



Task Order 0250, RT 205:
Identifying and Measuring Modularity Violations on Cyber-Physical Systems
Final Report
Technical Report SERC-2019-TR-006

June 7, 2019

Principal Investigator: Dr. Lu Xiao, Stevens Institute of Technology
Co-Principal Investigator: Dr. Michael Pennock, Stevens Institute of Technology

Research Team:

Joana Cardoso, Stevens Institute of Technology
Xiao Wang, Stevens Institute of Technology

Sponsor: ODASD(SE)

Copyright © 2019 Stevens Institute of Technology, Systems Engineering Research Center

The Systems Engineering Research Center (SERC) is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology.

This material is based upon work supported, in whole or in part, by the U.S. Department of Defense through the Office of the Assistant Secretary of Defense for Research and Engineering (ASD(R&E)) under Contract [HQ0034-13-D-0004, TO#0250].

Any views, opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense nor ASD(R&E).

No Warranty.

This Stevens Institute of Technology and Systems Engineering Research Center Material is furnished on an “as-is” basis. Stevens Institute of Technology makes no warranties of any kind, either expressed or implied, as to any matter including, but not limited to, warranty of fitness for purpose or merchantability, exclusivity, or results obtained from use of the material. Stevens Institute of Technology does not make any warranty of any kind with respect to freedom from patent, trademark, or copyright infringement.

This material has been approved for public release and unlimited distribution.

TABLE OF CONTENTS

Table of Contents	iii
List of Figures	iv
List of (Tables, Sequences).....	vi
Executive Summary	1
1 Introduction.....	3
1.1 Research Questions	4
2 Background.....	5
2.1 Architecting Cyber-physical Systems.....	5
2.2 Detecting Modularity Violations in Software Systems.....	6
2.3 Detecting Modularity Violations in Cyber-Physical Systems	7
3 Approach	8
3.1 Data	8
3.2 Task 1: Explore alternative modular decompositions of a cyber-physical system	9
3.3 Task 2: Develop a Domain Concept Learner to Extract Hardware Related Concepts	10
3.4 Task 3: Build decision framework demonstrator	11
4 Results	12
4.1 Model Decomposition Results	13
4.1.1 Development View Decomposition	13
4.1.2 Dependency Hierarchy Decomposition	16
4.1.3 Organizational Structure Decomposition.....	17
4.1.4 Modular Structure Time-Series Analysis	18
4.2 Domain Concept Learner Results	20
4.2.1 Data Preparation and Exploration of NLP Techniques.....	21
4.2.2 Clustering Concepts.....	22
4.2.3 Refining the Model.....	24
4.2.4 Mapping Hardware/Software Concepts to Modules	25
4.2.5 Manual Validation	26
4.2.6 Testing the Domain Concept Learner using MD PnP	30
4.2.7 Investigating the Limitations of the Domain Concept Learner for MD PnP.....	31
4.3 Analyzing and Diagnosing Modularity Violations	33
4.3.1 Modularity Violations in OpenWrt.....	33
4.3.2 Modularity Violations in MD PnP	36
4.4 Modularity Violation Demonstrator.....	39
5 Discussion	45
6 Conclusions and Future Work	46
7 List of Publications Resulted	48
8 References	49

LIST OF FIGURES

Figure 1 – Schematic of decision framework demonstrator	12
Figure 2 - Root Package View (Left MD PnP, Right OpenWrt).....	14
Figure 3 – MD PnP Fine-grained Package View	15
Figure 4 - MD PnP Fine-grained Package View with Co-changes	15
Figure 5 - MD PnP Fine-grained Package View with Co-changes > 10	16
Figure 6 - Design Hierarchy View (Left: MD PnP, Right OpenWRT).....	16
Figure 7 - Design Hierarchy View with Co-change (Left: MD PnP, Right: OpenWrt).....	17
Figure 8 - Organizational View (Left: MD PnP, Right OpenWrt).....	18
Figure 9 - OpenWrt-VfC Trend for the Top 5 Vendors.....	20
Figure 10 - OpenWrt- CMC-IMC-Trend	20
Figure 11 - Top 60 words from query	22
Figure 12 - t-SNE plot of the trained word2vec model with 500 hidden nodes using cosine distance	23
Figure 13 - A sub sample of conceptual clusters from OpenWRT	24
Figure 14 - A subset of modules from the system architecture organized by clustering on term similarity from the word2vec model	25
Figure 15 - Change Message Score Distribution	27
Figure 16 - Score Distribution for Manual Tagged Change Messages.....	27
Figure 17 - Violin Chart for Change Message Scores in Different Groups.....	28
Figure 18 - File Name Score Distribution	29
Figure 19 - File Name Score Distribution of Manual Tagged File Names	29
Figure 20 - Violin Chart of File Name Scores in Different Groups	30
Figure 21 – Histogram of scores assigned MP PnP change messages manually classified as hardware related or software related	32
Figure 22 – Changes histories for OpenWRT from the development view perspective.....	35
Figure 23 – OpenWRT cross module change analysis	36
Figure 24 - OpenWRT cross module change in coarse development view (hardware on left; software on right)	37
Figure 25 - Snapshot of domain concept learner training interface	39
Figure 26 - Snapshot of domain concept clustering interface.....	40
Figure 27 - Snapshot of scoring results of commit messages using the domain concept learner	40

Figure 28 - Module Decomposer Component	41
Figure 29 - Module Decomposer Component-Development	42
Figure 30 - Module Decomposer Component-Hierarchical.....	43
Figure 31 - Module Decomposer Component-Organizational	43
Figure 32 - Co-change Calculator that Separate Hardware and Software Changes.....	44
Figure 33 - Modularity Violation Visualizer/Analyzer	44
Figure 34 - Examples of modular views generated by the demonstrator tool.....	45

LIST OF (TABLES, SEQUENCES)

Table 1 – Project Summary Data	8
Table 2 – Stable Versions of OpenWRT	19
Table 3 - Firmware_utils Hardware and Software Concepts.....	26
Table 4 – VfC scores for the development view of OpenWRT	34
Table 5 – VfC scores for the organizational view of OpenWRT	34
Table 6 – VfC scores for the hierarchical view of OpenWRT	34
Table 7 - Software and Hardware Triggered Cross Module Change in MD PnP.....	36
Table 8 - VfC scores for coarse development view in MD PnP.....	37
Table 9 - VfC scores for the fine-grained development view in MD PnP	37
Table 10 - VfC scores for the organizational view in MD PnP	38
Table 11 - VfC scores for the hierarchical view in MD PnP.....	38

EXECUTIVE SUMMARY

In recent years, the Department of Defense has emphasized modular and open approaches to system development to improve interoperability, facilitate system evolution and technology insertion, and foster competition. Requirements such as the Modular Open Systems Approach (MOSA) have been imposed on acquisition efforts. However, it can be difficult for the Government to assess whether the resulting architectures and systems are truly modular.

In this research task, RT-205, *Identifying and Measuring Modularity Violations on Cyber-Physical Systems*, we investigated modularity violations in cyber-physical systems. Cyber-physical systems are composed of diverse subsystems consisting of both physical and software components. Cyber-physical systems are more representative of the systems that the DoD acquires than a pure software system. Our approach combined the analysis of alternative architectural perspectives with the analysis of project version control systems and project documents, and machine learning techniques. We were able to identify modularity violations from different architectural perspectives that were driven by hardware or domain factors instead of pure software issues. To accomplish this, our research approach consisted of four complementary activities:

- 1) We investigated techniques to decompose the architecture of a cyber-physical system into modules using three different criteria for addressing different stakeholders' concerns.
- 2) We leveraged machine learning and natural language processing techniques to identify hardware related concepts that are responsible for triggering changes in cyber-physical systems.
- 3) We combined the previous two techniques to identify modularity violations triggered by hardware issues in different architectural views of the cyber-physical systems
- 4) We implemented a proof-of-concept demonstrator to show how stakeholders can perform modularity violation analysis in cyber-physical systems.

For test cases, we used two open source cyber-physical projects, OpenWrt and MD PnP. In OpenWRT, we were able to determine that the majority (78%) of system maintenance actions were associated with hardware concepts, such as firmware support, chip support, and hardware bugs. We were also able to detect potential modularity violations and assess the likely driving factors. For example, at one level of the system architecture, we found 16 potential modularity violations driven by hardware issues. Of those, 11 are explicitly related to adding support for new chip types. This result strongly suggests that, despite the modular architecture of OpenWRT, it is unable to fully contain cascading changes from the introduction of new chip types. Such cascading changes are one of the chief concerns of the DoD as it acquires, maintains, and upgrades a complex web of cyber-physical systems for the warfighter. In an ideal modular system, the introduction of a new mission component should not require reengineering the other components of the system. Of course, this ideal is never achieved, but the goal here is to enable DoD to get as close as possible while remaining cost-effective.

We made similar observations for MD PnP. The difference is that, relative to OpenWRT, more changes are triggered by software concepts (48%) than by hardware concepts (16%). Of the modularity violations triggered by hardware concepts, representative causes include adding new devices, adding drivers/adapters for devices, and fixing bugs related to hardware devices.

These results provide evidence that it is possible to identify and measure hardware-driven modularity violations in a cyber-physical system by analyzing the software version control system. However, it is important to emphasize that analysis techniques developed are retrospective in nature. We found modularity problems that have already occurred. While this information can be useful when re-architecting a system or determining how to subcontract system maintenance, what would be more useful for DoD are leading indicators that can be assessed early enough in the acquisition process to prevent modularity violations from being created in the first place. That will be addressed through future work.

1 INTRODUCTION

Cyber-physical systems (CPS) have achieved widespread application in diverse areas including: civil infrastructure, energy, healthcare, transportation, automotive, smart appliances, and others. Usually cyber-physical systems are composed of diverse subsystems consisting of both physical and software components developed by different vendors. These components are deeply intertwined at various levels of abstraction (e.g. data flow, physical connections, and logical connections etc.) under changing contexts to achieve the desired functionalities and the respective quality attributes, such as performance, security, scalability, maintainability, etc. With the advance of technology, the recognition of new consumer needs, and the detection of deficiencies in current systems, components need to be upgraded, replaced, and fixed--- frequently, and in many domains. The key question is: can the upgrade, replacement, or problem fix happen quickly and without disrupting the rest of the system? In the traditional software engineering field, it is often not the case. Modules without any explicit structural dependencies are seen frequently changing together when new features or bug fixes were implemented. This phenomenon is called a modularity violation (MV).

To address this issue, stakeholders, such as the U.S. Department of Defense, have increasingly emphasized modular and open approaches to system development. The 5 goals of the Modular Open Systems Approach (MOSA) are to improve interoperability, facilitate technical refresh through system evolution and technology insertion, enable cost savings and cost avoidance, and foster innovation and competition. It is difficult, however, to assess whether the resulting architectures and systems are truly modular, making it difficult to gauge progress toward achieving these goals. Ultimately, it would be good to have sufficient confidence that the modules in a CPS can actually be upgraded, replaced, and fixed in a plug-and-play manner without affecting one another to maximally leverage the benefits of modularization. In software systems, the consequences of modularity violations could be higher bug-rates [1] and increased maintenance costs. It is possible that that modularity violations could be even more harmful in cyber-physical systems. Modularity violations, derived from the latent relationships among software and hardware components, prevent the long-term evolution, maintenance, and success of cyber-physical systems. In particular, due to the inflexibility and high cost of evolving hardware modules, modularity violations in cyber-physical systems could more easily lead to vendor lock-in compared to software developed to run on commercial-off-the-shelf hardware. The stakeholders lack techniques, metrics, and models that would allow them to detect, measure, and understand modularity violations in developed and acquired cyber-physical systems.

To address this issue, the objective of this research task was to investigate whether techniques developed to identify modularity violations in pure software systems could be adapted to detect modularity violations in cyber-physical systems. The primary challenge to adapting these methods lies with the hardware side of cyber-physical systems. Data to track changes to independent and distributed hardware components are often not readily available. However, we hypothesized that due to the fact that software changes are typically less expensive than hardware changes, hardware driven modularity violations may be addressed via software fixes, at least temporarily. This would mean that there would be evidence of hardware driven

modularity violations within the software version control system. To evaluate this hypothesis, we had to investigate the following three research questions.

1.1 Research Questions

Question 1: How should modules be defined in a cyber-physical system?

The nature of a CPS system and the goals of different stakeholders impact how modules should be defined/captured. In practice, the definition of a module will depend on the nature of system and the goals of different stakeholders. In some systems, stakeholders may be willing to trade modularity for performance. In other cases, it may be a matter of perspective. For example, low-level developers may view modules as hands-on and manageable working elements to implement functional details (e.g. a particular source file and a particular hardware board). In comparison, a project owner views modules as cohesive functional components to deliver the product value and competitiveness. Thus, there is a need to be able to handle modules of different granularity in the analysis.

Question 2: How can we identify hardware driven modularity violations when the relevant hardware concepts vary from system to system?

Given a CPS system in a specific application domain, how should modularity violations be identified and measured? During the incubator phase of this project, as a test case, we manually extracted 16 keywords from the commit messages and hardware devices supported by an open source cyber-physical systems called OpenWrt using expert judgment. However, these keywords reflected concepts unique to OpenWrt - a different system would entail different concepts. Manual extraction is also not scalable. A more extensible and general approach is needed to identify the system specific concepts necessary to diagnose identified modularity violations in a cyber-physical system.

Question 3: How can stakeholders make actionable decisions to adjust the modular structure of a CPS such that critical modularity violations in CPS can be mitigated?

Not all modularity violations are equally important. Rather they need to be prioritized based on the perspective of the decision maker, location within the system structure, the magnitude of impact, and frequency of occurrence. To address this, we needed to develop a proof-of-concept decision support tool to depict potential modularity violations on a project. The key is to enable the decision maker to examine the cyber physical system from different perspectives and identify the relevant modularity violations in each perspective based on impact and cost metrics.

The remainder of this report describes how we addressed these questions and is organized as follows: Section 2 describes the pertinent aspects of the challenges of measuring modularity violations in cyber-physical systems. Section 3 describes the approach to executing the research, while Section 4 describes the results of the research tasks in detail. Section 5 discusses the results and explains the key findings. Finally, Section 6 concludes the document by describing the next steps for this line of research.

2 BACKGROUND

2.1 Architecting Cyber-physical Systems

The term “cyber-physical systems” emerged around 2006, when it was coined by Helen Gill at the National Science Foundation [2, 3]. Gill defined a cyber-physical system (CPS) as an integration of computation with physical processes. Usually cyber-physical systems are composed of diverse subsystems consisting of both physical and software components developed by different vendors. These components are deeply intertwined at various levels of abstraction (e.g. data flow, physical connections, and logical connections etc.) under changing contexts to achieve the desired functionalities and the respective quality attributes, such as performance, security, scalability, maintainability, etc. It has been recognized that the design, modeling, and maintenance of cyber-physical systems is more challenging than for traditional engineering systems due to the intrinsic heterogeneity and nondeterministic nature of the interactions among cyber and physical components [4, 5, 6]. Architecting and evolving a system under these circumstances is a formidable problem.

To address this issue, stakeholders have increasingly emphasized modular and open approaches to system development. According to Baldwin and Clark [7], modularity allows for both independence of structure and integration of functions in large and complex systems. The goals of modularity are to improve interoperability, facilitate system evolution and technology insertion, and foster competition. However, it can be difficult for stakeholders to assess whether the resulting architectures and systems are truly modular. In other words, can modules in a CPS actually be upgraded, replaced, and fixed in a plug-and-play manner without affecting one another to maximally leverage the benefits of modularization?

