The place of software engineering in IoT

Yukta Thapliyal   
*Gina Cody School of Engineering and Computer Science*  
*Concordia University*Montreal, Canada  
[yuktathapliyal@gmail.com](mailto:yuktathapliyal@gmail.com)  
40137818

Lina Jiang  
*Gina Cody School of Engineering and Computer Science*  
*Concordia University*Montreal, Canada  
jianglina0914@gmail.com  
40072493Xu Chen   
*Gina Cody School of Engineering and Computer Science*  
*Concordia University*Montreal, Canada  
xuchenca@outlook.com  
40089697

Taranpreet Singh Bhatia  
*Gina Cody School of Engineering and Computer Science*  
*Concordia University*Montreal, Canada  
taranpreetsinghbhatia@gmail.com  
40127879Shujun Ge   
*Gina Cody School of Engineering and Computer Science*  
*Concordia University*Montreal, Canada  
geshujunoffice@gmail.com  
40078931

Yan Feng  
*Gina Cody School of Engineering and Computer Science*  
*Concordia University*Montreal, Canada  
fengyan20189@163.com  
40105416

*Abstract*— The topic “Place of Software Engineering in the Internet of Things” presents a tremendously wide spectrum; starting from adapting the existing software platforms to new ones for the emerging field of IoT. Consequently, it becomes imperative to look into divergent areas to form a better understanding. IoT is an emerging field, therefore currently there is a lack of defined set of methods and architectures for reference. Further one of the major blockade to evolution and development of IoT is that presently in the market, there is a lack of IoT skilled IoT software engineers. And in addition, there is a huge presence of vendor-specific applications for IoT which limits the outreach of differing IoT devices and software to connect with each other, and thereby stopping IoT to tap its maximum potential. In this paper, we have explored challenges in employing software engineering practices within the Internet of Things framework, and traversed through different software engineering methods which influence the performance of Internet of Things.

Keywords—Agile Development, Model Driven Architecture, Zachman’s Framework

# Introduction

Nineteen years into the new millennium and we are working our way to connect the world- to provide a seemingly seamless and efficient style of transmission of information. Internet of Things or IoT, the trending technology, enables multiple devices to interact amongst themselves, collect and transport data remotely while maintaining a unique identity. It is due to this advantage, that today IoT boasts a huge network of over 17 billion connected devices and is further projected to grow [1]. IoT has integrated itself within our lives with such finesse that most people utilize it in their day-to-day lives, from the smartwatches that keeps count of one’s daily steps, pulse rate, etc. to the Smart TV that can monitor different connected devices within the home such as a washing machine and even display if the washing cycle has been completed or not!

IoT enables "things", which can be devices or systems, to connect through a software infrastructure, and communicate amongst themselves to create aggregated data for specialized tasks. For example, a remote sensing and monitoring system in medical sphere can send vital parameters such as pulse rate, blood pressure and ECG (electrocardiogram) over the internet to doctors and physicians in real time. It helps in situations where efficient medical aid is central. Such a remote healthcare system will consist of a hardware module which will have various sensors to collect the required medical data. These sensors, which are heterogeneous “things”, will need to be able to convey the medical data to a database server. Then the server will ultimately deliver data to the medical professionals who are able to see the results on a displaying module, like PC or mobile phone. For this operation, the sensors will need to be configured and interfaced with a software platform in order to interact with the database server. Further, the semantics between the server and sensors should use a common concept to enable cooperation, so the server understands the received information in a meaningful way or the sensors can send data in a way that the server can decipher. Another thing to take into consideration is the handling of data by the server, which can involve classification and unique identification of different stream of incoming data. Additionally, the system needs to take into account of the final destination of the data, which is ultimately the displaying module. The displaying module will require a UI (User Interface), which can either be a software application or web page dedicated to its cause. This UI needs be updated in real time with respect to incoming data from the server and provide a comfortable and efficient design for the end user, i.e. the doctors and physicians in this case. Finally, non-functional requirements like security, efficiency, performance and scalability of the system cannot be ignored. Therefore, software engineering will be employed right from the beginning towards the end in IoT. However, the paper will delve into limited depth within some of the functions.

