Scientific Research Publishing

Journal of Software Engineering and Applications, 2016, 9, 479-490
http://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Development of a Wireless Environmental Data Acquisition Prototype Adopting Agile Practices: An Experience Report

Paul Celicourt[1], Richard Sam[2], Michael Piasecki[3]

[1]Civil Engineering Department, The City College of New York, New York, USA
[2]Electrical Engineering Department, The City College of New York, New York, USA
[3]CUNY's Environmental CrossRoads Initiative, The City College of New York, New York, USA
Email: pcelico00@citymail.cuny.edu, rsam00@citymail.cuny.edu, michael.piasecki@gmail.com

How to cite this paper: Celicourt, P., Sam, R. and Piasecki, M. (2016) Development of a Wireless Environmental Data Acquisition Prototype Adopting Agile Practices: An Experience Report. *Journal of Software Engineering and Applications*, **9**, 479-490.
http://dx.doi.org/10.4236/jsea.2016.910031

Received: August 20, 2016
Accepted: October 7, 2016
Published: October 10, 2016

Abstract

The traditional software development model commonly named "waterfall" is unable to cope with the increasing functionality and complexity of modern embedded systems. In addition, it is unable to support the ability for businesses to quickly respond to new market opportunities due to changing requirements. As a response, the software development community developed the *Agile Methodologies* (e.g., extreme Programming, Scrum) which were also adopted by the *Embedded System* community. However, failures and bad experiences in applying *Agile Methodologies* to the development of embedded systems have not been reported in the literature. Therefore, this paper contributes a detailed account of our first-time experiences adopting an agile approach in the prototype development of a wireless environment data acquisition system in an academic environment. We successfully applied a subset of the extreme Programming (XP) methodology to our software development using the Python programming language, an experience that demonstrated its benefits in shaping the design of the software and also increasing productivity. We used an incremental development approach for the hardware components and adopted a "cumulative testing" approach. For the overall development process management, however, we concluded that the Promise/Commitment-Based Project Management (PB-PM/CBPM) was better suited. We discovered that software and hardware components of embedded systems are best developed in parallel or near-parallel. We learned that software components that pass automated tests may not survive in the tests against the hardware. Throughout this rapid prototyping effort, factors like team size and our availability as graduate students were major obstacles to fully apply the XP methodology.

Keywords

Data Communications Devices, Rapid Prototyping, Real-Time and Embedded Systems, Systems and Software, Testing Strategies, Wireless Sensor Networks

1. Introduction

Despite its predominance in the last decades of the twentieth century, the single-pass model of software development, namely the "waterfall" model, suffers from serious shortcomings. These include its inability to cope with change when the requirements are not well defined on the onset, the need to re-write substantial portion of code, and the unpredictability of software quality due to late testing. Approaches of this type expose software development projects to high failure risks that may end up being cancelled due to large lag time in product delivery. This negates or at least complicates the need of businesses to quickly respond to new market opportunities. As a consequence, new paradigms for software development sought to develop modern and simple methods leading to: a timely development and delivery schedule, a mitigation strategy to reduce risks early in the development process, and the ability to incorporate changing requirements. Attention was geared towards the so-called "iterative and incremental development" (IID), a concept introduced in the mid-1950s. Thus, IID became the centerpiece of many software development approaches grouped under the *Agile Methodologies* umbrella since the publication of the Agile Manifesto (http://agilemanifesto.org/) in 2001. This latter defines a set of four core values and twelve principles for the individual *Agile Methodologies* with the most popular being Scrum [1] and XP [2]. They are sufficiently documented in the literature and for brevity and space limitations we do not present further details. A summary of several software development approaches including the *Agile Methodologies* can be found in [3].

