International Journal of Innovative Computing, Information and Control
Volume **15**, Number **3**, June **2019**

ICIC International ©2019 ISSN 1349-4198
pp. 1177–1191

APPROACH OF INTEGRATING BEHAVIOUR-DRIVEN DEVELOPMENT WITH HARDWARE/SOFTWARE CODESIGN

Mohammad Alhaj[1], Gilbert Arbez[2] and Liam Peyton[2]

[1]Faculty of Engineering
Al-Ahliyya Amman University
Amman 19328, Jordan
m.alhaj@ammanu.edu.jo

[2]School of Electrical Engineering and Computer Science
University of Ottawa
800 King Edward Avenue, Ottawa, Ontario K1N 6N5, Canada
{ garbez; lpeyton }@uottawa.ca

Received September 2018; revised January 2019

Abstract. *Using typical approaches in designing embedded systems to manage the specification and design of the hardware and software is not suitable in today's projects. In this paper, we propose a model-driven approach to integrate Hardware/Software codesign with Behaviour-Driven Development. Hardware/Software codesign approach allows the hardware and the software to be designed and implemented concurrently and optimizes the project design constraint, such as performance and cost. Behaviour-Driven Development spurs project stakeholders to collaborate to ensure the right software is developed to meet their needs and ensures that all project participants communicate in the same language. The approach exploits the advantages of each approach and provides the ability to describe the behaviour of the software as executable user stories in a Hardware/Software codesign environment. The approach is evaluated using a renewable energy project in collaboration with a private company in Canada to build a system for autonomous load management of self-forming renewable energy nanogrids.*

Keywords: Embedded systems, Behaviour-Driven Development, Hardware/Software codesign, Test-driven development, Nanogrid systems, Agile software

1. Introduction.

Agile software development methodologies have gained acceptance in the today's market [1]. As opposed to delivering one large software code at the end of the project like the waterfall approach, the software development process in agile methodology provides simple client approved incremental software product releases as soon as they are ready. A set of principles that support software development using agile methodology is described in [2]. These principles promote for continuous communication and collaboration between project stakeholders to manage the software development [3]; exchanging user stories to describe system behavior from the perspective of users; using test-driven approach to delivering software releases incrementally [4].

Embedded systems are characterized as of being heterogeneous where different software and hardware technologies and platforms may operate. Hardware and software are tailored to perform a specific task within a larger and complicated environment. Hardware/Software (HW/SW) codesign is used, in embedded computing, to allow the hardware and the software to be designed and implemented together and optimize the project design constraint, such as performance and cost. There is typically limited attention to developing user interfaces and the focus is mainly on software functional specifications

DOI: 10.24507/ijicic.15.03.1177

defined at the architectural level [4]. However, due to the growing importance of software development in embedded systems, stakeholders are exploring agile software development methodologies as being highly adaptive and iterative based on a collaborative team [5]. Behavior-Driven Development (BDD) is an emerging methodology which describes the behavior of the system as executable user stories. It focuses on how the system behaves for the users that interact with the system and ensures that all project participants communicate in the same language. BDD promotes Test-Driven Development (TDD) where automated test cases are used to drive software development [6]. It simplifies translation between software development language and the language of the user's domain.

This paper extends, updates, and provides more detail on earlier research results p-resented at the conference in [7]. In this paper, we extend the background and related work (Section 2); we describe in detail the architecture and process of integrating BDD with the HW/SW codesign approach (Section 3); we also present and analyze the results of a different case study from the conference paper, called load management user story (Section 4). The framework exploits the advantages of each methodology and provides the ability to describe the behaviour of the software as executable user stories in an HW/SW codesign environment. The approach is evaluated using a renewable energy project in collaboration with a private company in Canada to build a system for autonomous load management of self-forming renewable energy nanogrids.

The paper is organized as follows: Section 2 presents the background and related work; Section 3 presents an overview for our proposed approach; Section 4 presents a case study of self-forming nanogrid system; Section 5 presents the testing results; and Section 6 presents the conclusion and future work.

2. Background and Related Work.

Software development methodology organizes and structures the work of developing a software system into tasks that are performed at different stages of the software development process. The early traditional methodologies were relying on heavily initial requirements, detailed and documented stages of planning, rigid and consecutive phases of development. The most widely used is the waterfall methodology [9] where consecutive phases of software development and design flow like a waterfall. These phases are conception, initiation, analysis, design, implementation, testing, deploying and maintenance. The V-model [10] is an extension of waterfall where the phases of the development are associated with their phases of testing. Also, Capability Maturity Model (CMM) [11] is with five-level processes: initial, managed, defined, quantitatively managed and optimized.

The agile software development methodology became more accepted and adopted in today's software business market and was able to replace the traditional one. This is due to its simplicity, flexibility and the ability to meet the changes in the business requirements and evolution of technologies. There are many examples of agile methodologies supporting iterative software development process in today's market including Rational Unified Process (RUP) [12], an iterative software development process; eXtreme Programming (XP) [13], where small groups work together to iteratively produce the code; Scrum [14], an agile development framework where large processes are broken down into small pieces of user stories; Test-Driven Development (TDD) [15], a software development process that relies on test cases; and BDD [16].

HW/SW codesign is a methodology for combining the work of software and hardware development teams for complex electronic systems which integrate both software and hardware [17]. There has been work to apply HW/SW codesign in various practices and case studies such as management systems [18], image processing [19] and artificial neural networks [20].

With more than 1.5 billion people being out of electricity mainly in the rural areas, it is a challenge to build and maintain a sustainable and robust power delivery network by extending the electrical grid due to cost issues [21,22]. To overcome challenges of grid development, several research projects were proposed. In [23], a promising project proposed a solution using a scalable, self-forming nanogrid control architecture. The architecture is used to manage and monitor the electrical power supply to dynamically varying loads within a local electrical grid which may operate either isolated from, or connected to, the main grid. In [24], a proposed concept, called energy control centers, utilizes the emerging installation technologies of energy sources and storages where DC nanogrid in residential regions is interconnected to traditional AC utility grid. The proposed concept is an alternative future system solution for improving energy consumption and optimizing power management. In [25,26], DC Bus Signaling (DBS) is used to control nanogrid systems through prioritizing and scheduling load shedding. DBS strategy is used to convey the nanogrid information based on the level of the DC bus. Another approach in [27] introduces hybrid photovoltaic/diesel generation systems to supply a remote power plant using new control devices that maintain the optimum and balance energy flows.

