# *Developing innovative software in Brazilian public universities: Tailoring agile processes to the reality of research and development laboratories*

Igor Muzetti Pereira, Tiago Garcia de Senna Carneiro, Rodrigo Reis Pereira
Earth System Modeling and Simulation Laboratory (TerraLAB)
Federal University of OuroPreto, UFOP
Ouro Preto, Brazil
{igormuzetti,tiagogsc,rreisp}@gmail.com

***Abstract*—This paper considers the challenges involved in open source software development in a Brazilian public university, where software development is done by both undergraduates and graduates. In Brazil, scientific research and technological innovation are mostly performed in university labs and not in private companies. Universities should transfer technology to industry producing richness. They should not compete with private companies in the local market. In these labs, developers are still in training and cannot assume all roles in the development process. They might not be fully committed to projects and their lifestyle will not depend on their income. This paper presents a case study of open source software development in this environment. We customize a process that mixes up artifacts and iteration dynamics from Scrum, roles and test-driven aspects from Extreme Programming, and management practices and team structure from PMBoK. During three years, twelve software projects have been executed and monitored by diverse processes measures. Meanwhile, the process has been evolved to improve these measures. Data analysis shows that using a well-defined test process is conducive to the production of good quality software in academic labs. The availability of a project schedule and team productivity information encourages the students to work productively and efficiently. Our main contribution is to provide evidence that, through the process we have customized, teams formed mostly of undergraduate students can develop and maintain long-lasting and innovative software, which is being used by institutions spread around the world. This process can be used by other academic labs with similar characteristics.**

***Keywords— software and system engineering; software process improvement; software innovation inuniversity laboratories; computer science undergradute courses***

## I. INTRODUCTION

This paper considers the challenges involved in open source software development in Brazilian public universities, where software development is done by undergraduate and graduate students, led by one or more university professors. From a software engineering perspective, a university lab has distinctive characteristics; It is both a place for learning and for innovation. Students have a chance to learn how to work in a controlled production environment. At the same time, working on open source software which has outside users helps to create a sense of responsibility and a sense of customer demand that the students will face in their professional life. Therefore, a university lab is a microcosm with some similarities and some differences to a commercial software production company.

Undergraduates work between 12 and 20 hours per week. They are still in training (not professionals) and may have different levels of knowledge. They cannot assume all roles in the development process and may not be fully committed to projects. Their degrees do not depend on the success of projects and their lifestyle does not depend on their income. They can leave the team at any time without major problems to their careers.

These differences between academic labs and commercial companies lead to challenges for long-lasting software development and maintenance: how can we efficiently produce good quality software with teams composed mostly of undergraduate students?

Considering these challenges, this work presents a case study of open source software development in a university lab. We found out that an effective way to manage the lab was to set up a development process that combines scored and prioritized user stories, roles, shared project backlog, sprint planning and meeting dynamics from Scrum [1][2], test-driven focus from Extreme Programming (XP) [3], and team structure and management practices as described in the Project Management Body of Knowledge (PMBoK). Combining key parts of these models, we can organize the student team to produce good quality software. We are also able to derive quantitative metrics to assess individual and team performance.

Data analysis suggest that using a well-defined test process is conducive to the production of good quality software in academic labs. Along case study, the adoption of test-driven practices and automated testes has reduced the number of bugs perceived by customers. Students respond well to rapid feedback. The availability of a project schedule and team productivity allow students to recognize their faults and encourages then to work dedicated their time and produce better software. We aim to provide evidence that the customization of some agile practices with strict management practices led to a process with short development cycles. This

was with rapid feedback used in a laboratory environment in a university, where most of the employees are undergraduate students who develop free, durable software, and have multiple users dispersed across the planet.

## II. BACKGROUND

### A. Software project management and development processes

Project management theorists and practitioners have defined methodologies and standards followed by several industries to achieve success and certifications [4][5]. However, these are not usual practices in university labs. Reasons may include the lack of knowledge about management practices, the cost and effort involved in the implementation, or the focus on research and innovation activities.

Plan oriented development processes, or RUP - Rational Unified Process, deal with these challenges by adopting a heavy load of bureaucracy, obligating developers to strictly follow predetermined plans. Projects evolve in non-overlapping phases, using standardized processes and artifacts. Any change in project scope must be avoided so as not to compromise schedule and cost, otherwise the contract must be renegotiated[6].

Literature describes the use of XP methodology in the disciplines of a computer science course, which is something that differs from our case study. Ours runs in a research and development lab where most of the employees are undergraduate students. Their grades do not depend on their performance in the project and in this case if they leave the lab, their grades will be unaffected. The article [7] highlights the supervisors role that assists students to solve technical issues and to follow the practices of XP. In our study, there are two roles. The team leader helps his team in technical issues. The project manager helps several teams to follow the development process and uses the same measures to track them and comparing their performance. Project managers award good teams and work to improve the performance of other teams.

Melnik and Maurer (2003) demonstrate that the application of XP in academia can prove more difficult than in industry. This is because, first, students are not fully committed to their projects, and, second, they have to balance the learning of both XP with several other disciplines [8]. In an academic environment, the peaks in a team's velocity is shown to suffer due to the students approximation of the delivery dates. In comparison, professionals in industry retain a more constant speed of productivity throughout project. A report of the BOPE noting the productivity of teams `sprint phase´ is released on a weekly basis with the intention of avoiding a last minute cramming of work prior to the deadline which is found to improve the overall quality of deliverables. It has been demonstrated that projects that use XP in Industry have a flexible scope and fixed time, but in academic projects, the scope is fixed and time is flexible [8]. This flexibility is echoed in the projects of BOPE: a consequence of the research environment.

Some lessons were also identified in literature, including a pattern of resistance among students in the adoption of TDD (Test Driven Development), and a resistance to conceptualizing an idea prior to any development of code [9]. The adoption of TDD was not welcomed at first, but over time this practice was well regarded by student as they began to find that writing first the tests for new features was helpful. Planning estimates of the stories was a challenge for the students, dividing the stories into smaller stories to be estimated was not easy because the students did not know how long it would take to write the code. As the project progressed and they found the possibility of estimating based on similar stories, the students began to adjust their estimates. These lessons were also identified in our case study.