The benefits of applying *Agile Methodologies* to enterprise software development have been tangible and embedded systems developers have sought to embrace these concepts. Embedded System is defined here as a combination of hardware, including at least a microprocessor and software controlling the hardware as part of a system or device designed to perform a dedicated function. A number of researchers and practitioners have attempted to apply *Agile Methodologies* to embedded systems development. For instance, Ronkainen and Abrahamsson [4] explored the possibility of using agile development techniques in the development of low-level telecommunications software with stringent hardware constraints. They noted that avoidance of up-front documentation, the negative effects of refactoring on hard real-time system timing and the need to cope with changes in requirements during development are obvious challenges to fully-fledged use of agile principles. Similarly, Codeiro *et al.* [5] propose an agile methodology namely "The next Methodology (TXM)" combining practices from XP, Scrum and organizational patterns of agile software development for embedded software development under stringent hardware constraints. However, with the proli-

feration of low-cost and high performance processors and microcontrollers, this is no longer a critical issue.

In addition, Gary *et al.* [6] successfully applied *Agile Methodologies* to the software development of an image-guided surgical toolkit system while Manhart *et al.* [7] adopted *Agile Methodologies* to the software component of an automatic breaking system at Daimler-Chrysler. These two examples defy Boehm's [8] claim that *Agile Methodologies* might not be suitable for life-critical embedded systems. Greene [9] uses a combination of XP and Scrum practices in the development of firmware for the Intel Itanium processor. Karlesky *et al.* [10] developed and applied the Model-Conductor-Hardware design pattern in testing embedded software drawing from the Model-View-Presenter and Model-View-Controller design patterns. Van Schooenderwoert and Morsicato [11] discuss a combination of five testing techniques including the "Guru Checks Outputs" technique (manual check of results) during the production of a mobile spectrometer designed.

It is obvious that the above authors focused primarily on the software development aspect of their embedded systems and overlook the application of *agile methodologies* to the hardware development aspect. Unfortunately, none of the above authors have reported on their effort to make the software works with the real hardware. Furthermore, they have not illustrated how such methodologies can be applied to hardware development. But, Chen [12] highlighted that there is no good way to adapt agile software development techniques to hardware development. However, Drechsler and Breiter [13] concluded that hardware designers can make use of many development and management concepts of the software domain after studying the similarities between hardware and software development. Furthermore, Conforto *et al.* [14] confirmed that *Agile Methodologies* can be applied in industries other than software development based on an exploratory survey of 19 small- and medium-size Brazilian companies. Myllerup [15] asserts that it is "possible" to transfer *agile methodologies* into electronics- and mechanics-based product development projects. But, he also confesses that the nature of those projects (sourcing, manufacturing of prototypes) is not well aligned with methods that use short iterations with frequent deliveries. Monte [16] laid out the special challenges for agile adoption in hybrid software and hardware development and suggests the use of different agile approach for each component. The author also proposes the use of the CBPM/PBPM [17] to better utilize agile approach to the management of hardware development. The core idea of CBPM/PBPM is that things get done faster and in a more motivated way through organizations if team members voluntarily make promises to deliver their contribution to the project by a certain date instead of defining a sprint timetable (1 - 4 weeks) concept of Scrum. The approach would be to think about the basic, smallest chunks of hardware functionality that can be delivered using the most agile way.

There has been a noticeable reticence in reporting the failures and bad experiences in the literature on the application of *Agile Methodologies* to embedded systems development. For instance, Kaisti *et al.* [18] surveyed over 28 papers on the application of

agile methods to embedded-systems and discovered that failures and bad experiences have not been reported in any of the papers. Srinivasan *et al.* [19] concluded that the absence of failures with respect to adoption of agile methods is the startling gap in published literature on the subject. In this paper, we report on challenges and experiences of our first attempt to apply agile approaches to the rapid prototyping of an end-to-end wireless environmental data acquisition system in the context of two graduate students from two different engineering departments carrying out collaborative research. Our paper contributes a detailed account of our first-time experiences adopting an agile approach during the prototype development of the system in an academic environment. While we report on our failures and bad experiences, we also demonstrate the adaptation of *Agile Methodologies* to the development of both the hardware and software components of the system in our case study.