One of the challenges to architecting and managing cyber-physical systems is that there are techniques to represent either the cyber processes or the physical processes [8], with only a few exceptions that can represent both software and hardware components [9]. From the “cyber” perspective, there are different techniques to represent the architecture of software systems, such as architecture description languages, Unified Modeling Language (UML) models, and component models. From the physical perspective, there are different traditional engineering techniques to model the development of physical systems. One study on architecting cyber-physical systems found that, currently, researchers tend to focus on a specific attribute of interest rather than assessing the cyber-physical architecture as a whole [9].

However, there are some instances of research that take a more holistic approach to architecting cyber-physical systems. Rajhans, et al. [10] contributed a new cyber-physical architectural style to present the interconnections and interactions between physical and cyber components. They defined three related families of general components and connectors pertaining to the cyber domain, the physical domain, and their interconnections. From the architectural assessment perspective, Sinha [11] developed a theoretical framework for structural complexity quantification and its implications for the design of cyber-physical systems. Derler et al. [2]

focused on the challenges of modeling cyber-physical systems that arise from the intrinsic heterogeneity, concurrency, and sensitivity to timing of such systems, while describing some promising approaches that use domain-specific ontologies to enhance modularity and jointly model the functional and implementation architectures. Finally, Cristalli et. al. [12] provides a representative example of a modular approach to developing a cyber-physical system through their modular design for a Smart Robotic Cell, composed by several interconnected sub-systems.

What all of the above have in common is a focus on architecting modular cyber-physical systems. However, there is no guarantee that the realized system will exhibit the intended modularity. The need to assess the actual modularity of a cyber-physical system is particularly acute. As Lee [3] points out, cyber-physical systems have always been held to a higher standard of reliability and predictability than general-purpose computing. Low reliability and predictability constitute a significant barrier to the application of cyber-physical systems to safety-critical domains such as, traffic control, automotive safety, and health-care. While a modular design is certainly an option to improve reliability, predictability and maintainability, when employed, it is desirable to understand how well that goal is accomplished. Approaches to assess modularity in pure software systems have been developed, but to the best of the authors' knowledge no approaches have been developed that specifically address the unique challenges of assessing the modularity of cyber-physical systems. Thus, the question is whether existing approaches from the software domain can be adapted to the cyber-physical domain.

2.2 Detecting Modularity Violations in Software Systems

The term modularity violation was first proposed by Wong et al. [13] to describe the phenomenon where independent software modules frequently change together during the evolution of a system in the process of fixing bugs or adding features. Methodologies and tools have been built for analyzing modularity violations in software systems [14, 13, 15]. The identification of a modularity violation follows three steps:

1. *Reverse-engineering the modular structure of a software system from the code base:* The modular structure of a software system can be calculated based on the structural dependencies among software entities. Xiao et al. developed a new architectural representation, called Design Rule Space (DRSpace) modeling to capture the modular structure of a software system as multiple overlapping design spaces [15], applying Baldwin and Clark's design rule theory [7]. The modular structure can be visualized in the form of a Design Structure Matrix (DSM). The items in the DSM represent software entities, the cells represent the structural dependencies among entities. The entities can be clustered into modules based on selected dependency types for supporting analysis of different focuses.
2. *Extracting the evolutionary coupling (i.e. the number of co-changes) between software modules:* Evolutionary coupling can be calculated based on the data recorded in version control systems, which automatically keep track of changes made to software entities. Consequently, we are able to determine who changed what and when and for what

reason. Analyzing such data allows us to determine the frequency of concurrent modifications of any two software entities in the course of maintenance activities. In other words, it enables us to extract the evolutionary coupling between software entities. The more frequently two entities are modified together, the stronger is their evolutionary coupling – implying latent connections.

3. *Calculating the discrepancy between the above two data sets to point to modularity violations among software modules:* If two software entities/modules are structurally independent according to step 1, but they have high evolutionary coupling according to step 2, there is a potential modularity violation. According to prior empirical studies of industry software systems, modularity violations usually indicate implicit assumptions shared among software modules. For example, it could be an implicit assumption regarding the usage of time units in two modules [16].

In software systems, the consequences of modularity violations could be higher bug-rates (when the assumptions are not consistent among different parties) and increased maintenance costs (in order to keep the assumptions consistent) [14, 15, 17]. Studies of real-life open source projects indicate that up to 85% of bug-fixing efforts in a software system involve modularity violations [17].

2.3 Detecting Modularity Violations in Cyber-Physical Systems

The described approach for detecting modularity violations is limited in that it only addresses software systems implemented in a single programming language. Complex cyber-physical systems however, comprise both heterogeneous hardware and software sub-systems, which makes them particularly difficult to analyze. Indeed, incurring in modularity violations that result from latent relationships between software and hardware modules endanger the system's long-term evolution, maintenance, and success. In particular, due to the inflexibility and high cost of evolving hardware modules, modularity violations in cyber-physical systems could more easily lead to vendor-lock-in compared to a pure software environment.

In this report, we organize modularity violations in cyber-physical systems into three different types: 1) software vs. software, 2) software vs. hardware, and 3) hardware vs. hardware. The first type of modularity violation has been extensively investigated in prior work [13, 16, 15, 14, 17]. There, the approach to identify and measure modularity violations relies on the analysis of source code and operational data such as maintenance activity records. As previously mentioned, such information can be automatically and comprehensively tracked in version control systems. The immediate challenge that arises when trying to extend the same approach to identify hardware-specific modularity violations is that, oftentimes, it can be difficult to find records that extensively describe changes made to the different hardware components. Given these limitations, the question naturally follows, could a subset of hardware modularity violations be inferred from an analysis of records of software changes? More specifically, some software changes may be a consequence of hardware related modularity violations. If that were the case, analysis of version control systems could be used to detect some hardware related modularity violations outright

and flag other potential violations for further investigation. The issue is whether or not there is actually enough information in a version control system to infer a hardware modularity violation. Consequently, the goal of this study was to test the feasibility of leveraging software maintenance data to infer hardware related issues that are indicative of software-hardware modularity violations.

3 APPROACH

In order to answer the research questions, we needed to use real world cyber-physical systems as test cases. We selected two projects: OpenWRT and MD PnP which are described below. We then divided the research into three major tasks. Each of these tasks is described in detail below.

3.1 Data

To develop and evaluate the approach to detect and classify modularity violations, we selected two open source software infrastructures for two real-world cyber-physical systems to serve as test cases: OpenWrt and MD PnP. OpenWrt is a Linux based core that is commonly used to support the Internet of Things (IoT). MD PnP stands for the Medical Device "Plug-and-Play" interoperability program which aims to improve patient care by integrating information from medical devices. Table 1 summarizes basic facts of these projects. It shows the scale (measured by number of files, methods, and lines of code), the development team size, and the age (measured by the number of revisions and starting time). These projects were chosen because, in addition to providing sufficient research data, they also involved two very different application domains.

Table 1 – Project Summary Data

Project Name:	OpenWRT	MD PnP
# of files	1052	866
# of methods	6061	7872
# of lines of code	163,114	73,616
# of developers	137	12
# of revision records	38,099	1605

Range of history	2004 to 2017	2013 to 2017
------------------	--------------	--------------

3.2 Task 1: Explore alternative modular decompositions of a cyber-physical system

During the incubation phase of this research, we treated source files as modules. This is appropriate from the perspective of a low-level developer. However, a project owner views modules as cohesive functional components to deliver the product value and competitiveness. This results in a very different definition of a module.

As noted previously, the definition of a module will depend on the nature of system and the goals of different stakeholders. This research task examines different criteria to decompose and capture modules of different perspectives and granularities in a cyber-physical system:

1. *Decompose OpenWrt and MD PnP based on working task responsibility and ownership.* A large-scale cyber-physical system is often developed and maintained by a team or several sub-teams collaboratively. Task responsibility or ownership can be used to decompose the system into modules. The challenge is that the task responsibility/ownership may not always have clear boundaries among different contributors/vendors. We first extracted the mapping between atomic elements in the system and individual contributors. Then we developed a clustering algorithm to group the atomic elements into modules based on ownership. Each module contains a group of atomic elements that share specific ownership. The hypothesis is that the modularity violations identified from this perspective of modular decomposition may help to reveal latent assumptions shared among different contributors. Identifying these violations will help to avoid vendor lock-in.
2. *Decompose OpenWrt and MD PnP based on coupling among the atomic system elements.* The hundreds and even thousands of atomic system elements in a system are coupled with each other in different formats. We focused on two types of coupling: 1) static structural dependencies, and 2) emerging evolutionary coupling. Although run time dynamic coupling would also be an important aspect of information to study potential modularity violations, we were not able to acquire the run time dynamic coupling information due to the difficulty in replicating the proper run time environment of these two systems. We mined the project repository of OpenWrt and MD PnP to capture these three types of coupling among the atomic system elements. After the coupling analysis, we developed algorithms to cluster the atomic elements into modules such that elements within a module should have high coupling while those in different modules should have low coupling. Each module should represent a group of elements that achieve certain functionality together. Modularity violations identified from this perspective may reveal hidden interfaces among different functionalities.
3. *Decompose OpenWrt and MD PnP based on the system architecture.* Usually the architecture of a complicated system is composed of layers of separate concerns. A

common architecture should be composed of four basic layers from top to bottom: the interface layers, the application layers, the business layer (sometimes referred to as the domain layer), and the infrastructural layer. Each layer contains modules (components) of different areas of concerns. We first retrieved and processed the high-level design documents of the two systems. We then clustered the atomic elements in each system following the architectural layers defined in the high-level documentation. This coarse-grained modular structure helps to identify cross layer modularity violations.

4. *Compare and contrast the different modular decompositions for OpenWrt and MD PnP.* After using the four different criteria to decompose a system into modules, we compared the modular structure and the modularity violations of each decomposition perspective.

As an additional note, we initially planned to decompose *OpenWrt* and *MD PnP* based on cohesion. Each atomic system element in a cyber-physical system should have certain semantic meaning, reflected in its name, code, and even hardware boards. We intended to leverage such information to cluster elements with similar semantics into the same module. Therefore, the elements in a module should share consistent semantic meaning with each other. That is, each module is likely to represent a cohesive concern in the system. Modularity violations identified from this perspective will reveal the implicit connections across different concerns. However, we were not able to accomplish this goal as planned due to the unique challenges of extracting domain/hardware concepts in the different project domains. This part relies on successfully extracting the domain concept hierarchy of projects and map to the modules in cyber-physical systems. We found that there was often insufficient information to successfully map low level entities such as source files, while higher level entities could relate to many different domain concepts. Consequently, we will need to develop an approach to address these issues through future research.

3.3 Task 2: Develop a Domain Concept Learner to Extract Hardware Related Concepts

In order to understand when modularity violations are driven by hardware concepts, we need the ability to interpret change messages associated with the violation. These messages typically use domain specific terminology. During the incubator phase of this research, we used manually extracted keywords to classify changes. This approach is not scalable for an analysis of a full cyber-physical system and would need to be redone for each new system.

To address this issue, we developed a Domain Concept Learner that leverages natural language processing (NLP) techniques. To provide data to the NLP algorithms, we extracted and processed supporting wiki-pages and project documents that record domain-related hardware components. The NLP algorithms learn the semantic relationships among project specific terms allow one to group similar terms together. For example, in OpenWRT, the names of the supported routers should have a low semantic distance. Using this approach on each new project, we can quickly classify potential modularity violations to cyber-physical systems in different domains.

To accomplish this, we did the following:

1. *Evaluated candidate topic modeling techniques on OpenWrt.* We began by testing Latent Dirichlet Allocation (LDA) to the project documentation for OpenWrt. LDA is a common statistical technique for extracting groups of related terms from text documents. We chose LDA because of its established history as an effective technique for topic modeling. However, we quickly found that it did not provide the required flexibility to support our analysis. The reason is that while LDA can group related words, it does not provide a distance metric to assess how similar two words are. Consequently, LDA would have limited us to a purely categorical classification of terms. We then tested Word2vec and found it effective. (See Section 4.2). We then experimented with several clustering techniques to organize the results of the Word2vec model. These included agglomerative clustering, k-means clustering, and Gaussian mixture models. We found k-means to be the most useful. From there, we used the fit clusters to identify key, project specific terms that were related concepts of interest such as supported hardware devices. We then used these keywords to calibrate cosine distance based metric. This metric enables us to classify each change message based on its score where a high score indicates a hardware related change. Finally, we validated the results by randomly selecting change messages and manually classifying them. We then compared the results of the metric scoring with the manual classification to assess accuracy.
2. *Test topic modeling techniques on MD PnP.* To test the domain concept learner, we applied the exact same approach to MD PnP. This provided a clean evaluation of the approach, as we did not analyze MD PnP during the development of the concept learner. As with OpenWRT, we compared the results of the metric score with the manual classification to assess accuracy.
3. *Tailor concept learning approach to feed the decision support tool:* The domain concept learner was tailored to generate outputs that were directly integrated into the decision support tool. The idea is that the terms identified as hardware related by the concept learner can be used by the decision support tool to search version control histories for potential hardware related modularity violations.

3.4 Task 3: Build decision framework demonstrator

To integrate the analyses described above, we built a proof-of-concept software demonstrator that combines the methodologies developed in the two prior tasks. The demonstrator shows how the developed methods and metrics can be applied to the two case study projects we chose: OpenWrt and MD PnP.

The demonstrator contains three major components as depicted in Figure 1:

- *Module Decomposer:* This component takes the project data (e.g. configuration management systems and issue tracking database) and the user's decomposition strategy as an input to calculate a modular decomposition based on user's preferences.

- *Domain Concept Learner*: Takes the hardware supporting documentation, such as wiki pages or design documents as input to learn the topic model of the domain hardware concepts.
- *MV Analyzer*: Leverages the calculated hardware concepts and the modular decomposition as input to identify potential Modularity Violations involving hardware and software components in a cyber-physical system. The stakeholder will see the potential modularity violations identified in their projects. This will enable them to be aware of potential modularity violations and understand the underlying hardware concepts that triggered changes to software modules. The ultimate goal is to help stakeholders to make informed decisions to maximize the benefits of modularization.