Santos et. al (2012) highlight an improvement in implementing the use of XP in the computing disciplines where they adopt a wiki to document lessons learned. Lessons could be used in other courses [10]. Our process also maintains a collaborative wiki, where project documents and artifacts are recorded for monitoring and are always within reach.

Implementing a process within a academic environment can be a difficult task. A lot of time will be consumed training new students. Keeping a team following the process and project schedule can be strenuous. Requirements of innovative or research products are prone to change with some constancy. Another problem is the great volume of artifacts (documents, diagrams, etc) generated during the process. In general, they must be reviewed by the laboratory leader, i. e., the professor/researcher in charge, who quickly will become the process bottleneck. Thus, the process will not scale well.

Agile methods favor interactions between developers over using standardized processes and tools. They privilege responding to changes over following a plan. They prefer customer collaboration over contract negotiation and working software over comprehensive documentation [11]. They are softer and easier to implement. However, there may be some problems with their straight implementation in university laboratories.

The low level of bureaucracy may compromise project schedule and product quality if developers are not fully committed to project success and if they are still in training. In agile processes, product quality is tightly linked to the skills and experience of developers. The laboratory leader, who often acts as project customer, may be not available every day to interact with the development team. If there are many projects running, he/she will become the development process bottleneck. Again, the process will not scale well.

### B. Software process improvement in academic laboratories

Synergia laboratory located in the Department of Computer Science (DCC) of Federal University of Minas Gerais (UFMG) is organized as a traditional software factory. Although its environment retains academic characteristics, these are some dissimilarities with the environment tackled in this case study. Synergia has no difficulties in staffing capable and experienced teams. A good percentage of the staff have a masters degree. Graduate students receive a very attractive scholarship supplement when compared to scholarships granted by public research agencies [12].

Since 2000, Synergia has used a customized version of the software development process named Praxis [13]. Enhancements include fine-tuning of testing procedures, using

more process automation tools, and adopting project management procedures based on PMBoK [4]. Started in 2007, the implementation of a software process improvement (SPI) program results in an objective measurement to evaluate SPI adoption. Namely, the percentage of rework related to the test execution (% of rework = rework / (work + rework)) and the number of defects. At the end of a case study project, the amount of use cases without defects increased from 21% to 53%. The average number of defects reduced from 5.1 to 1.7 defects per use case [14].

Goldman et. al (2004) presents effective ways of teaching students and professionals how to develop high-quality software following the principles of XP process [15]. They define a new and important role in the process. The *libero* is an experienced developer who was not assigned any user stories; his task is to pair others, helping them finish their tasks. They conclude that in despite of the *a priori* fears of the consequences and effectiveness of XP, once developers and managers have real contact with a well-run XP project, the fears quickly dissipate.

Sato et. al (2006) evaluate several measures to track academic and governmental software projects executed via the XP process [16]. Project teams were composed by experienced coaches and mostly by developers without previous experience in XP. They concluded that all teams had a high willingness to improve in all XP practices. Retrospective meetings (also known as Reflection Workshops) are effective tools to help teams to understand the pace. They also propose new measures to diagnose how well testing and continuous integration is going in projects.

Lisboa et. al describe the steps of a development process based on the *software product line approach* and also describes an educational methodology to teach the state-of-the-art reusable software development in post-graduate courses [17]. They highlight the importance of the commitment of domain experts for the project success. They also comment on the deficiency in finding measures for feature analysis and the difficulty in defining the granularity of requirements and use cases.

Some researches evaluate approaches for teaching agile methods or for developing academic projects [18][19][20][21]. Some point out that the XP-like process resulted in good team communication and a broader knowledge of the project as a whole, highlighting that due to scheduling problems it is more difficult to make XP work in the academic environment than in the industrial. They also emphasize that, in the industry, the agile projects are normally of the "Flexible Scope-Fixed Time" nature, while in academia projects are always "Fixed Scope-Flexible Time" [18]. In this sense, projects conducted in our case study have more similarities with industrial projects. Also, our case study is more concerned with the development of open source software and less concerned with the teaching of agile methods.

### C. Organizational context

Our case study was conducted in the Earth System Modeling and Simulation Laboratory - TerraLAB (www.terralab.ufop.br). This lab is the result of a partnership between the National Institute of Space Research (INPE) and the Federal University of Ouro Preto (UFOP). It develops computer systems related to geo-processing and to environmental modeling. During the three year case study, the TerraLAB team changed a lot due to turnover rate. Twelve medium size software products were developed under the a process that has been continuously evolving. The size of project teams ranged from 4 to 19 developers. Some teams involved developers from several institutions. Currently, the TerraLAB local team is composed by 5 masters and 14 undergraduate students. After complete the initial disciplines, Masters students should work 40 hours per week. While, undergraduates students should 20 hours.

## III. BOPE DEVELOPMENT PROCESS

### A. Overview

In this work we tailor a new process that mixes up Scrum, XP and the guidelines from PMBoK. BOPE is a process driven by artifacts and the main artifacts are tests. This means that any tasks performed by the team must result in a standardized and measurable artifact. Both research and software projects can be managed through BOPE process. Even products such as publications demand activities that can be decomposed into tasks, in which input and output are standardized and measurable artifacts. Bibliographic review takes many articles as input and produces several summarized and highlighted articles as output. Scientific hypotheses should be tested through experiments in which plan and report have the same value of test plan and test report.

### B. Project structure

In BOPE, projects have at least the following information and structure:

- **Project scope**: It defines the context in which the project is involved. It defines the project mission and specific goals. It describes the intended benefits to the customer. To manage customer expectations, it describes what is expected to be produced from the project and what is not.
- **Product backlog (or project backlog)**:The product backlog is a list of main project deliverables, for instance: software, computational models, courses, monographs, publications, etc. It is ideal to know the time and cost required by the customer for each deliverable. The issue is to have enough information to build the initial project schedule and to prepare a technical and commercial proposal. *How many user stories should be done? How complex can they be? In what period? Who will do it? What resources will be necessary?* The product backlog can be expanded and changed throughout the project to satisfy the customer expectations.
- **User Stories**: The main user stories (or requirements) of each deliverable are identified and prioritized. User stories are lightweight descriptions of how software is used and must behave. Along with the project, they should be detailed, specific, measurable, achievable,

relevant and timeboxed [18]. Specific and measurable mean that user stories should be testable and that pairs {input, expected output} are known. Achievable means that if developers cannot deliver a user story in an iteration, it should be detailed in several user stories. Timeboxed means that it is important to know when to give up on a user story. Relevant means that user stories should provide as much value as possible to the customer.

