FACTORS INFLUENCING THE DEVELOPMENT TIME FROM TRL4 TO TRL8 FOR CUBESAT SUBSYSTEMS AT A UNIVERSITY

Evelyn Honoré-Livermore[1], **Elizabeth F. Prentice**[2], **and Sivert Bakken**[2]

[1]*Department of Electronic Systems, Norwegian University of Science and Technology, 7491 Trondheim, Norway*
[2]*Department of Engineering Cybernetics, Norwegian University of Science and Technology, 7491 Trondheim, Norway*

ABSTRACT

It is challenging to estimate the development time of subsystems for CubeSats with low maturity. At the same time, in order to secure funding, and to not run out of funding, estimating the development time is an important part of planning a CubeSat project. We have looked at factors that influence the development time of a hardware-software payload system from TRL4 to TRL8, with the goal of integration into a CubeSat bus for a research mission. Our analysis showed that the most critical factors affecting the development time were clear objectives, internal communication and team knowledge. A tentative relationship between maturity level and factors is presented to help university project managers plan resource needs and mitigating activities.

1. BACKGROUND

CubeSat projects at university have gained tremendous popularity the past decades. The objectives of the projects have been two-fold, providing experience of space systems and project to students, and providing scientific data from CubeSat missions such as Earth Observation. Some of the challenges these projects experience are project management; balancing coursework and satellite work; high turnover; and ensuring mission success [1]. Berthoud et al. (2019) discuss CubeSat project management based on three university case studies. They saw that there was a gap of 2.1 years [2] between industry and university CubeSat development time, and that there is a large number of "lessons learned" papers published on different aspects of university CubeSat projects. Among their findings from the case studies, the authors highlighted several sociotechnical characteristics of the team structure and project management (such as motivated staff, a nominal lifecycle of 2–3 years, passionate students, mixing coursework and non-coursework, version-control), and a strong emphasis on testing and integration to achieve a successful mission.

Aforementioned challenges result in cost or schedule overruns and add to the difficulty to manage and enable estimation of the development time of CubeSat subsystems at university. To achieve better estimates, we first need to understand the factors influencing the development time.

1.1. Product Development Time

To estimate the project schedule, the scope of work, resources available and funding available must be estimated and assessed. For CubeSat projects in academia, the resources commonly include volunteer students, faculty members, engineering support and researchers or Ph.D. students [3, 4]. The amount of funding needed depends on the mission of the project, and can for example be provided through governmental funding bodies or industry support. The scope of work relates to the phases the project encompasses, and the maturity of the systems involved and which maturity level they should reach within the project. It is common to re-assess the schedule regularly to communicate with stakeholders and ensure project success, as sources of funding may run out or resources leave the university or become unavailable. The lifecycle of a system is commonly divided into phases such as (1) Definition and analysis phase; (2) Technology development; (3) Engineering and manufacturing development or detailed design definition; (4) Production and deployment, and (5) Operations and support [5]. For this paper, we are concerned with phases (1)-(4), which commonly correspond to the time before launching and operating the spacecraft.

1.2. Maturity Levels

Maturity levels of technology, integration and system have been used to make more informed estimations of schedule. These are known as *Technology Readiness Level (TRL)*, *Integration Readiness Level (IRL)*, and *System Readiness Level (SRL)* [6, 7]. Originally developed for defense acquisitions, maturity levels have been adopted European Space Agency (ESA) and National Aeronautics and Space Administration (NASA) in their projects and are being used in other commercial and noncommercial sectors as well [8]. TRL is the most widely known and used maturity level, CubeSat subsystem vendors use TRL in product data sheets to provide information to their customers, and most can easily use the level when assessing which subsystems to purchase for their system. Some associate maturity levels with the product development lifecycle [9], which can help management determine if a gateway is completed successfully or not.

Funding bodies and larger defense companies may require maturity level assessments as a part of evaluating the project, and have procedures and methodologies in place for this [7, 9]. Weiping et al. (2011) discuss different approaches to using maturity level assessments and limitations of these. The most apparent limitation is how IRL and TRL are assessed by *subject matter experts* supported by comparison to previous work, established standards and **gut feelings**. Furthermore, the maturity levels themselves use wording that is open to interpretation depending on the person assessing and the context.