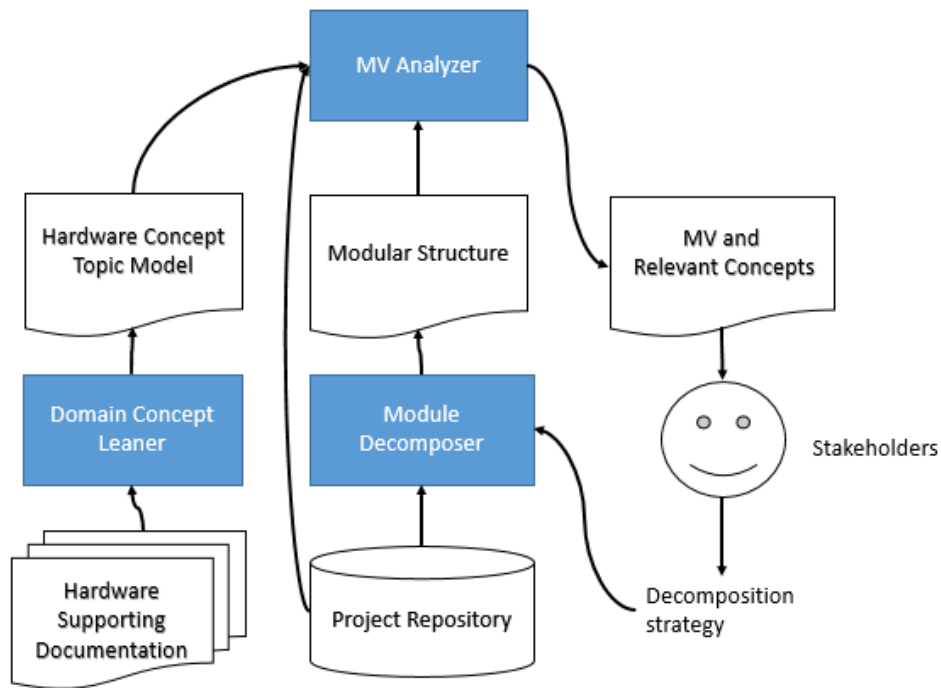


Figure 1 – Schematic of decision framework demonstrator

4 RESULTS

In this section, we present the research results of applying the approach described above to project data from two different open source cyber-physical systems: OpenWrt and MD PnP. The results can be summarized as follows:

1. Although OpenWrt and MD PnP both can be decomposed into separate modules using different criteria, there are always cross module changes that indicate potential modularity violations, i.e. the supposedly independent modules following a certain criterion are actually coupled in the system evolution. Modularity violations identified via

each different system decomposition reveal concerns for different groups of stakeholders.

2. By analyzing the project documentation and wiki information, we can train a concept learner to distinguish hardware related concepts from general software concepts. However, the effectiveness of the concept learner is impacted by the size of the training dataset, the problem domain, and the quality of the data. We found that the concept learner worked much better for OpenWrt than for MD PnP for these reasons.
3. By combining the module decomposer and the concept learner, we were able to discriminate modularity violations triggered by hardware concepts from those triggered by software concepts across multiple different modular decompositions. This enabled us to develop an in-depth understanding of what factors trigger modularity violations in these two cyber-physical systems.

4.1 Model Decomposition Results

We explored three different criteria to decompose the software side of the cyber-physical systems into modules from different perspectives. Each criterion represents a different aspect of concern in a system. We have also successfully generated the visualization of these different module views in a graph-based representation.

First, we explored the natural package decomposition. This is also known as the development view of a system. It represents how developers view the system as composed of different parts.

Second, we investigated the dependency hierarchy decomposition. This is computed based on the dependencies among files. The entire system is decomposed into layered modules. The source files in the upper layers (with larger layer numbers) depend on the source files in the lower layers (with smaller layer numbers), but not vice versa. Therefore, the dependency hierarchy decomposition potentially represents how a system can be developed sequentially, since the lower layer modules need to be built before the upper layer modules can be built.

Third, we examined the organizational structure decomposition. According to the Conway's law, the architecture of a system should mirror the organizational structure in which the system is built. Therefore, we leverage the organizational information of a system to decompose it into modules based on ownership. For the open source projects being studied, ownership of a system element was assigned to the developer who worked on the element the most.

4.1.1 DEVELOPMENT VIEW DECOMPOSITION

First, we recover the root package view of MD PnP and OpenWrt, as shown in Figure 2. The ovals represent modules, i.e. packages, in this case. The edges represent two types of relationship among modules: structural dependency and co-change frequency. The blue line represents the

occurrences of structural dependencies among modules. The red line represents the number of co-changes between two modules.

In MD PnP, the root packages are very well decoupled. There are dependencies only between “interop-lab” and “devices”, and between “interop-lab” and “data-types”. In OpenWrt, the root packages are strongly coupled from each other. Each root package depends on each other. The first impression is that MD PnP has much simpler modular structure than OpenWrt. Then, we add the co-change relationships (shown as red lines) into the root package view. The results show that although the package modules in MD PnP seldom have structural dependencies, co-change relationships appeared between nearly every pair of modules, even when two modules are independent from each other. Therefore, this is a potential indication of modularity violations at this level of modular structure. In comparison, the co-change relationship in OpenWrt is only among three out of 5 modules. In summary, despite the simplicity of the MD PnP’s root package structure, it seems to be more likely to contain modularity violations.

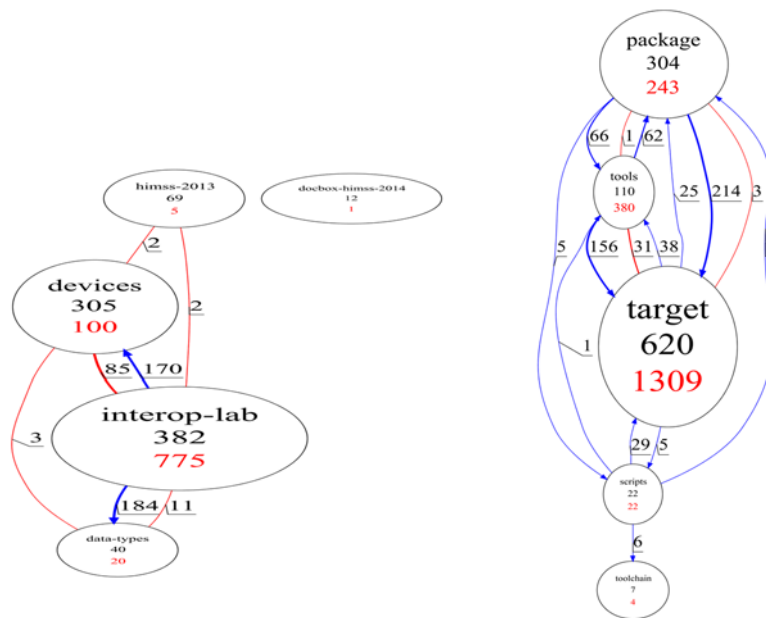


Figure 2 - Root Package View (Left MD PnP, Right OpenWrt)

Furthermore, we investigated the fine-grained package view of both projects. Figure 3 shows that MD PnP has a “facade” which takes requests from high-level applications, such as “demo-apps”, and assigns these “requests” to specific hardware items, such as “phillips”. The hardware modules further depend on the “common” infrastructure. This shows a well-structured modular structure in MD PnP. However, as shown in Figure 4, when considering the co-change relationships, everything changes with everything else. This indicates that modularity violations are very common at this level of modular decomposition. In order to focus on the most severe co-change relationship among modules, we used a co-change threshold of 10. This helps us to get Figure 5, which shows the most problematic modules in terms of potential modularity violations. Similar observations were made with respect to OpenWrt despite its fewer co-change relationships when compared to MD PnP.

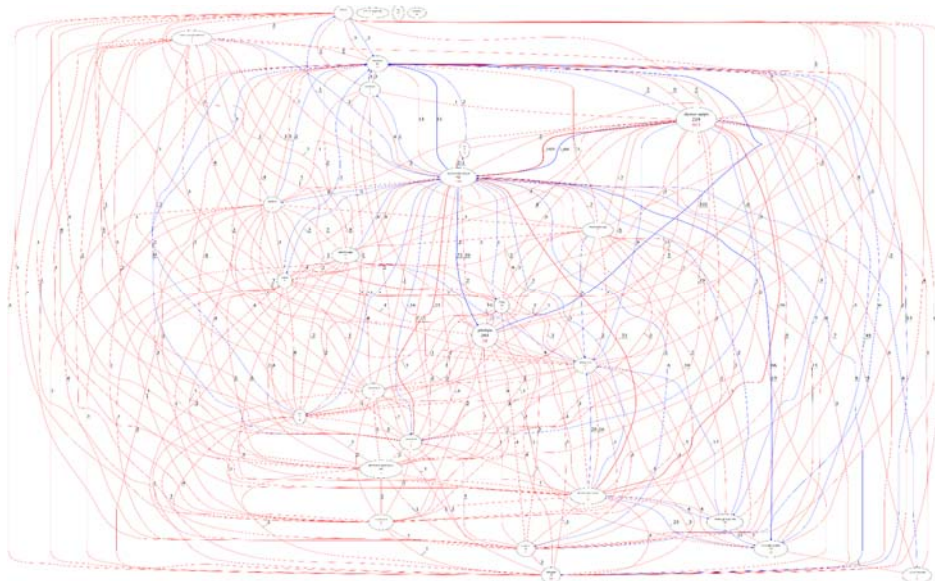
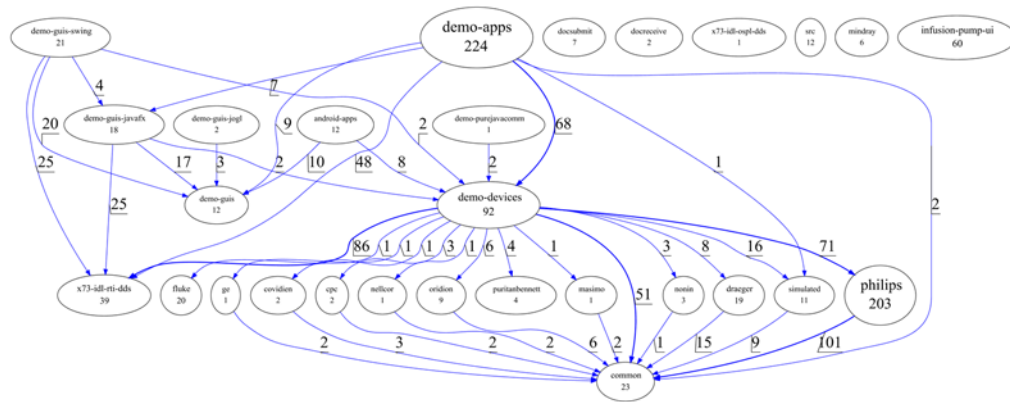


Figure 5 - MD PnP Fine-grained Package View with Co-changes > 10

4.1.2 DEPENDENCY HIERARCHY DECOMPOSITION

The dependency hierarchy decomposition is computed based on the dependencies among files. The entire system is decomposed into layered modules. The source files in the upper layers (with larger layer numbers) depend on the source files in the lower layers (with smaller layer numbers), but not vice versa. Therefore, the dependency hierarchy decomposition potentially represents how a system can be developed sequentially, since the lower layer modules need to be built before the upper layer modules can be built.

Figure 6 shows the dependency hierarchy decomposition of MD PnP and OpenWrt. According to our observations, there are 13 layered modules in MD PnP, and 8 layered modules in OpenWrt, which reveals the more complicated internal dependency hierarchy for the former project.

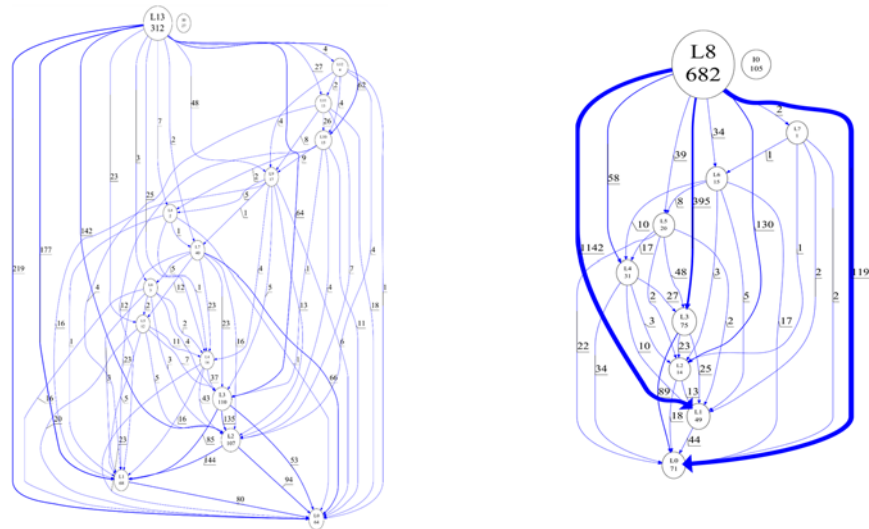


Figure 6 - Design Hierarchy View (Left: MD PnP, Right OpenWRT)

Figure 7 shows the co-change relationships for the dependency hierarchy modules. The results show that it is almost impossible to implement MD PnP sequentially, since there is a co-change relationship between almost every pair of modules. OpenWrt, on the other hand, has significantly fewer co-change relationships.

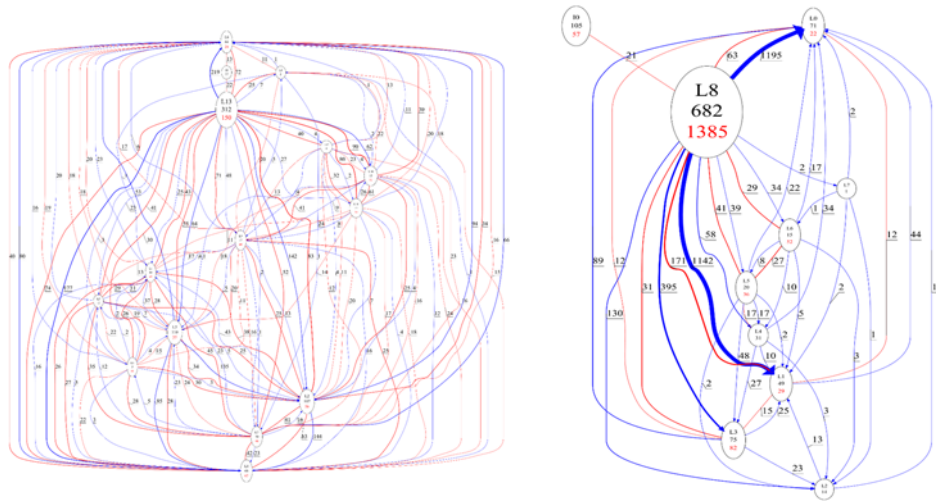


Figure 7 - Design Hierarchy View with Co-change (Left: MD PnP, Right: OpenWrt)

4.1.3 ORGANIZATIONAL STRUCTURE DECOMPOSITION

According to Conway’s law, the architecture of a system should mirror the organizational structure in which the system is built. Therefore, we leveraged the organizational information of a system to decompose a system into modules based on ownership. Ownership of a system element was assigned to the developer who worked on the element the most. The identities of the real developers were obscured by substituting names of chess pieces.

According to Figure 8 (Left), there are 7 main “vendors” in MD PnP. The “King” is the main contributor and his module decouples other contributors’ modules. In Figure 8 (Right), there are 21 main contributors in OpenWrt. Everyone’s work is related to everyone else’s. By adding the co-changes among organizational modules in MD PnP, we imply that there is a low likelihood of modularity violations at this level. However, the project is heavily dependent on “King.” He is a single point of failure. In comparison, co-changes among organizational modules in OpenWrt is much more common. This indicates a higher likelihood of modularity violations cross modules of different vendors.