One of the scope defined for the topic is the exploration of studies involving efficient software development methods that can handle the issue of optimizing energy in IoT systems. As IoT powers billions of devices over the world, there is a need for energy transparency in software development to prevent software from being a burden on hardware. Another area that has been identified in the review process is the adoption of good software engineering mechanisms in IoT. Namely Agile development methods and Model-driven Engineering has been looked into. Model-driven engineering has been identified to simplify the IoT software development process by providing abstractions in nodes and platforms and thereby enabling an IoT network to avoid unnecessary interactions. Also, a model-driven architecture can widen interoperability in IoT. While Agile methods and their relation to IoT was looked into simply because today’s business environment demands elasticity in infrastructure. Things of an IoT network which are usually deeply embedded devices need to be updated and upgraded as per the needs of the market, therefore, it is a point of interest to discover how Agile fits in IoT.

Another important aspect is security which is highly critical in IoT systems as a large number of sensitive data is shared online for processing. Existing security models do not match up to the varying and complicated demands of IoT. The Paper review section also considers user interface for IoT. Further, several other reference material has been looked into to find customer satisfaction, costs, organizational aspects, social implications and other determinants that are associated with the place of software engineering in the Internet of Things.

# Paper Review

## Software Engineering For The Internet Of Things

The easiest way to get a general IoT-oriented software engineering approach is to analyze the common features of existing proposals and application scenarios around IoTs. Then identify the key software engineering abstractions for IoTs by these common characteristics. [2]

From the beginning of the IoT emergence, its construction has challenged traditional software development. Traditional software development cannot adapt to the rapid development of IoT, so a new generation of software development methods is proposed. Rebeca C. Motta, Káthia M. de Oliveira, and Guilherme H.Travassos discuss Zachman’s Framework as an alternative architecture to support systems engineering of IoT applications. [3] Zachman’s Framework provides a holistic view of the enterprise architecture. Zachman’s Framework is supported by many modeling tools and languages. The framework is suitable for complex systems, defines and controls interfaces with some logical constructs and integrates all system components together. It is also used to evaluate the development process, requirements engineering, and business process modeling. The most important advantage of Zachman’s Framework is that it can integrate and align IT architecture and business goals. It organizes architecture elements and structures the information in a matrix. Its flexibility ensures that it can be used as a basic framework for building IoT software engineering.

The Zachman Framework is a fundamental structure for enterprise architecture. When it is used to IoT software system, it provides a formal and structured view of IoT software system. Zachman Framework is a kind of matrix which includes the perspectives in the rows and organizing fundamental questions in the columns. In the rows, the perspectives include in several software systems activities, such as executive perspective, business perspective, architect perspective, engineer perspective, technician perspective, user perspective and so on. In the columns, the organizing fundamental questions consist what, how, where, when, who, et.

However, as the application of Zachman’s framework has brought many advantages to the IoT software system, the development of IoT is still facing many challenges. There are many areas where Zachman’s Architecture falls short. Some of them are:

1) Architecture: Zachman’s framework is good, but not enough. Looking for a scalable, flexible, secure and cost-efficient architecture to suit the complex IoT is a challenge.

2) Data: The biggest challenge for data in the IoT is how to use data in any IoT environment. Data analysis is much harder than creation.

3) Interoperability: When different systems, software and things to interact for a purpose, standards and protocols might include issues.

4) Management: IoT includes various types of network connections. It is very complex, so management activities which include planning, monitoring and controlling are hard to do well.

5) Network: This is kind of a technical challenge, it relates to communication technologies, routing, access and addressing schemes.

6) Security: Security is the core issue of the IoT. It includes confidentiality, integrity, authentication, authorization and so on.

7) Social: The development of IoT needs to concern the human end-user, make them understand the situation and applications.

The above challenges are common problems in software engineering in IoT. The solutions of some problems is still unknown, such us security and social. These problems are complex and cannot be solved by relying on a single system structure. Zachman’s framework is a flexible, useful framework structure, which can be filled according to the actual requirements. It can help to deal with many different problems, according to what kind of requirements to be filled in. However, Zachman’s framework is just a basic structure, it need to be used with other good models, like agile model, to deal with these above challenges. [3]