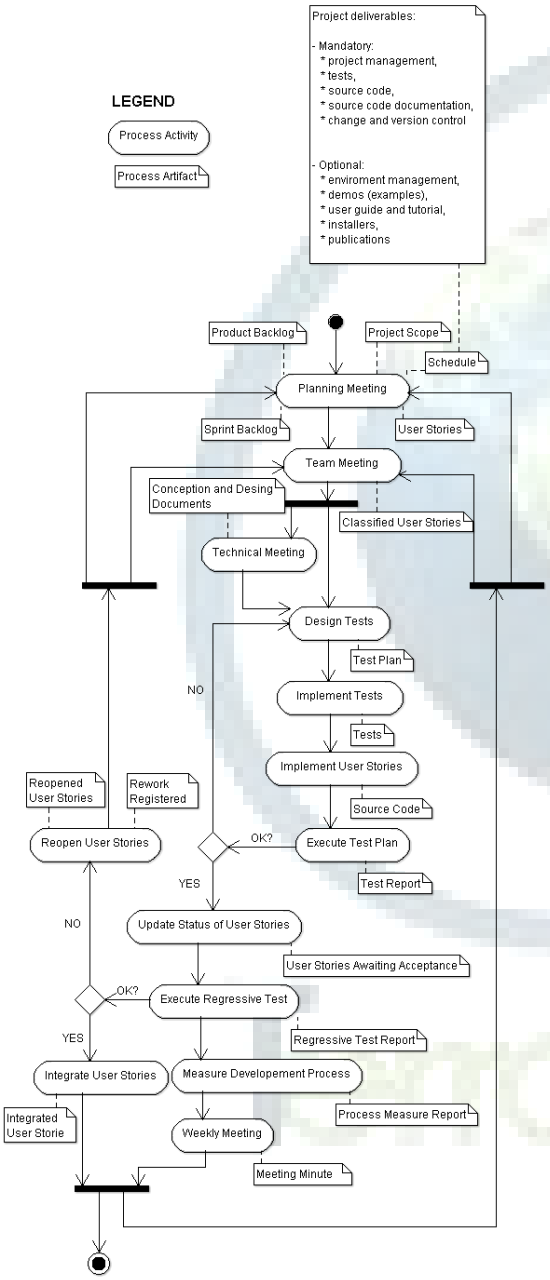


Figure 1. BOPE development process

- **Project Schedule**: The project schedule is planned generally for the entire project period. The schedule of each sprint is detailed throughout the project. Schedules can be modeled by a tree structure. The root node represents the entire project. The second level of node represents the projects deliverables. Next node levels represent sub-deliverables. Leaf nodes represent user stories (or requirements), which must be implemented in order to accomplish project deliverables. Considering available resources, it is possible to estimate the time needed to implement user stories. Each user story is implemented according to the BOPE activity diagram (Figure 1).

### *C. Activity diagram*

Projects are initiated by orders made by customers or by internally generated research demands. Long-lasting software products, such as TerraME, may give rise to two new projects every year. One for software maintenance and another for new feature developments. Both kinds of project are executed by following the process BOPE, illustrated in Figure 1.

In BOPE, the customer or the laboratory leader are always part of the development team. They have the same responsibility that the **product owner** has in Scrum. A development team has a **leader**, who is an experienced developer and who often is assigned no user stories. His task is to help developers to accomplish their activities and to verify the acceptance of user stories carried out by the team. A **project manager** tracks all projects, evaluating measures of the development process, of teams and of products. He also solves risks to all projects (schedule, cost and quality).

Before the project starts, a **planning meeting** occurs to gather enough information to identify project scope, project deliverables (**product backlog**) and main epics of each deliverable. It also results in an initial estimation of project schedule, required resources, and costs.

Project development evolves in **sprints** (large cycles) in which new versions of project deliverables are produced, adding value to project stakeholders. Each sprint is formed by several **iterations** (small cycles) in which the team executes test-driven development activities, measuring activities and feedback activities. Iterations are synchronous among projects, e.g., they start and finish at the same time.

In the beginning of each sprint, a planning meeting must occur to define the user stories of the next sprint (**sprint backlog**), to detail these user stories, and to estimate the sprint schedule and required resources. After, a **team meeting** must occur to select, prioritize and evaluate (maybe using the Scrum planning poker) user stories that will be developed in the next iteration. The project manager and the whole development team must attend these meetings. However, it is not necessary that the product owner attends all of the team meetings. The team leaders have autonomy to deal with them. The team leader should see the project manager as a customer, and the project manager should see the team leader as a service provider (Figure 2). The project manager always wants the tightest schedule and least consumption of resources. The team leader often wants freedom and peace to do their work. They must negotiate.

After the team meeting, the test driven development cyclical activities are ready to start. First, developers must design a test plan. Following that, they must implement the planned tests and implement the user stories to be approved in all tests. They may require a technical meeting with the laboratory leader in which to solve conception and design problems in tests or in user stories. The laboratory leader is often a researcher or a professor with a lot experience. This fact justifies his participation in conception and design activities. Finally, developers must execute the tests to verify that user stories were implemented in the correct way. If they were not, the test driven development activities must be restarted. The status of user stories approved in the test plan must be changed to "awaiting acceptance", indicating to the team leader to verify its correctness.

The team leader continuously verifies user stories awaiting acceptance, runs regressive tests and integrates the approved user stories with the product source code. If a user story is not approved by the team leader, he reopens it and registers the next hours consumed in its development as rework. At the end of each iteration, the team leader collects measures about the team, the process and the product. He then provides these measures to the project manager.

At the end of the iterations, laboratory leader and all teams sit together in the **weekly meeting.** The project manager presents the measures of all projects. Teams evaluate the development process, highlight lessons learned from the experience, and plan for small changes for the next iteration. This meeting should last no longer than one hour. Therefore, team leaders must be prepared to talk on behalf of their teams.

When all user stories are completed by developers, approved by team leaders, and validated by product owners, the project is ready to be closed. Otherwise, a new sprint will begin with a planning meeting.