In summary, it is obvious that the works above have addressed some features that are similar to our work. The architecture of embedded systems is characterized as of being heterogeneous since different software and hardware technologies and platforms may operate. It is considered to be complex since their components are forming a monolithic behaviour, where each component could operate for a distinct purpose, and interactions between components could be restricted to certain service points. The whole system will not function unless most or all of the components are working [4]. The major advantage of our proposed approach compared to the others is that it is using BDD agile software methodology that can be used to resolve designing complexity by performing HW/SW testing procedures early and frequently at different level of architecture abstractions. The adopted BDD methodology has a behavior driven approach [8] to ensure the system is validated against the complex set of scenarios that must be satisfied [3]. It is also a model-driven [28] in terms of the software solution generated. Our proposed approach is used to help the development teams to work concurrently and avoid any delay might be caused by the slow working progress of some teams and used to verify the correctness of integrated parts of the system at different system development stages.

3. **Approach Overview.**

3.1. **Building testing environments using HW/SW codesign.**

The proposed model-driven approach uses a closed loop testing environment principle that systemically develops and tests the hardware and software testing scenarios. It is used to separate the requirement concerns and provide concurrent testing environments that allow different development teams to operate separately and not to interfere with each other.

Figure 1 describes the process of building the testing environments in our proposed approach. Usually, user requirements of a project are provided in a form of documents to describe the functional and non-functional requirements of the system. Based on these requirements, user stories are created where there is a simplified description of the software feature from an end-user point view. A user story can be represented in a textual format using natural language, or graphical format using behaviour modeling languages, e.g., UML activity or flow chart.

Based on HW/SW codesign concept, our proposed approach supports three testing environments (as in Figure 1).

a) System Testing Environment provides the ability to test and validate the developed software in a real hardware environment. It contains the hardware grid such as the Printed Circuit Boards (PCBs), processors and other artifacts. The captured user stories are used to define the system configuration to build system testing environments. They are also used for developing the software and for generating testing scenarios.

   BDD methodology is applied within the HW/SW codesign approach to test and validate both the software and hardware of the system where further changes can be applied to the user requirement document.

b) Software Testing Environment provides the ability to test the software modules independently from the hardware. Testing stubs are used to simulate the behaviour of the hardware APIs. Initially, the developed software in the system environment is migrated and deployed in the software environment; then, BDD methodology is used to validate the behavior of developed software using testing scenarios.

c) Hardware Testing Environment provides the ability to test the measures of the hardware components independently of the software modules. Testing scenarios captured previously in the system environment are used to validate the behavior of the hardware components. The testing focuses on individual hardware component not an end to end testing.

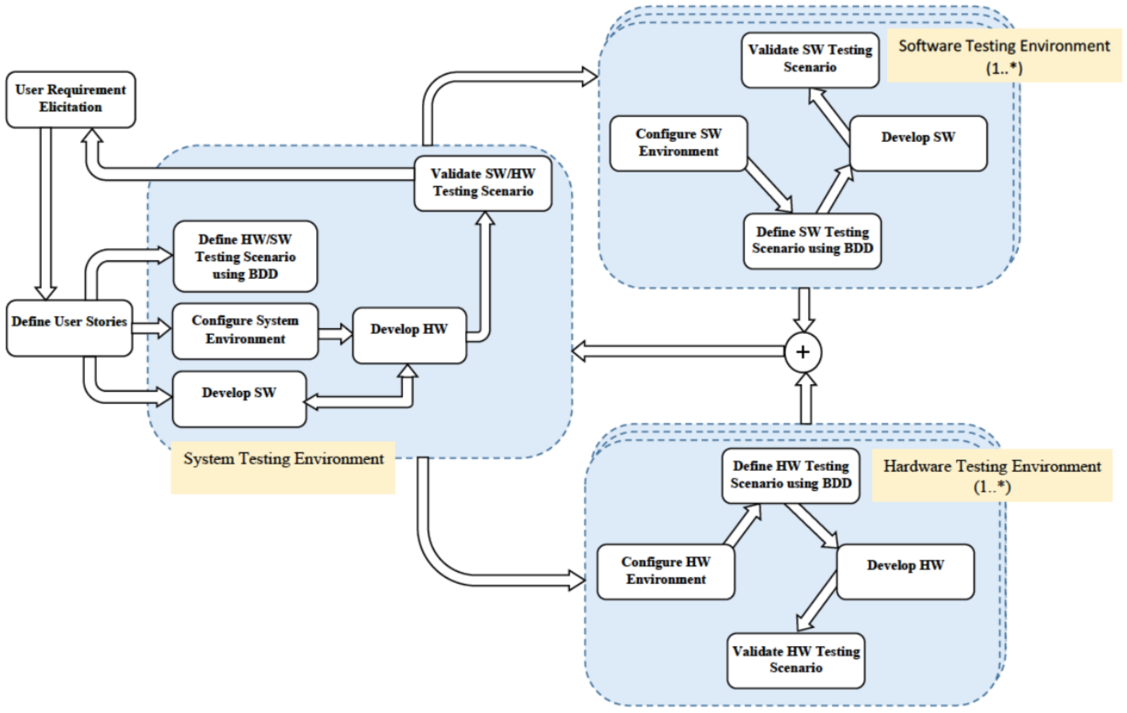

User Requirement Elicitation
Define User Stories
Define HW/SW Testing Scenario using BDD
Configure System Environment
Develop SW
Develop HW
Validate SW/HW Testing Scenario
System Testing Environment
Validate SW Testing Scenario
Software Testing Environment (1..*)
Configure SW Environment
Develop SW
Define SW Testing Scenario using BDD
+
Define HW Testing Scenario using BDD
Hardware Testing Environment (1..*)
Configure HW Environment
Develop HW
Validate HW Testing Scenario

FIGURE 1. Architecture and process of the testing environments

Notice that our proposed approach allows to build multiple environments of software modules and hardware components. This helps the developer teams to migrate and test the software and the hardware at different levels of architecture abstraction and then integrate them again after completing the tests.

3.2. **Creating testing scenarios using Behaviour-Driven Development (BDD).** Agile software methodology, such as BDD, can be used to resolve designing complexity by performing HW/SW testing procedures early and frequently. The target is to build user stories and validate them with respect to testing scenarios. Each user story is validated by multiple of testing scenarios which capture combinations of system configuration elements and alternative behaviour paths. The flow of alternative paths is described by traversing *time-based* states. Theoretically, there are an infinite number of combinations that can be captured. However, in practice we do not use the whole combinations of system configuration elements because it causes redundant testing scenarios. A template of testing scenario is described in Table 1. To standardize writing testing scenarios, we use a template described by D. North in [16] based on three directives.

a) *Given*: represents the initial context of the scenario. The initial context is defined using input parameters which are assigned or compared with constants.
b) *When*: represents the timestamp of an event occurrence or decision node during the testing scenario workflow.
c) *Then*: represents an action to be done after every event occurrence.