The TRL of a technology is assessed on a scale from 1–9, by "subject matter experts" [9, Table I], where the context and environment that the technology shall be used in becomes relevant at levels 5 and up. The IRL is also assessed on a scale from 1–9 by subject matter experts, which ranges from definition of the interface through structured communication to success in a deployed system [9]. The SRL is given by a combination of the TRLs of the technologies involved, and the IRL. Tompkins et al. (2020) list the major four calculation methodologies for determining the SRL, where the Sauser SRL method is given in Eq. 1.

$$SRL = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{m_i}\left[\left(\frac{1}{9}IRL_{SRL}^{n\times n}\right) * \left(\frac{1}{9}TRL_{n,1}\right)\right]\right] \tag{1}$$

where n is the number of elements in the TRL vector, each represented as TRL_i, the IRL_{ij} is measured as an integration between elements i and j. The factor m_i is the number of integrations per element (i.e. the non-zero elements in each row in the IRL matrix, see Eqs. 4 and 5. The IRL_{ii}, i.e. the integration of an element with itself, is assigned a level of 9. The values are then normalized and the SRL matrix is calculated by taking the matrix product of the IRL matrix and the TRL vector. Following the procedure of [9], we show the building blocks of Eq. 1.

$$[SRL]_{n\times 1} = [IRL]_{n\times n} \times [TRL]_{n\times 1} \tag{2}$$

The SRL of an element is given in Eq. 3, a combination of the maturity of the specific element or technology and its interfaces to the adjacent elements.

$$[SRL] = \begin{bmatrix} SRL_1 \\ SRL_2 \\ ... \\ SRL_n \end{bmatrix}$$

$$= \begin{bmatrix} IRL_{11}TRL_1 + IRL_{12}TRL_2 + ... + IRL_{1n}TRL_n \\ IRL_{21}TRL_1 + IRL_{22}TRL_2 + ... + IRL_{2n}TRL_n \\ ... \\ IRL_{n1}TRL_1 + IRL_{n2}TRL_2 + ... + IRL_{nn}TRL_n \end{bmatrix} \tag{3}$$

We are interested in the full SRL of our system, which is the arithmetic mean of each of the element SRL. The element SRL can be calculated by normalizing the SRL value, shown in Eq. 4 where m_i is the number of integrations of the element i including its own interface.

$$SRL_{element} = \frac{SRL_i}{m_i} \tag{4}$$

To get the composite SRL, we take Eq. 4 and calculate the average for each of the element SRL, shown in Eq. 5.

$$SRL_{comp} = \frac{\frac{SRL_1}{m_1} + \frac{SRL_2}{m_2} + ... + \frac{SRL_n}{m_n}}{n} \tag{5}$$

By combining Eqs. 2-5, we get Eq. 1, which is used to assess the maturity of a system at different phases in the lifecycle. The SRL is a numerical value from $0 - 1$, where 1 is full maturity, where we can expect that it is operationally deployed.

2. METHODS

We have followed a university CubeSat team from Phase 0/A to Phase D/E, in which the payload subsystem has been developed from TRL4 to TRL8. The payload includes both software and optomechanical hardware. The university CubeSat team consists of 20 students and 6-8 Ph.D. and Post.Doc. fellows. The students join for 1-2 semesters as part of their thesis.

The payload of the CubeSat under study is a hyperspectral imager with ambitious on-board processing capabilities. With the appropriate processing the mission aims to increase the response times and save bandwidth for operations. Low-level programming and specialized hardware is needed to achieve this with the chosen on-board computer that has limited resources in terms of power and computational capacity. While the concept of splitting light into a spectrum of wavelengths is not new (TRL1), the chosen instrument and optical design was TRL3/4 at the beginning of the project. The imager is based on Commercial-Off-The-Shelf (COTS) components and worked in a laboratory setting.

2.1. Case Study

The method used for the case study has been action research [10, 11]. The authors are active participants in the project, and have project management and subsystem management roles, which includes continuous improvement of processes and procedures. Data sources include project documentation, statistics from GitHub and semi-structured interviews (n=5) with some of the team members focused on interfaces and product development.

Technology development followed closely from the work of two main teams – software and hardware. Beginning at TRL4, these two teams worked primarily separate and technology grew independently. In order to approach TRL6, coordination between the two teams was necessary in building subsystem and system level prototypes. Two case studies on technology development are presented, one from each team. The studies illustrate that although development grew from separate sources and teams, many of the same factors influenced the pace of the project.