### *D. Process roles*

Figure 2 presents the roles a team member can perform in BOPE process. The laboratory leader often performs the role of *Business Analyst (or product owner)*. The *Team Leader* is an experienced developer, who's main task is to help developers to accomplish their activities and to verify the acceptance of user stories carried out by the team. Developers will be assigned to three kinds of roles, depending on their abilities: *System Analyst*, *Software Engineer*, and *Test Engineer*. The project manager tracks all projects, evaluating performance of the development process, of teams and of products. He also solves risks to all projects (schedule, cost and quality). Table 1 presents the activities carried out for each role.

BOPE intentionally violates the flat structure of agile teams. It also has a set of standardized artifacts that makes it looks like as a non-agile process. In this sense, we would like to argue that although agile processes value more working software instead of comprehensive documentation, they still value documentation, tools and processes. BOPE uses very simple artifacts to keep the process agile (Table II). Empirically we have been convinced that, in order to efficiently produce good quality software, hierarchical teams supported by minimal standardized artifacts and processes can overcome the inexperience of undergraduate students, mainly those students in the initial semesters. Having an experienced leader, these teams can also overcome the part-time dedication of supervisors in each project, allowing the lab to scale up its production.

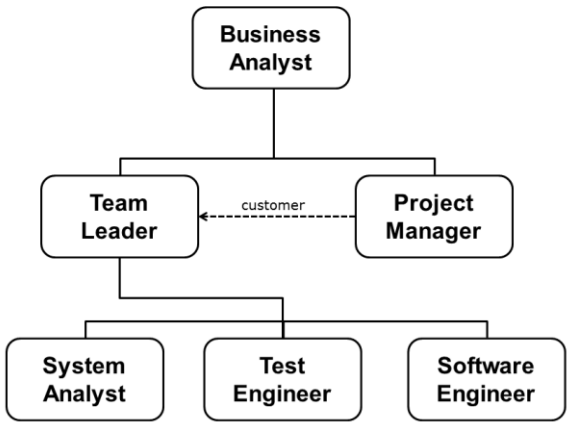


Figure 2. BOPE role hierarchy

TABLE I. ROLES X ACTIVITIES

| Roles | Activities |
|---|---|
| Business Analyst | Takes care of market research and sales services. Responsible for the relationship with the customer, he/she must manage customer expectations and interests. Participates in the planning meeting, which elaborates the project scope, defines project deliverables (product backlog). Participates in the technical meeting with the team and helps to clarify details about software conception and design. Evaluates the results of measurements of the process, of teams, and products through the weekly meeting. |
| Project Manager | Allocates resources, keeps the project team focused, helps teams to valuate tasks in the "planning poker". Manages risks in development activities. Responsible for collecting measures from team leaders and evaluating these measures, comparing projects and reporting the conclusion to the *Business Analyst* and teams. Prepares weekly status reports. Participates in all meetings. |
| | Technical leader of the development team. Responsible for the technical conception and design of the project tests and deliverables. Plans tests with *Test Engineers*. Sets system architecture and structure (component, deployment, class, and sequence diagrams) with *Software Engineers*. Guides developers during implementation. Executes regressive tests. Controls software version and changes. Collects measures about the development process, about products |

| | |
|---|---|
| Team Leader | and the team. Provide measures to the project manager. |
| Systems Analyst | Collects, analyzes, and tracks software requirements (user stories). Highlights features and technical boundaries of the system. Generates final product tutorial, examples of use, and delivery systems. |
| Test Engineer | Plans, implements and executes functional and integration test cases. |
| Software Engineer | Implements user stories and unit tests. Executes unit tests. |

### E. Process Artifacts

Table 2 presents the artifacts produced in each activity of the BOPE development process.

TABLE II. ACTIVITIES X ARTIFACT

| Activity | Artifact |
|---|---|
| Planning Meeting | Project scope, product backlog, sprint backlog, project schedule, main user stories. |
| Team Meeting | Classified user stories. |
| Technical Meeting | UML diagrams (class, sequence, state, component, or deployment diagrams) |
| Design Tests | Test Plan |
| Implement Tests | Tests source code |
| Implement User Stories | User stories source code |
| Execute Test Plan | Test report |
| Update Status of User Stories | Status of user stories modified to "awaiting acceptance" |
| Execute Regressive Test | Regressive test report |
| Measure Development Process | Measurement report |
| Weekly Meeting | Minutes of meeting |
| IntegrateUser Stories | Integrated source code of approved user stories |
| Reopen User Stories | Status of user stories modified to "reopened" and rework registered |

## IV. STUDY DESIGN AND EXECUTION

### A. Selection of case study projects

During the case study, twelve software projects running in the laboratory followed the same process. However, this paper reports the observations from only two software products: TerraME and SIGHabitar. We found these projects to be more complex, longer and involved larger teams. Data collected for other projects reproduced similar patterns and dynamics.

TerraME is a ten year old modeling and simulation toolkit [22], which serves as foundation to national efforts in defining public policies to deal with land use and cover change (LUCC) in the Amazon region [23][24][25], with emission of greenhouse gasses [26], with natural disasters and early warning [27][28], and with dengue fever control [29]. Some of these efforts led to partnerships to the development of TerraME extensions or applications, like LuccME[1], INPE-EM[2], TerraMA2[3], and DengueME (Dengue Modeling Environment - soon available). The TerraME local development team is composed of 3 masters students and 6 undergraduate students.

SIGHabitar is a business intelligence set of tools for developing land information system s[23]. The SIGHabitar whole team is composed of 2 masters students and 6 undergraduate students.

### B. Selection of process measures

During the case study, we selected some measures to evaluate the development process and allow its improvement. To overcome some challenges, some new measures (*in italics*) were created and evolved during the case study. They allow us to know team commitment, velocity and productivity. Some allow us to monitor project completeness, while others allow us to measure the amount of rework and effective adoption of test driven activities by the team. The measures used in this work are:

- ***Dedication*** = Number of worked hours in a fixed period of time
- ***Relative dedication*** = Dedication / Number of worked hours planned for the same period of time
- **Velocity** = Number (or points) of tasks performed and approved in a fixed period of time
- **Productivity** = Velocity / Sum of dedication of all team developers in the same period of time
- ***Relative velocity*** = Velocity / "Number" of tasks planned for the same period of time
- ***Relative productivity***= Relative velocity/ Average relative dedication of team developers
- **Project completeness** = Percentage of planned tasks performed and approved
- **Project increment** = Percentage of planned tasks performed and approved in a fixed period of time
- **Number of bugs** = Number of software faults detected by development team or by customers
- **Test factor**[16]= Number of lines of test code / Number of lines of production code

### C. Process improvement implementation

Since the beginning of the case study, we perceived that undergraduate students did not meet the hours of work per week under their contracts. Graduate students tended to work the contracted hours or more. This way, we deployed a project management tool that logs the time spent to execute each activity. Figure 3 and Figure 4 present the average relative dedication of team members along 2012 and 2013. Events A, B, C, D, and E are coincident with period of school tests. The low dedication of undergraduate students pushes the average dedication down.