Table 1. A template of testing scenario using BDD

Given	$param_1$, $param_2, \ldots, param_N$
Sequence	*At t0: When* $<$ *event_Occurence*$_1$ $>$ *Then*: $<$ *action*$_1$ $>$ *At t1: When* $<$ *event_Occurence*$_2$ $>$ *Then*: $<$ *action*$_2$ $>$ $\vdots$ *At tN: When* $<$ *event_Occurence*$_N$ $>$ *Then*: $<$ *action*$_N$ $>$

4. **Case Study: Behaviour-Driven Development Self-Forming Nanogrid System.** In this section, we describe a collaborative project with Solantro Semiconductor Inc. to design a self-forming nanogrid system. Our objective is to apply a BDD [8] within HW/SW codesign to develop an autonomous load management software that can be embedded in power architecture owned by Solantro to intelligently manage load demands to the power supply [22].

Figure 2 describes the architecture of the nanogrid system in [29]. A collection of individual photovoltaic bank ($\mathrm{PV}_1, \ldots, \mathrm{PV}_\mathrm{N}$) and battery bank ($\mathrm{B}_1, \ldots, \mathrm{B}_\mathrm{N}$) are connected through inverters to the Intelligent Distribution Panel (IDP) with a load bank ($\mathrm{L}_1, \ldots, \mathrm{L}_\mathrm{N}$). Solantro has established a hierarchy of control functions [29] to supply AC electrical power to dynamically varying loads within a local electrical grid [30], which may operate either isolated from, or connected to, the main grid. The system is using shedding loads based on priority from lowest to highest to maximize utility when constrained by the availability of supply. The higher priority control maximizes power usage by optimizing power consumption of loads to run as little as possible from batteries.

The IDP controller is embedded in the power architecture of the IDP panel and forms point of interfacing to the local/other networks, user interfaces and portals. It is responsible for:

a) Participating in power optimization strategies by allowing interaction with external and internal devices;
b) Monitoring voltage, frequency, phase, and power of the main grid, PV panels, batteries and loads;
c) Connecting and disconnecting the PV panels, batteries and loads with the main gird according to safety, regulatory, and policy direction;
d) Providing a secured granular view of actual loads and giving the user the opportunity and obligation to configure the system using IDP portal.

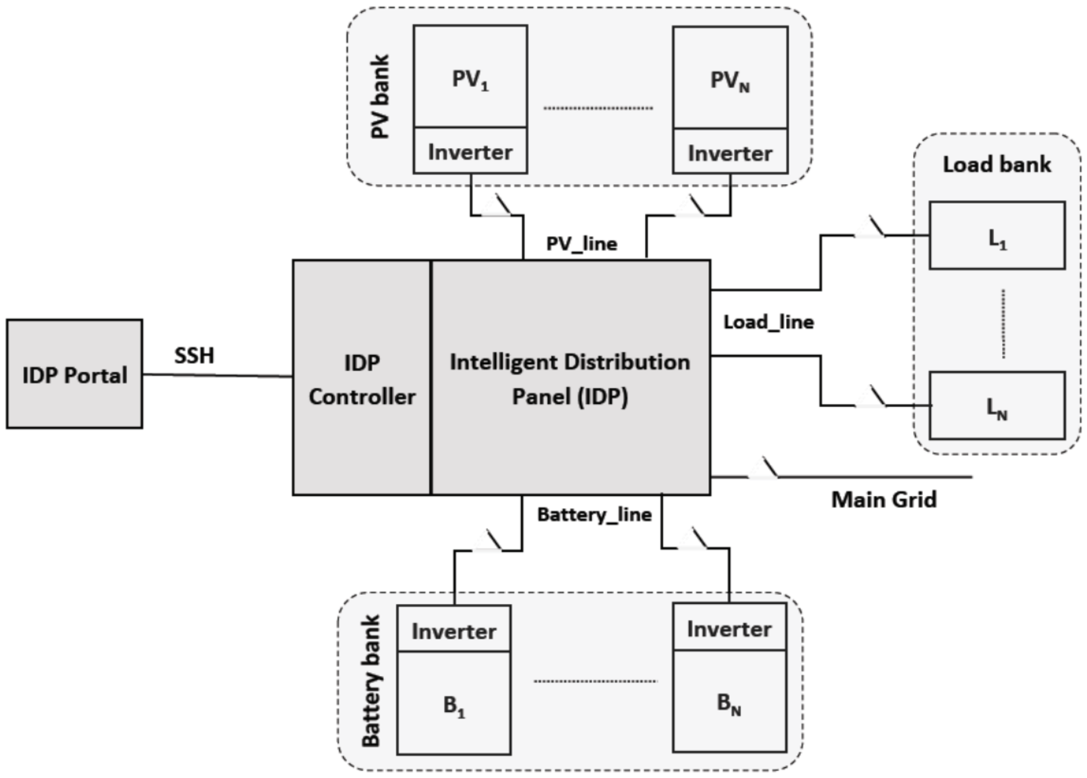


FIGURE 2. The architecture of the self-forming nanogrid system

4.1. **HW/SW codesign testing environments.** We created three testing environments based on our proposed approach, as described in the previous section.

- System Testing Environment: It is a black box testing that requires a complete and integrated system of both software and hardware to evaluate whether it complies with user requirements. The produced results are presented using IDP portals in a form of real-time signals, for example, the grid voltage or the open/close state of the relay.
- SW Testing Environment: It is a white box testing which validates only the developed software, while the behaviour of hardware components is simulated using testing stubs. The software behaviour of each process in the nanogrid system is tested by an SW module testing procedure, as in Figure 3.
  a) The load management testing is used to evaluate the load shedding behaviour of the nanogrid system.
  b) The data analysis testing is used to evaluate the behaviour of the modules that handle the computational functionality and produce the nanogird system state measures, such as voltage, current and power.
  c) The data acquisition testing is used to evaluate the behaviour of the modules that handle raw data acquisition from the hardware and opening/closing relays.
- Hardware Testing Environment: it is a black box testing which validates the behaviour of hardware components. The produced output is viewed on electrical instruments such as the oscilloscope. Software test drivers, including IDP controller modules, can also be used to send inputs to the hardware components and verify their behavior.