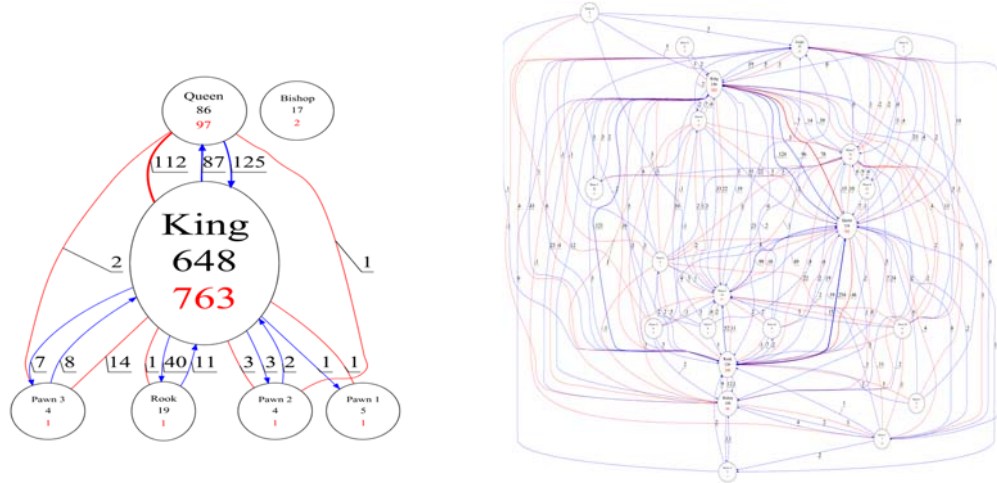


Figure 8 - Organizational View (Left: MD PnP, Right OpenWrt)

4.1.4 MODULAR STRUCTURE TIME-SERIES ANALYSIS

The modular structure of a system may change over time due to various reasons such as increasing product maturity or operational feedback. To better understand the evolution of the modular structure of a CPS, we evaluated three metrics at various time steps to generate time series data for the two projects.

Cross Module Change (CMC): the percentage of changes that involve source files belonging to different modules compared to the total number of changes that happened in the system. The higher the value, the more severe modularity violations.

$$CMC = \frac{\text{Number of Cross - Module - Changes}}{\text{Total Number of Changes in a System}}$$

Inner Module Change (IMC): the percentage of changes that only involved source files belonging to the same module compared to the total number of changes that happened in the system. Note that CMC+IMC always equal to 1. The higher the IMC value, the better the modular structure.

$$IMC = \frac{\text{Number of Inner - Module - Changes}}{\text{Total Number of Changes in a System}}$$

Value for Cost (VfC) for a Module: the maintenance cost (approximated by the number of changes since the actual cost is not available) on a module divided by the size of a module. The higher the value, the more expensive it is to maintain a particular module.

$$VfC_{\text{module}} = \frac{\text{Number of Changes in Module}}{\text{Number of Files in Module}}$$

Next, we identified three stable versions of OpenWrt, as shown in Table 2. We chose these three versions because they are the versions released by the developers and reflect delivered products to customers.

Table 2 – Stable Versions of OpenWRT

Version	Date	Months between Versions	#Commits between Versions
v17.01.0	Feb-17	8	3036
v17.01.4	Oct-17	8	1600
v18.06.0	Jul-18	9	2492

Our goal is to analyze how the modular structure based on the three different criteria changes over these three versions measured by the three metrics we proposed. Modules that involve a drastic increase in complexity (i.e. the metrics) should merit more attention from the stakeholders.

We applied all three decomposition criteria. The examples below show the most drastic changes. As shown in Figure 9, in the organizational structure, the VfC of the top 5 vendors changed dramatically. This indicates that the responsibility portion of the vendors changed drastically over time. For example, Vendor 2 used to have very high VfC at the first version, but its VfC reduced drastically in the third version. In comparison, the VfC of Vendor 3 and 4 drastically increased from the first version to the third version. This indicates that Vendor 3 and Vendor 4 are more likely to cause lock-in over time, and replace the influence of Vendor 1. This is because higher VfC of module indicates that the module is requiring more maintenance updates by the specific vendor. When the vendor with increasing VfC leaves the project, the project suffers from higher risk of maintenance difficulties.

We also investigated the trend of CMC and IMC based on the other two different decomposing criteria, including the development structure and the hierarchical structure. We can observe the change of these metrics from different perspectives using other criteria. Here, we just will use the organizational structure to illustrate our study as shown in Figure 10. We observed that the CMC and IMC respectively reduced and increased in the second version but increased to the same level in the third version as in the first version. This indicates that overall modular structure following the organizational decomposition criteria increased in the second version, but does not change when comparing version 1 and version 2. This indicates that the modular structure is improved in the middle release, probably reflecting an organizational structure improvement, but it became the same as version 1 in version 3.

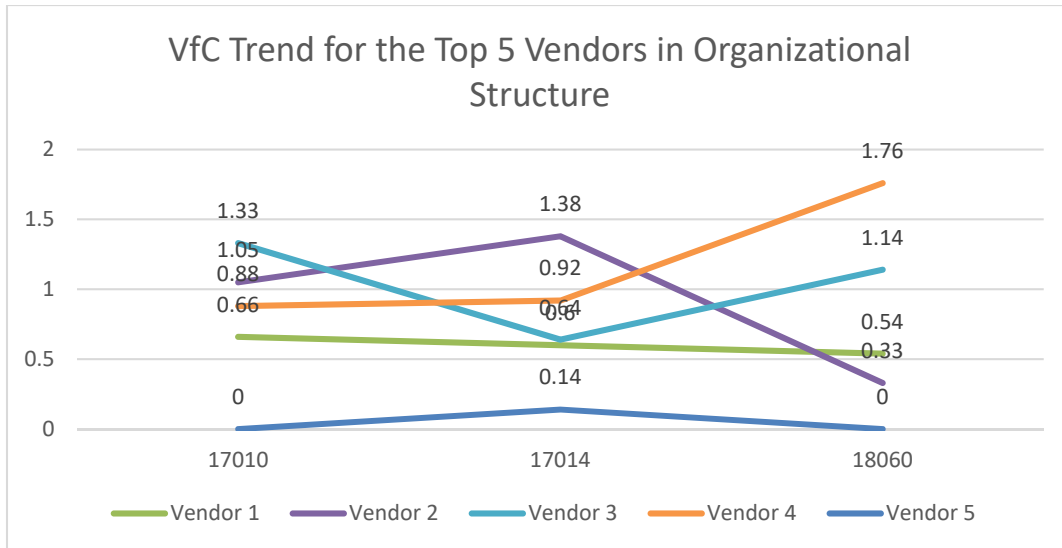


Figure 9 - OpenWrt-VfC Trend for the Top 5 Vendors

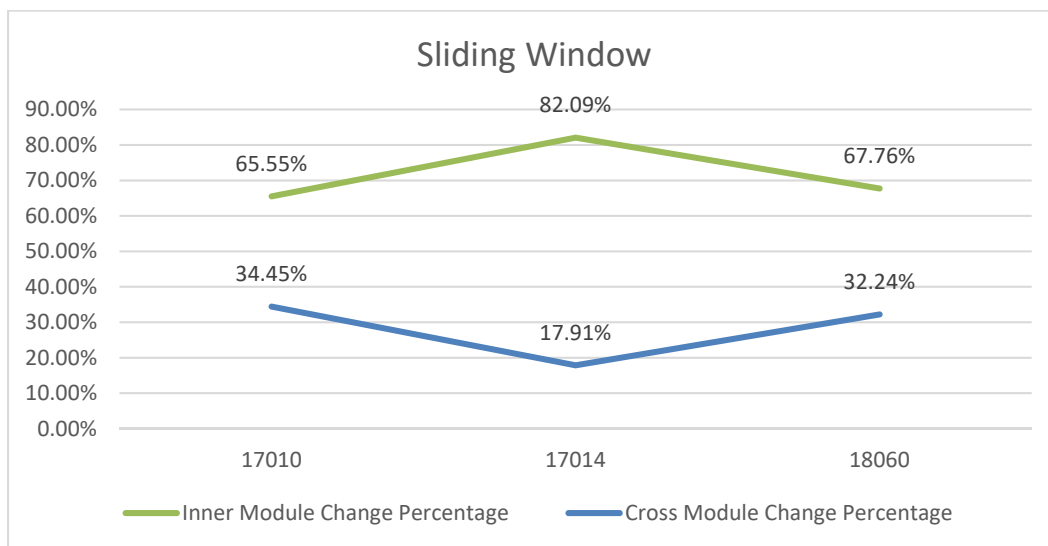


Figure 10 - OpenWrt- CMC-IMC-Trend

4.2 Domain Concept Learner Results

The goal of the domain concept learner is to leverage natural language processing techniques to extract domain, e.g., hardware related, concepts of a cyber-physical system to identify potential modularity violations. As indicated in Section 3.3, the approach to develop the domain concept learner began with OpenWrt as the training case and then use MDPnP as the test case.

4.2.1 DATA PREPARATION AND EXPLORATION OF NLP TECHNIQUES

The first step was to compile the corpus of project documentation. For OpenWRT, we investigated three types of narrative data:

- The project documentation contained in a wiki on the project website. We wrote a script to traverse the wiki pages and scrape relevant text about OpenWRT and its development. This resulted in a set of 818 documents extracted from the wiki pages. Next we extracted the revision message history from the project code repository in GitHub. We included 41,687 revision messages in our analysis.
- The narrative comments stripped from the source files of OpenWrt. OpenWrt is a multi-language program, containing mostly C/C++ code, and other script languages, such as perl, shell, etc. We wrote a program to automatically traverse the source code directory and strip all the natural language comments in the C/C++ source files. These comments provide descriptions of the concepts and functions in the source code.
- The change messages left by developers of OpenWrt for describing the causes and purposes of the change. Such information helps us to understand why changes are happening in the OpenWrt project. More specifically, our ultimate goal would be to automatically measure and separate hardware concept driven changes from software concept driven changes.

To analyze the corpus of project documentation to extract concepts, we initially applied a common topic modeling approach called Latent Dirichlet Allocation (LDA). While the topics generated by this method did appear to have some coherence when applied to the wiki pages, LDA and related approaches rely on a coherent document structure to extract topics. The revision history has no such structure. Consequently, this limited the utility of LDA.

Therefore, we switched an alternative NLP technique called word2vec, which is based on word similarity learned from the features of the given documentation. Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct the linguistic context of words. This class of models is often used to analyze unstructured text such as social media postings. Thus, it was a more suitable choice to handle the OpenWRT change history.

The word2vec model yielded much more useful results than the LDA family of algorithms. In addition, the word2vec model supports analysis based on semantic distance. For example, we were able to generate a list of hardware related terms by applying the following query:

```
word_distances = model.most_similar(positive = ["hardware", "device", "router",  
"radio", "wifi", "mips", "ramips", "mtd", "broadcom", "routerboot", "router",  
"firmware", "bluetooth", "energy", "power", "soc", "chip" ], negative = ["api",  
"call", "class", "code", "readability", "style", "data", "function", "gdb",  
"infinite", "loop", "bug", "json", "kernel", "leak", "method", "null", "parameter",  
"plugin", "process", "recursive", "script", "string", "syscall", "variable"], topn  
= False)
```

This query allows us to specify both positive and negative words. The query returns words that are semantically close to the positive words and semantically distant from the negative words. In this example, the positive words were hardware related terms such as “device” and “chip” while the negative words were software-related terms such as “loop” and “code.” The query allows us to generate a rank-ordered list of hardware related terms that can be used in the analysis of modularity violations. For example, Figure 11 lists the top 60 words from the above query out of a total vocabulary of 11,952 words in the project documentation. While many of the listed terms are hardware targets for OpenWRT, there are terms in the list that are not as useful for our analysis.

board	cf	plus	verdex
profile	ep93xx	fi	qca9563
at91	zyxel	rt5350	dk01
ehci	techdata	udc	agl300nh
netgear	compex	ar9331	k330
linksys	mt7620a	imx23	305x
cns21xx	omap35xx	routerboard	rb750up
mikrotik	qualcomm	wi	u7623
rt3883	pro	extender	ls1043ardb
rt288x	ata	amcc	aga
ppc40x	openmesh	meraki	awake
apm821xx	gumstix	dlan	sc16is752
pxa	alice	pirelli	mx60w
buffalo	huawei	gate	mt7621a
avila	rb1xx	devolo	7links

Figure 11 - Top 60 words from query

4.2.2 CLUSTERING CONCEPTS

To better organize the vocabulary, extract concepts, and reduce noise words; we next applied clustering techniques to the trained word2vec model. A key advantage of word2vec over LDA is that word2vec enables the computation of a distance between each pair of words in the vocabulary. This allows for clustering where words with shorter semantic differences should be included in the same cluster.

First, we applied a type of hierarchical clustering called agglomerative clustering. This approach is common for ontology extraction, and the hope was that we could use it to not only extract clusters of topics, but also relationships among those topic clusters. For example, a topic could be a sub-topic of another, more abstract topic. Unfortunately, despite many experiments, we were unable extract useful clusters using this method.

To better understand the structure of the word distances, we next applied a visualization technique called t-SNE. This enabled us to project the 500 dimension vectors that describe the words into two dimensions. An example using cosine distance is shown in Figure 12. Given the amount of dimension reduction in the visualization, the plot should not be interpreted literally, but it did show that there was structure to the vocabulary.

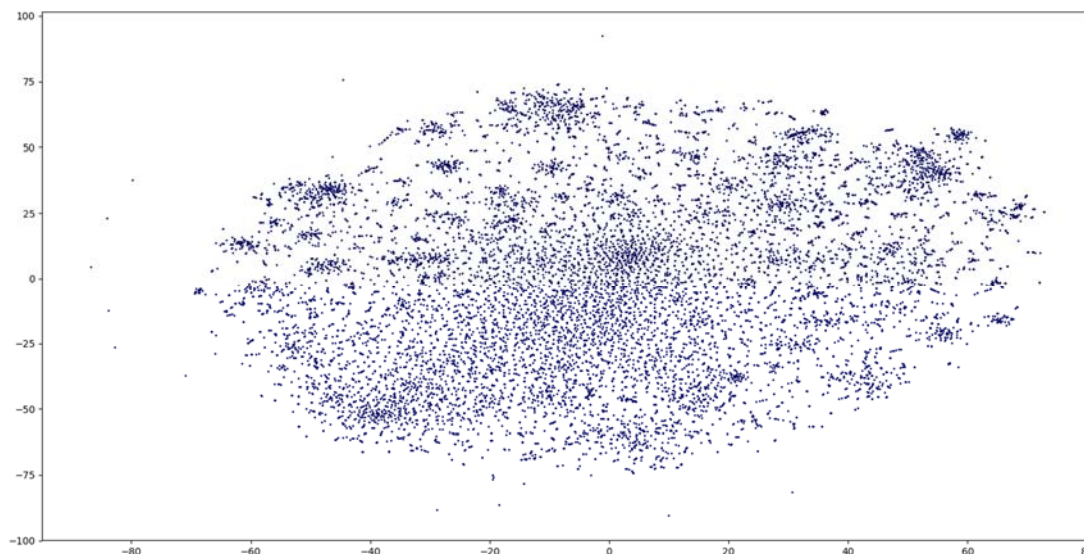


Figure 12 - t-SNE plot of the trained word2vec model with 500 hidden nodes using cosine distance

Consequently, our next step was to apply another approach called k-means clustering. However, unlike hierarchical clustering, the number of clusters must be specified. We experimented with a range of cluster numbers. The resulting clusters were much more coherent and useful than those found using hierarchical clustering. We were able to summarize each cluster by a name that reflects the overall content of the cluster. As expected, some of the keyword clusters are not related to hardware concepts at all. For example, cluster named “project infrastructure” contains keywords like “documentation, welcome, git, release, doc, wiki”. These are keywords used in managing the eco-system of OpenWrt. As another example, we also observed a cluster named “software development” which contains keywords like “x86, header, developer, typo, legacy, python”, which are software related keywords used by the developers in the development process. In comparison, there are also keyword clusters that are closely related to hardware concepts. For example, the cluster named “IoT devices or targets” which contains keywords like “brcm2708, ixp4xx, ipq806x”. The combination of the word2vec model and k-means clustering demonstrated that it was feasible to extract key concepts and terms from the project documentation. However, given the large number of keywords extracted from the data, not all are equally relevant to our study. Thus, the next step was to identify the most relevant keywords and their associated relationships.