[1]http://www.terrame.org/doku.php?id=luccme
[2]http://inpe-em.ccst.inpe.br
[3]http://www.dpi.inpe.br/terrama2/

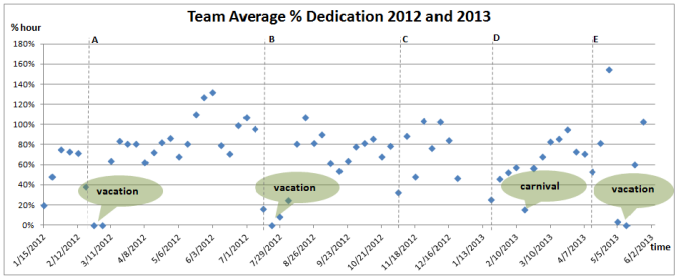


Figure 3. Average TerraME team % dedication during case study

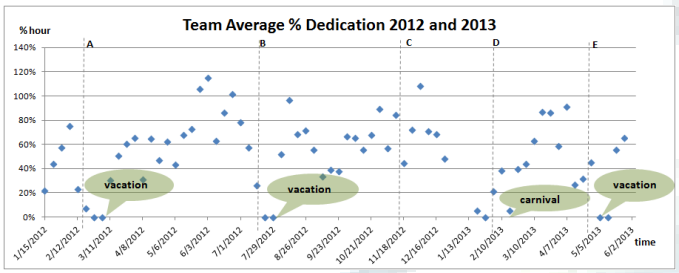


Figure 4. Average SIGHabitar team % dedication during case study

Figure 5 shows separately the relative dedication of undergraduate and graduate students in 2013. After the holidays in the beginning of the year, graduate students quickly resumed the pace of work. The same is not true for undergraduate students. During the period of little dedication, many conversations during weekly meetings sought to motivate them. However, only communicate measures of team dedication, velocity and productivity were effective for this purpose. This occurred for the first time in 3/3/2013. From 4/7/2013 onwards, undergraduates reduced their work dedication again due to the period of final school tests. In average, graduate students dedicated 80% or more of their contracted work time

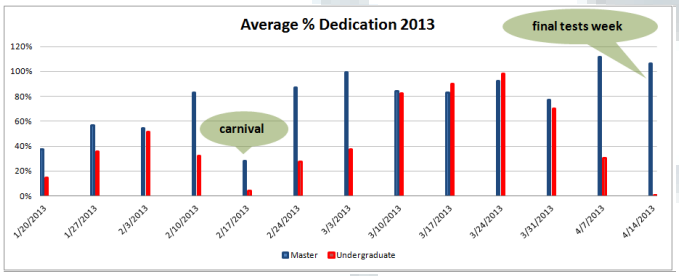


Figure 5. AverageTerraME team % dedication in 2013

Figure 6 presents the number of software bugs detected by the customer and by the TerraME team, throughout this case study. This is the main measure used to evaluate product quality and to evaluate the implementation of software process improvement. We started using a development process based on RUP practices, dominated by rigid plans and no use of software tests. And we ended the case study with a development process based on agile methods, founded on test-driven activities and on short iterations with rapid feedback. We tried different durations for sprints and iterations. The best settings were for three months and one week, respectively.

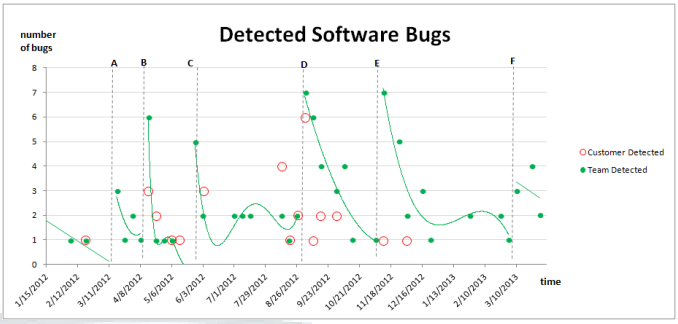


Figure 6. Number of bugs detected by team and by customer in several TerraMEsoftware releases

In Figure 6, the event A is the project milestone in which the team began to close the version 1.1 of TerraME application launched at the event B. Thereafter, users started to download the software and to register perceived bugs. In the next sprint, the team started to close the version 1.1.2 of TerraME 3 iterations before it was launched in 6/25/12 - event C. Until this sprint, tests were executed manually and users perceived several bugs in the software release. Thus, the team learned one more lesson and began to automate tests. In event D, 2 iterations before version 1.2.0 of TerraME was launched in 9/10/12, the team provided the alpha release to the customers and started to close the final version while running automated tests. Many new features in version 1.2.0 are related to scientific visualizations and graphical user interfaces (GUI), making test automation a hard task. In event E, version 1.2.1 alpha of TerraME was provided to the customer with the automated tests and the team put all of its effort into developing its own test framework for the Lua programming language, with special features to verify visualizations and GUIs. TerraME 1.2.2 was provided to the customer in the event F. TerraME 1.3 is still under development.

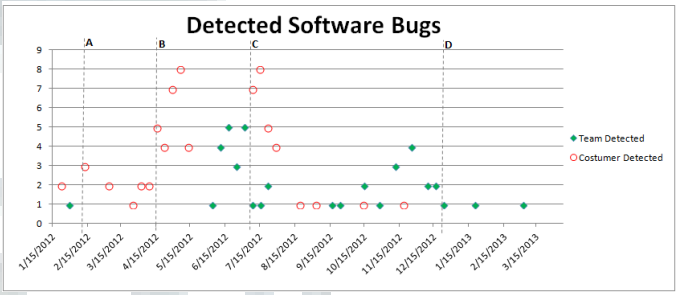


Figure 7. Number of bugs detected by team and by customer in several SIGHabitar software releases

Figure 7 presents the number of software bugs detected by the customer and by the SIGHabitar team, along the case study. Events A, B, C and D highlight dates in which SIGHabitar releases were launched. Most tests were executed manually because they involve verification of mobile tools in field work and verification of spatial data transformation tools. The automation of these tests is many times more sophisticated than the development of the software components being tested.