4.2. **SW module testing procedure.** We use a multi-process architecture running on Linux OS to develop the nanogrid software, as in Figure 3. This improves the software

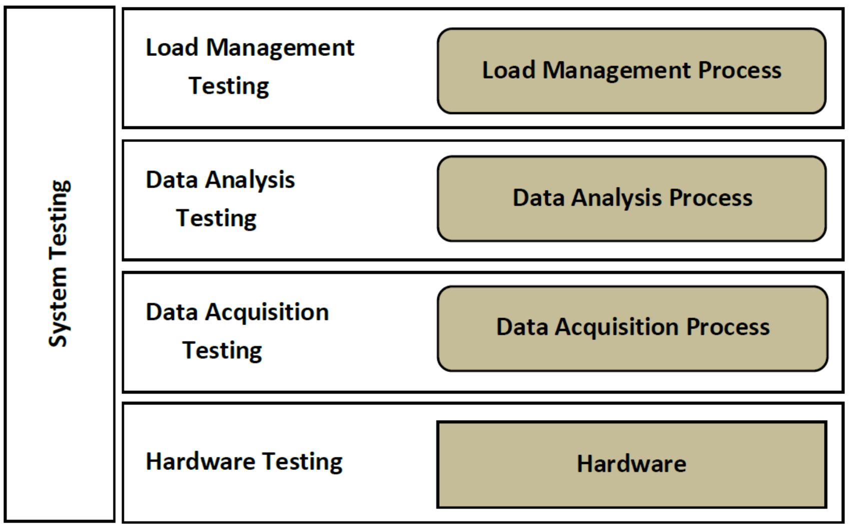


FIGURE 3. The structure of the testing procedures

performance by distributing the software operations between the processes. It also reduces the chances of process wait time in cases of synchronous messaging between the software and the hardware component. We also introduced shared memories and signaling mechanisms to allow the three processes to interrupt and communicate with each other during the operation. Shared memories contain data structures used to pass data between the processes. When the software is executed, the program starts three processes at the same time. These processes are:

- The Data Acquisition process: A low-level process that is mainly responsible for communicating with the hardware by reading the raw data, i.e., hexadecimal samples, from the HW component buffers and stores it into the Raw Data structures. It also handles the operation of opening/closing the IDP relays.
- The Data Analysis process: A mid-level process that is responsible for the main computational functionality by reading the raw data from the shared memory obtained by the Data Acquisition process and calculating the values that reflect the state of the nanogrid system necessary for controlling and monitoring (such as voltage, current, frequency, real and reactive power and power factor pf). The nanogrid state is saved in the Nanogrid State Data Structures.
- Load Management process: A top-level process that is responsible of monitoring and controlling the system behaviour. The calculated values, obtained by the data analysis process and stored in the shared memory, are used to handle the system logic. Additional shared memory serves for interprocess communication with the low-level Data Acquisition process to open and close relays.

4.3. **Behaviour specified by user story.** The target is to build user stories of nanogrid system operations and validate them with respect to testing scenarios. Each user story is validated by multiple of *testing scenarios* which capture combinations of alternative paths and system configuration.

Figure 4 describes the user stories of self-forming nanogrid power system. The initial user story is the dark start when the system starts up in the morning with no energy available in the nanogrid. PV panels are illuminated one at a time as the sun rises. As energy becomes available batteries are charged and loads are connected. Next, the load management user story describes the process of system load monitoring by connecting and disconnecting the load relays based on the voltage and frequency of the grid. When

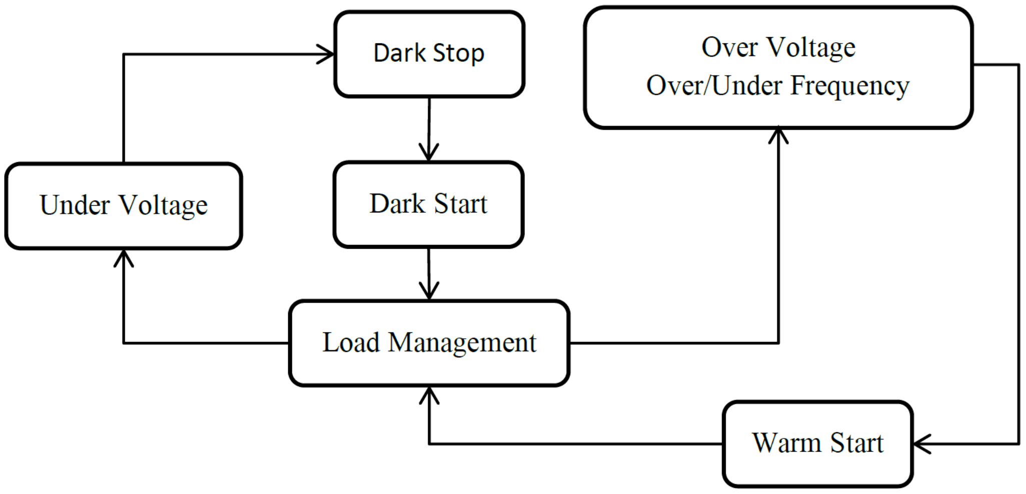


FIGURE 4. User stories of self-forming nanogrid power system

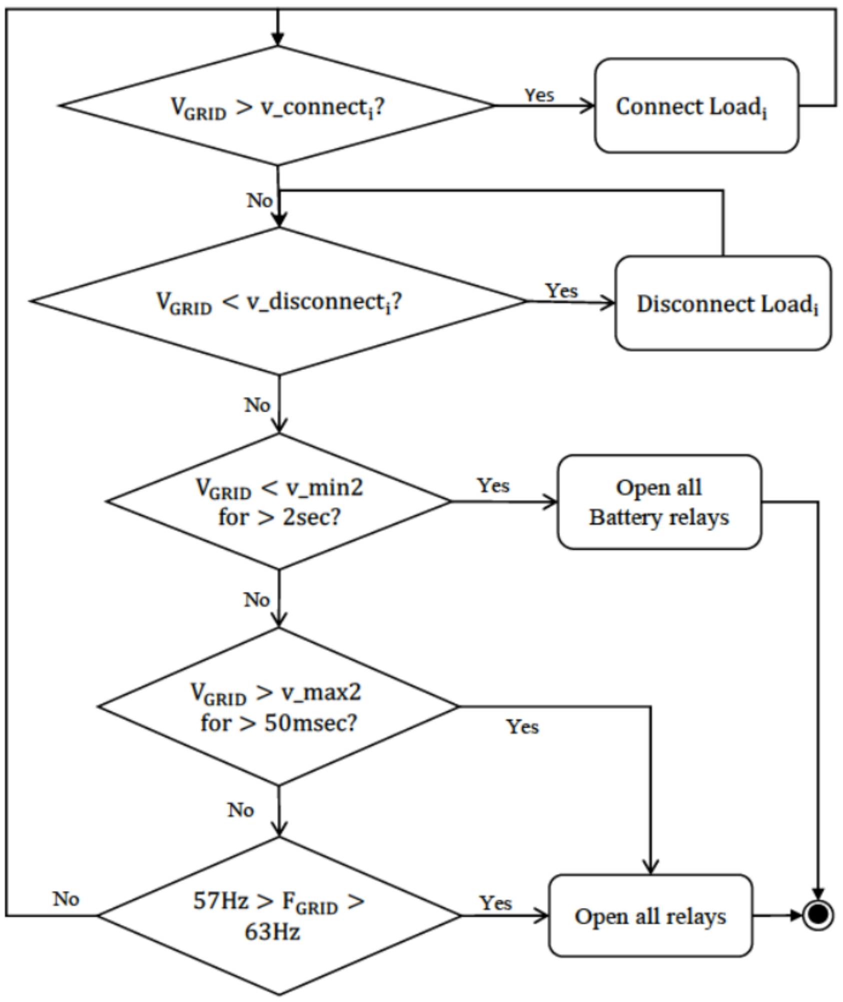
an under voltage occurs due to the drop of the illumination, the system will progressively shed loads as the State of Charge (SoC) of the batteries depletes. The highest priority loads will continue to be powered until the SoC of the batteries reaches the lower safe limit for the batteries used. Lower priority loads will be powered until the nanogrid voltage falls below their specified shut-off thresholds. When an over voltage or under/over frequency occurs, the system re-starts up with energy available from the PV panels or batteries. The PV panels are illuminated as the sun rises. As the alternative current is present on the nanogrid, batteries are charged.