4.2.3 REFINING THE MODEL

While we were able to demonstrate that the fit model grouped related terms together and enabled us to extract hardware related concepts, we still needed to keep refining the word2vec model and clustering approaches as well as developing a network of system concepts that could be mapped back to the architectures developed via Task 1.

First, several challenges with regard to text pre-processing were resolved. The most important of these dealt with naming conventions of system components that used '-' or '_' in the name. These characters are commonly used by developers when naming software objects but are less common in standard prose. As a result, our prior text processing routines were breaking the names of key objects into separate words. We added modifications to the text processing to keep these names intact. This has improved our ability to map from the text analysis to the system architecture. Figure 13 depicts a subset sample of conceptual clusters from OpenWRT. For example, Cluster 2 is a collection of hardware targets for OpenWRT. The highlighted terms are also the names of modules from the architecture described in Task 1. This is the first step to identify which of the issues described in the previous section are hardware related.

2	3	4	5	6	7
linux	image	file	package	add	patch
target	support	default	openwrt	fix	es-3
generic	device	use	makefile	update	es-4
es-2	board	user	config	build	-default
ar71xx	usb	data	network	remove	pending-4
ramips	driver	option	base-files	version	international
brcm63xx	profiles	configuration	etc	make	submitting
lantiq	platform	interface	src	change	alike
brcm2708	switch	http	lib	documentation	pagesource
adm5120	ethernet	server	control	upgrade	attribution-share
brcm47xx	wifi	set	ipkg	content	lzma-loader
mt7601	phy	port	net	lede	swconfig
ath79	code	rule	modules	page	backport-4
ixp4xx	register	using	services	missing	coldfire
ipq806x	gpio	start	init	new	map
s3c24xx	wireless	address	scripts	enable	uml

Figure 13 - A sub sample of conceptual clusters from OpenWRT

Second, we experimented with several variations on clustering of terms based on the word2vec model. The objective was to group similar terms together so that we can easily extract subsets of hardware related terms to use in the analysis for Task 1. We experimented with Gaussian mixture models as alternative to k-means clustering, but we did not see a substantial difference in results. Consequently, we decided to use k-means for subsequent analysis. We also conducted analyses to determine the most suitable number of clusters. For this phase we continued to use k-means clustering. However, we started varying the number of clusters in an attempt to

construct a hierarchy of system concepts. Figure 14 depicts a subset of these results. For clarity, only the terms that were also object names from the architecture were retained in the figure. The goal was to see how the clusters in one layer relate to those in another. For example, Cluster 2, a hardware related cluster, in the top layer appears to be composed of clusters 6 and 13 in the next layer down. This suggests that clusters 6 and 13 may be hardware sub-concepts that merit further investigation.

Clusters	0	1	2	3	4	5	6	7	8	9					
Module terms	6rd	uboot-ar71xx	linux			package		swconfig	kernel	include					
	gpio-button-hotplug	w1-gpio-custom	target			config		map	tools	boot					
	button-hotplug	rbcfg	generic			network			utils						
	otrx	trelay	ar71xx			scripts			system						
	rsileds	owipcalc	ramips			ppp			hostapd						
	flock	resolveip	brcm63xx			broadcom-wl			libs						
		padjffs2	lantiq						openssl						
		rtc-rv5c386a	adm5120						firmware-utils						
		iwcap	brcm47xx						musl						
		usbreset	mtd						glibc						
		rotary-gpio-custom	ath79						ead						
		ixp4xx-microcode	ar7						libnl-tiny						
		avila-wdt	firmware						mtdev						
		leds-apu2	at91						px5g						
		spidev_test	cns3xxx						mklibs						
		oseama	oxnas						gettext						
		fbtest	nvrpm						uboot-oxnas						
		fritz-tools	mcs814x						libiconv						
		maccalc	adm8668						sstrip						
		fwtool	mpc85xx												
		spi-gpio-custom	apm821xx												
		patch-image													
		osafeloader													
Clusters	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Module terms	map	fritz-tools	tools			rsileds	linux		package	broadcom-wl			lantiq	config	generic
		maccalc	utils			resolveip	target		network	swconfig				kernel	ramips
		fwtool	hostapd			padjffs2	ar71xx		scripts	ead				include	brcm63xx
		spi-gpio-custom	libs			flock	adm5120		ppp	6rd				mtdev	brcm47xx
		patch-image	openssl			rotary-gpio-custom	ath79			gpio-button-hotplug				boot	at91
		osafeloader	musl			ixp4xx-microcode	adm8668			uboot-oxnas				system	cns3xxx
			glibc			leds-apu2				libiconv				ar7	oxnas
			libnl-tiny			spidev_test				uboot-ar71xx				firmware	mcs814x
			mtdev			oseama				button-hotplug				nvrpm	mpc85xx
			px5g							otrx				firmware-utils	apm821xx
			mklibs							w1-gpio-custom					
			gettext							rbcfg					
			sstrip							trelay					
										owipcalc					
										rtc-rv5c386a					
										iwcap					
										usbreset					
										avila-wdt					
										fbtest					

Figure 14 - A subset of modules from the system architecture organized by clustering on term similarity from the word2vec model

4.2.4 MAPPING HARDWARE/SOFTWARE CONCEPTS TO MODULES

The ultimate goal of this research project is to identify and measure modularity violations in cyber-physical systems. In Task 1, we explored three modular decomposition criteria. In Task 2, we focused on exploring approaches for understanding key hardware concepts in cyber-physical systems. The next step is to map the conceptual model developed through Task 2 to the modular structures developed through Task 1. This mapping is what will enable us to identify which modularity violations that driven by hardware concepts.

Our first attempt was to map hardware and software concepts to the modules based on the package structure for the OpenWrt. We retrained the word2vec model with an expanded corpus and used the most similar function from *gensim* to extract the most similar words to both “hardware” and “software”. For each module, we extracted the top 10 most relevant words to “hardware” but not “software”, and the top 10 most relevant words to “software” but not “hardware”. For example, the module “firmware-utils” is an important utility package for

supporting firmware related processing. We found that this module is both relevant to software concepts and hardware concepts. As show in Table 3, the module “firmware_utils” is relevant to the following hardware and software concepts. In this case, the hardware and software concepts are indeed appropriately differentiated. The result shows that the current approach can extract and separate hardware and software concepts mapped with each module.

We conducted a more extensive manual verification of 50 cases and, we found that 60% of hardware concepts can be truly confirmed as hardware related. For the remaining 40%, we were not able to unambiguously classify the terms manually due to the nature of the project. In other words, even after manual review of the project documentation, we could not easily determine whether some terms should be classified as hardware or software related. In OpenWRT, software and hardware are so intertwined that many terms seem to have relationships to both hardware and software concepts. In retrospect, we should not have expected a purely binary classification to be effective. While there are undoubtedly low-level concepts that are purely hardware or purely software, these concepts are ultimately grouped into higher-level concepts that relate to both. “firmware-utils” is a good example of this. Thus, we needed to classify concepts as hardware, software, or both. In future work, we may also need to add a “none of the above” group to account for concepts that are purely related to project management.

Table 3 - Firmware_utils Hardware and Software Concepts

Firmware_utils	
Hardware concepts	mktplinkfw, tplink-safeloader, crc32, wrt350nv2, wrt400n, squashfs, mktplinkfw2, seama, gzip, generation,
Software concepts	mkimage, sstrip, polarssl, iconv, wolfssl, libiconv, -cmdline, -standalone, gettext, patch-image,

4.2.5 MANUAL VALIDATION

To test the accuracy of the domain concept learner, we applied it to two different types of project data from OpenWrt: Change messages (42,250 total) and Source file names (926 total). If the domain concept learner works, it should be able to group the evaluated items as hardware related, software, related or unknown/mixed.

For each change message, we calculate a relevance score that measures its relevance to hardware concepts. The score is in the range of [-1, 1]. The distribution of the scores of all the 42250 change messages is shown in Figure 15.

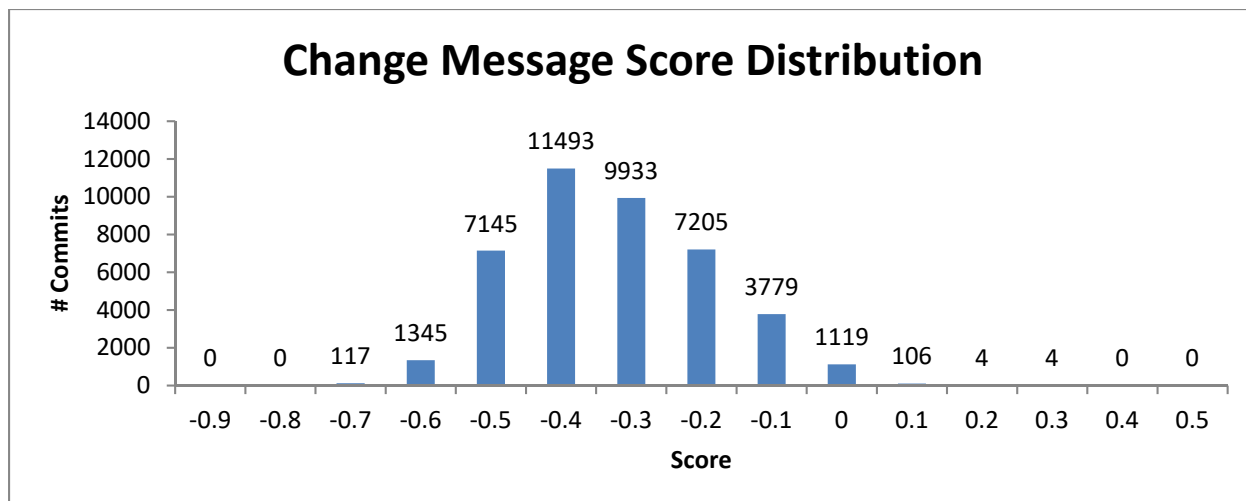


Figure 15 - Change Message Score Distribution

Next, we randomly selected 264 messages from these 42250 change messages. Without looking at the calculated score, we manually categorized each message into three types: 1) Software related, which are purely related to software concepts; 2) Hardware related, which are purely related to hardware concepts; and 3) Un-determined, due to mixing hardware and software terms, lacking necessary information, or we cannot understand.

Then we compared the scores of the change messages being manually tagged into “software related” and “hardware related”. The distribution of the score is shown in Figure 16. The figure suggests that the score of software related change messages and that of the hardware related change messages are clearly separated into two distinctive groups.

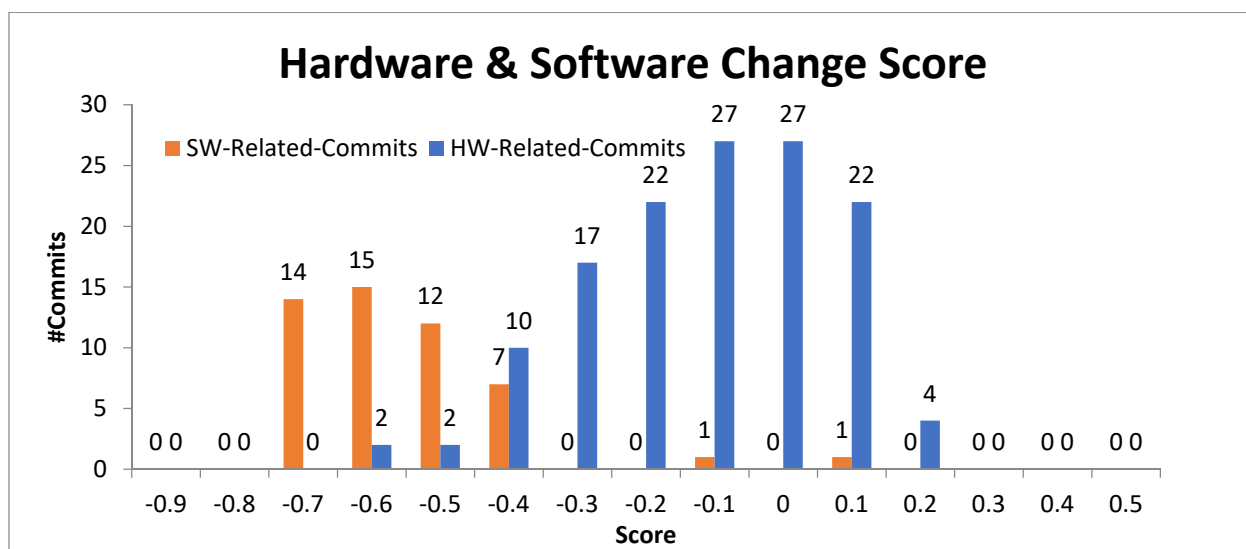


Figure 16 - Score Distribution for Manual Tagged Change Messages

The results can be seen more clearly on the violin chart of the score distribution of the three manually tagged groups, as shown in Figure 17. These results verified that the score of software related changes and hardware related changes are separated very clearly. In addition, we observed that software changes reside in the score region of $[-0.4, -1]$, hardware changes reside in the region of $[-0.3, 1]$, while the remaining reside in the region of $[-0.4, -0.3]$. This suggests the changes cannot be categorized as purely software or hardware related. The manual verification shows the effectiveness of the concept learner.

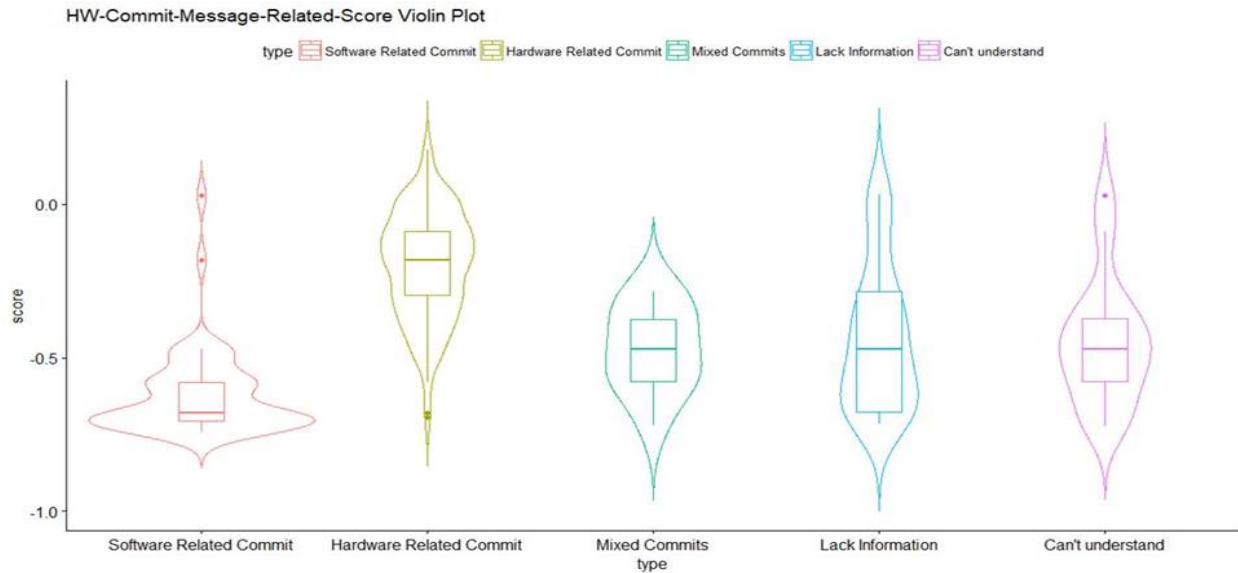


Figure 17 - Violin Chart for Change Message Scores in Different Groups

We conducted a similar verification process on the source files of OpenWrt. That is, for each source file, we analyze its file name using the concept learner and calculate a score that indicates its relevance to hardware concepts. The score follows the distribution shown in Figure 18 for a total of 923 source files. However, the manually tagging verification indicates that both software and hardware related source files could have very low score, as shown in Figure 19. As confirmed in Figure 20, the violin chart of the file name scores in the three different groups (software, hardware, undetermined), we cannot separate source files into purely software or hardware using the concept learner. We believe this is because the majority of file names had few occurrences in the corpus. Consequently, the algorithm had little contextual information to learn from.