Our environmental data acquisition system consists of multiple end-nodes collecting data from sensors (such as air and water temperature, soil moisture) and transmitting the data and metadata to a central node where the data is automatically annotated with the metadata and stored into an instance of the Consortium of Universities for the Advancement of Hydrologic Sciences, Inc.'s Observations Data Model (CUAHSI ODM) [20]. The central node operates in such a way that an end-node can integrate the network at any time without disturbing the network. This data model is designed to store point observations along with sufficient corresponding metadata to allow users to unambiguously interpret and understand it and to provide traceable heritage from raw measurements to useable information. In addition, the data model comprises community defined semantics and syntaxes using controlled vocabularies needed to enable interoperability of hydrologic information systems. We developed software components to control the end-nodes and organize them as a network. In addition, software components enable automated organization of the incoming data with metadata from the end-nodes into the ODM. In addition, we developed hand-soldered Printable Circuit Boards (PCB) that are attached to the Raspberry Pi microcomputer (www.raspberrypi.org) that provides the processing power for the end-nodes. In this paper, we simply provide a succinct description of the system without diving into the details about its components which will be the subject of a subsequent paper.

This paper is organized as follows: we first introduce the approach and challenges we faced during the hardware and software development. Because we adopted an agile approach in which the test-driven development principle plays a significant role, we then present the testing approach and challenges we faced. Next, we transition to the discussion section before we conclude the paper.

2. Approach and Challenges to the System Development

We started our project without a formal development process in mind. In fact, initially we were not aware of any process or approach that could help us deliver a working system quickly. As it was our first attempt developing such a prototype, we started off by deploying hardware accessories and software scripts that would do as little as reading

measurements from a digital temperature sensor (a DS18B20 thermometer, https://www.adafruit.com/product/374).

Given our small team size (2 developers) and limited availability to conduct true team work in addition to the initial strategy of trial-and-error, we concluded that the CBPM/PBPM was a more suitable management process to continue testing and adding new functionalities and features to the system. Features include items such as adding logical and physical supports for analog sensors or wireless communication to the PCB among others. For every working feature added, we made time to meet and discuss the next step. While the CBPM/PBPM method is based on making and fulfilling commitments, we realized that promises are more satisfactorily fulfilled on time when initial works had already been started (what we called "half-baked promise") before the promise was made and a certain date proposed. In other words, we found that the concept of "half-baked promise" is an important step to successful promise delivery. A similar technique is used in most STEM Graduate Schools where PhD students are required to carry out initial works towards their thesis such as publishing one or two papers before they actually defend their thesis proposal before their PhD Committee.

2.1. Hardware Development

We started out using breadboard and jumper wires to connect sensors and Analog-to-Digital Converter (ADC) to the Raspberry Pi for the hardware components. For anything that did not work well initially (such as incorrect sensors wiring), we could disconnect, make the necessary changes and reconnect again. While this was a simple and straightforward way "to fix things", with rising complexity because of new jumper cables added to the board, it became apparent that changes and fixes were much harder to implement. As a consequence, we transitioned to the PCB (**Figure 1**) which also prompted us to learn about existing processes that would allow us to continue the development in a more systematic way.

We have been exposed to the challenging situation where individual components that worked fine in a previous version of the PCB did not work in the next version. We realized, in most of cases, that these were due to design schematics (made with Eagle, http://www.cadsoftusa.com/download-eagle/) that were not always properly translated into the physical prototype. For instance, most of the components of the second version of the PCB were not working from the first try prompting us to unsolder a few components and solder them back on the circuit board. If not possible, we reorganized the wiring depending on the components being added to the PCB which served as the basis to create a new PCB version.

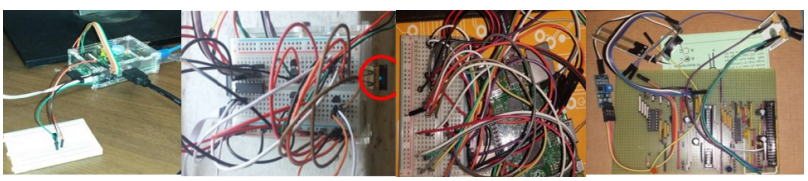

Figure 1. Stages of the hardware component development.

The creation of multiple PCB versions and activities such as using jumper wires and breadboard, review of wiring, and adding new components incrementally seem to be analogous to the XP incremental and Test-Driven Development (TDD) principles. We need to mention that we did not test and add every component to the breadboard or PCB separately and integrated the components after testing. Not only would this not guarantee that the components would work fine when integrated, all the components are not necessarily independent. For instance, when we want to test an analog sensor, we could not avoid using an ADC Chip and vice-versa. Therefore, we had to make sure that both the ADC and the sensor are properly wired.