In this paper, we will present load management user story, as in Figure 5. The IDP controller continuously handles connecting/disconnecting loads to the grid by checking grid voltage and frequency with respect to connect/disconnect voltages defined in the configuration file. The controller also handles three protective scenarios: a) when grid voltage goes below v_min2 (216 volt) for more than 2 second, it opens the relays of the batteries; b) when grid voltage exceeds v_max2 (264 volt) for more than 50 msecond, it opens all the relays of IDP; and c) when the grid frequency is below 57 Hz or exceeds 63 Hz, it opens all the relays of the IDP.

4.4. **System configuration.** The configuration file defines the list of variables that is used to initialize the software, such as the PVs, Batteries and Load parameters, and the GPIO pin numbers. Figure 6 describes the xml configuration file with the following sections.

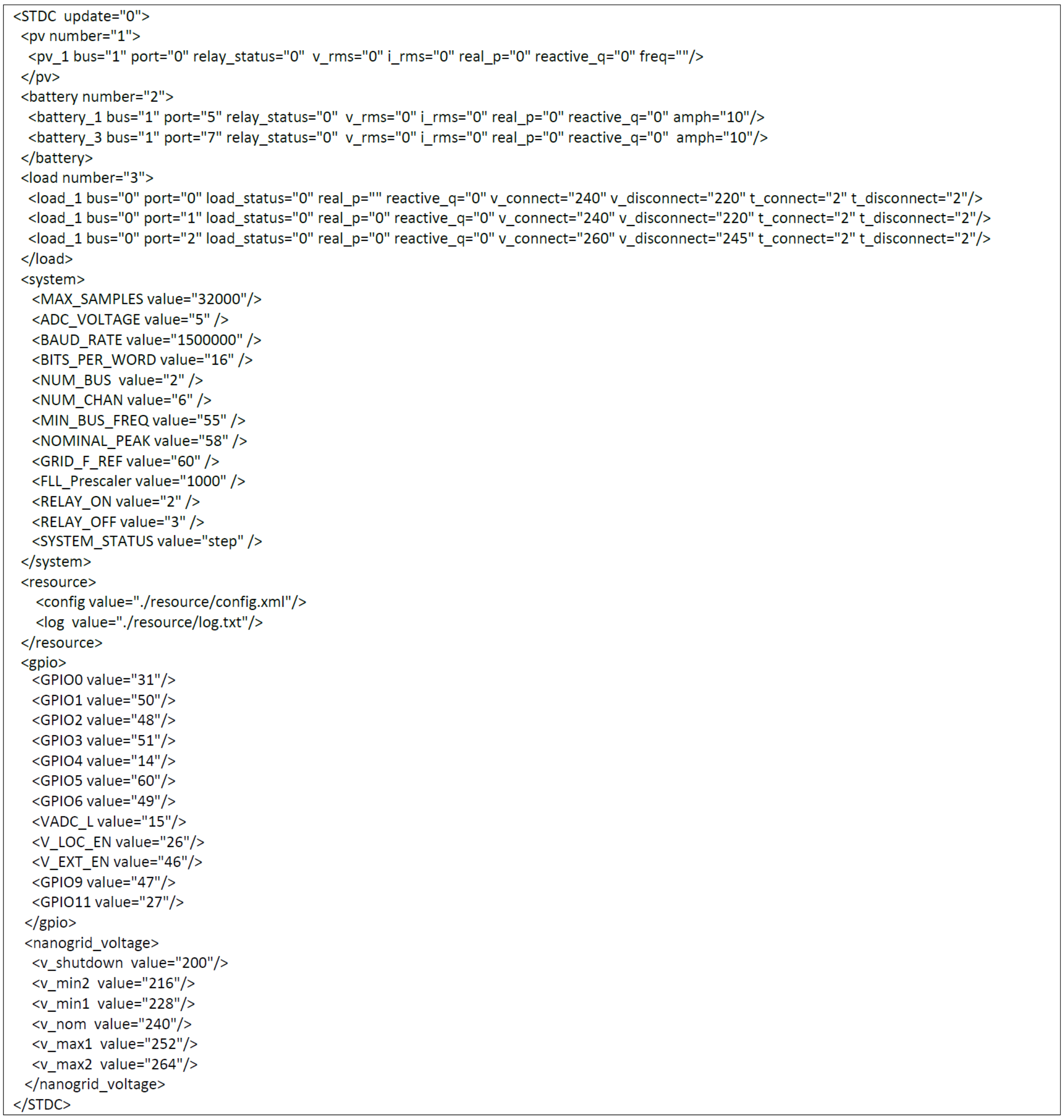
- Properties of PVs and Batteries: bus number, port number, frequency, Root Mean Square (RMS) voltage and current, real and reactive power.
- Properties of Loads: bus number, port number, load status, voltage of connect and disconnect, time to connect and disconnect.
- System properties: sampling rate, baud rate, bit per word, number of buses, number of ports and system status.
- General-Purpose Input/Output (GPIO) pin numbers.
- Levels of grid voltage: shutdown, minimum, nominal, maximum voltages.