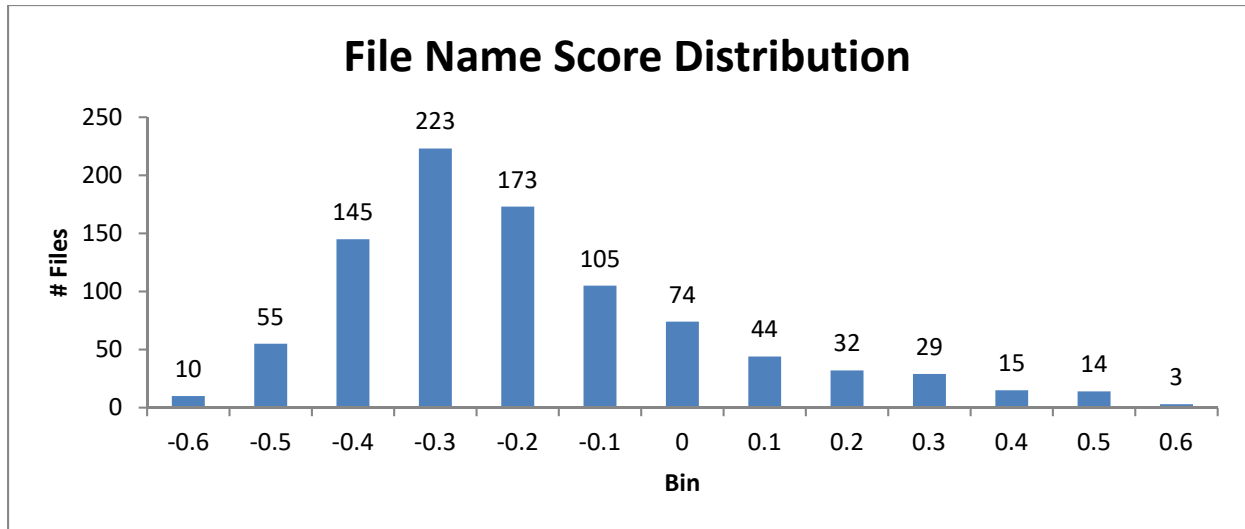


Figure 18 - File Name Score Distribution

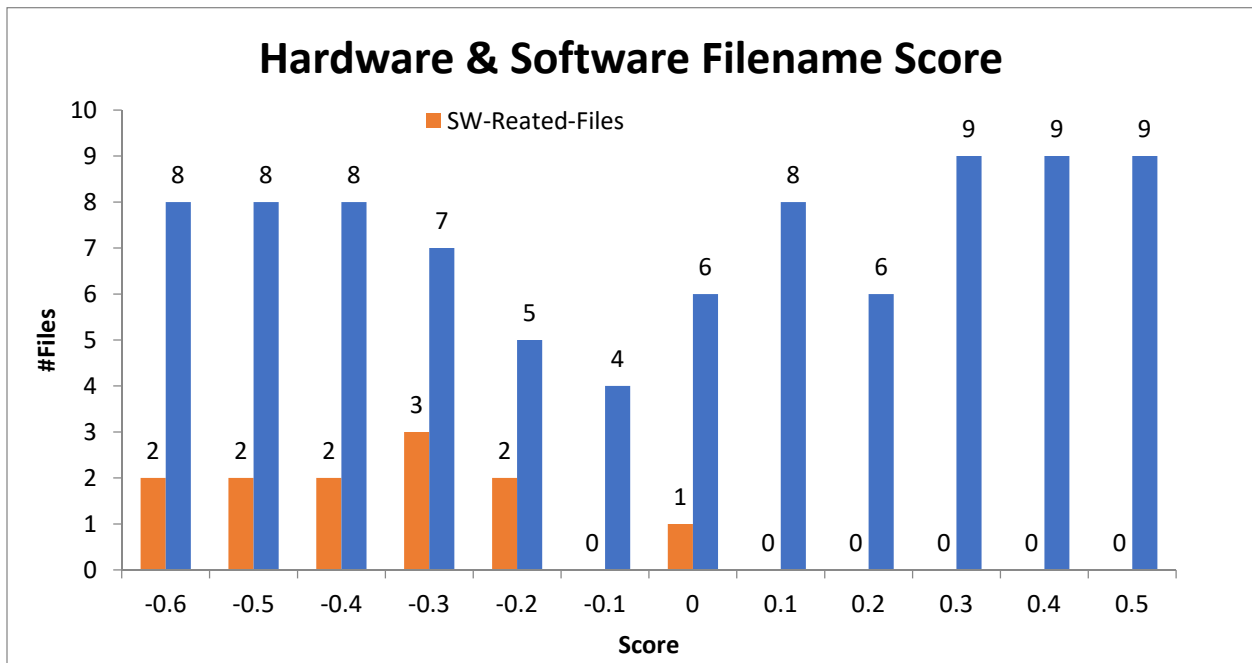


Figure 19 - File Name Score Distribution of Manual Tagged File Names

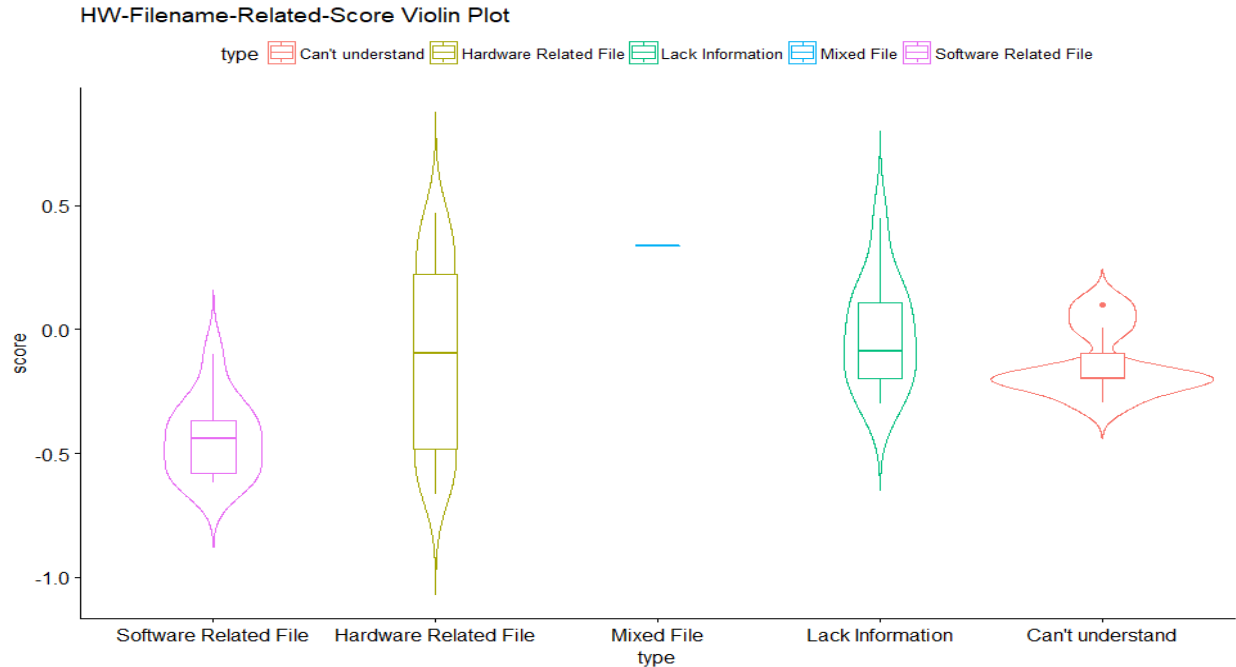


Figure 20 - Violin Chart of File Name Scores in Different Groups

4.2.6 TESTING THE DOMAIN CONCEPT LEARNER USING MD PnP

After determining that the domain concept learner approach worked on OpenWRT, we began testing on MD PnP. As with OpenWRT we extracted documentation from the project website and change histories and comments from the GIT repository. MD PnP has much less documentation than OpenWRT, but since MD PnP is an academic project, we were able to enlarge the corpus using academic publications.

After pre-processing we applied the word2vec algorithm to the MD PnP corpus. We ran parametric excursions minimum word count thresholds of 1, 5, 10, 15, and 20. Lower minimums risk introducing too much noise while higher minimum risk missing important information. The excursions enable us to determine the balance of signal to noise. To do this, we clustered the words using both k-means and Gaussian mixture models based on the cosine distances learned during each excursion.

The next step was to determine which keywords would be used to assign scores in the domain concept learner. We also ran an elbow analysis on the k-means clustering and found the elbow at 5 clusters. We then sorted contents of these clusters using word frequency. Finding the relevant hardware and software clusters we extracted the following keywords:

- **positive:** hardware, device, adapter, electrocardiogram, ecg, pump, display, xray, anesthesia, component, equipment, button, monitor, ventilator, webcam, beaglebone, screen, power, oximeter, instrumentation, tracker, station, camera, ultrasound, switch, scan, usb, laptop, router, electrode, rtls, gas, port, fpga

- **negative:** app, java, network, build, program, data, code, class, function, byte, file, software, virtual, implementation, loop, simulator, api, script, bug, algorithm, bits, fix, hack, method, plugin, call, style, infinite, json, kernel, leak, null, parameter, process, string, variable, debug

Here positive merely refers to the fact that we want to assign higher scores to hardware related words, negative means that we want to assign lower scores to software related words. The almost immediate observation was that there is considerable overlap between the negative words selected for both MD PnP and OpenWRT. However, there was not much overlap among the positive. This result reaffirmed the motivation behind creating the domain concept learner. While the “cyber” concepts may be common across systems, the “physical” concepts are likely to be system specific. For example, in MD PnP many of the positive words were medical terms.

To get these lists of positive and negative words down to something more manageable, we manually extracted 16 keywords that might highlight potential changes in HW concepts within the medical field (or for MD PnP in particular). The rationale behind the extraction was:

- Selected words that suggest the act of changing something:
 - o change, adjust, modify, switch, refine, substitute, fix, correct, hack
- Selected words specific to the medical fields and that represent equipment that exists in the MD PnP laboratory:
 - o ventilator, oximeter, pump
- Selected words of more general equipment and HW components used in the MD PnP laboratory:
 - o machine, monitor, FPGA, beaglebone

With the keywords selected, we were then able to compute the same scores for MD PnP change messages, comments and filenames as we did for OpenWRT. We then performed a manual validation on these results just as we did for OpenWRT. When we did, we found that we did not achieve strong separation between hardware and software changes messages. We conducted several experiments on variations of scoring algorithm, but the results were not improved.

4.2.7 INVESTIGATING THE LIMITATIONS OF THE DOMAIN CONCEPT LEARNER FOR MD PnP

In order to understand why the concept learner did not work as well on MD PnP, we manually examined all of the 1600 change messages from MD PnP and categorized the messages into the following categories:

- Hardware related: such as db205f49: “adds nellcor, masimo, Philips”
- Software related: such as a572d2c3: “Adds support for builtin topics to InstanceModelImpl and also support for iterating without registering for callbacks.”
- Hardware and Software Mixed: such as d017d82f: “Pass an optional transitionNote with StateMachine transitions. Also do not throw RuntimeException from Serial Intellivue

when serial port unavailable. Instead connect now returns false and the process terminates gracefully.”

- Unsure: such as 1dae724d: “Loosens constraints on participant liveliness. Refines QoS policies for heartbeat samples.”
- Lack Information: 2d594835: “placeholders for the future”
- Administrative: 55d09c9d: “Merge branch 'master' into patient”

The distribution of change messages based on manual verification is shown below:

Number of selected commit messages	1600	100%
Hardware	255	16%
Mixed-Hardware and Software	104	7%
Software	762	48%
Unsure	120	8%
Administrative	162	10%
Lacking Information	197	12%

The data indicates that most changes in MD PnP are due to software concepts. Only 16% changes are triggered by hardware concepts. 7% of the changes involved mixed software and hardware concepts. The results suggest that the data in MD PnP are indeed separated into Hardware and Software concepts. However, our current Concept Learner is not effective in differentiating the two types. As shown in Figure 21, the scores of hardware and software change messages significantly overlap.

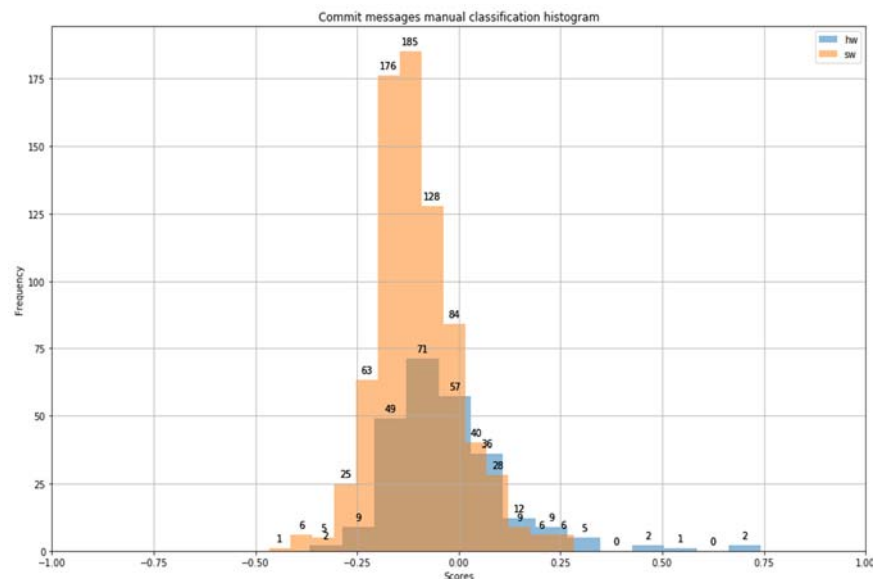


Figure 21 – Histogram of scores assigned MP PnP change messages manually classified as hardware related or software related

We believe that the limitation of the Concept Learner on MD PnP is due to two reasons:

1. The size and quality of the training dataset from MD PnP is much lower than that from OpenWrt. When we use the project documents from OpenWrt, we were able to find a large amount of supporting documents with well-defined hardware concepts. However, we did not find such good quality and quantity data from MD PnP. As with any machine learning technique, sufficient data is required to train the model.
2. By the nature of MD PnP, it also involves domain concepts, such as medical terms, which can be mapped into both software and hardware components. However, in this study, we focused the concept learner on binary classification. This approach lost some of the nuance that may be necessary for proper classification. A broader classification scheme will be considered in future research.