## Energy Efficient Software Engineering In Iot

The term energy efficiency in IoT refers to the use of software engineering methods to have energy transparency, to display the program’s energy consumption in software development and avoid software being a burden on the hardware. In IoT, the ‘things’ that are employed are hardware devices which are controlled by the software. Often the fact that software is forcing the hardware to consume a fair amount of energy is identified in a later stage, either during testing or in extreme cases it can even be after the project has been made live. If it is identified that software is exploiting the hardware during testing, it can lead to incurring additional costs and can delay the delivery of the product. However, if the problem is identified after the product has been made available for use, it can mean severe consequences. In the worst-case scenario, with time the hardware can start deteriorating or even malfunction due to over-exertion by software. If this happens in a mission-critical system, the consequences can be colossal. Therefore, it becomes essential that incompetent software does not waste the energy of competently built hardware. To get an idea of energy estimations exerted by software in IoT systems, three main techniques are widely popular: 1. Profile-based, 2. SRA based and, 3. Component repository

Since programmers have limited knowledge of which part of the program is consuming the most energy and how much energy the overall program is costing, having tools that give feedback about the energy cost can be the very first step in energy transparency. However, presently in the market, there is a lack of defined tool or set of practices for energy transparency. One of the reasons for this problem is the presence of various layers in system stack, causing struggle in realization as to how the program affects the system. Additionally, it is the lower-level components in the stack that deal with hardware and, software developers generally work on a higher level of stack with limited knowledge of sophisticated hardware equipment. To look into the solution for this problem, one way is to identify energy consumption at three main levels: 1. source code, 2. Compiler’s Intermediate Representation (IR), and 3. Instruction Set Architecture (ISA) [1]. One of the methods to do so is profile based energy consumption which involves collecting execution statistics and thereby utilizing the data to understand how software at different levels is utilizing energy [4]. It can involve the introduction of Performance Monitoring Counters (PMCs) on events to get an idea of how a particular event is using power and get an estimation of like methods. The profile-based method also involves Code Instrumentation which extracts execution statistics at different levels like the compiler’s levels. Also, a factor to take into consideration while doing finding energy estimations is that at the source code, the program has the most information however the energy modelling suffers greater precision losses. Similarly at ISA, the program losses are profound while energy precision losses are less.

Another technique to get energy consumption estimation is based on Statistics Resource Analysis or SRA. One of the methods involved in this technique is to determine the energy in terms of program input size at ISA and compiler’s IR level which is fairly easier than profile based estimation [4]. Further, another advantage is that this technique does not depend on the programming language. K. Georgiou, S. Xavier-de-Souza and K. Eder in their paper, acknowledge SRA as a superior method in comparison to profile based technique in terms of efficiency and ease [1].

As IoT is an emerging field that does not have a set of defined practices, different software practices and architectures are currently being deployed in IoT. If there existed a reference architectures for IoT, it would become easier to take software parts for re-use. However, in the absence of such an architecture of a set of popular architectures, having an Energy-based Component Repository (ECR) that manages the energy information of different components can be useful [4]. But, one also needs to keep in mind that energy is consumed differently depending on processes and communication mechanisms. Therefore, it is not enough to have components within the repository but also necessary to compare them within identical software architectures of IoT services. D. Kim, J. Choi and J. Hong discuss this method in detail in their paper. They use Energy consumption of three different components with same functionality based on two methods: 1. Architectural-based evaluation and 2. Code-based evaluation to find the better component and decide which should be used in the IoT network. Calculating component energy can be done while considering, 1. Component energy and 2. Internal behaviour of component. The basic idea of component repository is that a particular component that utilizes less energy when configured a certain way with the software can be reused in different IoT systems who are looking to work in a similar domain. For example, an IoT system needs to be designed for remote healthcare monitoring and there exists a component repository that has information on pulse sensors that can be configured in certain ways to consume different units of energy. Now based on the repository, different pulse sensors can be selected and compared with their specification and the best ones can be selected which fulfils the remote monitoring system’s functionality.

## Agility in IoT

Agile methods are widely used today for software development activities and it can prove to be insightful how these can be adapted and implemented in IoT, specifically in industrial IoT solutions. To fit agile methods into IoT, we need to consider the internal structure of the team working on the IoT solution, as well as the management. An efficient and easy way to find the structure of the team is to conduct structured interviews with managers, team leaders, and project members [5].