4.5. **Load management testing scenario.** In this section, we capture a sample of testing scenario from load management user story (as in Section 4.4). Figure 7 describes testing scenario with three loads: two batteries and one PV. Initially, there is enough



User Story	**Load Management**
Actors	Main Grid, Set of Loads, Set of Batteries, Set of PVs
Input	Grid Voltage > 0
Sequence	1. Begin 2. Loop through all loads a. Check Grid voltage b. If Grid voltage > Connect voltage, i. Close load relay. c. If Grid voltage < Disconnect voltage, i. Open load relay. d. If Grid Voltage < v_min2 for greater than 2 second i. Open the all Battery relays e. If Grid Voltage > v_max2 for greater than 50 msecond i. Open all relays ii. End f. If Grid Frequency < 57 Hz or Grid Frequency > 63 Hz i. Open all relays ii. End
Output	Open all Battery relays or Open all IDP relays

FIGURE 5. User story of Load Management process



```
<STDC  update="0">
 <pv number="1">
  <pv_1 bus="1" port="0" relay_status="0"  v_rms="0" i_rms="0" real_p="0" reactive_q="0" freq=""/>
 </pv>
 <battery number="2">
  <battery_1 bus="1" port="5" relay_status="0"  v_rms="0" i_rms="0" real_p="0" reactive_q="0" amph="10"/>
  <battery_3 bus="1" port="7" relay_status="0"  v_rms="0" i_rms="0" real_p="0" reactive_q="0"  amph="10"/>
 </battery>
 <load number="3">
  <load_1 bus="0" port="0" load_status="0" real_p="" reactive_q="0" v_connect="240" v_disconnect="220" t_connect="2" t_disconnect="2"/>
  <load_1 bus="0" port="1" load_status="0" real_p="0" reactive_q="0" v_connect="240" v_disconnect="220" t_connect="2" t_disconnect="2"/>
  <load_1 bus="0" port="2" load_status="0" real_p="0" reactive_q="0" v_connect="260" v_disconnect="245" t_connect="2" t_disconnect="2"/>
 </load>
 <system>
   <MAX_SAMPLES value="32000"/>
   <ADC_VOLTAGE value="5" />
   <BAUD_RATE value="1500000" />
   <BITS_PER_WORD value="16" />
   <NUM_BUS  value="2" />
   <NUM_CHAN value="6" />
   <MIN_BUS_FREQ value="55" />
   <NOMINAL_PEAK value="58" />
   <GRID_F_REF value="60" />
   <FLL_Prescaler value="1000" />
   <RELAY_ON value="2" />
   <RELAY_OFF value="3" />
   <SYSTEM_STATUS value="step" />
 </system>
 <resource>
    <config value="./resource/config.xml"/>
    <log  value="./resource/log.txt"/>
 </resource>
 <gpio>
   <GPIO0 value="31"/>
   <GPIO1 value="50"/>
   <GPIO2 value="48"/>
   <GPIO3 value="51"/>
   <GPIO4 value="14"/>
   <GPIO5 value="60"/>
   <GPIO6 value="49"/>
   <VADC_L value="15"/>
   <V_LOC_EN value="26"/>
   <V_EXT_EN value="46"/>
   <GPIO9 value="47"/>
   <GPIO11 value="27"/>
  </gpio>
  <nanogrid_voltage>
   <v_shutdown  value="200"/>
   <v_min2  value="216"/>
   <v_min1  value="228"/>
   <v_nom  value="240"/>
   <v_max1  value="252"/>
   <v_max2  value="264"/>
  </nanogrid_voltage>
</STDC>
```

FIGURE 6. System configuration of self-forming nanogrid project

power produced such that $V_{\mathrm{GRID}} > 0$ volt and $F_{\mathrm{GRID}} = 60$ Hz. At t1, t2, t3 and t4, $V_{\mathrm{GRID}} = 242$ volt. At t4, $F_{\mathrm{GRID}} = 65$ Hz.

5. Testing Results.

5.1. System testing environment.

The system testing environment was performed in a lab where the IDP panel was assembled and the developed IDP controller was deployed in Linux OS using Beagle Bone Black (BBB) micro-processor [31]. Several voltage sources were connected to represent PV panels in the nanogrid system. Batteries, however, were already installed. In order to display the output, a Laboratory Virtual Instrument Engineering Workbench (LabView) application [32] is also connected to interact with the system and view relay status and real-time measures for the main grid, PVs, Batteries and Loads such as voltage, current, frequency and power.

Given	PV = {PV_1}, Battery = {B_1, B_2}, Load = {L_1, L_2, L_3}, V_{GRID} = 242 volt, F_{GRID} = 65 Hz, xml config file.
Sequence	1. Initially $V_{GRID} > 0$ 2. At t1: when V_{GRID} > v_connect$_1$ a. then, close the relay of the L_1. 3. At t2: when V_{GRID} > v_connect$_2$ a. then, close the relay of L_2 4. At t3: when V_{GRID} < v_disconnect$_3$ a. then, open the relay of L_3 5. At t4: when $F_{GRID} = 65$ a. then, open the relays of L_1, L_2, L_3 b. open the relay of B_1, B_2 c. open the relays of PV_1

FIGURE 7. Load management testing scenario

TABLE 2. Results of system testing scenario

	Input		Output		
	Grid voltage	Grid frequency	Relay status of {L_1, L_2, L_3}	Relay status of {B_1, B_2}	Relay status of {PV_1}
Initial	> 0	60	{Open, Open, Open}	{Close, Close}	{Close}
End	242	65	{Open, Open, Open}	{Open, Open}	{Open}

The results of system testing scenario of the load management user story are described in Table 2. It validates the results of the testing scenario where initially, grid voltage is greater than zero, the frequency is 60 Hz, the PV and batteries are connected to the IDP while the loads are disconnected. At the end of the process, the grid voltage is 242 volts with 65 Hz frequency and all the PVs, batteries and loads are disconnected.

5.2. **SW testing environment.** We perform three SW module testing procedures: Load Management Testing, Data Analysis Testing and Data Acquisition Testing. In the SW testing environment, the developed software is implemented and tested without the hardware being installed. Instead, three types of test stubs are created to replace the behaviour of the hardware during the testing. The test stubs come in a form of text files.

a) Input files represent the hardware components that provides data to the IDP controller, i.e., Analog to Digital Converters (ADCs).
b) In/Out files represent the hardware components where the IDP controller reads, writes or updates data, i.e., relays.
c) Log files record events and messages that occur during the testing.

The testing results described in Table 3 were produced by running the SW module testing scenarios. The load management testing describes the top level behavior of the nanogrid system. The data analysis testing describes the values of input parameters in a form of 2500 sample array in hexadecimal format. In order to produce the output measures, the input samples are first converted from hexadecimal to decimal. The converted decimal samples are then calibrated, and the RMS value is calculated. The data acquisition module, not shown in the table, describes the values of input parameters in a form of a continuous reading of 12-bit binary, the size of the ADC buffer. The values of the output parameters are stored into a form of 2500 sample array in hexadecimal format.