4.3 Analyzing and Diagnosing Modularity Violations

By combining the results of the decomposition analysis with the domain concept learner, we were able to analyze and diagnose potential modularity violations for each system we studied. The following section describes these results for OpenWRT and MD PnP respectively.

4.3.1 MODULARITY VIOLATIONS IN OPENWRT

First, we selected 542 changes in OpenWrt that happened between June-2016 and July-2018. Based on the sampled manual verification, we chose a threshold of 0.47 to separate hardware changes and software changes. It turns out that 121 of the 542 changes are software related; while 421 out of the 542 changes are hardware related. This indicates that OpenWrt changes are heavily driven by hardware related concepts. To further confirm the effectiveness and correctness of the scoring from the concept learner, we sampled 10% of the 542 changes. The following table indicates that the False Positive for Software and Hardware changes is about 1% to 6%. Therefore, we confirmed that the separation based on the concept learner scoring is effective.

Next, we investigated how software and hardware concepts are likely to trigger changes and potential modularity violations in different modular perspectives. The key findings will be discussed as follows:

In each modular perspective, some modules are more hardware related and some are more software related. We achieve this leveraging the VfC measurements proposed in Section 4.1.4. We calculate two versions of VfC for each module, one based on only software changes and one based on only hardware changes. For example, in the development view, Table 4 shows module “network” and “libs” are more relevant to software concepts; while module “linux” is more related to hardware concepts.

Table 4 – VFC scores for the development view of OpenWRT

Inner Module	Module Size	SW Change VFC	HW Change VFC	SW/HW VFC Ratio
linux	455	0.9	1.34	0.67
firmware-utils	91	2.13	1.47	1.45
kernel	85	0.57	0.54	1.06
network	69	0.82	0.48	1.71
libs	68	0.12	0.03	4.00

In the organizational perspective, different vendors show different expertise that focus on software or hardware specifically. As shown in Table 5, two vendors are software experts, while vendor 5 is a hardware expert.

Table 5 – VFC scores for the organizational view of OpenWRT

Inner Module	Module Size	SW Change VFC	HW Change VFC	SW/HW VFC Ratio
Vendor 1	151	1.4	0.8	1.75
Vendor 2	130	0.67	0.6	1.12
Vendor 3	107	0.46	0.48	0.96
Vendor 4	32	0.17	0	Null
Vendor 5	31	0.17	0.42	0.40

Most interestingly, in the hierarchical perspective, the lower layers, which provide the most fundamental functions of the system, seem to be more relevant to hardware concepts. This indicates that hardware concepts are indeed the fundamental change trigger in OpenWrt. The detailed data is shown in Table 6. The blue lines are the hardware related layers.

Table 6 – VFC scores for the hierarchical view of OpenWRT

Inner Module	Module Size	SW Change VFC	HW Change VFC	SW/HW VFC Ratio
I0	88	0.1	0.28	0.36
L0	58	0.15	0.32	0.47
L1	23	1.12	1.63	0.69
L2	90	0.48	0.38	1.26
L3	11	1.56	0.85	1.84
L4	20	1.29	0.94	1.37
L5	10	0	1.25	0.00
L6	1	0	0	N/A
L7	594	1.29	1.24	1.04

Both software concepts and hardware concepts could become the “shared secrets” that trigger modularity violations in different perspectives. In particular, the hardware secrets are usually related to supporting a specific type of hardware device in OpenWrt. For example, in the Figure 22, we separate software related changes and hardware related changes in the development modular view of OpenWrt. We observed that software and hardware concepts are equally likely to cause cross module changes (i.e. modularity violations). The distinction is that all the 16 hardware related modularity violations are between only three components: “Linux”, “Firmware-

utils”, and “Boot”. In comparison, the software related modularity violations are more likely to be sprinkled around different modules in the system. The most interesting finding is that 11 out of the 16 hardware modularity violations explicitly mentioned that the change is based on other chips, such as the TP-Link Archer C7 v4, a dual-band AC1750 router, and the Qualcomm/Atheros QCA9561+QCA9888. Therefore, the modularity violations are shared support for hardware devices from the software side.

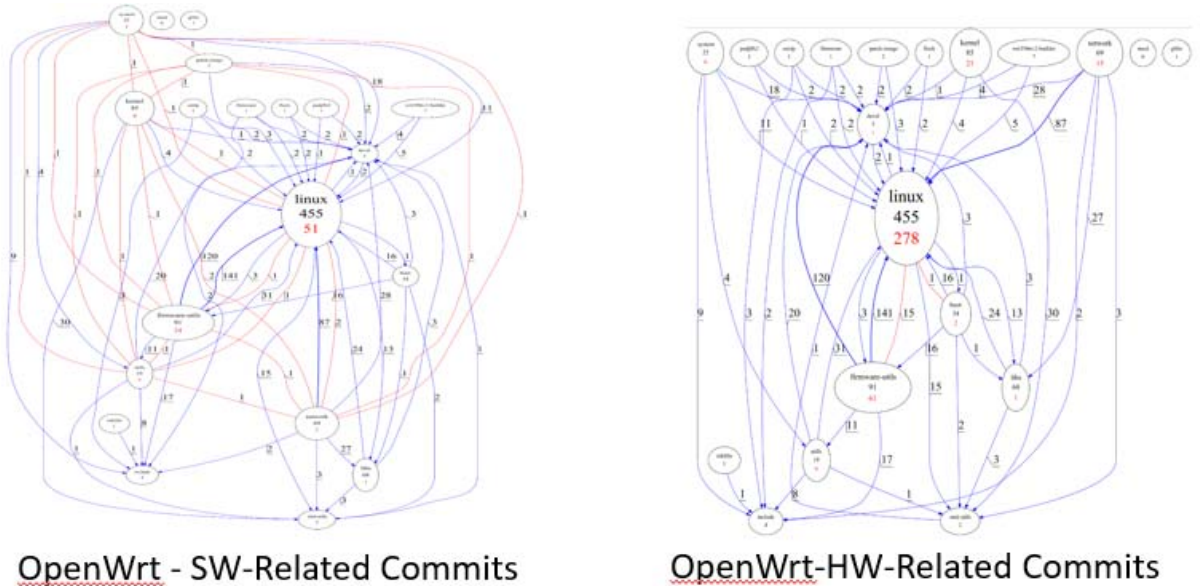


Figure 22 – Changes histories for OpenWRT from the development view perspective

We made similar observations from the other modular view: organization and hierarchical. Both software and hardware concepts are likely to cause modularity violations in OpenWrt. We investigated the general types of software and hardware changes that cause modularity violations (Figure 23). We found that: software triggers to modularity violations are usually house-keeping activities, bug fixing, and achieving other quality goals, such as performance; hardware triggers to modularity violations are usually due to the support, or updates of hardware devices, such as a single chip or multiple chip support.

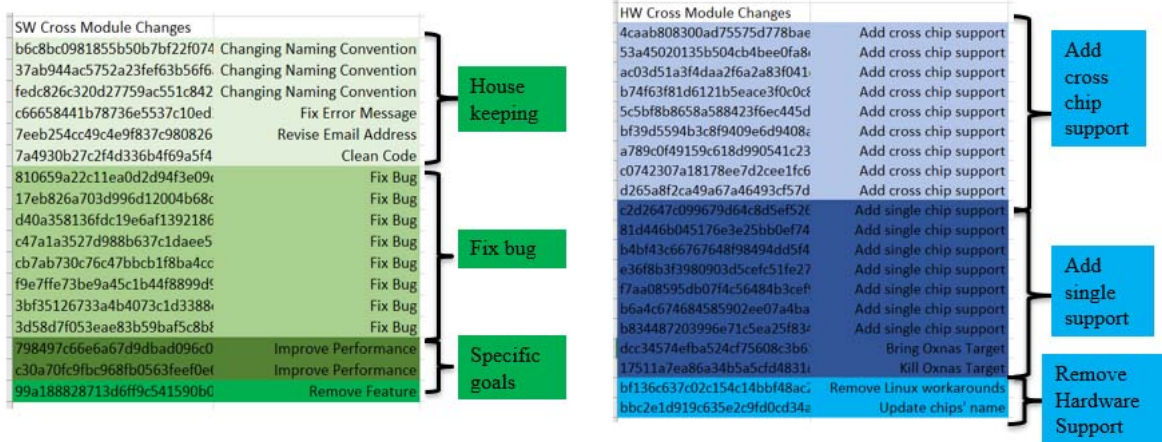


Figure 23 – OpenWRT cross module change analysis

4.3.2 MODULARITY VIOLATIONS IN MD PnP

After completing the analysis on OpenWRT, we performed an identical analysis on MD PnP. Compared to OpenWrt, there are 2.6 times more software triggered changes (619) than hardware triggered changes (236) in MD PnP, a significant difference between the two projects.

However, in the coarse development view, hardware concepts are more likely to cause cross modules changes than the software concepts. The detailed data are shown in Table 7. In this view 5% of cross module changes are triggered by software issues. In contrast, 20% of cross module changes are triggered by hardware issues. The implication is that the software concepts are better contained within each module, while hardware concepts are more likely to cause cross module concerns in MD PnP. We further visualized the distribution of software and hardware cross modules in a graphical view as shown in Figure 24. Cross-module changes, whether software or hardware triggered, are mostly between the modules “devices” and “interopr-lab”. As shown in Table 8, the VfC scores indicate that these two modules involve an almost equal amount of software and hardware changes. However, module “data-types” mainly involves software concept triggered changes. This observation is consistent with the naming convention of the modules.

Table 7 - Software and Hardware Triggered Cross Module Change in MD PnP

Development-Coarse	Change Type	Change Times	Change Percentage
Software Change	Inner Module Changes	477	94.64%
	Cross Module Changes	27	5.36%
Hardware Change	Inner Module Changes	170	79.81%
	Cross Module Changes	43	20.19%

Table 8 - VfC scores for coarse development view in MD PnP

Inner Module	Module Size	SW Change VfC	HW Change VfC	SW/HW VfC Ratio
interop-lab	382	1.76	1.78	0.99
devices	305	0.3	0.37	0.82
himss-2013	69	0.05	0	N/A
data-types	40	0.71	0.12	6.06

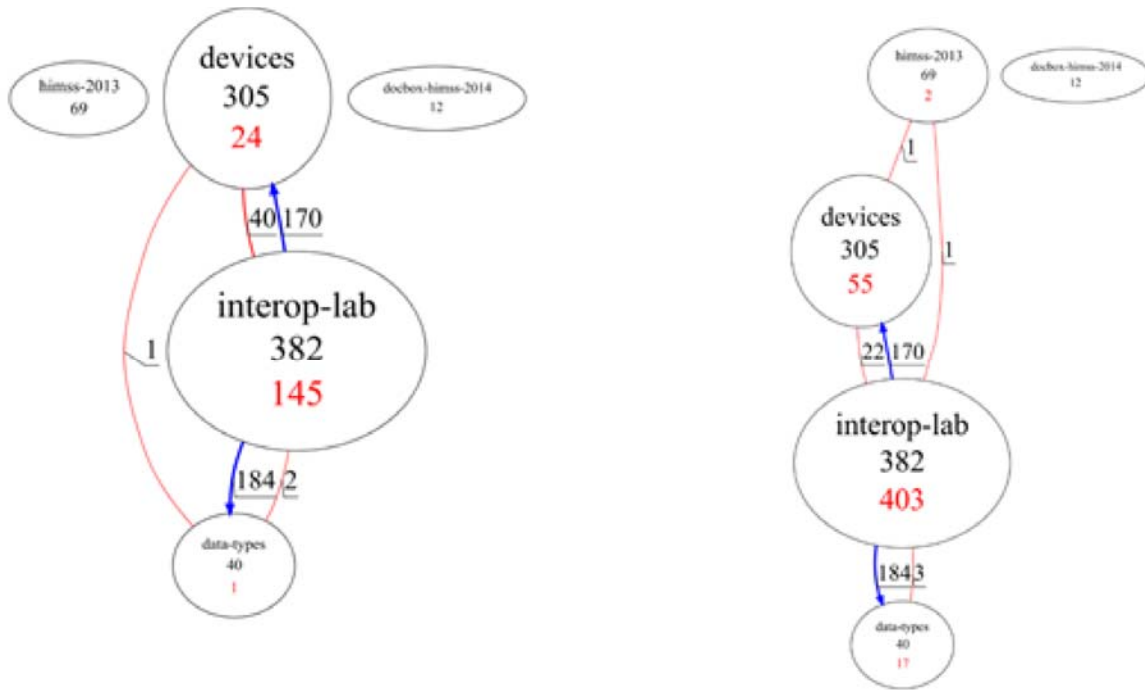


Figure 24 - OpenWRT cross module change in coarse development view (hardware on left; software on right)

In the fine-grained development view, there are nearly equal amounts of cross module changes and inner module changes; and the software concepts and hardware concepts are equally likely to trigger cross module changes. However, in this view, we can better distinguish which fine-grained modules are driven by software changes and hardware changes respectively. As shown in Table 9, the modules “demo-apps”, “philips”, “x73-idl-rti-dds” and “common” are more likely to be driven by software changes; while the module “demo-devices”, “demo-guis-swing”, and “fluke” are more likely to be triggered by hardware changes.

Table 9 - VfC scores for the fine-grained development view in MD PnP

Inner Module	Module Size	SW Change VfC	HW Change VfC	SW/HW VfC Ratio
demo-apps	224	1.93	1.3	1.48
philips	203	0.17	0.07	2.35
demo-devices	92	1.72	2.91	0.59
infusion-pump-ui	60	0.06	0	N/A
x73-idl-rti-dds	39	0.78	0.12	6.32
common	23	0.77	0.21	3.72
demo-guis-swing	21	1.02	1.37	0.74
fluke	20	0.09	0.24	0.37

In the organizational view, we observed that some vendors are more likely to perform software triggered changes while others are more likely to perform hardware triggered changes. As discussed earlier, this suggest different vendors' expertise emphasizing software changes or hardware changes. The detailed data are shown in Table 10. Vendor 1 is more likely to make hardware related changes. Vendor 5 exclusively makes hardware changes. Vendor 2 is 4.45 times more likely to make software related changes; and vendors 3 and 4 exclusively make software changes.

Table 10 - VFC scores for the organizational view in MD PnP

Inner Module	Module Size	SW Change VFC	HW Change VFC	SW/HW VFC Ratio
Vendor 1	648	1	1.12	0.89
Vendor 2	86	1.48	0.33	4.45
Vendor 3	19	0.1	0	N/A
Vendor 4	5	0.36	0	N/A
Vendor 5	4	0	1.02	0.00

Finally, we analyze the hierarchical view (Table 11). First, different layers will have different VFC scores, indicating how likely it is that each module is driven by software or hardware changes. There are three modules with almost the same amount of software and hardware changes. Seven modules are more likely to be triggered by software changes, and four modules are more likely to be triggered by hardware changes. In addition, five of the seven hardware driven modules are in the first three layers. This is consistent with what was observed in OpenWrt. Hardware concepts serve as fundamental assumptions for these cyber-physical systems. Thus, these modules are in lower layers and are depended upon by other modules. A complementary observation is that the four software driven modules are at relatively higher layers (layer 4 to 7).