The semi-structured interviews were conducted with representatives which had been a part of integration activities at different periods of the project. The interviewees were asked to describe their subsystem, if they were able to draw a block diagram with its interfaces, describe the different types of interfaces, the information sources used to understand the interfaces, if they had utilized sequence diagrams or Interface Control Document (ICD)s, and what was challenging or easy with the development of their subsystems.

2.2. Assessing the Readiness Levels

We assessed the TRL and IRL of the system at the current status, in lifecycle phase (3), as the project has not reached phases (4)-(5) yet. For the purpose of providing information to estimating development time and schedule, the limitations of subjectivity and bias when assessing maturity levels become less problematic when used for internal evaluation and communication because it will be the same people doing relative assessment of the technologies over a longer period of time [12].

3. RESULTS

The original project schedule was made in late 2017, before the project had its Mission Design Review (MDR) and before deciding which spacecraft bus and subsystems to integrate the payload with. The project schedule has been revised at the major gateways (Preliminary Design Review (PDR) in December 2018, a new PDR in June 2019, and Critical Design Review (CDR) in March 2020), in addition to small adjustments continuously.

This section describes the development of software and hardware, and identifies the main factors influencing the development time. The semi-structured interviews are used to illustrate how a typical team member experiences the development process.

3.1. Software team development

In this CubeSat project, students and staff have contributed to different software stack components i.e. Hardware, Operating Systems, Middleware, Applications, and user interface layer. To achieve this a lot of development considerations needed to be made, agreed upon and followed-up to ensure the desired level of coherence and maintainability of the software stack. Initially, there was no such coherent workflow available to guide the students in the development and the documentation and code were less intelligible as a result. The need for such a workflow became evident after software PDR.

Thus, a workflow was proposed to better guide the software stack development coherently. This is now a part of the on-boarding procedure for new students. Given the limited time students can commit to the project, some only for one semester, the workflow needed to be attainable. That is, there was a limitation within the workflow to utilize development tools that the students should be familiar with concerning programming language, revision control system, build environment, and so on. The first batch of students that were introduced to the baseline workflow found that it took some time to adapt, but that the benefits in terms of collaborative work made it worthwhile. Through the use of the workflow, in close collaboration with other subsystem teams, it has become more familiar and better exploited across the team.

Most incoming students on the team had some experience with the programming languages C/C++ for low-level programming from university classes. Embedded systems, such as the one planned for the CubeSat under study, have a functional compiler in C, and the support for C found in Application Programming Interface (API) is not as common as for other low-level languages. There are some challenges related to using low-level programming and specialized hardware. The C language can be hazardous as it provides the programmer more direct control over the memory usage and run-time behavior. This can again provide better utilization of the computational capacity available. There are *safer* languages than C that rely on a larger runtime, a more complicated feature set, and maybe even virtual machines to work. However, the compilers available for C perform well even with a limited computational capacity. Using a *safer* language would result in a longer onboarding process. Through the use of such tools in the software development stack as compiler warnings, linters[1], and other static analysis tools to detect preventable issues the potential pitfalls of C can to some extent be mitigated.

Another major point was finding a shared platform to develop on. Git by GitHub was the chosen high-level tool used for the development of all software and issue tracking, where the branching strategy known as GitHub Flow was selected [13]. The software CDR highlighted the importance of using GitHub and its functions. This has also been used to enable a scrum approach to software development [14]. The issue tracking provided a common platform to discuss software development, making it easier to talk about software, document software, ask questions, or request new features. The GitHub Flow branching strategy aims to have a working master branch with as little overhead as possible. This lessened some complications with integration as the developers were continuously reminded that their contributions were expected to be deployed on the target hardware.

Specialized hardware that enables the use of field-programmable-gate-arrays, i.e. re-configurable hardware logic, is used to further expand the onboard processing capabilities. This comes at the cost of increased complexity in terms of development and integration. Developing in hardware description languages is time demanding, and does not encourage a lot of online processing flexibility. Development of software tests, and writing

[1]A tool that analyzes code to find errors and bugs.

code that is modularized and testable can and has been shown to further improve the robustness of the deployed software. However, this can be hard to motivate in general, and especially as part of academic work. Prioritizing contributions to the software stack versus progression on personal academic work has proven to be a challenge, but the team has developed strategies to find synergies between the two.

3.2. Hardware team development

Hardware development was divided into two phases, the technology development phase where much of the design work was completed, and the engineering and manufacturing phase where assembly and testing occurred. Through these phases we were able to push our technologies through TRL levels. Hardware development focused on the primary payload, or hyperspecral imager. More details on development follows.