F. Christoph and H. Thomas show that Scum, which is a popular agile method in software engineering, can be efficiently utilized in industrial IoT solutions. In their ongoing research to discover the adaptation of agile methods for obtaining solutions in industrial IoT, they conducted research in companies ranging in revenues from 3 – 71 billion Euros. Their methodology was to collect data was interviews with project members, team leaders, and managers who are actively involved in delivering IoT products at each company. The result for one of the case studies concluded that Scum was well adopted in the IoT development teams, both in hardware and software development.

Another industrial example of Agile IoT is in automation infrastructure looked into by L. Russell, R. Goubran, F. Kwamena, and F. Knoefel. [6] They aim to provide increased resiliency in critical infrastructure by using agile IoT redeployment. To change existing sensors ordinarily deployed, signal processing middlewares were employed. Signal processing middleware is a type of middleware that lies between the application and the underlying operating system, network protocol and hardware. This middleware is mainly used to overcome the gap between hardware and software infrastructure. Existing sensors are made or repurposed to the agile way of communication between hardware and software. Sensors deployed can detect the signals, process them and send them to the software for analysis which is the one-side communication and analyzed information sent back to the sensors to make hardware react to the stimuli for parallel communication.

For manufacturing, the studies conducted show that the design follows a layered pattern. [6] The layers are outlined as functioning teams managing the IoT infrastructure at the bottom layer, i.e. devices and networks, and also the IoT applications which are running on top. In between both of these layers, the management of applications within the IoT infrastructure is done where the configuration and execution of various application of IoT takes place. In IoT enabled automation, the production is realized in the hierarchical structure. In an agile manufacturing environment, this structure changes with the addition of new components. This change can lead to new bus topology with changed variables and modification in programming to provide new functionalities for the process to be controlled. This is due to the software being specialized in a fixed production facility and its corresponding processes. For Example, to monitor the fridge open-close event, one way could be the use of mechanical switches and wiring systems in a minimal possible way. This is a low-cost monitoring system built by using inexpensive sensors. However, the alternative to this is by using the temperature sensor to the outside of the fridge to monitor the surrounding air and analyse the air temperature to conclude if the fridge is open or close.

## Model Driven Software Engineering Approach in IoT

Due to the high heterogeneity and fast iteration of the IoT nodes, efforts are made to free the developers from adapting endless new hardware platforms. The model-driven approach, though in its early stage, reduces the development overhead and gives wider interoperability by providing high-level abstraction of applications. In general, a set of developing tools and methodologies are used to create the models and generate the executable codes for different platforms automatically or with little changes.

However, because of the limited hardware resources, it’s usually hard to adapt high-level abstraction to the sensor nodes. [7] A sensor node is a self-contained entity that can collect, process and transfer data, usually deployed in large quantity and made up with microcontroller as its low cost. The developers have to determine the detailed hardware and software requirements at a low level, making it hard to reuse and adapt to the new system. The FRASAD framework with MDA (Model-Driven Architecture) approach, is developed by Xuan Thang Nguyen’s team to solve this and improve the flexibility, compatibility, and maintainability in the sensor node domain. [7]

The traditional WSN (Wireless Sensor Network) typically includes the Operating System Layer, the Hardware Abstraction Layer, and the Hardware Layer. In this FRASAD approach, two new upper layers: the Application Layer and the Operating System Abstraction Layer are introduced for higher abstraction as well as hiding the lower platform. A rule-based application model described by Domain Specific Language (DSL)，which uncouples the application and the operating system, is developed for automatically generating the platform specific executable code. Each rule executed periodically is composed of select, processing and action clauses. [7]

The framework is then evaluated in the simplicity of usage and portability through a user study compared with two other approaches and generating code for different platforms. The result turns out that FRASAD is easier in use and codes can be generated in most of the test cases automatically or with slight changes in source code. [7] Despite its ease and quickness to develop in comparison to using native language on specific platforms, such as nesC language on TinyOS, it can only be applied to devices running on an operating system, which usually gives more functionalities at the cost of power. More abstraction levels mean there is additional effort in adapting the operating system.

Model driven approach in energy validation is also an interesting area looked into by P. Iyenghar and E. Pulvermueller. [8] They looked into a model-driven workflow for early energy and timing validation and argue that it is capable of providing better software design options. They contest that as currently most of the software in IoT area is based on hand-written code, higher abstraction is required for time and energy evaluation. Their workflow involves four steps: 1.Reversing hand-written C or C++ code to obtain the UML design model with higher abstraction through widely used UML tools such as IBM Rational Rhapsody Developer. [9] This step is also called reverse engineering and UML element such as UML classes and sequence diagrams are generated. 2.Specifying performance requirements timing and energy with UML design model. 3.Generating the timing-energy model based on the design model through the Model-to-model transformations supported by Eclipse Modelling Framework. 4.Validation with SymTA/S tool for the feedback. SymTA/S is a timing analysis tools capable of evaluating the time used in each task and energy consumption.

