# Agile Development Methodology for Embedded Systems: A Platform-Based Design Approach

Lucas Cordeiro[1,3], Raimundo Barreto[1], Rafael Barcelos[3], Meuse Oliveira[2], Vicente Lucena[1], and Paulo Maciel[2]

[1]Departamento de Ciência da Computação - Universidade Federal do Amazonas (UFAM), Brazil
rbarreto@dcc.ufam.edu.br and vicente@ufam.edu.br

[2]Centro de Informática - Universidade Federal do Pernambuco (UFPE), Brazil
{mnoj, prmm}@cin.ufpe.br

[3]BenQ Mobile Phones - Research and Development Center, Brazil
{lucas.cordeiro, rafael.barcelos}@benq.com

## Abstract

*This paper describes an agile development methodology which combines agile principles with organizational patterns and adapts them to build embedded real-time systems focusing on the system's constraints. The hardware/software partitioning and platform-based design are used in the proposed methodology to support the embedded system designer meet the system's constraints in an iterative and incremental way and to reduce substantially the design time and cost of the product. To discuss the strengths and weakness of this methodology, a case study involving a pulse oximeter is also presented.*

## 1 Introduction

The micro-controllers becoming cheaper, smaller and more reliable make it economically attractive to be used as computer systems in several appliances. Approximately 3 billion of micro-controllers ($\mu$C) are sold each year and smaller $\mu$C (4-,8-, and 16-bit) are dominating the market and adding value to products [6]. The embedded computer systems are used in a wide range of system from machine condition monitoring to airbag control systems. As the system complexity increases, its development lifecycle is also affected. Because of that, system development methodologies must be applied in order to manage the team size, to manage the product requirement *(scope)* and to meet the project's constraints *(time-to-market and costs)*.

Nevertheless, many development methodologies that are used to produce software that runs on the personal computers (PC's) are not appropriate for developing embedded real-time systems. This kind of system contains very different characteristics such as dedicated hardware and software, and constraints that are not commum to PC's based systems (e.g. energy consumption, execution time, memory footprint). Moreover, many embedded systems engineers do not have good software engineering skills. They have hardware development skills and often use programming languages to solve the problems at hand in an empirical way [5]. Another important point is that some classes of embedded real-time systems may put lives or business-critical functions at risk *(mission criticality)*. Therefore, these systems should be treated differently from the case where the only cost of failure is the project's investment.

Based on this context, we propose a development methodology based on the agile principles such as *adaptive planning, flexibility, iterative and incremental approach* in order to make easier the development of embedded real-time systems. To achieve that, this methodology is composed by best practices from Software Engineering and Agile methods (Scrum and XP) which aim at minimizing the main problems present on the embedded software development context *(i.e. requirement volatility and risk management)*, and by others practices that are needed to achieve embedded real-time systems *(i.e. platform-based design* [13]). On this paper, this methodology and its components (roles, process and tools) are described.

The remainder of this paper is organized as follows: Section 2 summarizes the related works. Section 3 overviews the agile methods and patterns that were integrated into the proposed methodology. Section 4 describes the proposed agile methodology and section 5 shows the application of

the proposed methodology in the development of the pulse oximeter. Finally, section 6 summarizes this paper and identifies the next steps from this research.

## 2 Related Works

Embedded software development teams usually do not make use of development methodologies or any other more complex software engineering concept [5]. There are different reasons that explain this fact, but the main one is the developers' lack of maturity related to software engineering practices. Nevertheless, we identified in the literature through a bibliographical review, three different development methodologies that allowed us to evaluate the state of the art in this context and to support us during the definition of our proposed approach.

One of the results from this review is a paper that describes the experience of applying Agile approaches to the development of firmware for the Intel Itanium processor family [5]. In this paper, Greene identified the agile practices that his team successfully applied, but he did not take into account the hardware related development, one of the main parts of this kind of development. Greene only mentioned that another Intel team was applying agile concepts and they were having good results. Even so, the comments retrieved from this paper regarding the application of agile concepts were very useful during the definition of our approach since it supports the benefits of using agile concepts in contexts beyond object oriented software development.

