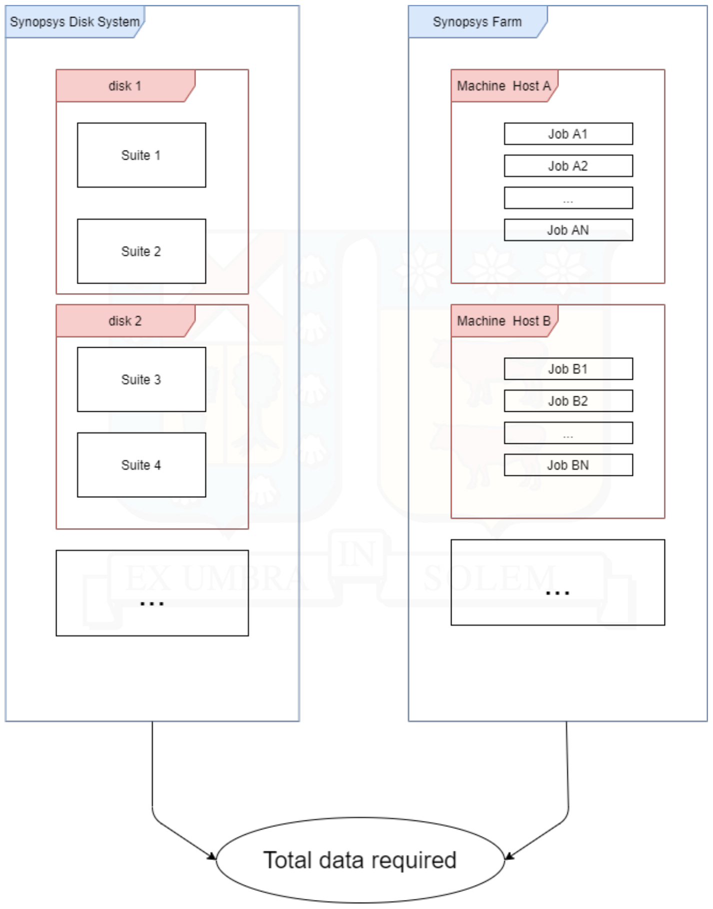


Figure 3.1: Required sources of data for data model.

The data comes from two main sources as exemplified in Figure 3.1. In Figure 3.1 each machine host is running a number of jobs. A job is a process prepared to be executed by a computer, in this case one of the computers used as host in Synopsys´ farm. In the context of the project a job contains the process needed to realize an experiment of a Suite. Also, in Figure 3.1, the disk system inside Synopsys environment that is related to the Tracking process is shown. Each disk contains the results of experiments executed in the computer farms of Synopsys.

In the motivational example at the beginning of this report it was shown that PV organizes the experiments of each Suite by using an existing system called *PRS*. As a simplified description, that system automates the execution of each single experiment by assigning its execution and required resources to a *Farm* of host computers. Also, it assigns memory to certain disks in Synopsys network to save the results of each experiment, in a way that the data is sorted by Suite in a recognizable way. In order to gather all the necessary data for the data model, it is needed to extract and organize relevant data from both the disks containing the experimental results and the status of each experiment in the *farms*.

As a solution, a Python script was developed to collect this data, which will be known from now on as Health Collector. This script is responsible of the standardize step shown in Figure 2.1. The functionality of Health Collector is described in Figure 3.2.

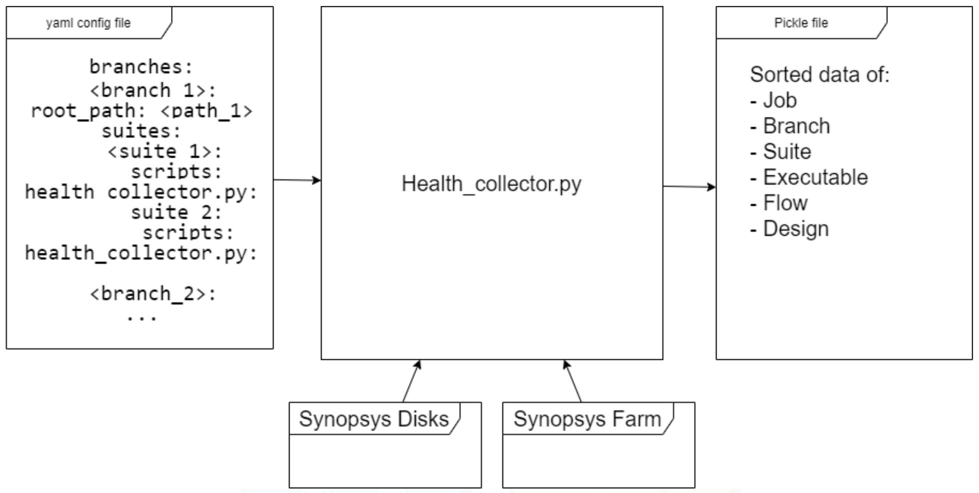


Figure 3.2: Functionality of Health Collector script. A pickle file is a type of binary file.

The output of Health Collector is a pickle file. This is an binary file that contains a Python dictionary, but is more compressed in size than a regular text file. This kind of Python data structure is useful to sort data hierarchically while also optimizing future iteration done over it to search specific data. Output data is sorted hierarchically as shown in Figure 3.2. This follows a similar hierarchy of PRS disk allocation of experiment results, but also adds the *Tool* used at the top of the hierarchy. In addition, it adds farm related data of each experiment at the design data level of the hierarchy. Is here where Health Collector achieves the goal of unifying data from both sources: disks and farm.

3.2.2 Data model

To transfer the terminology and characteristics of the Tracking process, and based on the interviews with PV team, Figure 3.3 illustrates the concepts of this process that may require to be stored as data in the data management system of the project.