An IoT-enabled use case of emission monitoring is used for evaluating the workflow. [8] The model generated the result of three modes in total power usage and scheduling of tasks, with the comparison of them, the developers can decide the better one. It turns out the workflow enables early feedback on performance and will help in optimizing the code with reduced development overhead. However, this workflow is not widely tested in other platforms. In the developing period for new product there are less likely codes ready for analyzing, even the hardware platform may remain to be decided according to the result of energy analysis, thus this workflow is more suitable for existing product with the purpose of optimization.

Model driven approach in industry for IoT is still in the academic realm, however there is an effort to apply them to industry for solving the distribution and heterogeneity in IoT. [10]

The ThingML approach is trying to handle the abstraction of heterogeneous nodes and platforms, resource constraints, as well as avoiding unnecessary interactions. It consists of a modelling language, tools and a methodology for specifying the platform-independent logic, warping existing libraries, processing events, handling dynamically discovered sensors and generating code for multiple platforms in a different language. The code generator is available for three languages (Java, C/C++, JavaScript) and even more available hardware platforms. This means the developers can develop platform-independent code and implement them according to the hardware price and required functionality, which gives more flexibility and interoperability is also solved.

An important category in the IoT system is Mission-Critical IoT Systems (MC-IoT). For a mission-critical system to run, it becomes imperative for the systems to have high availability, reliability, safety & security, regulatory compliance, scalability, and serviceability. The running failure of the MC-IoT system can have serious consequences. Model-driven engineering (MDE) has been found to have a potential in the MC-IoT system. Automated mechanisms of MDE can ensure the self-adaptation of distributed MC-IoT systems. Reusability, another property of MDE, ensures sustainable development of the MC-IoT system. [11]

The major software difficulty in MC-IoT relates to its lack of standardized platform which leads to difficulty in cross-platform development. Also, if the nature of the MC-IoT system is dynamic, i.e. it changes based on available resources, unpredictable contextual information, etc., then inaccurate information may cause the IoT system to malfunction which can definitely lead to much harm. Therefore, there is a requirement for MC-IoT systems to have dynamic discoverability of available resources, i.e MC-IoT systems should have a mechanism to discover available resources and constraints. But, it is not only enough to discover constraints but also to respond accurately to these resources at any event. Therefore, software engineering within the sphere of MC-IoT systems should be carried out in such a way that these problems are efficiently tackled and it has been found that Model-Driven Engineering can be a good solution for the challenges in MC-IoT systems. [11]

In the instance of E-Health Fall Detection System developed by a company called Tellu, more than 25 components are implemented with ThingML, which helps in solving requirements such as distribution, heterogeneity, and reliability. [10]

## User Interface for IoT

Although IoT technology continues to receive attention in the areas of cities, services, industry, and commerce, there still exists many issues that need to be solved. Popular research on the Internet of Things focuses on how to design its main framework and how to analyze and process large amounts of data but lacks attention to the user interface of the Internet of Things technology. But user interaction interfaces often play a significant role in large software and systems. The Internet of Things communicates by exchanging information between intelligent devices, but its ultimate purpose is to provide value to humans. And this value can only be reflected in the user-friendly interface through user interaction with visual charts and information. Therefore, designing a friendly user interface has become a very important part of the design of IoT systems.

Brambilla, Marco & Umuhoza, Eric & Acerbis, Roberto propose the IoT system modelling composed of mainly content models and Interaction models. [12] Content model which implements IoT services by defining the concepts that users, organizational structures, and their customers need and the concepts they need to define. Interaction model which includes IoT-specific operations and events from IoT devices. Because of the design and implementation of the user interface using a model-driven design, almost all of the main behavior of the system can be overridden and included in one model. If the technical architecture needs to be changed, it means a new transformation of the model. Because the main code is almost always generated automatically, which greatly reduces the time the designer writes the code, the designer has more time to focus on how to solve the business problem. The system can better meet the needs of users.