The second one is the methodology proposed by Gajski [4] which aims to develop embedded systems by formally describing the system's functionalities in an executable language rather than a natural language. The executable specification is refined through the system-design tasks of allocation, partition, and refinement. Estimators are also used in order to explore design alternatives. Since the system components are defined formally then components are implemented by just compiling the component's functional description into machine code. This methodology has already been applied to several embedded systems projects and has influenced our proposed methodology. However, this methodology assumes that all requirements are captured before applying the partitioning algorithms.

Finally, Manhart and Schneider [8] related a successful industrial experience when partially adopting agile methods in the production of software for embedded systems. Indeed they made slight modifications in a well established software development process for the automotive branch adopting some agile elements in order to adequate their process to new needs as flexibility and high speed software production. As pointed out in the paper many other application areas may benefit from their experiments, nevertheless the authors did not presented any measurement results that could prove their expectations.

The difference of the proposed methodology compared with other methodologies can be described as follows: *(i)* our methodology aims to tradeoff flexibility and performance by adopting highly programmable platforms, *(ii)* hardware/software estimation and partitioning techniques are used in order to explore design alternatives and meet the system's constraints, *(iii)* by making use of the iterative and incremental approach, the product development can be broken in a sequence of iterations and implemented in an incremental way, *(iv)* as the system functionalities increases iteration by iteration then the proposed methodology offers clearly an iterative process where the designer can validate the partition of a system specification produced by algorithms, *(v)* and last but not least, the proposed methodology adopts an adaptive planning which makes it possible to embraces changes even late in the development process.

## 3 A Brief Look at the Agile Methods and Patterns

In this section, a brief look at the agile principles, methods, and patterns that were used in the proposed methodology is presented. It identifies the main product development and management practices of the XP and Scrum methods respectively.

### 3.1 Extreme Programming

The most recognizable agile method is eXtreme Programming (XP) which is very communication-oriented and team-oriented [1]. XP is composed of 12 core practices and some of its main characteristics that were integrated into the proposed methodology include: Refactoring practice *(i)* which is the process of changing a software system in such a way that it does not alter the external behavior of the code and at the same time improves its internal structure.

In the Continuous Integration practice *(ii)*, the code is compiled and tested in an automated process every time it is checked-in. Test-driven development practice *(iii)* means that the unit tests are written by the developers before coding. These unit tests are automated tests that test the functionality of pieces of the code. In the Coding Standard practice *(iv)*, everyone involved in the project needs to follow the same code style. It specifies a consistent format for source code, within the chosen programming language.

XP promotes an evolutionary approach to design the system by using the first three practices described above. The main benefit of this approach is that the system grows in an incremental way and it aims to reduce project's risk and uncertainty too early *(risk management)*. Section 4 describes how these XP practices were adapted into the proposed methodology.

### 3.2 Scrum

Scrum is a simple and straightforward approach to manage the software development process based on the assumption that environmental (i.e. people) and technical (i.e. technologies) variables are likely to change during the process [12]. Scrum is composed of 14 practices and some of its main characteristics that were integrated into the proposed methodology include: Sprint practice *(i)* is the iteration organized in 30-calendar-day. The Sprint Planning practice *(ii)* consists of two meetings.

In the first meeting, the product backlog which contains a list of features, use cases, enhancements, and defects of the system is refined and re-prioritized by the product owner, stakeholders and goals for the next iteration are chosen. In the second meeting, the Scrum team figures out how to achieve the requests and creates the sprint backlog that contains detailed tasks to be accomplished in the current iteration. In the Sprint Review practice *(iii)*, the Scrum team presents the results obtained at the end of each iteration by showing the working software for the product owner, customers and other stakeholders. In the Daily Scrum practice *(iv)*, daily meetings are held at the same place and time with special questions to be answered by the Scrum team.

Scrum employs the empirical process control model, i.e. the practices aim to inspects the condition of activities and empirically determines what to do next in order to produce the expected outcomes *(product)*. The productivity and quality strongly depend on both skills and motivation of the people involved in the process. Section 4 shows how the Scrum practices were adapted into the proposed methodology.

### 3.3 Patterns for Agile Software Development

The agile patterns described by [3] can be combined with XP and Scrum agile methods with the purpose of structuring the software development process of the organizations. These patterns are split into four different pattern languages as follows: The *project management pattern language* provides a set of patterns that help the organization manage the product development, clarify the product requirements, coordinate project's activities, generate system's build, and keep the team focus on the project's primary goals.