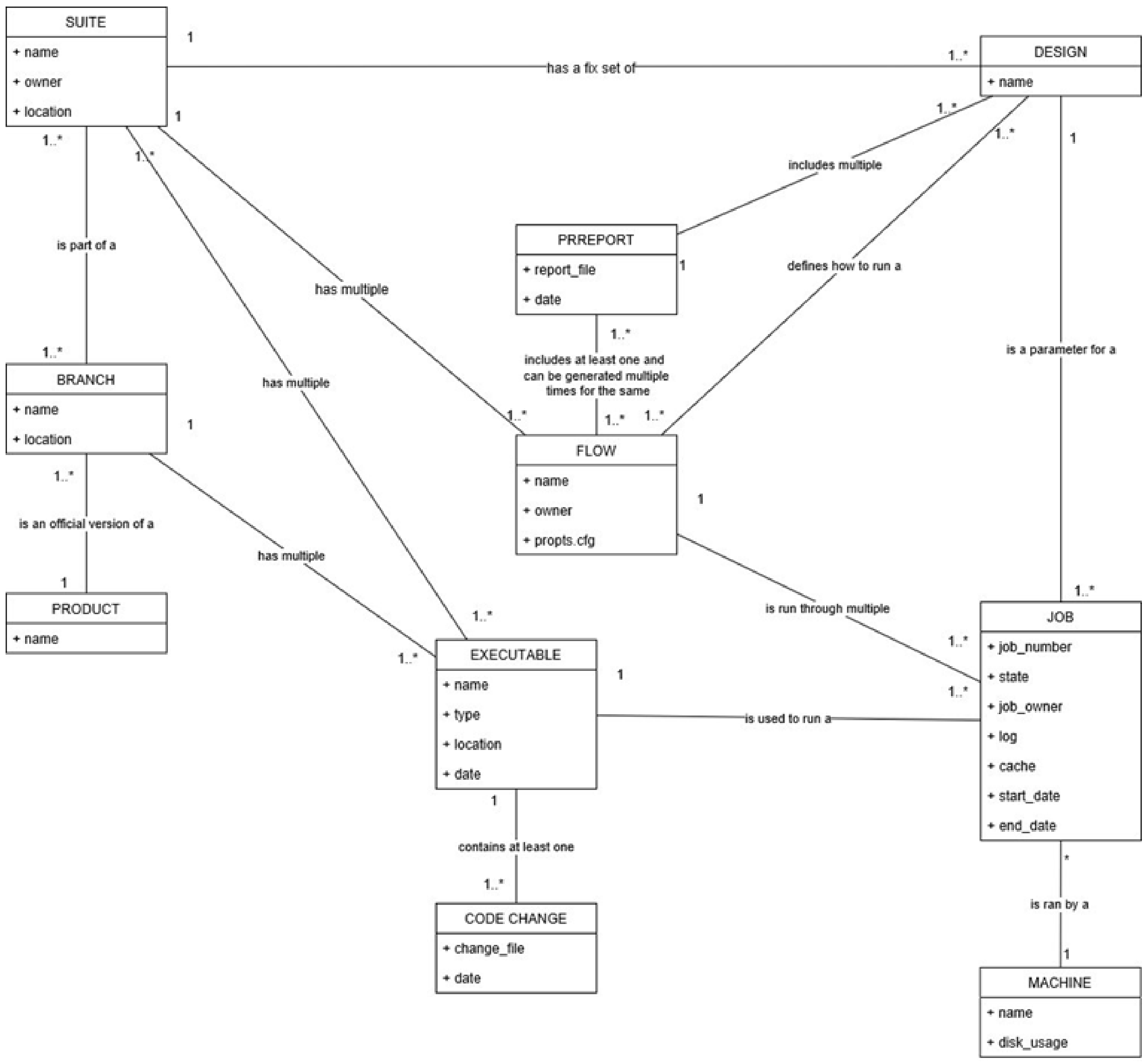


Figure 3.3: Data model built upon commentaries from PV members.

To implement this model as a database, it needs to be transformed into a data model compatible with PostgreSQL. For this, further discussions with the PV team done focusing on the hierarchy of each element of Figure 3.3. It was agreed that the main entity of the model should be the Suite, but to add more value of future analysis for PV team the branch was put one level above in the hierarchy, while Product (also known as Tool) was put above it. This allows PV team to analyze the behaviour of each Suite across different release versions (or branches) of each Product. Below the Suite entity, Flow, Executable and Design need to be sorted. In the end, it was decided to follow the hierarchy as follows: Executable and Flow, and Design after flow. This order eliminates the direct relation between Suite and Design, but leaves it implicit by relating Design below Flow. The new hierarchy can be visualized in Figure 3.4

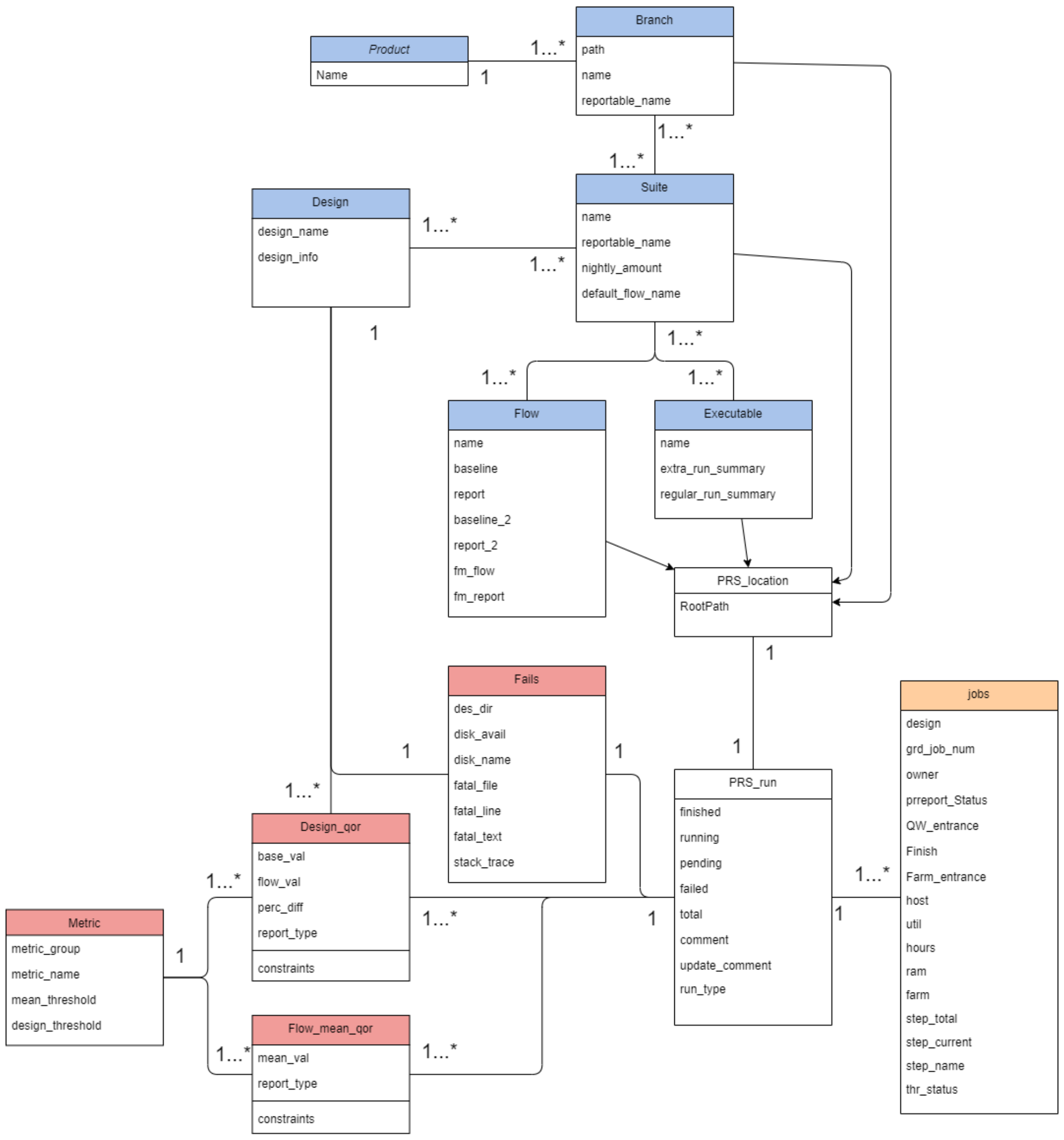


Figure 3.4: Database implement for the project.

Colors in Figure 3.4 are added to make some conceptual differences. Blue entities of the diagram represent the PRS hierarchy found in the directories inside the disks where experiment results are stored. However, in those disks Flow directories contain directories named by Executable, hinting a hierarchical order between them, but discussion with PV team clarified that this sorting of directories is just a simplified solution to organize experiment results. In the final database implementation Flow and Executable were put at the same hierarchical level below Suite. This was done to represent correctly the conceptual relation between Suite, Flow and Executable.

3.2.3 PostgreSQL server integration

The original plan was to use MySQL for the news feed database and PostgreSQL for the data warehouse. However, all database related tasks were implemented using only PostgreSQL. This was because starting a MySQL server required administrative permission inside Synopsys environment, while a PostgreSQL server could be started by any user in some of the machines inside the network of the company.