Further, Interactive Flow Modelling Language (IFML) which is a fairly new modelling language (developed in 2012-2013) can be taken advantage of to create a graphical user interface in IoT. Although model-driven software has a lot of advantages such as heterogeneity, large-scale and emergent properties, context awareness and uncertainty, dynamic discoverability of resources, security, and trust, it will also meet a lot of challenges. The main challenge is that we can not find the most appropriate example. For instance, when people read the specification on how to execute a specific task, such as designing a highly available solution, there is no model-driven software development content in the document. This means that there is no standardized models are available, so in the future, this is an issue we urgently need to solve..

## Security in IoT

With the development scale of IoT, the system currently has a variety of devices that are widely distributed. This, however, exposes IoT towards a bigger security risk than any other network. Integrating different kinds of devices in software engineering further magnifies the security issue in IoTs. The developers are also at risk of losing control over the evolution of reused IoTs system elements. This can undoubtedly increase costs and waste resources. This complicated and changeful security situation of the IoT renders the traditional security architecture incapable of fully meeting the IoT security requirements. Exploring and establishing an active effective security defence approach based on the features of the IoTs is the only way out. One way is to design a dynamic defence frame, referring to traditional security models. [13]

A feasible approach to security in IoT starts with security threat detection, which leads to danger computation, followed by security response, security defence strategy formulation and security defence. This acts as a defence circle to protect the core code. All original data from IoT traffic needs to be detected in the first step. Once a problem occurs in a place, the subsequent steps must be adjusted accordingly.

The experiment conducted by Caiming Liu, Yan Zhang and Huaqiang Zhang [13] demonstrates the possibility of the architecture, which is a dynamic defence frame by applying principles in immunology into the key links. This study simulated its performance after cloning attacks, mutated cloning attacks, replay attacks, and mutated replay attacks. The result shows that the system kept safe for three hours to collect initial data and adopted defence measures.

The IoT environment is dynamic and is very different from the traditional static computer environment. This feature has a major impact on the entire software development process, especially in security testing. Although the past software engineering methods are no longer fully applicable to the IoTs, they still have reference significance. From the method proposed in this article, we can see that the security method for the IoTs is still based on software engineering, including some other ideas. However, popular software engineering methods, like agile development, are difficult to take into account for security issues.

# Discussion

After going through the extensive literature for the paper, four main areas in the IoT platform for software development and implementation that need attention have been identified:

1) Lack of standardised architecture or set of methods for reference.

2) Absence of trained software engineers.

3) Conflicts within approach.

4) Widespread presence of vendor specific application.

## Lack of Standardised Architecture or Set of Methods for Reference

Although the Internet of Things (IoT) is developing rapidly, it is still an emerging industry compared to other Computer fields. Additionally, early research in the IoT system focused mainly on communication or operational issues, ignoring the development of a common unified approach for the whole IoT environment. This situation will adversely affect the design of large and complex IoT systems in the future and limit the promotion of other IoT systems. Although some current works of literature present the development frameworks for the IoTs with corresponding guidelines, they have no general applicability because they are designed for very specific functions lacking general abstractions.

For example, in energy estimation methods there are three approaches proposed by different researchers discussed here: 1.Profile Based method, 2. SRA based method and 3. Repository architecture. While Profile based approach can prove to be quite useful and accurate as we are getting energy information directly from performed/simulated tasks, on the contrary it can also be a tedious way to collect information as that numerous events and tasks are carried out in an IoT system. To assess each event or to introduce checks after them can prove to be impractical in most cases.

In case of SRA, determining energy estimations based solely on input size can be a crude method. Also, at ISA and compiler’s IR, it is difficult to capture energy in terms of program-input size. In addition, SRA techniques can be quite a challenge for multi-threaded and multi-core programs.

And for component repository method, one of its main disadvantage is the need for inclusive scenarios of component behaviour. If an unknown behaviour comes up, the energy levels might not be adjusted for the behaviour and cause the system damage. For example, the pulse sensor used in the remote health monitoring system encounters an unexpected signal for which the component repository did not provide any information and the system is left unprepared to meet the signal’s energy demands. Another downside to having a component repository is maintaining a huge amount of data. The data will need to be constantly updated when new information is found for a certain component. Further, it can become problematic on who has access to change the component repository. Further, if the access to the component repository is kept fairly open, there may be an increased risk of infiltration of erroneous data. Also, the concept of component repository has the potential to be successful if companies and individuals who deal with IoT agree to share their findings on a common platform, which can be quite a difficult task to achieve. However on the other hand, if such a component repository can be maintained and the method becomes popular, it can save IoT solution companies considerable resources such as time and money and can also enhance the quality of developed systems as the repository will have tested and approved components.