The *piecemeal growth pattern language* provides a set of patterns that help the organization define the high-level management and amount of team members per project, ensure and maintain customer satisfaction, communicate the system requirements, and ensure a common vision for all people involved in the product development team. The *organizational style pattern language* provides a set of patterns that help the organization eliminate project's overhead and latency, ensure that the organization structure is compatible with the product architecture, organize work to develop products by geographically distributed teams, and ensure that the market needs will be met.

The *people and code pattern language* provides a set of patterns that help the organization define and keep the architecture style of the product, ensure that the architect is materially involved in implementation, and assign feature development to people in nontrivial projects. The *software configuration management pattern language* is not part of the organizational patterns but they were integrated into the proposed methodology. These patterns were defined by [2] and they offer patterns that help the development team define mechanisms for managing different versions of the work products, develop code in parallel with other developers and join up with the current state of development line, and identify what versions of code make up a particular component.

## 4 Proposed Agile Development Methodology

The proposed methodology aims to define roles and responsibilities and provide processes, lifecycle, practices and tools to be applied in embedded real-time system projects. It contains three different processes groups that should be used during the system development: *system platform, product development and management*.

The system platform processes group aims to instantiate the platform for a given product. It means that the system designer must choose the system components that will be part of the architecture and API platforms from a platform library. After that, the system designer has still the possibility to customize the architecture and API platforms in order to meet the application constraints. The customization process is carried out by programming the designer-configurable processors and runtime-reconfigurable logic integrated into the platform. The customization process is carried out by successive refinements in an iterative and incremental way into the proposed methodology.

The product development processes group offers practices to develop the application's components and integrating them into the platform. The functionalities which make up the product are partitioned into either hardware or software elements of the platform. The partitioning algorithms used to carry out this task take into account the energy consumption, execution time, and memory size of the application's components. The mechanical design is also part of this processes group, but it is out of the scope of this paper. The partitioning technique is also applied in an iterative and incremental way.

The product scope, time, quality, and costs parameters are monitored and controlled by the product management processes group. These parameters also influence the sys-

tem platform and product development processes groups. When the project starts with an infeasible project plan which needs corrective actions to be carried out then this processes group aims to get the project back on the track and ensure that the project's parameters are met. The product management processes group consists of the practices promoted by the Scrum agile method as well as the agile patterns described in Section 3. The next subsections are concerned with describing the processes groups, roles and responsibilities, and the processes lifecycle of the proposed methodology.

### 4.1 System Platform Processes Group

The system platform processes group is composed of the following processes: *product requirements, system platform, product line, and system optimization*. The *product requirements process* aims to obtain the system's requirements (functional and non-functional) that are relevant to determine the system platform in which the product will be built. The *platform instance process* helps the development team define the system platform by making use of a set of design tools and benchmarks.

After defining the system platform, the *product line process* helps the development team setup the repository in which the system platform components will be available to the product development. This process also allows the development team to implement and integrate system's functionalities into the system and release new product versions into the market. After implementing and integrating the system's functionalities into the product development line, the *system optimization process* provides activities to ensure that system's variables such as execution time, energy consumption, program size and data memory size satisfy the application constraints.

### 4.2 Product Development Processes Group

The product development processes group is composed of the following processes: *functionality implementation, task integration, system refactoring, and system optimization*. The *functionality implementation process* ensures that test cases are created for every product's functionality. This process helps increase the product quality and reduce the creation of complex functions. The *task integration process* provides means to integrate new implemented functionalities into the development line of the product without forcing the other team members to work around it.

The *system refactoring process* helps the development team identifies opportunity to improve the code and changing it without altering its external behavior. After refactoring the code, the *system optimization process* allows the development team to optimize small part of the code by making use of profiler tools that monitor the program and tells where, for instance, it is consuming time, energy, and memory space. This process guarantees that software metrics meets the system constraints.

### 4.3 Product Management Processes Group

The product management processes group is composed of the following processes: *product requirements, project management, bug tracking, sprint requirements, product line, and implementation priority*. The *product requirements process* (that also belongs to the system platform processes group) aims to obtain the system's requirements (functional and non-functional) that must be part of the product. The *project management process* allows the development team to implement the system's requirements by managing the product and sprint backlog, coordinating activities, generating system's build, and tracking the product's bug.