2.2. Software Development

Our software development evolved from writing simple Python scripts to read digital sensor measurements wired directly to General Purpose Input/Output (GPIO) ports of the Raspberry Pi to modules/packages that perform automated network organization and data management. On the end-nodes side of the system, the software modules read and process measurements from multiple sensors, decode received command and encode response to commands and ultimately package responses and transmit them using the ZigBee wireless communication protocol (www.zigbee.org). On the data management side, we develop software modules that process announcement messages and response to commands, encode commands to be sent using the ZigBee protocol, annotate incoming end-nodes data with corresponding metadata to comply with the ODM specifications among others.

Funds to acquire the hardware components for the prototype were not yet available from the beginning of the project. So, instead of waiting to acquire the Xbee radio modules (www.digi.com/lp/xbee) for data transmission using the ZigBee protocol, we started developing bits of our software component around the HTTP protocol using the Python Web Framework named Bottle (www.bottlepy.org) to simulate data transmission between a Raspberry Pi and a laptop computer. The main reason that we chose the HTTP protocol was mainly the affordability and ease of hardware accessories acquisition as modern hardware development kits are often provided with a WIFI dongle. The idea was to use the framework to quickly experiment the mechanics of message encoding, transmission and decoding between a data acquisition system and a computer.

We found a significant difference between our work with the HTTP protocol and the message encoding, transmission and decoding using the ZigBee protocol and also the library packages used. Consequentially, a substantial amount of work was necessary to transition from one protocol or communication interface to the other. We learned that tightly coupling the software and hardware developments is critical to avoid dramatic impacts of either component on the other. An additional critical issue was that it can be difficult to discern if either the software or the hardware is at fault if something malfunctions when there is a lag between hard- and corresponding software development. We concluded that it is much safer to develop the software along with the hardware and not wait late for testing.

We also realized that we needed a better way to continue the software development while the software components running on the laptop, which are responsible for network coordination and data organization, had grown sufficiently to slow our progress. Consequently, we started learning about agile development processes of which we adopted the XP approach. Our first steps were a) to create a number of automated tests using PyUnit, the Python standard unit testing framework, b) to rely heavily on refactoring and c) continuous update and integration. Prior, we used the "Guru Checks Outputs" technique, an approach that proved to be insufficient, tedious, and error-prone.

With the adoption of the TDD and incremental development principles of XP, we improved our development process dramatically even though we had to go through a steep learning curve. Before we embraced the XP approach in the software development process, the software components used to be fragile in the sense that parts can break at any time when adding new components to the existing ones. As a consequence, we spent some times debugging the system which slows down our development effort. In addition, we sometimes experience a cascading effect of codes breakage that constrained us to go back and fix several other components of the ensemble, which is time-consuming. However, when we shifted our approach to XP, we mitigated such issues. Consequently, this suggests that the adoption of *Agile Methodologies* to embedded systems development can potentially yield a more robust and reliable product. We sometimes went back and made little fixes such as change in data structure or algorithm modification as part of our refactoring efforts. But, these changes are not appropriated to the cascading effect of codes breakage. Thus, we concluded that such an experience demonstrated the benefits of XP in shaping the design of the software and also increasing productivity.

Furthermore, the XP approach requires the application of additional practices such as pair programming, metaphor, 40-hour weeks, real on-site customer, planning game among others. As graduate students, we played the role of the customer ourselves by setting priorities on features that needed to be developed first and next. Due to our team size and the different school-related activities we are involved as students, we could not apply many of the XP practices. For instance, we could not work 40 hours a week but instead we worked under an irregular schedule which fits better our availability. In terms of metaphor, we did not use this practice as we did not have to explain the system to any people external to our team and we did not also have external collaborators on the project. We also have not developed elaborated User Stories as if we were dealing with a real customer. Instead we discuss and/or take notes in our regular project planner notebook instead of cards. In addition, because we played a developer-customer role, we did not pay attention to technical words or jargon in our User Stories. We need to mention that User Stories decisions were made at the system level and we then derived what needs to be done both on the software and hardware sides. We did not also have formal Acceptance Tests following the implementation of the User Stories at the end of each iteration, instead we performed what we would qualify as

on-the-fly Acceptance Tests. In other words, we tested on-the-go as we were developing. As for Continuous Integration, we used the git version control system tool and commit (no push) updates and fixes every few hours to a local repository. We have also strived to keep the system in a working state at any time. This leads us to the pair programming practice which we could not apply too. The reason is that we had only one software developer and one hardware developer in addition to our research supervisor.