Technology development phase: The TRL4 model was a functioning hyperspectral imager used for desktop measurements or unmanned aerial vehicle flights. However, its optics had been especially designed for imaging from low earth orbit. The lab was an empty room, available facilities on campus were yet undiscovered by the team, no one had training to use other facilities, there was no database nor documented guidance on how to proceed, and nothing had been purchased so there was no relationship with industry suppliers. Very few students involved had prior experience in the CubeSat discipline nor did the project advisors. At this point, a lot of time was spent planning and defining requirements both for the mission and systems on board. At this time, the hardware team focused on a full exploration of the capabilities of the imager along with understanding its assembly/disassembly and nuances.

At TRL5, the hyperspectral imager had been dissected into components and much of the time went into understanding if individual parts could withstand the space environment. Here, individual lenses, detectors, etc. were tested in vacuum and thermal chambers. Design modifications and new assembly procedures were developed based on experience gained from testing.

Engineering and manufacturing phase: Coordination with the software team became extremely important to achieve TRL6. Hardware and software were combined to demonstrate a working prototype. Here, many missing or faulty interfaces, cables, connectors, etc. were discovered. The work at this stage was severely underestimated, even down to the timeline of getting parts machined inhouse. In addition to getting the payload working and mounted in the satellite, the prototype also went through basic environmental testing to prepare for design qualification.

To reach TRL7, is the completion of environmental testing on the qualification model. At this stage, most parts had already been ordered, test objectives outlined, test plans written, and the design was frozen. Most time went into waiting for machine time at external facilities and working through test plans and reports.

As with software, the major factors influencing each of these stages in hardware development can be broken down into categories: internal facilities (lab and university), team knowledge, parts supply chain, clear objectives, and external facilities.

The chart in Fig. 1 illustrates a breakdown of the primary factors that influenced each TRL for the hyperspectral imager development from a hardware perspective.

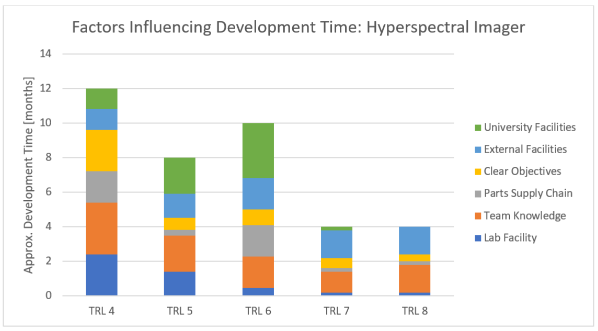

Figure 1: Factors influencing the development time of the hyperspectral imager.

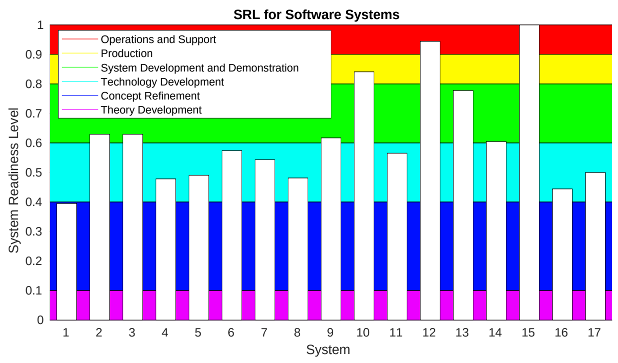

Figure 2: System Readiness Levels for software modules. Systems 1-9 are NTNU payload, systems 10-14 are spacecraft software modules, and systems 15-17 are ground software modules.

In summary, important factors influencing hardware and software development are:

1. **Lab facility:** Target Hardware set up correctly, what tools and machines are in the lab, design and testing software, data storage and management, spare hardware parts and materials, ESD protected areas[2], special corona restrictions and control, remote access to target hardware. Available target hardware for testing.
2. **University facilities:** machine availability, prioritization of projects, training and access required to

[2]Where all surfaces, people and objects are kept at the same potential to avoid damage to electronics.