The *bug tracking process* allows the product leader to manage the lifecycle of the project's issues (bug, task, and enhancement) and provide the needed information about the product quality through the release notes for the end user. The *sprint requirements process* allows the development team to analyze, evaluate, and estimate the system's functionalities before starting a new project's sprint. This information is included into the sprint backlog which will help the development team partition the system functionalities into either hardware or software before starting the sprint.

The *product line process* guarantees that the system functionalities implemented during the sprint will be integrated into the product development line. This process also helps the development team to release new product versions into the market. The *implementation priority process* helps the product leader manage any kind of interruptions that may impact the project's goals. This process guarantees that the project's tasks are 100 percent completed after initiated.

### 4.4 Roles and Responsibilities

The proposed methodology involves four different roles and the responsibility of each role is described as follows:

**Platform Owner:** Platform owner is the person who is officially responsible for the products that derive from a given platform. This person is responsible for defining quality, schedule and costs targets of the product. He/she must also create and prioritize the product backlog, choose the goals for the sprints, and review the product with the stakeholders.

**Product Leader:** Product leader is responsible for the implementation, integration and test of the product ensuring that quality, schedule, and cost targets defined by the platform owner are met. He/she is also responsible for mediating between management and development team as well as listening to progress and removes block points.

**Feature Leader:** Feature leader is responsible for managing, controlling and coordinating subsystem projects, pre-integration projects, external suppliers that contribute to a defined set of features. The feature leader also tracks the progress and status of the feature development (deliverables, integration and test status, defects, and change requests) and reports the status to the product leader.

**Development Team:** The development team which may consist of programmers, architects, and testers are responsible for working on the product development. They have the authority to make any decisions, do whatever is necessary to do (according to the project's guidelines), and ask for any block points to be removed.

If the product to be developed is small, i.e. it is composed of few components and does not require other development teams to implement the product's functionalities then one product leader and the development team are enough for the product development. On the other hand, if the product is composed by several components and requires other development teams to implement the product's functionalities then the Feature Leader role must be involved in the processes. In this context, one product leader requires feature leaders to manage, control and coordinate components' projects. Therefore, for medium and larger projects, one product leader and several feature leaders and development teams may be involved in the processes.

### 4.5 Processes Lifecycle

The proposed agile methodology consists of five phases: *Exploration, Planning, Development, Release, and Maintenance*. In the *Exploration phase*, the customers provide requirements for the first product release. These requirements are included into the product backlog by the platform owner. After that, the platform owner and product leader estimate the requirements with no item larger than 3 person-days of effort. In this phase, the development team identifies the application constraints and estimates the system's metrics based on the product backlog items. With this information at hand, the development team is able to define the system platform that will be used to develop the product in the next phases.

In the *Planning phase*, the platform owner and customers identify more requirements and prioritize the product backlog. After that, the development team spends one day to estimate the sprint backlog items and decompose them into tasks. The tasks that make up the sprint backlog must take from 1 to 16 hours to be completed. Explanatory design and prototypes may also be developed at this phase in order to help clarify the system's requirements.

In the *development phase*, the team members implement new functionalities and enhance the system based on the items of the sprint backlog. The daily meetings are held at the same time and place with the purpose of monitoring and adapting the activities to produce the desired outcomes. At the end of the each iteration, unit and functional tests are executed in a continuous integration build. System optimization also takes place during this phase. The last sprint provides the product to be deployed in the operational environment.

In the *Release phase*, the product is installed and put into practical use. During this phase, it usually involves the identification of errors and enhancement in the system services. Therefore, the platform owner and customers decide if these changes will be included in the current or subsequent release. This phase aims to deliver the product and the needed documentation to the customer. The *Maintenance phase* may also require more sprints in order to implement new features, enhancement and bug fixes raised in the release phase.

## 5 Applying the Proposed Methodology

This section is concerned with describing the application of the proposed methodology in the development of the pulse oximeter equipment. We chose the pulse oximeter as a case study because it was already developed in another work by our research group using an ad hoc development methodology [10]. Therefore, we describe the pulse oximeter in this section only as an example of the application of the proposed methodology in the domain of embedded systems. Generally speaking, the pulse oximeter is responsible for measuring the oxygen saturation in the blood system using a non-invasive method. The architecture of this equipment is shown in Figure 1.