3. System Testing Challenges

Developing embedded software while testing the codes "live" on the corresponding hardware is cumbersome and time consuming which slows down the software development, a circumstance that cannot be avoided entirely however. For the end-nodes, we used a dual-target approach where tests and application codes are first written and run on the development machine and periodically run on the target environment. However, for the data management and network coordination application, we adopted a three-step testing approach in most of the cases. First, we tested application codes for a particular module or a set of classes in an isolated mode. Then we integrated with the existing application codes and ran the tests. The idea was to find out whether the new module works when integrated and a call to this module from the larger system does not break either the new codes or the previously working codes. The final step was to test the behavior of the full system after integration of new codes and turned out to be the most challenging in the testing phase.

A common approach to perform embedded software testing is to use virtual versions of the hardware such as mocks [18] and simulations and emulations. However, our experiments demonstrate that this is not the same as testing on the true hardware where live conditions such as heat, movement and other environmental impacts must be taken into consideration, to ensure reliability. We found that hardware specifications and capability need to be taken into consideration as well. For instance, during our testing against the end-nodes, we found that "communication latency" plays a significant role in breaking the codes that passed our automated tests.

Another limitation we faced concerned the message size an Xbee radio module can transmit. We came to understand that using mock testing might have let us pretend, for example, that any message size can be transmitted in a single transmission. When we encountered a limitation of this type we were prone to be side-tracked in an attempt to immediately fix this problem which in turn set us back and derailed our development time schedule. In fact, we spent a couple of days fixing the problem needing to develop tests and application codes for both the node sending the message to slice it into smaller pieces and the message receiver to re-assemble the message slices. This was not a straightforward task as more than one node can send sliced messages asynchronously. We tackled this issue using the three-step approach to test the codes. The lesson here was that codes that work in an isolated fashion are prone to be broken when integrated into a larger system or falls under the influence of communication latency of the hardware system.

As for the hardware components, we did a "cumulative testing" of the components added to the PCB in the sense that any stage, we tested the functionality of all previous components added to the PCB up to the most recently added one. Overall, our approach permitted that we were able to often quickly detect whether it was the software or the hardware that is at fault when a problem surfaced. An exception to this general behavior was one case where we first added a Real-Time Clock (RTC) to the PCB and the RTC could not keep up the current time. After spending a considerable amount of time trying to debug the system that saw us address power supply of the RTC which we thought might have been defective as the RTC has worked before. We discovered that is was actually a problem with the RTC driver that was not loaded when the Raspberry Pi boots up.

4. Discussion

In this paper, we introduce our approach to software and hardware development of our prototype to automatically stream and manage environmental data from sensor-to-data-management system. Despite the benefits offered by the XP approach adopted, we could not fully apply such an approach to the software development due to the factors mentioned above. In addition, we also found that getting suppliers actively involved is also critical to the development of embedded systems under *Agile Methodologies*. For instance, we still cannot conduct experiments with Modbus/RS485 and SDI-12 sensors due to order not fulfilled on time which sets us back again in the time schedule.

We have learned that testing embedded software by avoiding the interaction with the hardware is similar to duct-taping the codes. In particular, testing in an isolated fashion simply helps the developer to be on track to have the code working both when integrated to a larger application and interacting with the hardware. The codes will eventually break when integrated in a larger system and worse may happen when it interacts with the software. Sometimes, major alterations are needed to get the off-hardware-tested codes to work with the hardware. This experience prompts us to critically view the application of the Mock Testing approach in which the mock is created with no real functionality, but rather mimics the behavior of a module's interface.