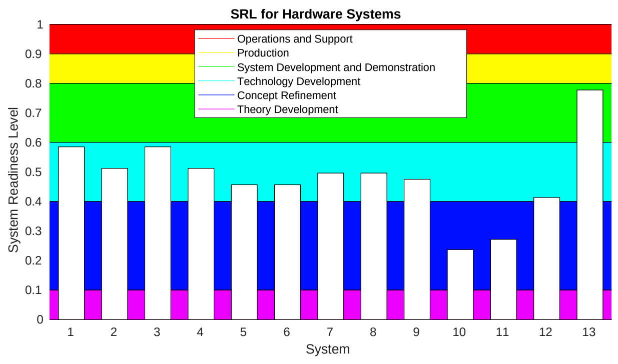

Figure 3: System Readiness Levels for hardware. Systems 1-6 are NTNU payload, systems 10-13 are the thermal, vacuum, radiation and vibration subsystems.

use the facilities, special corona restrictions and control.

3. **Team knowledge:** Prior knowledge from existing team members and advisors, a database of documentation or templates, professional network, and ways to reach them with questions, a common workflow.
4. **Parts supply chain:** shipping procedures, known suppliers, special suppliers for space materials, invoicing, methods for speeding up delivery of parts if necessary.
5. **Clear objectives:** A roadmap of the project, well-defined roles and responsibilities, technological requirements that are well defined, organization, a common workflow and concise software stack.
6. **Internal communication:** A common workflow and concise software stack used for issue tracking and version control. Willingness and ability to utilize the recommended tools.
7. **External facilities:** understanding of what facilities have available and what they require, what their schedules look like, how they prioritize university projects, costs involved.

3.3. Maturity levels

The maturity levels can be linked to the factors identified as shown in Fig. 1. This can help plan the next phases, where project managers should identify mitigation actions to reduce the impact of the factors. Furthermore, we recommend using the system readiness levels to focus development efforts. For example, in Fig. 2, we see that **system 1 (SS1)** has an SRL of 0.4, while some of the other NTNU payload software modules are higher (0.6 and above). This means that management should focus resources on developing **SS1** until it reaches a higher maturity level. For hardware, we see the same for **systems 10 and 11 (HS10, HS11)**, which are the environmental interfaces. The project has not yet finalized the thermal subsystem design (HS10), while the payload has been tested in a vibration environment (HS13).

3.4. Interview results

All interviewees (n=5) answered that they would be able to draw a block diagram of their subsystem and its interfaces. The interviewees were able to identify most of the physical interfaces and data links, and some (n=4) identified the interfaces with people within the organization and external (suppliers and support functions at the university). When asked explicitly, the interviewees were able to describe in detail the people and operational interfaces. Some interviewees (n=2) had created ICDs for their documentation, while all interviewees (n=5) have used them for their subsystem development. Most have created readmes for their software, and software team members had created architecture diagrams as well.

People did not use the previous master theses as knowledge source as much as the project documentation. Some interviewees (n=2) introduced the concept of different viewpoints when describing their subsystem, for example the network layers (the 7-layer open systems interconnection model), as one way of describing their subsystem, and which layers they were working on. Most interviewees agreed stated it was important to have the right descriptions of the subsystems used to be able to integrate them properly. For example, how the operating system maps onto the chosen hardware, and that having diagrams showing how the different software modules are interconnected is helpful. How the payload subsystem connects to the interfaces is not supposed to change, but one challenge is to get the right information about the interfaces. It takes time to get the interfaces from third-party suppliers, and one hypothesis is when the technology the team is integrating with is also being developed concurrently to provide the capabilities the mission needs.

Interviewees stated that development was easier when the scope and interfaces were defined. The development challenges were rooted in a lack of specification if not all functions had been identified, and when the architecture decided early in the project did not accommodate the introduction of new functions or capabilities well. It is difficult to know exactly what is needed when the requirements have not been fully developed at the top or derived to specifications. As a young student it was difficult to determine which architecture was suitable for the payload and mission needs. It is easy to start development "too early" without a proper process or guidance.

4. DISCUSSION AND CONCLUSION

The experience from developing the payload identified the following factors influencing the development time: (1) lab facility; (2) university facilities; (3) team knowledge; (4) parts supply chain; (5) clear objectives; (6) internal communication; and (7) external facilities.

The interviewees did not mention *(1) lab*, *(2) university* or *(7) external facilities*, or *(4) parts supply chain* as factors influencing their work. However, these factors are at a managerial or group leader level, and the interviewees were all team members, which may be why these were not mentioned during the interviews.