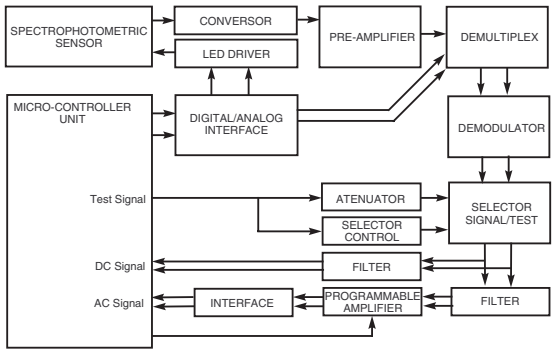


**Figure 1. Pulse Oxiemeter Architecture.**

The micro-controller controls the synchronization and amplitude of the led driver, which dispatches non-simultaneous stream pulses to the infrared and red leds. Both leds generate, respectively, infrared and red radiation pulses that cross the finger of a patient. After crossing the finger, a photo-diode catches the radiations level. A sequence of operations occurs until data reaches the micro-controller. Lastly, the micro-controller performs the calculation related to oxygen saturation level based on data received, and shows the result on a display. The final product contains about 5000 lines of C code. Due to timing and energy constraints, the sensor signals excitation and conditioning were implemented using hardware components while the control algorithm and the signal conversion system were implemented in software by the micro-controller.

The main system's characteristics are depicted as follows: *(i)* the level of the oxygen saturation and cardiac frequency must be shown every second, *(ii)* The user must be able to change the alarm configuration, *(iii)* the user interface of the pulse oximeter equipment must have a keyboard and a graphical display, *(iv)* the design of the system should be highly optimized for life-cycle cost and effectiveness, *(v)* the amount of software defects should be as low as possible, and *(vi)* the power dissipation of the final system should be about 2,35 Watts.

If we develop this product using the proposed methodology then there would be about three people involved, two developers and one product leader. The product would take approximately four sprints of three weeks to be developed, tested and delivered (rough estimation). The next subsections describe only the processes of the proposed methodology that focus on achieving the aims of the pulse oximeter equipment.

### 5.1 Process for Managing the Product Requirements

This process would help us identify the market needs for the pulse oximeter product line and manage the product requirements. At the beginning of the project, we could arrange a brainstorming meeting in order to capture high-level requirements of the product. After that, we could create an initial product backlog with the purpose of capturing more requirements and creating a first product prototype. The first project iteration would allow us to answer questions such as whether the technology needed for the system exists, how difficult it would be, and whether the engineers would have enough experience using that technology.

We could put much emphasis on delivering the system's functionalities *(i)*, *(ii)*, and *(iii)* in the beginning of the sprints. Delivering these functionalities with highest business value (the business value could range from 1 lowest to 5 highest), could help our customer (e.g., a representative of a hospital) and the product leader get feedback on functionality earlier and allow them to spot any misunderstanding more quickly. At the end of each iteration, the product leader and customer could verify if the product was still feasible or not. If the project was not feasible then it could have been canceled just after the end of the iteration *(risk management)*.

If we develop a new product in which the requirements cannot be expected to be fully available earlier in the development then this process would help us identify more requirements and update the product backlog as the project evolves. We could start the project with a set of functional and non-functional requirements. After running some project's sprint, our understanding about the product would increase and we could identify the requirements that were not captured at the beginning of the project.

### 5.2 Process for Managing the Project

This process would help us refine and prioritize the product backlog that contains the system's functionalities. In the sprint planning, the product leader and our customer could choose the goals of the next sprint based on the highest business value and risks of the product backlog items. After that, we could have a meeting to consider how to achieve the sprint's goals and to create the sprint backlog. The sprint backlog should contain only tasks in the 4-16 hour range.

During the system development, the sprint backlog could be updated on a daily basis as the activities were being accomplished. The product leader could hold daily meetings at the same place and time with the team members with the purpose of monitoring and controlling the complexity of the tasks. These daily meetings would provide a great feedback to the product leader and create the habit of sharing the knowledge. After starting the sprint, we could implement first the functional requirements and then focus on the non-functional requirements of the system. This approach would help us obtain better optimization results because we would be trying to optimize the global system instead of only parts of the system which sometimes could not lead to the global optimization.

During this phase, system's builds could also be generated on a daily basis which could help our customer clarify the requirements and asses the risks earlier in the development process. At the end of the sprint, the product leader and development team could show the results of the work to the customers. This meeting aims to present the product increment, technology and business situation. These artifacts would help the product leader and customers decide the goals of the next sprint. In addition, after each sprint review there would be a retrospective meeting which has the purpose of collecting the best practices used in the sprint and identifying what could be improved for the next sprint.