TABLE 3. Results of SW module testing scenario

(a) Load management testing

	Input		Output		
Event	Grid voltage (Volt)	Grid frequency (Hz)	Relay status of {L_1, L_2, L_3}	Relay status of {B_1, B_2}	Relay status of {PV_1}
Initial	> 0	60	{Open, Open, Open}	{Close, Close}	{Close}
t1	242	60	{Close, Open, Open}	{Close, Close}	{Close}
t2	242	60	{Close, Close, Open}	{Close, Close}	{Close}
t3	242	60	{Close, Close, Open}	{Close, Close}	{Close}
t4	242	65	{Open, Open, Open}	{Open, Open}	{Open}

(b) Data analysis testing

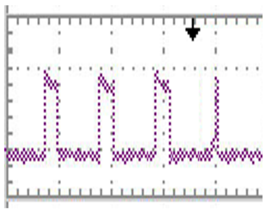
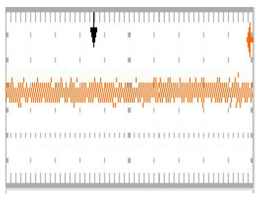
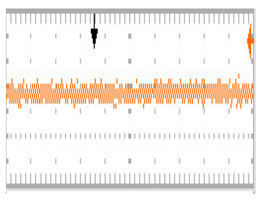
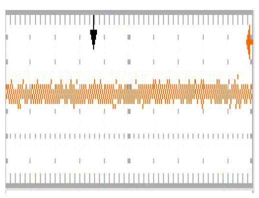
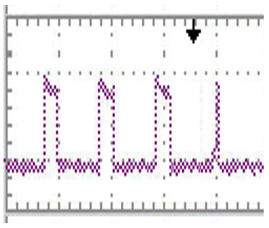
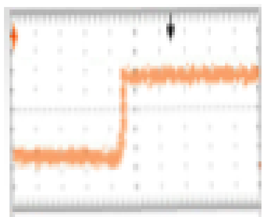
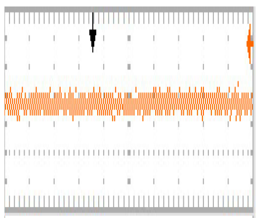
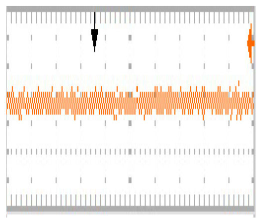
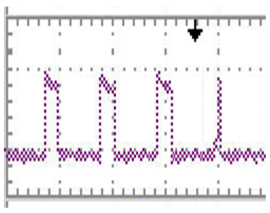
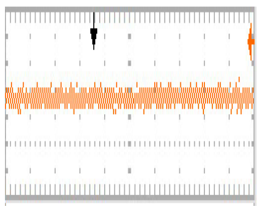
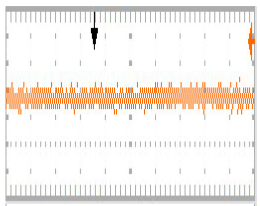
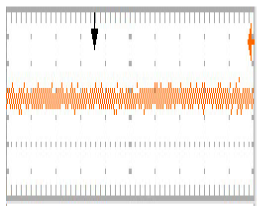
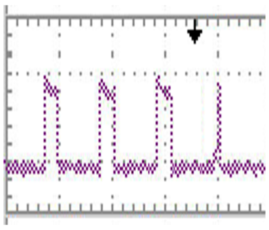
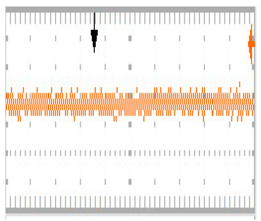
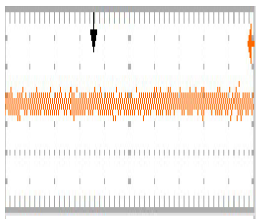
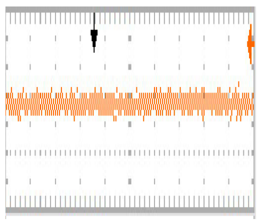
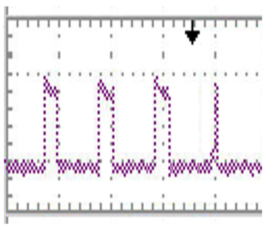
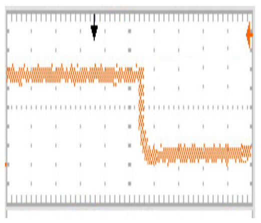
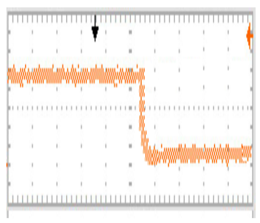
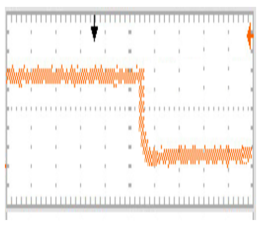
	Input	Output	
Event	Grid voltage (2500 samples in hex)	Grid voltage (Volt)	Grid frequency (Hz)
Initial	ae8, ae6, ae2, add, ada, ad9, ad6, ad0, acd, acc, aca, ac4, ac0, ac0, abe, ab9, ab5, ab2, ab0, aab, aa6, aa3, aa2, a9f, a9a, ...	> 0	60
t1	a8d, a88, a87, a84, a7f, a7a, a77, a75, a71, a6c, a69, a68, a65, a5f, a5b, a59, a56, a51, a4c, a4a, a48, a43, a3e, a3a, a39, ...	242	60
t2	a75, a71, a6c, a69, a68, a65, a5f, a5b, a59, a56, a51, a4c, a4a, a48, a43, a3e, a3a, a39, a36, a30, a2c, a2b, a27, a21, a1d, ...	242	60
t3	a56, a51, a4c, a4a, a48, a43, a3e, a3a, a39, a36, a30, a2c, a2b, a27, a21, a1d, a1b, a19, a13, a0e, a0b, a08, a04, 9fe, ...	242	60
t4	a39, a36, a30, a2c, a2b, a27, a21, a1d, a1b, a19, a13, a0e, a0b, a08, a04, 9fe, 9fb, 9fa, 9f6, 9f0, 9eb, 9e9, 9e6, 9e1, ...	242	65

5.3. **Hardware testing results.** In hardware testing, we use lab instruments to display the input/output signals producing the hardware components involved in the testing scenarios. Table 4 describes a sample of the signals produced, by the main grid and relays of the Loads, Batteries and PVs during the scenario testing of load management user story. The voltage samples of the grid is an input; while the relays of Load L_1, Battery B_1 and PV_1 are output. Notice that at t1, the Load L_1 is connected to the grid and at t4 all relays are disconnected from the grid.