Here we see that in absence of defined set of architecture and designs, a lot of consideration goes in determining the energy estimation, which is only one aspect of IoT system. And in addition to methods mentioned here, there exists other methods proposed by different individuals in this field of energy estimation for IoT. Which means that there is a lack of standardised methods for reference.

## Absence of IoT Trained Software Engineers

A general IoT-oriented software engineering approach is difficult to develop at this stage because at present the development time of the IoTs has been short and the technology is still immature. Therefore, some developers use past experience and software engineering methods that are not IoT-oriented to develop the IoTs incorrectly or only develop special IoT applications. Also, there exists various layers in an IoT systems and further there exists various abstract layers in programming. The developers generally work on the highest layers, close to source code at the lowest layers. While at the other end of the IoT systems, the things that are employed are hardware components. And most of the times, the developers do not understand the configuration of these devices things for which they are writing program for. And since the IoT system in question can include varying type of devices, it becomes impossible for software developers to be aware about all devices. These issues are all obstacles to determine set of training standards for future IoT software engineers.

## Conflicts within approach

Facing the diverse and unfamiliar systems of IoT, many developers cannot give clear solutions, there still exists some controversy in providing absolute requirements and solutions. Some might think that agile processes can help in dealing with these problems. While others might argue that there is a need to extract inspiration from existing research, and develop a new generation of development environment by using past and new software engineering techniques.

In IoT, agile methodology can be increasingly used for planning the resources and to assign work for the team in the software development process. In the process of acquiring the resources for IoT systems, it can prove to be useful to employ agility as IoT is an emerging field and requires some flexibility to deal with changes as new findings can come up in later stages. Presently there exists a dearth of highly skilled developers for IoT and simple platforms for IoT development in the market. Therefore in the absence of a common tool, agile methods can prove to be useful to develop them.

However, agile methods to IoT do not come with only advantages. IoT is an emerging field, plan-driven approach might be required initially to have good documentation for future use and for record-keeping to identify problematic and successful areas. Also, since agile methods are incremental, it might not be suitable for IoT where at the final end hardware components are deployed since hardware does not have flexibility like the software.

Some of the developers have also been adding in efforts in developing the approach and tools to IoT with Model driven approach. However, it is not estimated whether the model-driven approach is worth. The more platforms to be implemented, the more developing overheads will be reduced by applying this approach. With continuous tool developing, the maintenance of legacy system will be easier with lower costs. Currently the software support for legacy products only last several years and extend the time will enhance the competitiveness and reputation for a company. The model driven approach gives the possibility of longer product life time, but the benefits remain to be evaluated with costs. Be as that may be, we can also see that the model-driven approach is still at its early stage. In most of the papers collected, majority of authors built their entire toolchain from almost zero, the only commonly used tool being the UML tool. And if searched by the approach name, authors maintained their own approach, that is to say, there is no popular toolchain for model-driven approach.

While MDA has been underpinned as a revolutionary approach by some, it being a field that is itself underway remains a dissuading factor for its most of its proponents. Another deterring factor to its realisation is that hardware manufacturers are less likely to build a universal toolchain for a model-driven approach, which is a threat to their own products. And it’s really hard for individual developers to optimize the tool for several platforms. The cloud service providers may have the motive to develop such a kit to attract more developers but the embedded systems, which comprise a large proportion of the IoT network, are really far from them in technology. Though the heterogeneity of IoT devices in one company can be solved by a model-driven approach in the future, big companies build their own standard and protocol for achieving more users.

The trend of language and hardware platform is always changing and such a high-level abstraction approach frees most developers from learning endless platform-specific knowledge and speeds up the development once the development kit is ready. The problem is that the translator requires too much development overhead and the developing environment is still on its early stage.