As discussed in Section 2.2.3, MySQL is more suitable to request high volumes of volatile data, but using PostgreSQL did not show any relevant slowdown during development.

Implementation of a PostgreSQL server required to choose a host computer inside Synopsys environment that had ports available for connection

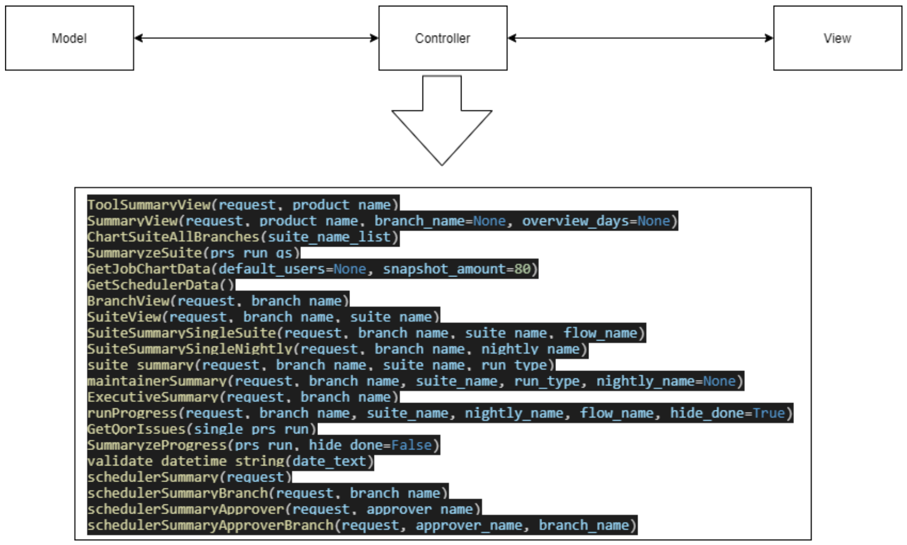
In the previous section it was described the output of the Health Collector. This output is now used to populate the PostgreSQL database.

3.3 Web interface system

Users of this project must be able to interact with the data in a way that does not add more complexity to their daily tasks. For this, it was decided to create a web interface using Django as explained previously in this document. Django uses a Model-Controller-View design pattern, where database manipulation is done at Model level, web interface is displayed at the view level, and requests between Model and View are handled at the Controller step.

In Figure 3.5 are shown the implemented Python functions for the Controller level.

3.3.1 Model View Controller



```
ToolSummaryView(request, product_name)
SummaryView(request, product_name, branch_name=None, overview_days=None)
ChartSuiteAllBranches(suite_name_list)
SummaryzeSuite(prs_run_qs)
GetJobChartData(default_users=None, snapshot_amount=80)
GetSchedulerData()
BranchView(request, branch_name)
SuiteView(request, branch_name, suite_name)
SuiteSummarySingleSuite(request, branch_name, suite_name, flow_name)
SuiteSummarySingleNightly(request, branch_name, nightly_name)
suite_summary(request, branch_name, suite_name, run_type)
maintainerSummary(request, branch_name, suite_name, run_type, nightly_name=None)
ExecutiveSummary(request, branch_name)
runProgress(request, branch_name, suite_name, nightly_name, flow_name, hide_done=True)
GetQorIssues(single_prs_run)
SummaryzeProgress(prs_run, hide_done=False)
validate_datetime_string(date_text)
schedulerSummary(request)
schedulerSummaryBranch(request, branch_name)
schedulerSummaryApprover(request, approver_name)
schedulerSummaryApproverBranch(request, approver_name, branch_name)
```

Figure 3.5: Model-Controller-View diagram used for the web interface part of the project. Highlighted are the functions implemented to generate the data displayed in the views.

3.4 Feature integration

Finally, these needs are translated into functional requirements, as shown in Figure 3.6, to organize the development of the project, while also serving as milestones to accomplish for the completion of it. Functional requirements in Figure 3.6 are marked in blue boxes. Each functional requirement was then divided in smaller features that define the functional requirement itself. A deeper description of the development of each feature will be made next.

3.4.1 News feed

This requirement establishes the feature of a dashboard page in the web interface. This dashboard displays recently added Tracking issues to the data base, by displaying a box with the relevant data.

3.4.1.1 General

This feature of the News feed is a web interface that contains all the Tracking issues stored in the data base, sorted by most recent to oldest. For this, a query is done when loading the web page to get the data of the Tracking issues. As this data set gets larger over time due to the constant influx of new issues, first iterations of this feature started to query more data every following day during testing, so loading the page got slower every day. To solve this, a pagination solution was added, by using the *Pagination* package of *Python*. With this, queries to the database in each page load where limited to ten, and a pagination menu at the bottom of the page was added to query older entries in the data base.

3.4.1.2 Highlights

This feature generates an *HTML* element with a summary of relevant Tracking issues from the data base. The contents are manual conFigured from the server to display certain Suite data. The HTML element generated can be displayed in other views of the web interface.

3.4.2 Session management

This requirement specifies that users of the web interface must be restricted from certain information. There are three kinds of sessions needed: PV, outsider and Admin. Admin session has access to all information in the web pages, while also being able to view and use buttons on each page to access the data base administration page manually edit the data base contents. PV session has access to all information and pages, but cannot see the buttons to access the data base administration page. An outsider session has restricted access to pages and information.

Restriction in pages of the web interface was achieved by using *Django´*s default session management package. This allows to verify which session is active when loading each page. To assign a session to each user, a user management package was implemented to create users for the web interface system, then each user is manually assigned a session flag to categorize it.

The final implementation of this requirement lacked the categorization of user. This was due to not being able to integrate the established Synopsys´ user accounts assigned to each employee with the web server used by the company. As each employee of the company is assigned a user name to verify access through the VPN of Synopsys while using a web browser, a request to the *Information Technology* (IT) department was done to integrate the session management from the company to the project. However, the project was done using *Python 3.6* version, while the server of the company supports only *Python 2.7*. This incompatibility and the dependencies of this project of *Python 3+* packages meant at the end that the session feature was not implemented at the end. As a workaround to avoid unauthorized access to the data base management page, a single admin user was created and maintained to access this page.

3.4.3 Data management

This requirement specifies that a data base must be implemented, storing data related to the Tracking process.

3.4.3.1 Job data

This feature adds a job table to the data base, which contains the data of the job process running o queuing at the computer farms that is related to a specific experiment that is being tracked. To implement this, the data collected by the Health Collector script is processed to create a data base entry with data related to job identification, date of submission, resources used and other data.

3.4.3.2 QoR data

This feature adds an entry to the tables *Design_qor*, *flow_mean_qor* and *Metric* from Figure 3.3. This data is related to the *Quality of Results (QoR)* of each experiment, which are metrics related to the optimization and placement process done to a digital design. These data is extracted from the experimental results stored in disk by an automated system or the company, and is stored as log files per each experiment. To scrap the log files for valuable data to save in the data base, multiple discussions where held with PV representatives to discuss the meaning of the metrics and their relevance. These discussion also helped to make correlations between the metrics, which influenced in the display of data in some of the pages of the web interface.

3.4.3.3 Disk data

This feature adds a table to the database containing disk usage data from the jobs running or queuing in the computer farms. Early in development this feature was removed, as this information could be synthesized from the jobs table from the data base. This feature was intended to create a page in the web interface to display a summary of the disks involved in the Tracking process. In the end, when generating that page, a query is done to the data base for all job data of jobs currently running, then that data is summarized and categorized in the *SummaryView* function shown in Figure 3.6. The time of loading the page was in practice always less than half a minute, so it was not considered necessary to implement a new data base table.