6. **Conclusion and Future Work.**

We proposed an approach that integrates classic HW/SW codesign with the Behavior Driven Development (BDD) as an agile software methodology. The approach is used in complex systems to coordinate the development teams to work concurrently and avoid any delay might be caused by the slow working progress of some teams and help to verify the correctness of integrated parts of the system

TABLE 4. Result of HW testing scenario

	Input	Output		
Event	Grid voltage	Relay status of $\{L_1\}$	Relay status of $\{B_1\}$	Relay status of $\{PV_1\}$
Initial				
t1				
t2				
t3				
t4				

at different system development stages. The approach exploits the advantages of BDD also to provide the ability to describe the behaviour of the software as executable user stories in HW/SW codesign environment, and ensure target systems are validated against a complex set of scenarios that must be satisfied. The approach is applied on a renewable energy project in collaboration with a private company in Canada to build a system for autonomous load management of self-forming renewable energy nanogrids. The load demand of the system can be intelligently managed using an embedded controller within the nanogrid architecture. Various challenges at the development phase were easily solved by using the proposed agile approach, such as HW/SW delays, 50 Hz signal noise, antenna effect, zero crossing. In the future, we are planning to validate the proposed approach using other projects.

Acknowledgment. The authors would like to thank Solantro Semiconductor Inc. for giving us an opportunity to work closely with their team and use their facilities. We would also like to thank NSERC for funding this research with an engage grant.

REFERENCES

[1] K. Petersen, C. Wohlin and D. Baca, The waterfall model in large-scale development, *Lecture Notes in Business Information Processing*, 2009.

[2] *Manifesto for Agile Software Development*, http://www.agilemanifesto.org/.

[3] I. Lazar, S. Motogna and B. Parv, Behaviour-driven development of foundational UML components, *Proc. of the 7th International Workshop on Formal Engineering Approaches to Software Components and Architectures (FESCA 2010)*, 2010.

[4] D. Dahlby, Applying agile methods to embedded systems development, *Journal of Software*, vol.41, pp.101-123, 2004.

[5] *Accelerating Embedded Software Development via Agile Technologies*, Technology Institute, 2013.

[6] K. Beck, *Test-Driven Development: By Example*, Addison-Wesley, 2003.

[7] M. Alhaj, G. Arbez and L. Peyton, Using behaviour-driven development with hardware-software co-design for autonomous load management, *The 8th International Conference on Information and Communication Systems (ICICS)*, Irbid, Jordan, 2017.

[8] M. Cohn, *User Stories Applied: For Agile Software Development*, Addison-Wesley, 2009.

[9] K. Petersen, C. Wohlin and D. Baca, The waterfall model in large-scale development, *International Conference on Product-Focused Software Process Improvement*, 2009.

[10] L. Cimasoni, *The Use of Methodologies for the Development of IT Projects*, Bachelor Thesis, Business Informatics, University of Fribourg, Fribourg, Switzerland, 2009.

[11] M. C. Paulk, B. Curtis, M. B. Chrissis and C. V. Weber, Capability maturity model, version 1.1, *IEEE Software*, vol.10, no.4, 1993.

[12] P. Kruchten, *The Rational Unified Process: An Introduction*, Addison-Wesley Professional, 2004.

[13] K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2004.

[14] K. Schwaber, SCRUM development process, in *Business Object Design and Implementation*, 1997.

[15] D. Astels, *Test Driven Development: A Practical Guide*, Prentice Hall Professional, 2003.

[16] D. North, *Introducing BDD*, http://dannorth.net/introducing-bdd/.

[17] J. Teich, *Hardware/Software Codesign: The Past, the Present, and Predicting the Future*, 2011.

[18] T.-Y. Lee, P.-A. Hsiung and S.-J. Chen, A case study in hardware-software codesign of distributed systems – Vehicle parking management system, *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, NV, USA, 1999.

[19] R. Joost and R. Salomon, Hardware-software co-design in practice: A case study in image processing, *The 32nd Annual Conference on IEEE Industrial Electronics*, DOI: 10.1109/IECON.2006.347790, Paris, France, 2006.

[20] J. Parri, J.-M. Desmarais, D. Shapiro, M. Bolic and V. Groza, A case study on hardware/software codesign in embedded artificial neural networks, *Applied Computational Intelligence in Engineering and Information Technology*, vol.1, pp.225-237, 2012.

[21] S. Rolland and G. Glania, *Hybrid Mini-Grids for Rural Electrification: Lessons Learned*, 2011.

[22] D. Li, S. Poshtkouhi, O. Trescases, R. Orr and B. Bacque, *Intelligent AC Distribution Panel for Real-Time Load Analysis and Control in Small-Scale Power Grids with Distributed Generation*, Ottawa, 2015.

[23] B. Bacque, T. K. Gachovska, R. Orr, N. Radimov, D. K. Li, S. Poshtkouhi and O. Trescases, Solving the last mile problem for energy self-forming nano-grids, *IEEE Canada International Humanitarian Technology Conference (IHTC2015)*, Ottawa, 2015.

[24] D. Dong, I. Cvetkovic, D. Boroyevich, W. Zhang, R. Wang and P. Mattavelli, Grid-interface bidirectional converter for residential DC distribution systems – Part One: High-density two-stage topology, *IEEE Trans. Power Electronics*, vol.28, no.4, 2013.

[25] J. K. Schonberger, *Distributed Control of a Nanogrid Using DC Bus Signalling*, Ph.D. Thesis, University of Canterbury, 2006.

[26] J. K. Schonberger, R. Duke and S. D. Round, DC-bus signaling: A distributed control strategy for a hybrid renewable nanogrid, *IEEE Trans. Industrial Electronics*, vol.53, no.5, 2006.

[27] W. Dalbon, M. Roscia and D. Zaninelli, Hybrid photovoltaic system control for enhancing sustainable energy, *Power Engineering Society Summer Meeting*, Chicage, USA, 2002.

[28] P. Fraternali, S. Comai, A. Bozzon and G. T. Carughi, Engineering rich Internet applications with a model-driven approach, *ACM Trans. the Web (TWEB)*, vol.4, no.2, 2010.

[29] T. Vandoorn, J. D. Kooning, B. Meersman and L. Vandevelde, Review of primary control strategies for islanded microgrids with power-electronic interfaces, *Renewable and Sustainable Energy Reviews*, vol.19, pp.613-628, 2013.

[30] B. Nordman, K. Christensen and A. Meier, Think globally, distribute power locally: The promise of nanogrids, *Computer*, vol.45, no.9, pp.89-91, 2012.

[31] *BeagleBoard.org – Black*, http://beagleboard.org/BLACK, [Accessed 1 January 2016].

[32] *Laboratory Virtual Instrument Engineering Workbench (LabView)*, https://www.ni.com/en-lb/shop /labview.html, [Accessed 1 January 2016].