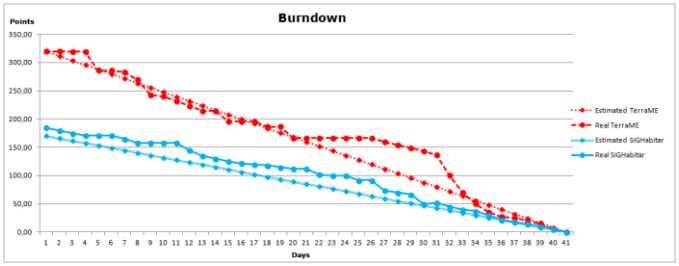


Figure 8. Burndown chart for the last eightiterations

Figure 8 presents the burndown chart for the last 41 days of development. In the first iteration of TerraME project, developers wait until the last moment to change the status of their tasks to "awaiting acceptance". Thus, the team leader had to verify all tasks in the last day of the iteration. This fail in following the BOPE process, was corrected in the weekly meeting. In the fifth iteration (starting at day 20), no task was accomplished. Undergraduate students stopped working due to school final tests. Master students were busy in tasks that take more than one iteration to be accomplished, for instance, writing this article or writing dissertation chapters.

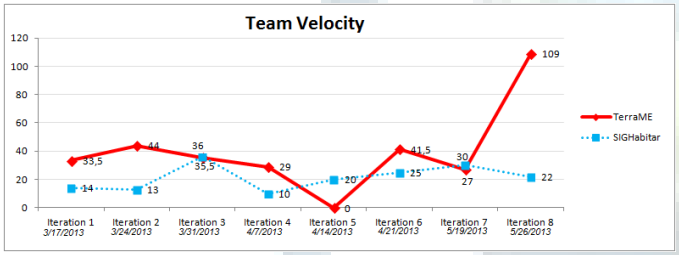


Figure 9. Team velocity in 2013

Figure 9 shows the velocity of teams during 2013. No tasks were been accomplished in the iteration 5 in TerraME. In iteration 8, the TerraME team leader has been asked to spent the first days of the iteration planning tests and delivering them for implementation, as defined in the process. Prioritizing test planning speeds up team performance. The project SIGHabitar in this reporting period was already in its last sprint, and tasks (points) made by the team were conducted without many variations.

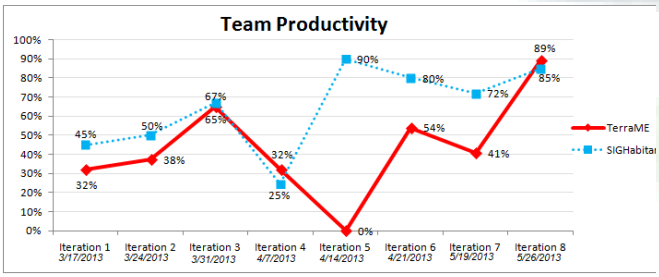


Figure 10. Team productivity in 2013

Figure 10 illustrates the productivity of teams along 2013. TerraME team productivity in iteration 5 is also null. Although the velocity was higher in iteration 2, the productivity was not because dedication was also high. Team productivity was high in iteration 3 because velocity was satisfactory and less effort was made. In iteration 8, productivity is higher because the team leader changes their behavior keeping the developer busy and motivated. Productivity of the SIGHabitar team at iteration 5 was the highest in this period, despite not having consumed many points, they devoted less than expected and their tasks added a lot of value to the customer.

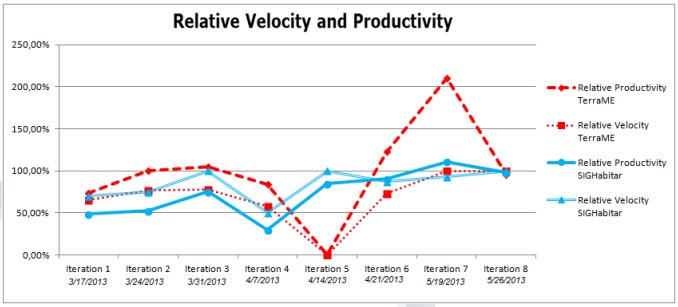


Figure 11. Team relative velocity and productivity in 2013

The former measures oscillate a lot and can be manipulated. Teams may assign too many points to their tasks to improve their velocity. Teams may work less time to improve their productivity, doing tasks as fast as they can with no commitment to quality. Then, we create two new measures that, at least, tell the project manager if teams are following what was planned: relative velocity and relative productivity. They also oscillate less. As in iteration 5 no task was completed by the team of TerraME, the relative velocity and relative productivity were 0. Throughout iterations, as real velocity approaches planned velocity, the team relative velocity will approach 100%. If team relative velocity is 100% and average relative dedication of team members is 100%, then the team relative productivity will be 100%. Figure 11 presents these measures along 2013. In iteration 7, the TerraME team project planned to work less than the actual expectation, increasing relative productivity.

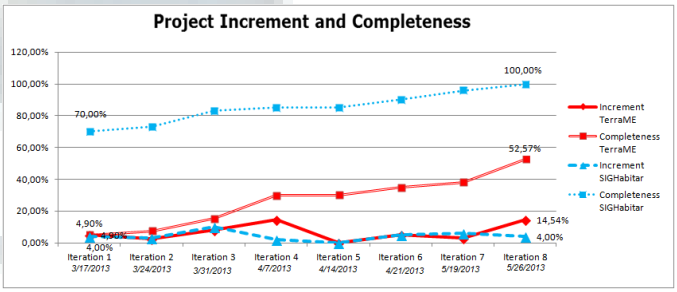


Figure 12. Project deliverables completeness and increment

Figure 12 shows the project completeness and increment along 2013. Only 4.9% of TerraME deliverables were completed in the first iteration. The highest increment was in the forth iteration. Project completeness reached 52.57% in iteration 8 and increased only 14.54% in the last iteration. As in iteration 5 team velocity is null, project completeness has not incremented from the forth to the fifth iteration. The SIGHabitar project was completed in iteration 8.