3.4.3.4 Dashboard log

This feature add a log file to register the activity done in the dashboard of the *News feed* requirement. This was discarded during development as it was not considered useful after further discussion with PV representatives.

3.4.3.5 Reports

This feature adds a table to the data base containing data related to the reports created by each PV employee. This feature was removed due to the lack of the session management requirement. The impossibility of identifying users in the system meant that reports could not be linked to users through data base tables relationships.

3.4.4 Actions

This requirement is established to provide users with an interface to execute typical actions done during the Tracking process to interact with a Tracking issue.

3.4.4.1 Start culprit

This feature is intended to allow the user to start investigation of a Tracking issue from the web interface, by using the already developed scripted tools currently used by PV team. This feature was not implemented due to lack of time at the end of the project.

3.4.4.2 Relaunch job

This feature is intended to submit again a job that was already running or queuing in the computer farms. In the end, the solution was to display the path to the directory of the disk containing the experiment results, which then the user can use to relaunch the job by using a terminal of a virtual machine. This was done due to not fully understanding the user permission requirements needed to submit a job, which are dependant of the user and the Suite information.

3.4.4.3 Publish statement

This feature is intended to allow a user to publish a statement in the News feed so that other users may see it. This feature was implemented by updating the news feed code to add an issue in the database containing the statement written by the user, and a flag to differentiate it form Tracking issues. *Subscription* This feature is intended to allow users to subscribe to certain topics, such as users, Suites, Branches, Flows, etc. This would allow users to have a personalized News feed that displays data of the topics subscribed. This feature was not implemented due to lack of time at the end of the project.

3.4.5 On-demand summary

This requirement is established to provide a report requested by a user at any time, to get summarized data of the current status of the Suites used for the Tracking process.

3.4.5.1 Nightlies

This feature is intended to add a summary of jobs related to a specific Executable, internally called *Nightly*. Relation between a job and a Executable can be observed in Figure 3.3. To implement this feature, a query of all *PRS_location* tables related to a specific Executable is done, then the respective 1 to 1 relation with *PRS_run* table is used to get the entries of this table. After that, jobs related to the obtained PRS_run entries are queried and processed to create the summary. This processing requires knowledge on which metrics of resources used by each job are relevant to monitor, and how these metrics can be related to typical Tracking issues. Discussions about the criteria to filter these issues were held constantly during development, as it required digital circuit knowledge.

This feature took around half an hour to be executed when requesting the summary through the web page. This is due to the amount of queries needed to get all the needed data and the increasing amount of data stored in the data base during development.

3.4.5.2 Summary

This feature is intended to provide an overall summary of all the Suites across all branches. This was implemented through the controller function *ToolSummaryView* shown in Figure 3.6. This Python function queries all jobs in the data base, and prepares a summary of the data for each Suite across each branch. This function is more efficient in execution time than the ones used for the Nightlies feature, as it only needs to query all the jobs entries in the data base, without needing previous queries to other tables besides PRS_run to get the data needed to categorize the jobs by Suite and Branch.

3.4.6 Disk

This requirement is established to provide users with a summary of current status of disk. This summary is generated automatically by the web server periodically during the day.

3.4.6.1 Data status

This feature is intended to provide a summary table of the resources used and available of the disks used during the Tracking process. It is implemented by querying all jobs and PRS_run entries of the data base, and then sorting them by the name of the disk where the experiment results of each job is stored.

3.4.6.2 Disk trends

This feature is intended to store snapshots of the disk usage over time. It was removed due to being redundant with the previous feature.

3.4.7 Live summary

This requirement, and the feature named identically, is established to provide an automatically generated summary of each Suite with data from the Tracking process. To implement this feature, the functions *SuiteSummary, SuiteSummarySingleNightly, SuiteSummarySingleSuite, Suite_summary and SummarizeSuite* were created to query job data of Suites at different levels of details. These levels are differentiated by the filters used, such as filtering by Nightly, Suite or Branch.

3.4.8 Reports

This requirement is established to support users to redact reports to deliver to other users. It was removed during development due to time constraints at the end of the project.

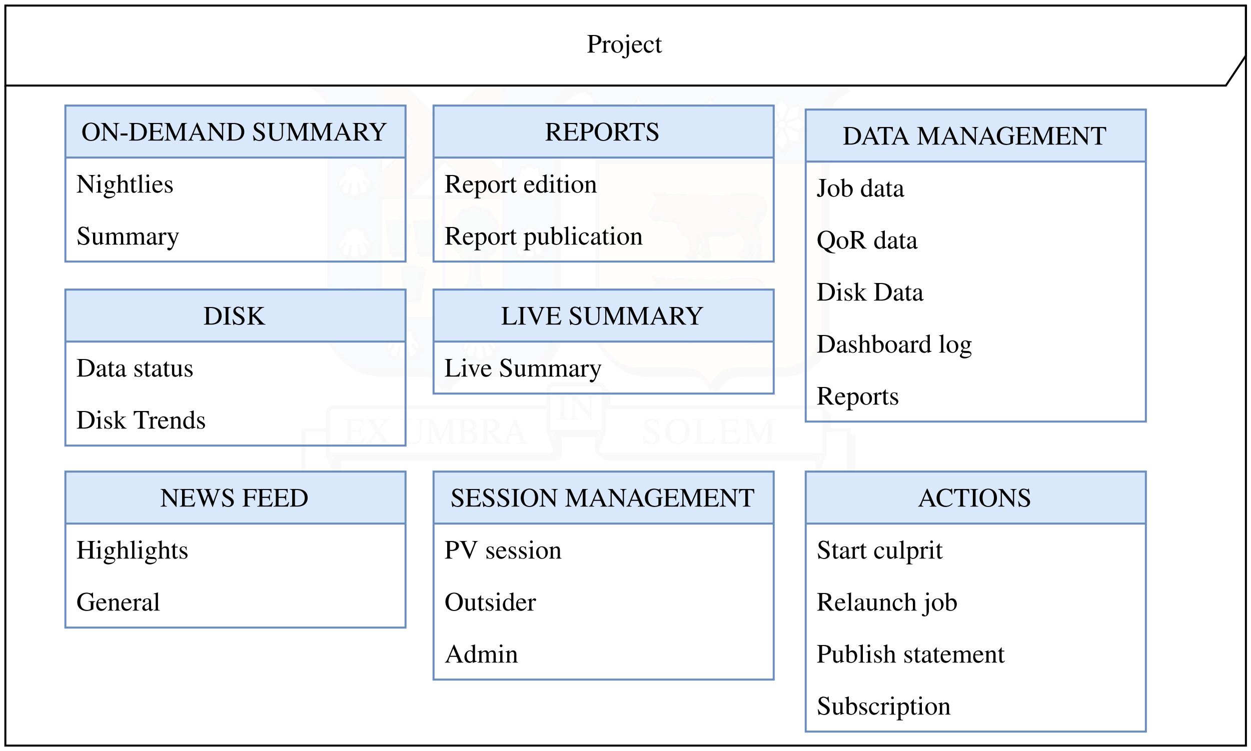


Figure 3.6: Functional requirements of the project.