The application of a single agile method to the development of embedded systems is hard to achieve. In our case, we attempted to adapt agile software development techniques such as TDD and Continuous Integration to the hardware development. But, this was not achieved purely in the same way as it is done in the software components development. Software inherently permits a top-down and fine-grained approach in testing the components almost at all levels. However, the hardware permits only testing and integration of components as a black box. Hence, applying the refactoring practice in hardware development is no different. Nevertheless, we found that these XP practices and principles including the use of User Stories were more susceptible to be integrated in embedded hardware development.

We have also identified a few shortcomings of the Pyunit testing framework. For instance, the "dictionary" data structure (made of *key-value* pairs) in Python can handle

very complex data organization consisting in almost any primitive or derived data structures including itself either as *key* or *value*. However, the methods to compare dictionary-based data organization (*assert Dict Equal* and *assert Dict Contains Subset*) in Pyunit are not capable of handling the unlimited data arrangement possibilities of this data structure. Consequently, a function may produce an expected result, but the corresponding test fails depending on the complexity of the data structure of either the *key* or the *value*. In such case, the developer may do either or both of the following:

a) Implement tests of low granularity such as checking number of elements in the structure.
b) Make an extra effort to write tests of high granularity that take into account the intricacy of the data structure, but such tests are very likely to be non-reusable.

Finally, the data management application is developed around the *Django Web Framework* (www.djangoproject.com) that offers object-relational mapping functionalities. The mechanics of creating tests with PyUnit is different than with Django. Thus, the developer needs to learn testing techniques for both frameworks including their inherent features and/or shortcomings.

5. Conclusions

In this paper, we present our experiences adopting *Agile Methodologies* to both the hardware and software development of a wireless environmental data acquisition system. The application of *Agile Methodologies* to our prototype development effort has been effective in ensuring a more reliable and robust system. In addition, the *Agile Methodologies*, especially the XP methodology contributes to increasing our productivity while shaping the design of the software components of the system. Due to our availability and team size to perform true team work, we adopted the Promise/Commitment-Based Project Management which was better suited than Scrum for our development process management.

The experiments prompted us to realize that *Embedded System* is a niche domain for the application of agile development methodologies. Some principles such as TDD, refactoring, User Stories and Continuous Integration are more straightforward and simpler to apply than others. These principles, especially TDD and refactoring, played a critical role in the development of the prototype system. We have experienced the benefits and we will stick to those principles in future development efforts while exploring other XP principles and other methodologies such as Scrum.

Our future work plan will consist in exploring the implications of *Agile Methodologies* on the end-to-end development (from conception to release) of embedded systems adopting a Minimum Viable Product (MVP) approach to test assumptions and incorporate customers' feedbacks.

Acknowledgements

This work is supported by the Grove School of Engineering at The City College of New York, the City University of New York's Environmental Cross Roads Initiative and the

IEEE Foundation. We thank the anonymous reviewers for their insightful comments and suggestions which contribute to the improvement of this paper.

References

[1] Schwaber, K. (2004) Agile Project Management with Scrum. Microsoft Press, Redmond.

[2] Beck, K. (2000) Extreme Programming Explained: Embrace Change. Addison-Wesley Professional, USA.

[3] Qasaimeh, M., Mehrfard, H. and Hamou-Lhadj, A. (2008) Comparing Agile Software Processes Based on the Software Development Project Requirements. *International Conference on Computational Intelligence for Modelling Control & Automation*, Vienna, 10-12 December 2008, 49-54. http://dx.doi.org/10.1109/CIMCA.2008.54

[4] Ronkainen, J. and Abrahamsson, P. (2003) Software Development under Stringent Hardware Constraints: Do Agile Methods Have a Chance? In: Marchesi, M. and Succi, G., Eds., *Extreme Programming and Agile Processes in Software Engineering*, Springer, Berlin Heidelberg, 73-79. http://dx.doi.org/10.1007/3-540-44870-5_10

[5] Cordeiro, L., Mar, C., Valentin, E., Cruz, F., Patrick, D., Barreto, R. and Lucena, V. (2008) An Agile Development Methodology Applied to Embedded Control Software under Stringent Hardware Constraints. *ACM SIGSOFT Software Engineering Notes*, **33**, 5. http://dx.doi.org/10.1145/1344452.1344459