The velocity shows to the project manager the amount of work the team has performed. However, much of the work done can be operational tasks, for instance, installing development environment, making backups of project resources, etc. This kind of task does not always result in benefits to customers, e.g., an increment in project completeness. Although in TerraME project a high velocity was obtained in iterations 2 and 6 (Figure 9), the higher

increments in project completeness occurred in iterations 4 and 8. Although team relative velocity was low in iteration 4, most of the tasks completed in this iteration accomplished pending user stories. It is easier to complete user stories after some time from the beginning of a project.

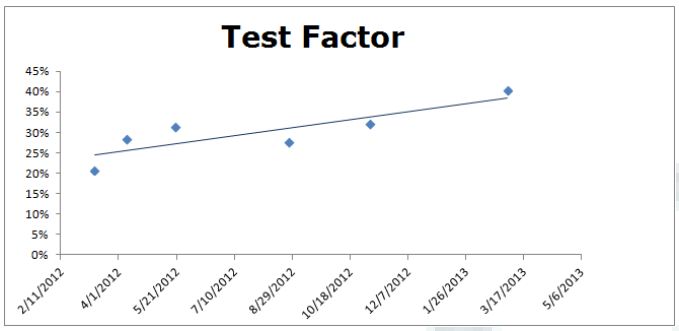

Figure 13. Test adoption along TerraMEcase study

Figure 13 presents the test factor measure [12] to determine how well the TerraME team is adopting test-driven practices.

## V. LIMITATIONS

Although this case study has considered twelve software projects during three years, it has been conducted in just one laboratory of a unique university. This fact can compromise our conclusions and limit the extent of the obtained results to other environments. However, we believe that other academic laboratories can capitalize on the development processes we have evolved during this study.

Since there is no *a priori* knowledge on the problems that will need to be solved in maintenance projects, it is hard to estimate their schedules and costs. Measures that consider plans are badly affected by this fact and lose their meaning, namely relative velocity and relative productivity. However, it is also easy to manage and execute project maintenance following the BOPE process, since short interactions with fast feedbacks make it easy to respond to changes.

## VI. CONCLUSION

In this paper we presented a case study in which an agile process, named BOPE, was evolved into engineering software in laboratories of Brazilian public universities, where products are most often related to scientific research and to technological innovation. Team members are still in training, have academic activities as their priorities and are partly dedicated to projects. Special attention was given to laboratories in which teams are mostly composed of undergraduate students.

BOPE intentionally violates the flat structure of agile teams. It also has a set of very simple standardized artifacts (table II) and measures (section IV- B) that allows quantitative and qualitative tracking of projects. Empirically we have been convinced that, in order to efficiently produce good quality software, hierarchical teams supported by minimal standardized artifacts and processes can overcome the inexperience of undergraduate students. Experienced team leaders can help their team to overcome the part-time dedication of supervisors (professor/researcher) in each project, allowing the process to scale up its production.

Throughout the case study, we collected diverse measures to evaluate and improve the development process. Data analysis has shown that undergraduate and graduate students behave in different ways in relation to their involvement in projects (Figure 5).

As show in Figures 6 and 7, at each release of software launched, the number of bugs detected by customers has decreased and the development team starts to detected most of them before they are perceived by customers. The adoption of the BOPE process by teams has also increased since the test factor measure has significantly augmented along projects (figure 13).

The availability of a project schedule and team productivity information encourages the students to work productively and efficiently (Figure 11). Velocity and dedication start being calculated and discussed with teams at iteration 1. Since then, their performance has gradually improved. Relative performance and relative productivity have also improved since then, showing that teams gradually estimate better their work in each iteration and gradually increased their commitment in doing what has been planned.

Data analysis also suggest that using a well-defined test process (Figure 1 and 13) is conducive to the improvement of product quality (Figure 9 and 10). Conceiving tests before the solutions forces developers to design good Application Programming Interfaces, because they should think in how to program the client code first. This fact promotes code reuse since developers have freedom to think in how they would like to use the API in different situations, before they begin coding. Once the test's conception and design are approved by team leaders, they only need to implement tests and to program solutions to pass in the tests. This is easier than writing and erasing debug code. It will be always possible to test changes in the solution code against regressive tests, assuring product quality. However, there are limitations in this approach, as tests cannot show the absence of bugs in the solution, only the presence.

The main contribution of this work is to provide evidence that, through BOPE, teams formed mostly of undergraduate students can develop and maintain long-lasting and innovative software. This process can be used by other academic labs with similar characteristics.

## AKNOWLEDGEMENT

We would like to thank the TerraLAB team.[4]

## REFERENCES

[1] Cohn, M. Succeding with Agile: Software Development Using Scrum. Boston: Addison-Wesley, 2009.

[2] Rising, L.; Janoff, N.S. The Scrum Software Development Process for Small Teams. IEEE Software, v. 17, n.4, 2000, pp. 26-32.

[3] Beck, K. Embracing Change with Extreme Programming. IEEE Computer, v.32, n. 10, 1999, pp. 70-78.

♣ This work was partially funded by CAPES, CNPq (CTINFO 09/2010) and UFOP.

[4] PMI. A Guide to the Project Management Body of Knowledge (PMBOK Guide). Newtown Square, Pa: Project Management Institute, 2004.

[5] Basili, V. R.; Caldiera, G.; Rombach, H. D. The goal question metric approach. *Chapter in Encyclopedia of Software Engineering*. Wiley. 1994, *2*, pp. 1–10.

[6] Krutchen, P. The Rational Unified Process - A Introduction. Reading, MA: Addison-Wesley, 2003.

[7] O. Hazzan and Y. Dubinsky, Teaching a software development methodology: the case of extreme programming. In the Proceedings of 16th Software Engineering Education and Training (CSEE&T). ISSN: 1093-0175, pp. 176-184, March 2003.

[8] G. Melnik and F. Maurer, Introducing Agile Methods in Learning Environments: Lessons Learned. Lecture Notes in Computer Science, vol. 2753, pp. 172-184, 2003.

[9] M. M. Müller, J. Link, R. Sand, and G. Malpohl, Extreme Programming in Curriculum: Experiences from Academia and Industry. Lecture Notes in Computer Science, vol. 3092, pp. 294-302, 2004.