4 | Implementation of the project

As previously mentioned, the project was divided into smaller sub-projects in a DevOps approach, which in the end meant that testing was done by PV employees as soon as new features were developed. To implement features, a Git repository protected inside Synopsys network was created to store the Django web interface project, then a Django server was hosted in one of the host machines of the company and started. In the side of data base implementation, a PostgreSQL server was hosted in the same host machine and configured to be able to receive queries from any host inside Synopsys´ network. Each feature was implemented trying to follow the Gantt chart from Figure A.1. However, development found issues along the way.

Regarding data base implementation, the PostgreSQL server proved to function properly during development, but it got suddenly shut down during various instances, at least ten times during the development of the project. This was due to the server being hosted inside a machine host of the company that also hosted other processes and services from employees of Synopsys. This caused, when this machine host failed or was restarted for reasons outside of this project influence, that the PostgreSQL server stopped suddenly, which in turn caused gaps of data that needed to be added manually to the data base from the back-ups of the output files of the Health Collector script described previously. This process of restoring the data base consumed time of development.

Regarding the web interface implementation, the Django server used was also hosted in the same machine host as the data base server, as mentioned previously. However, shutting down the web server did not caused any data issues, and only provoked delays on testing the interface by other PV employees.

A more relevant issue during the web interface implementations was the acknowledgement and implementation of feedback from employee testers each time a feature was implemented. New feedback meant in various cases that additional time of development was required, which in the end caused that some features were not implemented at all, as explained during chapter 3.

The original plan, as shown in Figure A.1, was to end development at the end of March 2020. However, the previous issues extended the development process until late April 2020. Also, as previously mentioned, some features had to be discarded.

After the project was finished, some tools designed during development where recycled to support other tasks and projects of PV team, which will be explained next. Recycling in this context means that features that once were designed to be part of the project were later on adapted to be used in other project or to support another task.

4.1 Recycling Health Collector for Scheduler project

Among the tools used internally by Synopsys employees, there is one that is used by both PV employees for testing purposes as well as R&D engineers for new feature implementation. The tool is known as Scheduler, which is a web page that allows the user to prepare and run experiments in the farms, but with a very simplified interface that uses templates for setting up said experiments. This tool has a version used and maintained by PV team for the DC related tools. During a long time, a request by R&D departments was to being able to monitor the status of jobs associated to each experiment by using the web page interface. This request is born as the other option for R&D employees is to ask PV where each experimental result is stored and then use

Linux commands from inside a virtual machine to search for specific checkpoints written in various log files. A web interface capable of displaying both summarized and detailed data of the jobs would ease the work for R&D.

Due to the previous situation, PV team decided to update the Scheduler tool to comply with R&D needs. During this update, the Health Collector part of the project described in this document was recycled.

First, and following Figure 3.2, the yaml configuration file needed as input to Health Collector was filled with the data of the suites supported by the Scheduler. Then, the Health Collector script was set up in a cron job to collect data from those suites and dump the output as a pickle file containing a Python dictionary with the processed data in a specific disk. This cron job is run every hour rewriting the output file. After that, Scheduler was updated to process that pickle file and update the web interface with the status of the jobs associated to each suite.

This update is still used and there are no plans to revert or replace it.

4.2 Recycling data base to report farm status

During the latest months of the project, Synopsys farms started to get too many requests of experimental jobs. This caused that a lot of experiments by both R&D and PV teams expected to run in a regular basis during the week stayed in queue to be processed at the farms.

Before this, the data warehouse feature of the project had been saving snapshots of the status of jobs related to PV suites, which also includes jobs from experiments of R&D teams launched using the Scheduler tool mentioned previously. These snapshots were then processed to make a report showing the amount of jobs submitted to the farms, categorized by which come from PV suites or R&D employees. Most importantly, it showed this data through time in a span of four months, revealing that the issue was an abrupt increase of experiments by R&D. Thanks to this report, PV had information worth enough to convince R&D teams to reduce and to plan better the submission of experiments.

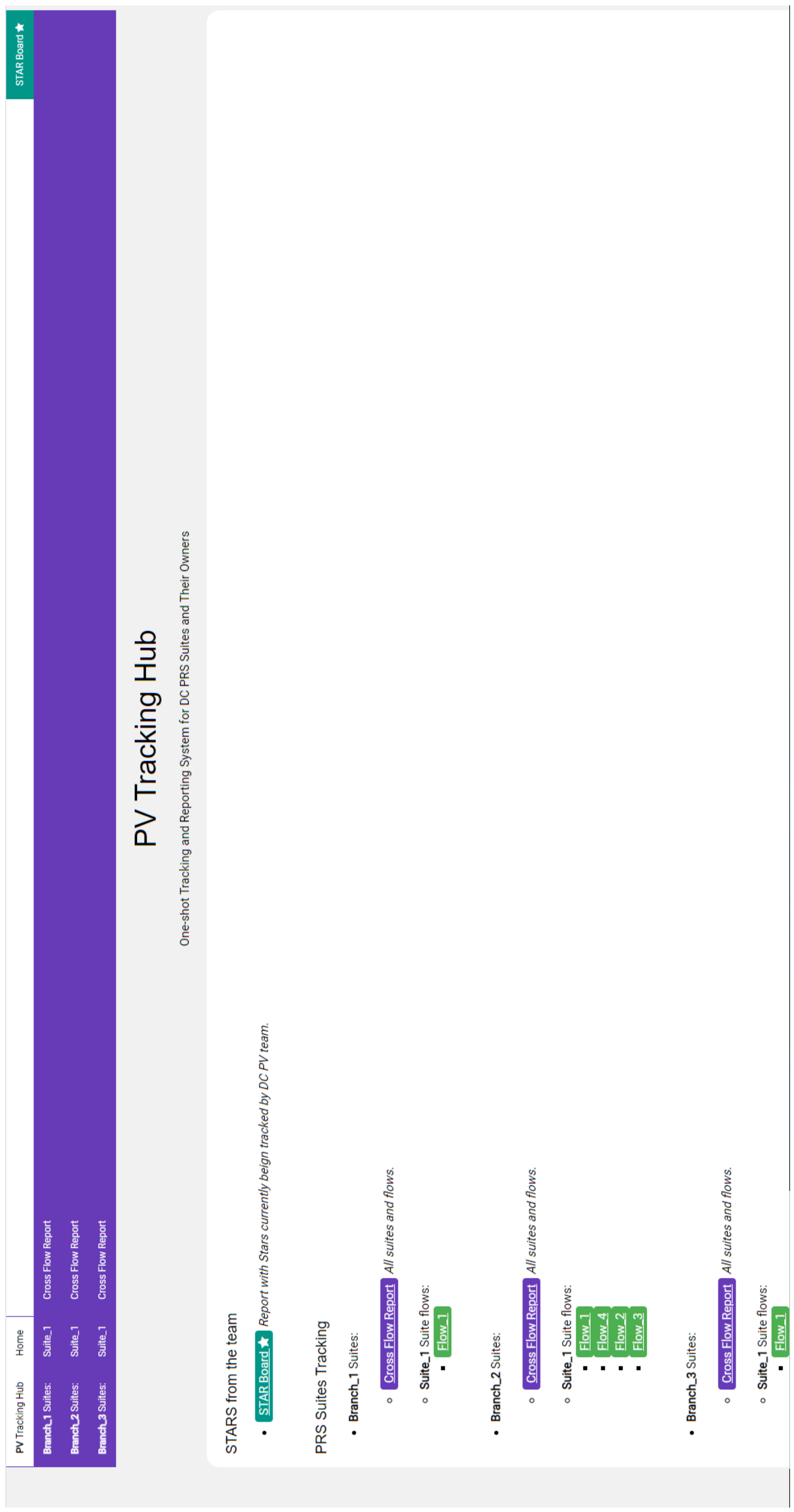
4.3 Recycling Health Collector for PV Hub

An internal project inside PV team is the PV Hub project. PV Hub is a web interface where PV employees dedicated to do *Tracking* of *Suites* can monitor the status and metrics of experiments. It is similar to the project described in this document, but it is designed to not work with an online data base nor save snapshots of data through time, so it is focused to daily usage rather than data analysis. This means is not necessarily designed to be useful for management, but rather PV employees.