### 5.3 Process for Instantiating the Platform

This process would help us estimate the pulse oximeter metrics in order to define the system platform. To obtain the execution time and energy consumption metrics, we could specify the system's functionalities in the Unified Modeling Language (UML) by creating the class, collaboration and sequence diagrams. UML 2.0 profile could also be used to specify the system functionalities [7], but at the time we wrote this paper there was no tool available to convert the UML 2.0 diagrams into a programming language. The main benefits of the UML 2.0 is that it uses special stereotypes and design rules for modeling the behavior of the application and platform components and it also enables their parameterization.

The CASE (Computer Aided Software Engineering) tools like Together and Rational Rose could be used for the entry of the system model. After specifying the system model in UML using these tools, the code could be generated automatically in the language selected by the system designer (e.g., SystemC, Java, and C/C++). Nguyen et. al. [9] provides a tool that enables the system designer to specify the system model in UML and automatically convert it into SystemC code [9]. After generating the code in the selected language, hardware/software estimation tools could be used to estimate the execution time and energy consumption. We could use the estimation tool developed by our research group that is capable of estimating the execution time and energy consumption based on Assembly code [11].

Therefore, after estimating the system's metrics, we could provide this information to our hardware/software partitioning tool that is being developed as an Eclipse plug-in by our research group. This tool looks for the best partitioning that meets the design constraints. Since most design decisions are driven by constraints then we should incorporate the application constraints into our objective function so that partitions that met constraints would be considered better than those that did not meet. Finally, after instantiating the platform based on the application constraints then we could start developing the product.

### 5.4 Process for Implementing New Systems Functionalities

This process would help us implement the system's tasks of the pulse oximeter in a systematic way. According to the business value of the system's functionalities defined in the *process for managing product requirements*, we could start implementing the tasks responsible for measuring the oxygen saturation level in the patient's blood. In order to implement this functionality, we should first write the unit test for this functionality and thereafter we should successfully compile the unit test before really writing the functionality's code. If there would be compilation problems then the unit test for this functionality should be fixed.

After successfully compiling the unit test, we could start coding the functionality by following product's coding standard defined at the beginning of the project. The functionality's code would be considered completely implemented only after the team member successfully runs the unit test that was created for the functionality. Therefore, it would ensure that the functionality would be in compliance with its specification and the calculation of the oxygen saturation level in the patient's blood would be correctly performed. If there would be some need for splitting this functionality into different tasks then there would be the need for creating the unit tests for each code piece of the functionality.

### 5.5 Process for Refactoring the Code

After implementing the system's functionalities, we could identify in further sprints opportunities to improve an existing code. For instance, we could identify during the pulse oximeter project that the level of the oxygen saturation and cardiac frequency functionalities have some tasks in common. Both functionalities need to collect data from the sensor and identify the maximum and minimum signal pulse. Therefore, the application of this process would lead to elimination of duplicated code, reduction of the amount of system's functions, and improve the system performance.

But before improving the code for those tasks in common, we should first create branches in the system repository for not breaking an existing working code. After that, we should verify if there is some need for updating the unit test of the functionality. If there is no need to update the unit tests then we could start improving the code without altering its external behavior. After refactoring the code, we could run the unit test in order to verify if the changes are working correctly. If there is no compilation problem and the unit test dos not fail then we could integrate our changes into the product development line.

After integrating the code, the regression tests could be run in order to check if there is no compilation and semantic problems. If there is no problem then the refactoring would be completed. Another important point when applying the refactoring process is that system functions might also be moved from one system component to another, i.e., the system designers might decide to move a function from software running on one of the processors to a hardware block. For instance, the signal conditioning of the pulse oximeter could be moved from software component to a full-custom logic, an application-specific integrated circuit (ASIC), or reconfigurable logic component. This process would also lead to improve the system performance.

### 5.6 Process for Optimizing the System

This process would help us identify system's variable that could be optimized in order to meet the system's constraint. To optimize the system's variables, we should first establish the metrics and ensure that the refactoring process has already been applied. After that, we could run our profiler tool [11] to monitor the program and tell where it would be consuming time and energy. In this way, we could be able to find small parts of the program where these system's variables could be optimized. Thereafter, we could optimize the variables under attention by hand. As in refactoring, we could also carry out the changes in small steps. After each step, we should compile, test and run the profiler tool again. If the variable has not been optimized then we should return the changes in the version control system and continue the optimization process until the variables could satisfy the constraints.