Regarding *(3) team knowledge*, the interviewees mentioned that it was helpful with all the project documentation such as ICDs, and that there were more senior team members available on Slack to answer questions. However, some interviewees identified that it was challenging to know which questions to ask because it takes time to build the basic knowledge needed. In relation to this, the *(4) parts supply chain* and main subsystem provider influenced the Norwegian University of Science and Technology (NTNU) team more through interface definitions.

The semi-structured interviews showed a strong agreement with *(5) clear objectives*, where well-defined requirements and required capabilities and functions were often not present at the start of a student thesis project. Interviewees identified that development would be easier with well-defined scopes and interfaces, which is linked to having clear objectives of the work. Furthermore, a lack of required capabilities and functions can cause the architecture choice to be incompatible with future refinement of functions and mission.

Finally, some of the interviewees mentioned that *(6) internal communication* was challenging cross-team or cross-function. For example, that the software development was not fully aligned with the operations need. However, the Scrum stand-up meetings help with internal communication, because they provide low level day-to-day tasks and lowers barriers for informal communication.

To enable future planning, we recommend project managers of university CubeSat projects to use SRL, TRL and IRL to help prioritize tasks. It is also a useful communication tool with team members, to ensure alignment and common understanding of the different systems involved. The use of IRL and SRL in addition to the commonly used TRL highlights the need for integration of subsystems which may be forgotten if students are focusing on separate theses. Furthermore, we found that the factors influencing development time vary depending on the maturity level of the technology. For instance, the need for *clear objectives* are critical in the early planning phases and verification phases, or that *university facilities* play a key role in moving from TRL6 to TRL7. The maturity levels can be used for communication with university departments to help plan the workshop and testing availability.

Future work includes using these assessments at multiple gateways of the project, and measure quantitatively how long time each system takes to progress in terms in TRL, IRL and SRL. Furthermore, the authors are also interested in learning about other university CubeSat teams and their experiences with maturity level assessments.

ACKNOWLEDGMENTS

This work was supported by the Norwegian Research Council through the Centre of Autonomous Marine Operations and Systems (NTNU AMOS) (grant no. 223254), the MASSIVE project (grant no. 270959).

REFERENCES

[1] Alminde, L. et al. “Experience and methodology gained from 4 years of student satellite projects”. In: *2nd International Conference on Recent Advances in Space Technologies*. Vol. 1. Turkey, Istanbul: IEEE Xplore, 2005, pp. 94–99.

[2] Richardson, G. et al. “Small Satellite Trends 2009-2013”. In: *AIAA/USU Conference on Small Satellites SSC-VII-3*. 2015.

[3] Cho, M. and Mazui, H. “Best practices for successful lean satellite projects”. In: *4th UNISEC Global Meeting*. Series Best practices for successful lean satellite projects. 2016.

[4] Berthoud, L. et al. “University CubeSat Project Management for Success”. In: *Conference on Small Satellites*. 2019.

[5] Tan, W., Ramirez-Marquez, J., and Sauser, B. “A Probabilistic Approach to System Maturity Assessment”. In: *Systems Engineering* 14 (2011), pp. 279–293.

[6] Mankins, J. *Technology Readiness Level*. Tech. rep. Office of Space Access and Technology, NASA, 1995.

[7] Sauser, B. et al. *A Systems Approach to Expanding the Technology Readiness Level within Defense Acquisition*. Tech. rep. 2009.

[8] Rybicka, J., Tiwari, A., and Leeke, G. A. “Technology readiness level assessment of composites recycling technologies”. In: *Journal of Cleaner Production* 112 (2016), pp. 1001–1012.

[9] Tompkins, Z., Grenn, M., and Roberts, B. “Improving System Maturity Assessments by Incorporating a Design Structure Matrix”. In: *IEEE Transactions on Engineering Management* 67.1 (2020), pp. 122–140.

[10] Tripp, D. “Action research: a methodological introduction”. In: *Educação e Pesquisa* 31 (2005).

[11] Yin, R. K. *Case Study Research and Applications: Design and Methods*. 6th ed. SAGE Publications Inc, 2017.

[12] Austin, M. F. and York, D. M. “System Readiness Assessment (SRA) an Illustrative Example”. In: *Procedia Computer Science* 44 (2015), pp. 486–496.

[13] GitHub, I. *Understanding the GitHub flow · GitHub Guides*. `https://guides.github.com/introduction/flow/`. (Accessed on 07/03/2020). 2017.

[14] Rising, L. and Janoff, N. S. “The Scrum software development process for small teams”. In: *IEEE Software* 17.4 (2000), pp. 26–32.