The *Health Collector* script was recycled for this project to allow it to gather the daily data needed for display. In Figure 4.1 the home page of PV Hub is displayed. In this interface links for each *Branch*, *Suite* and *Flow* are sorted in a logical manner in the below-left part of the page for PV purposes, and each one redirects to a report page, like the one in Figure 4.2.

In Figure 4.2, a mock-up example of the daily report page is shown. Above-left there is the code *D20210526_15_35*, which identifies a specific Executable created in May 26, 2021, at 03:35 PM. This page contains hundreds of other Executables, but only one is displayed in the Figure to simplify the example. At the right of the Executable identifier, there are two green buttons that redirect to HTML reports of the experimental results for this Executable. Below the Executable identifier is a text box and a button to leave a comment on the results of that Executable. Below the text box is a table containing a summary of crashes and failures of each experiment of the Executable, identified by the name of the digital circuit used as test case. This digital circuit test cases can vary in amount between Suites, but oscillate between one to sixty designs. Below this table, there is another table with a highlight selection of metrics that fulfills a criteria to be considered a degradation or an improvement compared to the results of the previous Executable. Metrics of a single experiment can vary between twenty and a hundred

All the data needed to generate the report of each Executable on this page is gathered by using the Health Collector script developed for the project reported in this document.


PV Tracking Hub Home STAR Board ★

Branch_1 Suites: Suite_1 Cross Flow Report

Branch_2 Suites: Suite_1 Cross Flow Report

Branch_3 Suites: Suite_1 Cross Flow Report

PV Tracking Hub

One-shot Tracking and Reporting System for DC PRS Suites and Their Owners

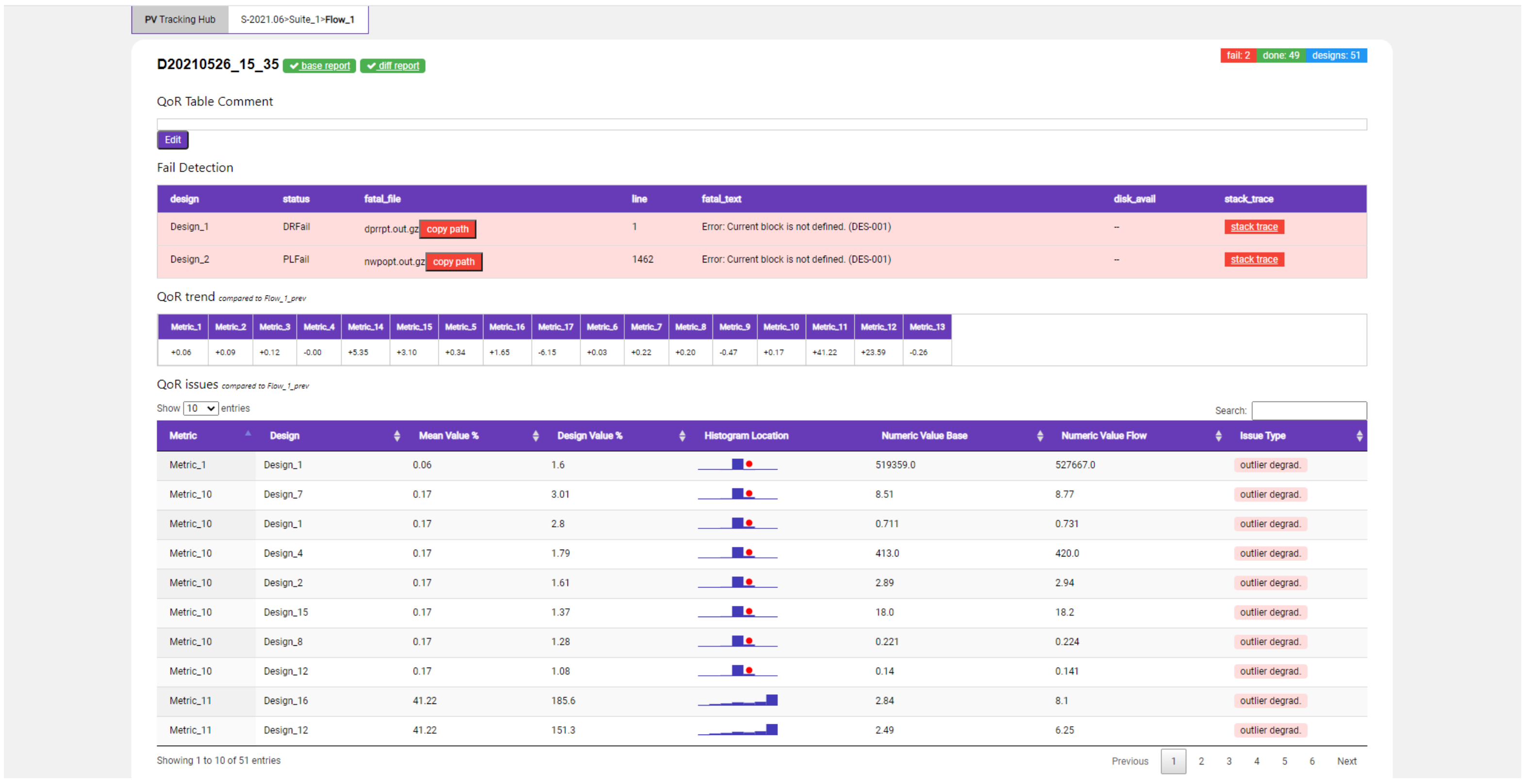
STARS from the team

- STAR Board ★ *Report with Stars currently beign tracked by DC PV team.*

PRS Suites Tracking

- **Branch_1** Suites:
  - Cross Flow Report *All suites and flows.*
  - **Suite_1** Suite flows:
    - Flow_1
- **Branch_2** Suites:
  - Cross Flow Report *All suites and flows.*
  - **Suite_1** Suite flows:
    - Flow_1
    - Flow_4
    - Flow_2
    - Flow_3
- **Branch_3** Suites:
  - Cross Flow Report *All suites and flows.*
  - **Suite_1** Suite flows:
    - Flow_1

Figure 4.1: PV Hub main web page mock up. Data was changed to generic names in order to not show unauthorized information.