## 6 Conclusion

This paper described an agile development methodology and its application in the development of the pulse oximeter. In order to create the methodology, we chose two agile methods XP and Scrum as well as organizational patterns named in this paper as agile patterns. Scrum is explicitly intended for the purpose of managing agile software development projects. On the other hand, XP is a collection of well-known development practices. The agile patterns provide means to structure the software development process of organizations.

When XP, Scrum and agile patterns are combined they cover many areas of the system development life-cycle. However, the combination of Scrum, XP and agile patterns does not mean that they can directly be used to develop embedded systems. Slightly changes were needed to: *(i)* adopt processes and tools to optimize the product's design rather than take paths that lead to designs that have no chance of satisfying the constraints, *(ii)* support software and hardware development through a comprehensive flow from specification to implementation, *(iii)* instantiate the system platform based on the application constraints rather than overdesign a platform instance for a given product, and *(iv)* use system platform to conduct various design space exploration analyses for performance.

To illustrate the use of the processes and tools of the proposed methodology, we described how it could be applied to develop the pulse oximeter equipment. In this case study, we used UML notation to specify the application functionalities and convert them into programming languages to apply the estimation and partitioning tools. For further steps, we are researching models that can carry enough information about the ultimate physical implementation at a high abstraction level and planning experimental studies where the methodology will be observed. After that, our goal is to introduce the proposed methodology step-by-step into the industry by using traditional measurement framework.

## 7 Acknowledgements

This work is partially supported by Brazilian Council of Research CNPq under grant number 55.3164/2005-8.

## References

[1] K. Beck and C. Andres. *Extreme Programming Explained - Embrace Change*. Second Edition, Addison-Wesley, 1999.

[2] S. Berczuk and B. Appleton. *Software Configuration Management Patterns*. First Edition, Addison-Wesley, 2002.

[3] J. O. Coplien and D. Schmidt. *Organizational Patterns of Agile Software Development*. First Edition, Prentice Hall, 2004.

[4] D. Gajski, F. Vahid, and S. Narayan. A system-design methodology: Executable-specification refinement. *European Conference on Design Automation, Paris, France*, 1994.

[5] B. Greene. Agile methods applied to embedded software development. *Proceeding of the Agile Development Conference (ADC'04).*, 2004.

[6] P. Koopman. Embedded system design issues (the rest of the story). *Proceedings of the International Conference on Computer Design (ICCD96)*, pages 310–317, 1996.

[7] P. Kukkala, J. Riihimki, M. Hnnikinen, T. Hmlinen, and K. Kronlf. Uml 2.0 profile for embedded system design. *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05).*, 2005.

[8] P. Manhart and K. Schneider. Breaking the ice for agile development of embedded software: An industry experience report. *Proceedings of the 26th International Conference on Software Engineering (ICSE04)*, page 3647, 2004.

[9] K. Nguyen, Z. Sun, , and P. Thiagarajan. Model-driven soc design via executable uml to systemc. *Proceedings of the 25th IEEE International Symposium on Real-Time Systems, Page(s) 459-468*, pages 459–468, 2004.

[10] M. Oliveira. *Desenvolvimento de Um Protótipo para a Medida Não Invasiva da Saturação Arterial de Oxigênio em Humanos - Oxímetro de Pulso*. Master thesis, Center for Informatics at Federal University of Pernambuco, 2002.

[11] M. J. Oliveira, S. Neto, P. Maciel, R. Lima, A. Ribeiro, R. Barreto, E. Tavares, and F. Braga. Analyzing software performance and energy consumption of embedded systems by probabilistic modeling: An approach based on coloured petri nets. *ICATPN 2006, LNCS 4024, pp. 261281, 2006.*, page 261281, 2006.

[12] K. Schwaber and M. Beedle. *Agile Software Development with Scrum*. First Edition, Series in Agile Software Development, Prentice Hall, 2002.

[13] A. S. Vicentelli and G. Martin. Platform-based design and software design methodology for embedded systems. *IEEE Design and Test of Computers*, 18(6):23–33, 2001.