[6] Gary, K., Enquobahrie, A., Ibanez, L., Cheng, P., Yaniv, Z., Cleary, K., *et al.* (2011) Agile Methods for Open Source Safety-Critical Software. *Software*: *Practice and Experience*, **41**, 945-962. http://dx.doi.org/10.1002/spe.1075

[7] Manhart, P. and Schneider, K. (2004) Breaking the Ice for Agile Development of Embedded Software: An Industry Experience Report. *Proceedings of the* 26*th International Conference on Software Engineering*, Edinburgh, 23-28 May 2004, 378-386. http://dx.doi.org/10.1109/ICSE.2004.1317460

[8] Boehm, B. (2002) Get Ready for Agile Methods, with Care. *Computer*, **35**, 64-69. http://dx.doi.org/10.1109/2.976920

[9] Greene, B. (2004) Agile Methods Applied to Embedded Firmware Development. *Agile Development Conference*, Salt Lake City, UT, 22-26 June 2004, 71-77. http://dx.doi.org/10.1109/ADEVC.2004.3

[10] Karlesky, M., Williams, G., Bereza, W. and Fletcher, M. (2007) Mocking the Embedded World: Test-Driven Development, Continuous Integration, and Design Patterns. *Proceedings of the Embedded Systems Conference*, San Jose, CA, 1-5 April 2007, 1518-1532.

[11] Van Schooenderwoert, N. and Morsicato, R. (2004) Taming the Embedded Tiger-Agile Test Techniques for Embedded Software. *Proceeding of the Agile Development Conference*, Salt Lake City, UT, 22-26 June 2004, 120-126. http://dx.doi.org/10.1109/ADEVC.2004.21

[12] Chen, E. (2015) Bringing a Hardware Product to Market: Navigating the Wild Ride from Concept to Mass Production. CreateSpace Independent Publishing Platform, North Charleston, South Carolina, USA.

[13] Drechsler, R. and Breiter, A. (2007) Hardware Project Management: What We Can Learn from the Software Development Process for Hardware Design? *Proceedings of the* 2*nd International Conference on Software and Data Technologies* (*ICSOFT* 2007), Barcelona, 22-25 July 2007, 409-416.

[14] Conforto, E., Salum, F., Amaral, D., da Silva, S. and de Almeida, L. (2014) Can Agile Project Management Be Adopted by Industries Other than Software Development? *Project Man-*

agement Journal, **45**, 21-34. http://dx.doi.org/10.1002/pmj.21410

[15] Myllerup, B. (2011) Why Agile Does Matter in an Embedded Development Environment. Scrum Alliance. Retrieved 7 May 2016. https://www.scrumalliance.org/community/articles/2011/march/why-agile-does-matter-in-an-embedded-development-e

[16] Monte, M. (2012) Challenges of Adopting Agile in Combined Hardware and Software Environments. cPrime. Retrieved 26 April 2016. https://www.cprime.com/2012/08/challenges-of-adopting-agile-in-combined-hardware-and-software-environments/

[17] Sull, D.N. and Spinosa, C. (2007) Promise-Based Management: The Essence of Execution. Harvard Business Review, April, 78-86.

[18] Kaisti, M., Rantala, V., Mujunen, T., Hyrynsalmi, S., Könnölä, K., Mäkilä, T. and Lehtonen, T. (2013) Agile Methods for Embedded Systems Development: A Literature Review and a Mapping Study. *EURASIP Journal on Embedded Systems*, **2013**, 1-16. http://dx.doi.org/10.1186/1687-3963-2013-15

[19] Srinivasan, J., Dobrin, R. and Lundqvist, K. (2009) "State of the Art" in Using Agile Methods for Embedded Systems Development. 33*rd Annual IEEE International Computer Software and Applications Conference*, **2**, 522-527.

[20] Horsburgh, J.S., Tarboton, D.G., Maidment, D.R. and Zaslavsky, I. (2008) A Relational Model for Environmental and Water Resources Data. *Water Resources Research*, **44**, 1-12. http://dx.doi.org/10.1029/2007WR006392