D20210526_15_35

QoR Table Comment

Fail Detection

design	status	fatal_file	line	fatal_text	disk_avail	stack_trace
Design_1	DRFail	dprrpt.out.gz	1	Error: Current block is not defined. (DES-001)	--	stack trace
Design_2	PLFail	nwpopt.out.gz	1462	Error: Current block is not defined. (DES-001)	--	stack trace

QoR trend *compared to Flow_1_prev*

Metric_1	Metric_2	Metric_3	Metric_4	Metric_14	Metric_15	Metric_5	Metric_16	Metric_17	Metric_6	Metric_7	Metric_8	Metric_9	Metric_10	Metric_11	Metric_12	Metric_13
+0.06	+0.09	+0.12	-0.00	+5.35	+3.10	+0.34	+1.65	-6.15	+0.03	+0.22	+0.20	-0.47	+0.17	+41.22	+23.59	-0.26

QoR issues *compared to Flow_1_prev*

Show 10 entries

Metric	Design	Mean Value %	Design Value %	Histogram Location	Numeric Value Base	Numeric Value Flow	Issue Type
Metric_1	Design_1	0.06	1.6		519359.0	527667.0	outlier degrad.
Metric_10	Design_7	0.17	3.01		8.51	8.77	outlier degrad.
Metric_10	Design_1	0.17	2.8		0.711	0.731	outlier degrad.
Metric_10	Design_4	0.17	1.79		413.0	420.0	outlier degrad.
Metric_10	Design_2	0.17	1.61		2.89	2.94	outlier degrad.
Metric_10	Design_15	0.17	1.37		18.0	18.2	outlier degrad.
Metric_10	Design_8	0.17	1.28		0.221	0.224	outlier degrad.
Metric_10	Design_12	0.17	1.08		0.14	0.141	outlier degrad.
Metric_11	Design_16	41.22	185.6		2.84	8.1	outlier degrad.
Metric_11	Design_12	41.22	151.3		2.49	6.25	outlier degrad.

Showing 1 to 10 of 51 entries

Figure 4.2: PV Hub daily report page mock up. Data was changed to generic names in order to not show unauthorized information.

5 | Conclusion

This project was not about a discovery of new methodologies for software development or the invention of new EDA tools for the company, but rather the project was focused in using existing tools of software development to design a system to support the PV team of Synopsys, a task that is not the main priority of the PV team.

To design this service, a requirement engineering approach was used to understand the needs of the PV team regarding the issues that members of the team encounter during the Tracking process of issues in the code of software tools property of Synopsys. With the requirements defined, it was decided that a good way of supporting the Tracking process was to develop a web service that could facilitate the discovery and reporting of issues in Synopsys´ tools, along a data base capable of storing data of said issues for further analysis by the team. The structure of this solution can be seen in Figure 2.1.

After the solution was defined, an analysis of current technologies was done to determinate which framework for web development was suitable for the project. This analysis concluded that Django-Python was the better framework. Also, after concluding that a relational data base was a good data base type for the project, another analysis was done to determinate which RDBMS was suitable for the project, concluding that PostgreSQL was suitable. Both PostgreSQL and Django-Python complied with the security restrictions of the company to protect Synopsys data and Synopsys´ client data. Also, both were available in Synopsys environment and did not needed to gain too much more expertise on them before using them for development.

Then, a DevOps approach was used to develop both a data management system using PostgreSQL and a web interface system using Django. The web service development was divided into smaller projects, as shown in Figure 3.6, to haste both development and implementation for testing. For the data base project, first the data model of Figure 3.3 was created with the help from PV representatives, then the data base model of Figure 3.4 was designed to be implemented using PostgreSQL. With this data model in mind, a software tool called Health Collector, which can be seen in Figure 3.2, was developed to gather the necessary data needed to populate the data base.

DevOps also implies that while in development, the project is put operational for testing by a selected group of PV employees. This allowed the gathering of feedback from testers to correct or add smaller features to the project. However, this caused delays in the proposed Gantt chart for development shown in Figure A.1, so the project was finished later and some features were left behind. Deployment of the data base server using PostgreSQL was interrupted many times due to issues with the machine hosting the server, which caused to use development time to restore data not saved in the data base while the interruptions occurred. The Django web server also was hosted in the same machine host, but the issue with the machine host only delayed the testing time of the web service by PV employees.

The project was concluded during April of 2020 with the data base and web interface being used by certain PV employees since December 2019 and in a complete state until May 2020. Other urgent project inside the company required to deviate computational and human resources from this project to those ones, so in the end the servers of the project, which are the PostgreSQL and Django servers, were shutdown during May 2020 to avoid realizing related maintenance tasks.

However, that was not the final destination of the project, as the data base related part of it provided a new tool for data gathering for PV and the data model serves as template for future internal projects. Also, the data saved during the time the servers were online, which spanned a time frame of 4 months of relevant data, was then used to report the behaviour and status of the farming computers used to run experiments.

5.1 Knowledge requirements

The project realized is akin to any web development project that a person with knowledge on that area may be able to develop. However, this project in particular required to understand the concepts of digital electronics regarding digital circuits, digital gates&cells and digital phenomenons such as slack and power consumption. This was necessary to make a proper filtering of the data gathered and displayed in the web interface, so it could be of value to other PV employees and R&D engineers. So, rather than any web developer, the project required an engineer with digital electronics knowledge that also had web development skills.

5.2 Benefits to the company

A benefit provided by this project was that the data saved during the lifespan of the online data base was then used to analyze the behaviour of computer farms. This helped to attend a request by upper management to investigate why the farms where overwhelmed with experiments.

Another benefit of this project was that the Health Collector tool created during development to gather data for the data base was then re-purposed for the *Scheduler* project, which is still heavily used by PV and R&D teams across the company. The upgrade made to Scheduler by using Health Collector allows it to display the current status of experiments being executed on the computer farms, which saves time for employees monitoring said experiments that they would previously be using on searching and compiling this data in the result disk storing the experiment results or by searching this data in the computer farms directly. Now they can check this data in the same web interface in Scheduler they use to execute their experiments.

Lastly, most of the project ideas were recycled for the *PV Hub* project. PV Hub provided a quick implementation of the ideas developed during the project described in this document, that is the case of the data model, metric gathering and status analysis, but without worrying about the implementation of an online data base management system.

5.3 Challenges

One of the hardships of the project was to fully understand the concepts and logic of the Tracking process. Concepts such as Suite, Branch and Flow have specific meaning in the context of Tracking. More specifically, the way how the experiments store their results in disks suggest that a Suite is part of a Branch, and this is how the data model described in Figure 3.4. However, Branches are just versions of the code of a specific Tool developed by Synopsys, and Suites are a set of parameters and configurations designed to simulate the use of a Tool by some client. This means that it would have been more appropriate to define the Branch entity in the data model as parameter instead of an independent entity, so that the logic behind the Tracking process could have been translated more accurately.

On that note, the entity Product that represents the Tool ended up being not valuable as a separate entity, so it should have been better to define it as a parameter of the Suite.

These observations of the issues with the data model may prove to be useful when a new data base is designed in the future for some other project where data analysis and storage is relevant.

Another challenge of the project was to work with protected data by Synopsys. As previously mentioned, the company work with sensible data from its clients, which includes proprietary digital designs, know-hows and software tools. This implied that a project like the one reported in this document, which is composed of two online servers, required special care to avoid leaks of data. this meant to use secure web protocols as *https* that required verification tokens among other measures, and to be careful in not accidentally leaking data outside the VPN of the company.

5.4 Future work

The ideas implemented for the project may serve as a base for future internal tool developments of the PV Team.

Work done to create a data model and database may be re-used or re-implemented to fill the lack of data analysis done by the team. There are additional considerations and restrictions that could be used to upgrade the the data model designed here, so it can provide better conceptual value to management and co-workers. Such is the case of the Branch issue described previously.

Also, development of PV Hub may receive an addition influx of human resources to further development, such as to develop a proper online server for both database and web service of PV Hub. Additionally, features designed for the project described in this document may be implemented for PV Hub in the future to add more value for the company.

As an additional and final note, the pandemic hasted the contraction of the developer of this project, which meant that the now PV employee had to take care of new tasks that leave this project to a non priority status. However, as explained in this chapter, the work done for this project was far from wasted, but helped to support other projects inside the company.