[10] V. Santos, A. Goldman, C. D. Santos. Uncovering Steady Advances for an Extreme Programming Course. CLEI Electronic Journal. Volume 15, Number 1, Paper 1. 2012.

[11] Beck, K.; Beedle, M.; van Bennekum, A.; Cockburn, A. ; Cunningham, W.; Fowler, M.; Grenning, J.; Highsmith, J. ; Hunt, A.; Jeffries, R.; Kern, J.; Marick,B.; Martin, R. C.; Mallor,S.; Shwaber, K.; and Sutherland, J. The agile manifesto. Technical report, The Agile Alliance, 2001.

[12] Pimentel, B.; Pádua, W.; Pádua; C. I.; Machado, F. Synergia – A Software Engineering Laboratory to Bridge the Gap between University and Industry. 28th International Conference on Software Engineering, Shanghai , China, 2006.

[13] Paula Filho, W. P. Praxis 2.1. Available from http://www.wppf.uaivip.com.br/praxis/2.1/Praxis%202.1.htm , 2005 (in Portuguese).

[14] Peixoto, D.; Batista, V.; Resende, R.; Pádua, C. I., A Case Study of Software Process Improvement Implementation; International Conference on Software Engineering and Knowledge Engineering (SEKE 2010). California, USA, 2010.

[15] Goldman, A.; Kon, F.; Silva, P. J. S.; Yoder, J. W. Being Extreme in the classroom: Experiences teaching XP. *Journal of the Brazilian Computer Society*. 2004. *10*(2), 4–20. doi:10.1007/BF03192356

[16] Sato, D., Bassi, D.; Bravo, M.; Goldman, A.; Kon, F.. Experiences tracking agile projects: an empirical study. *Journal of the Brazilian Computer Society*, 2006. *12*(3), pp. 45–64. doi:10.1007/BF03194495

[17] Lisboa, L. B.; Nascimento, L. M.; Almeida, E. S.; Meira, S. R. D. L.. A Case Study in Software Product: Lines An Educational Experience. *2008 21st Conference on Software Engineering Education and Training*. 2008, pp. 155–162. doi:10.1109/CSEET.2008.17

[18] Wainer, M. Adaptations for Teaching Software Development with Extreme Programming: An Experience Report. Lecture Notes in Computer Science,2003, v. 2753, pp. 199-207.

[19] Bunse, C.; Feldmann, R. L; Dörr,J.. Agile Methods in Software Engineering Education. Lecture Notes in Computer Science, 2004, v. 3092, pp. 284-293.

[20] Hazzan, O.; Dubinsky, Y.. Teaching a software development methodology: the case of extreme programming. In the Proceedings of 16th Software Engineering Education and Training (CSEE&T), 2003, pp. 176-184.

[21] Santos, V.; Goldman, A.; Santos, C. D.. Uncovering Steady Advances for an Extreme Programming Course. CLEI Electronic Journal. 2012, v. 15, n.1.

[22] Melnik, G.; and Maurer, F.. Introducing Agile Methods in Learning Environments: Lessons Learned. Lecture Notes in Computer Science, 2003, v. 2753, pp. 172-184.

[23] Silva, J. T. C. ; Rezende, J. F. V. ; Fidêncio, E. ; Melo, T. ; Neves, B. ; LIMA, J. ; Carneiro, Tiago . SIGHabitar Business Intelligence based approach for the development of Land Information Systems: The Multipurpose Technical Cadastre of Ouro Preto, Brazil. In: ICCSA - International Conference on Computational Science and Its Applications, 2012, Salvador,BA. Anais, 2012

[24] Carneiro, T. G. S. ; Andrade, P. R. ; Câmara, Gilberto ; Monteiro, Antônio Miguel Vieira ; Pereira, R. R. . An extensible toolbox for modeling nature society interactions. Environmental Modelling & Software, 2013, v. 46, pp. 104-117.

[25] Moreira, E.; Costa, S.; Aguiar, A. P.; Câmara,G.; Carneiro, T. G. S. Dynamical coupling of multiscale land change models, Landscape Ecology, v.24, n. 9, 2009, pp. 1183-1194, 2009.

[26] Patterson, D.; Armando Fox, A..Engineering Long-Lasting Software: An Agile Approach Using SaaS and Cloud Computing. 2nd Beta Edition. CreateSpace. 2013.

[27] Aguiar, A.P.; Câmara, G.; Escada, M.I.S. Spatial statistical analysis of land-use determinants in the Brazilian Amazon: exploring intra-regional heterogeneity. Ecological Modelling, vol 209, n. 1-2, 2007, pp. 169–188.

[28] G. Câmara; Aguiar, A. P.; Escada, M. I.; Amaral, S.; Carneiro, T. G. S.; Monteiro, A. M. V.; Araújo, R.; Vieira, I. C.; Becker, B. Amazon Deforestation Models. Science, v. 307, 2005, pp. 1043-1044.

[29] Aguiar, A.P.; Ometto, J. P.; Nobre, C. A.; Lapola, D. M.; Almeida, C.; Vieira I. C.; Soares, J. V.; Alvala, R.; Saatchi, S., Valeriano, V. ; Castilla-Rubio, J. C. Modeling the spatial and temporal heterogeneity of deforestation-driven carbon emissions: the INPE-EM framework applied to the Brazilian Amazon. Global Change Biology, v.18, n.12, 2012, pp. 3346-3366.

[30] Lopes, E. S. S.; Namikawa, L. M.; Reis, J. B. C. Risco de escorregamento: monitoramento e alerta de áreas urbanas nos municípios no entorno de Angra dos Reis - Rio de Janeiro. In: 13° Congresso Brasileiro de Geologia de Engenharia e Ambiental, 2011, São Paulo. Anais. 2011.

[31] Reis, J. B. C.; Cordeiro, T. L.; Lopes, E. S. S. Utilização do Sistema de Monitoramento e Alerta de Desastres Naturais aplicado a situações de escorregamento - caso de Angra dos Reis. In: 14° Simpósio Brasileiro De Geografia Física Aplicada, 2011, Dourados, MS. Anais. 2011

[32] Lana, R. M.; Carneiro, T. G. S.; Honório, N. A.; Codeço, C. T. Multiscale analysis and modeling of Aedes aegypti population spatial dynamics. Journal of Information and Data Management, v. 2, n.2, 2011, pp.211-22