## Presence of vendor specific applications

While conducting our literature review, one of the main challenges was to find appropriate studies that could be generalised for understanding. The reason being the presence of vendor specific application that have provided solutions for a specified system. Vendor specific application hinders the scope of interoperability. It does not allow for open source platforms or other widely accessible tools for open source programmers to develop and evolve IoT. Further, vendor specific applications in IoT create a close knit community and again can be linked to all other problems of untrained IoT software engineers and lack of defined set of methods for reference.

# Conclusions

For the evolution of IoT in future, there needs to be a focus on building a common unified approach for IoT to solve most of its problems. Per contra, the dynamic environment of IoT and the central unanswered question on what is the specific trend in IoT, makes this reality difficult to establish. One of the way to solve this complication would be to use existing software methods and adapt them appropriately based on requirement and document them simultaneously for future use. Another would be to promote open source projects or platforms so that various minds could work on rapid development of IoT systems. However, since IoT deals with immense data and sometimes sensitive data as well, maintaining the integrity of information in case of open development can be a colossal issue.

Over and above, it is essential to train software engineers in field of IoT to take better considerations for embedded technology, which makes up crucial part of the network. Most of the time software engineers for IoT are unable to comprehend the needs of embedded system and how their program will affect the hardware in the long run. It is not a matter of how proficiently the software has been built, but a matter of how well it is suitable for the hardware in present and with prolonged use as well.

##### References

1. K. Georgiou, S. Xavier-de-Souza and K. Eder, “The IoT energy challenge: A software perspective,” IEEE, vol. 10, no. 3, pp. 53-56, Sep, 2018.
2. F. Zambonelli, “Towards a discipline of IoT-Oriented software engineering,” in CEUR Workshop Proceedings, 2016, pp. 1-7.
3. Rebeca C. Motta, Káthia M. de Oliveira, Guilherme H.Travassos. “ On Challenges in Engineering IoT Software Systems.” SBES 2018, Sao Carlos, Brazil, September 17–21, 2018.
4. D. Kim, J. Choi and J. Hong, “Evaluating energy efficiency of Internet of Things software architecture based on reusable software components,” International Journal of Distributed Sensor Networks, vol. 13, no. 1, pp. 18, Jan, 2017.
5. F. Christoph and H. Thomas, “Adapting Agile Methods to Develop Solutions for the Industrial Internet of Things,” in Proceedings of the 25th European Conference on Information Systems(ECIS), Guimaraes, Portugal, pp. 2852-2861, Jun 5-10, 2017.
6. L. Russell, R. Goubran, F. Kwamena, and F. Knoefel, “Agile IoT for Critical Infrastructure Resilience: Cross-Modal Sensing as Part of a Situational Awareness Approach,” IEEE, vol. 5, no. 6, pp. 4454-4465, Dec, 2018.
7. X. T. Nguyen, H. T. Tran, H. Baraki and K. Geihs, “FRASAD: A framework for model-driven IoT Application Development,” Internet of Things (WF-IoT) 2015 IEEE 2nd World Forum on, pp. 387-392, 2015.
8. P. Iyenghar and E. Pulvermueller, “A model-driven workflow for energy-aware scheduling analysis of iot-enabled use cases,” IEEE Internet of Things Journal, vol. 5, no. 6, pp. 4914-4925, Dec, 2018.
9. IBM Rational Rhapsody Developer, Ver 8. Accessed: November 30,2019. [Online]. Available: https://www.ibm.com/developerworks/ downloads/r/rhapsodydeveloper/index.html
10. B. Morin, N. Harrand, F. Fleurey, “Model-based software engineering to tame the iot jungle,” IEEE Software, vol. 34, no. 1, pp. 30-36, 2017.
11. F. Ciccozzi, I. Crnkovic, D. D. Ruscio, et al., “Model-Driven Engineering for Mission-Critical IoT Systems,” IEEE Software, vol. 34, no.1, pp. 46-53, Jan, 2017.
12. Brambilla, Marco & Umuhoza, Eric & Acerbis, Roberto. (2017). Model-driven development of user interfaces for IoT systems via domain-specific components and patterns. Journal of Internet Services and Applications. 8. 10.1186/s13174-017-0064-1.
13. C. Liu, Y. Zhang, and H. Zhang, “A Novel Approach to IoT Security Based on Immunology,” in 2013 Ninth International Conference on Computational Intelligence and Security (CIS), Leshan, China, Dec, 14-15, 2013.