Bibliography

[1] “The role of Requirement Engineering in Software Development Life Cycle”. Techrepublic, https://www.techrepublic.com/resource-library/whitepapers/the-role-of-requirement-engineering-in-software-development-life-cycle/ . Accessed November 2, 2019 2.1

[2] 02DCE (2020), "Software Engineering | Requirements Engineering Process", https://www.geeksforgeeks.org/software-engineering-requirements-engineering-process/ 2.1

[3] Axel van Lamsweerde (2009), “Requirements Engineering: From System Goals to UML Models to Software Specifications”, Université catholique de Louvain, Belgium. 2.1

[4] Hsiang-Jui Kung, LeeAnn Kung, Adrian Gardiner (2013), “Comparing Top-down with Bottom-up Approaches: Teaching Data Modeling”, Georgia Southern University, USA. 2.1

[5] Hui Li, Juan Chu, “Power analysis system based on data warehouse”, https://pdfs.semanticscholar.org/c887/82ce17a9ef5adf5bfa27e5403a7926f53a1e.pdf, Computer Department of Shandong University, Ji Nan 250061, Shandong province, China 2.1

[6] "Web API". Wikipedia, https://en.wikipedia.org/wiki/Web_API. Accessed November 10, 2019 2.1

[7] "Virtual Private Network". Wikipedia, https://en.wikipedia.org/wiki/Virtual_private_network. Accessed June 6, 2021. 3.2.1

[8] Phek Lan Thung, Chu Jian Ng, Swee Jing Thung, Shahida Sulaiman, “Improving a Web Application Using Design Patterns: A Case Study”, School of Computer Sciences Universiti Sains Malaysia 11800 USM, Penang, Malaysia 2.1

[9] “From Devs to Ops: An introduction”. AppDynamics, https://www.appdynamics.com/media/uploaded-files/White_Paper_-_An_Intro_to_DevOps.pdf . Accessed October 10, 2019. 2.1, 2.2.2

[10] “Python vs. PHP in 2020”. https://hackr.io/blog/python-vs-php-in-2019. Accessed December 18, 2019. 2.2.2

[11] Django documentation. https://docs.djangoproject.com/en/3.0/. 2.2.2

[12] Laravel Documentation. https://laravel.com/docs/7.x

[13] laravel quickstart guide. https://laravel.com/docs/4.2/quick 2.2.2

[14] PostgreSQL documentation. https://www.postgresql.org/docs/12/index.html 2.1, 2.2.3, 2.2.3

[15] MySQL documentation. https://dev.mysql.com/doc/ 2.1, 2.2.3, 2.2.3

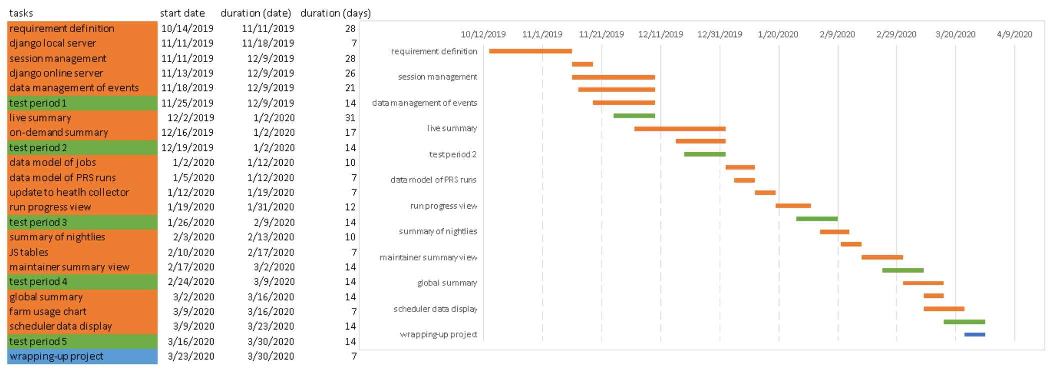
[16] SQLite documentation. https://www.sqlite.org/docs.html 2.1, 2.2.3

[17] "PostgreSQL vs MySQL: Everything you need to know". https://hackr.io/blog/postgresql-vsmysql. Accessed November 15, 2019. 2.2.3

[18] "RDBMS". Wikipedia, https://en.wikipedia.org/wiki/Relational_database#RDBMS Accessed December 21, 2019

2.1, 2.2

A | Appendix



tasks	start date	duration (date)	duration (days)
requirement definition	10/14/2019	11/11/2019	28
django local server	11/11/2019	11/18/2019	7
session management	11/11/2019	12/9/2019	28
django online server	11/13/2019	12/9/2019	26
data management of events	11/18/2019	12/9/2019	21
test period 1	11/25/2019	12/9/2019	14
live summary	12/2/2019	1/2/2020	31
on-demand summary	12/16/2019	1/2/2020	17
test period 2	12/19/2019	1/2/2020	14
data model of jobs	1/2/2020	1/12/2020	10
data model of PRS runs	1/5/2020	1/12/2020	7
update to heatlh collector	1/12/2020	1/19/2020	7
run progress view	1/19/2020	1/31/2020	12
test period 3	1/26/2020	2/9/2020	14
summary of nightlies	2/3/2020	2/13/2020	10
JS tables	2/10/2020	2/17/2020	7
maintainer summary view	2/17/2020	3/2/2020	14
test period 4	2/24/2020	3/9/2020	14
global summary	3/2/2020	3/16/2020	14
farm usage chart	3/9/2020	3/16/2020	7
scheduler data display	3/9/2020	3/23/2020	14
test period 5	3/16/2020	3/30/2020	14
wrapping-up project	3/23/2020	3/30/2020	7

Figure A.1: Gantt chart of the project agreed upon among the developer and the stakeholders.

B | License

Glossary

ACID Atomicity, Consistency, Isolation, Durability. 11

EDA Electronic design automation is a category of software tools for designing electronic systems such as integrated circuits and printed circuit boards. The tools work together in a design flow that chip designers use to design and analyze entire semiconductor chips. Since a modern semiconductor chip can have billions of components, EDA tools are essential for their design.. ii

Farm A group of computer resources divided as hosts that are used to execute computing processes sent by Synopsys employees.. 16

NDA A contract in which the entities involved agree not to disclose information covered by the agreement. Synopsys has NDAs with many of its clients to not disclose information from them.. 3

PV One of many teams that work in EDA related products of Synopsys. They aim to ensure the quality of Synopsys tools.. ii

R&D One of many teams that work in EDA related products of Synopsys. They design, implement new features to the software tools licensed by Synopsys.. ii

Stakeholder Entity that has an interest in the project. It can be a client, a final user, a manager, etc.. 6–10, 13

STAR A report filled in a specific web page to describe and track a code issue, or other kind of issues.. 2

Suite A set of configurations and scripts libraries for digital designs that define the usage of an EDA tool (such as Design Compiler, Fusion Compiler or other Synopsys tools) for quality control reasons.. ii, 1–3

Suite maintainer maintainer: An employee assigned with the task of monitoring amp; launching testcases regarding a specific suite, while also reporting and addressing various kind of issues derived from the tests results and execution.. ii, 2

Testcase A set of conditions and configurations which determine if certain feature is satisfactory.. ii

Tool A piece of software licenced by Synopsys to its clients. It is designed to support EDA development of digital circuits.. 3, 17

Tracking A common internal term used to refer the process of monitoring issues in a suite.. ii