Table 11 - VFC scores for the hierarchical view in MD PnP

Inner Module	Module Size	SW Change VFC	HW Change VFC	SW/HW VFC Ratio
I0	27	0.38	0	N/A
L0	63	0.9	0.24	3.83
L1	68	1.33	1.2	1.11
L10	15	2.75	0.99	2.79
L11	13	2.78	0.57	4.88
L12	4	1.94	1.85	1.04
L13	313	0.67	0.66	1.01
L2	107	0.89	0.28	3.22
L3	110	0.45	0.13	3.31
L4	14	2.03	9.53	0.21
L5	12	1.08	3.09	0.35
L6	3	13.77	29.65	0.46
L7	40	1.03	3.15	0.33
L8	2	9.04	3.71	2.44
L9	17	4.1	2.18	1.88

Finally, we looked into the change messages that developers wrote when they made changes to MD PnP system. The goal is to gain an intuitive understand of why and what are the triggering factors for cross module changes (modularity violations) in different architecture perspectives. We found that each case is associated a unique reason. But there are three general triggers. First, the developers are trying to bind a new hardware device to the system, and this change impacts

multiple modules. For example, a developer says “Bind device node labels to makeAndModel;” when he/she make changes to source files from modules that belong to two different vendors. Second, the developers are trying to add device driver or adapter, and the change impacts source files from multiple modules. For example, a developer says, “another pass at spring-based device driver”, when changing source files from two different hierarchical layers at the same time. The last general type is fixing issues with existing devices or domain related concepts. For example, a developer says “fixed bug for PB840 related to alarms which dont have one of the limi...” when fixing a bug involving two different modules in all three architecture perspectives. PB840 is a typical ventilation device used in medical domain. Compared to OpenWrt, the factors that trigger modularity violations in MD PnP seem to differ from case to case in the details. In OpenWrt, we observed consistent triggering factors, such as chips, frequently triggering modularity violations.

4.4 Modularity Violation Demonstrator

The decision framework demonstrator for analyzing modularity violations consists of two stages. The first stage trains the domain concept learner using the project documentation. The results of this analysis are then fed into the modular violation demonstrator, which enables the user to view the cyber-physical system from different perspectives and levels of resolution. It also enables them to identify candidate modularity violations and determine which are hardware vs software related. Figure 25 shows the training process for the domain concept learner portion.

The screenshot displays a web-based interface for the 'Domain Concept Learner'. At the top, there is a button labeled 'Click here to toggle on/off the raw code.' Below this, the interface is divided into two main sections. The first section, titled 'Domain Concept Learner', shows two log entries: '2019-04-24 17:53:57.923521 : INFO : importing external libraries' and '2019-04-24 17:53:57.923521 : INFO : loading clean-up routines'. The second section, titled 'Word2Vec Model', displays a detailed log of the training progress. The log shows the model is in EPOCH 5 and provides progress updates every 100 examples, including the number of words per second, in_queue size, and out_queue size. The progress starts at 63.71% and reaches 86.87% by the end of the shown log entries.

```

Out[5]: Click here to toggle on/off the raw code.

Domain Concept Learner
2019-04-24 17:53:57.923521 : INFO : importing external libraries
2019-04-24 17:53:57.923521 : INFO : loading clean-up routines

Word2Vec Model
2019-04-24 18:07:00,182 : INFO : EPOCH 5 - PROGRESS: at 63.71% examples, 13747 words/s, in_queue 0, out_queue 0
2019-04-24 18:07:01,241 : INFO : EPOCH 5 - PROGRESS: at 64.42% examples, 13737 words/s, in_queue 0, out_queue 0
2019-04-24 18:07:02,743 : INFO : EPOCH 5 - PROGRESS: at 65.41% examples, 13724 words/s, in_queue 0, out_queue 0
2019-04-24 18:07:03,779 : INFO : EPOCH 5 - PROGRESS: at 66.08% examples, 13712 words/s, in_queue 0, out_queue 0
2019-04-24 18:07:04,866 : INFO : EPOCH 5 - PROGRESS: at 66.75% examples, 13684 words/s, in_queue 0, out_queue 0
2019-04-24 18:07:05,884 : INFO : EPOCH 5 - PROGRESS: at 67.43% examples, 13681 words/s, in_queue 0, out_queue 0
2019-04-24 18:07:06,925 : INFO : EPOCH 5 - PROGRESS: at 68.12% examples, 13667 words/s, in_queue 0, out_queue 0
2019-04-24 18:07:08,823 : INFO : EPOCH 5 - PROGRESS: at 69.61% examples, 13643 words/s, in_queue 0, out_queue 0
2019-04-24 18:07:10,570 : INFO : EPOCH 5 - PROGRESS: at 71.24% examples, 13594 words/s, in_queue 0, out_queue 0
2019-04-24 18:07:12,531 : INFO : EPOCH 5 - PROGRESS: at 73.19% examples, 13516 words/s, in_queue 0, out_queue 0
2019-04-24 18:07:14,424 : INFO : EPOCH 5 - PROGRESS: at 74.97% examples, 13452 words/s, in_queue 0, out_queue 0
2019-04-24 18:07:16,290 : INFO : EPOCH 5 - PROGRESS: at 76.65% examples, 13394 words/s, in_queue 0, out_queue 0
2019-04-24 18:07:17,975 : INFO : EPOCH 5 - PROGRESS: at 78.20% examples, 13359 words/s, in_queue 0, out_queue 0
2019-04-24 18:07:19,799 : INFO : EPOCH 5 - PROGRESS: at 79.89% examples, 13309 words/s, in_queue 0, out_queue 0
2019-04-24 18:07:21,662 : INFO : EPOCH 5 - PROGRESS: at 81.73% examples, 13253 words/s, in_queue 0, out_queue 0
2019-04-24 18:07:23,501 : INFO : EPOCH 5 - PROGRESS: at 83.37% examples, 13204 words/s, in_queue 0, out_queue 0
2019-04-24 18:07:25,335 : INFO : EPOCH 5 - PROGRESS: at 85.13% examples, 13159 words/s, in_queue 0, out_queue 0
2019-04-24 18:07:27,254 : INFO : EPOCH 5 - PROGRESS: at 86.87% examples, 13104 words/s, in_queue 0, out_queue 0

```

Figure 25 - Snapshot of domain concept learner training interface

The training process will generate a concept clustering based on the data supplied. For example, the K-means Clustering is one of the algorithms applied during the training process (Figure 26).

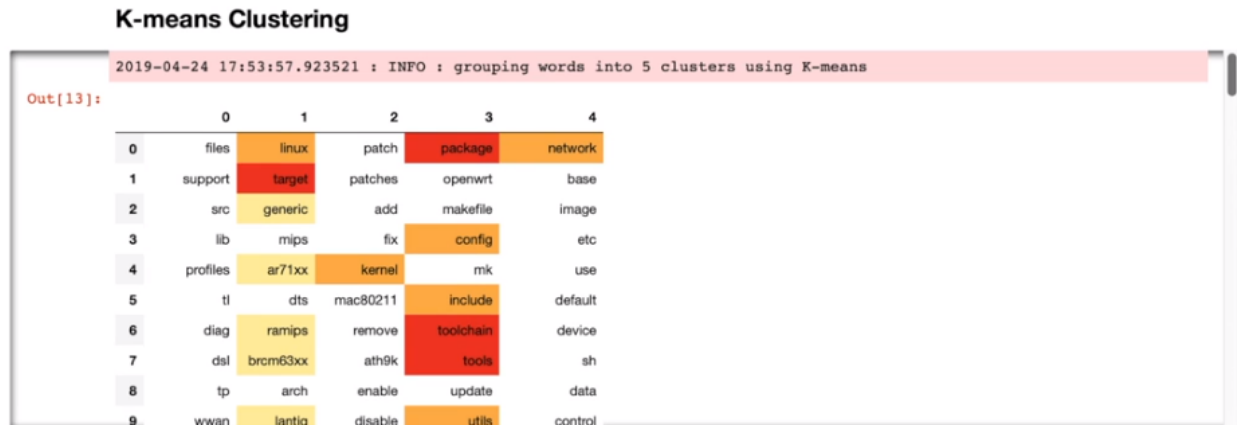


Figure 26 - Snapshot of domain concept clustering interface

After the model is trained, we can employ the concept learner to evaluate the relevance of an artifact of interest to hardware concepts. For example, in OpenWrt, we focused on the developers' change messages. Figure 27 shows the output of the evaluation process. For each single change message, the concept learner will output a floating point number to indicate its relevance to hardware concepts.

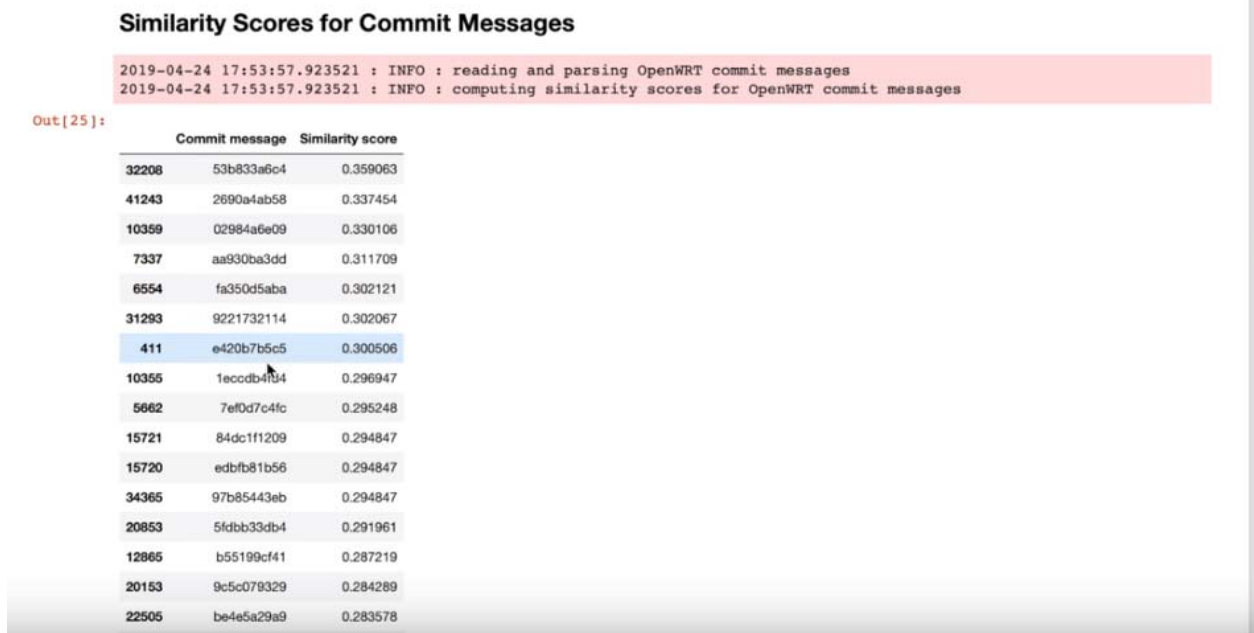


Figure 27 - Snapshot of scoring results of commit messages using the domain concept learner

The above information serves as the data interface to be fed into the Modularity Violation Demonstrator. As such, an analyst can focus on changes with a hardware relevance score higher than a given threshold. In this way, the analyst can flexibly tune what level of analysis is performed for modularity violations.

Figure 28 shows the internal structure of the modular decomposer. It first leverages existing tools to preprocess the project data, including the source code and the change history. Next, the user can choose to run through one of the three decomposing criteria as discussed in the previous sections, the development decomposer, the hierarchical decomposer, and the organizational decomposer. Next, based on any decomposing criteria, the user can calculate the co-change relationship among modules in the selected criteria. Finally, we have a Visualizer to take the data calculated from previous steps to render the modular structure, combining both the structural dependencies and the evolutionary coupling (i.e. co-changes) among modules. The different modular views in previous sections are generated using our module decomposer component.

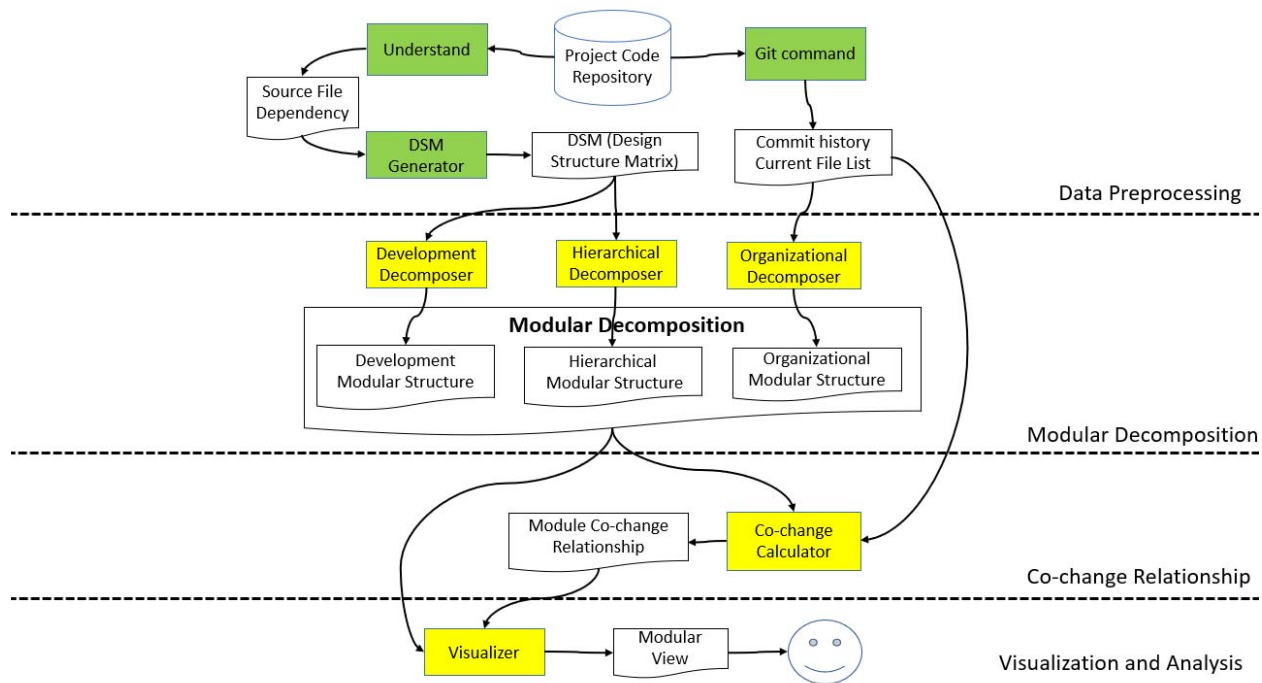


Figure 28 - Module Decomposer Component

Figure 29 to Figure 31 shows the three interfaces for generating the modular views following the three different decomposition criteria. Figure 29 shows the development view decomposer. The user can specify two levels of granularity: coarse and fine-grained. Figure 30 shows the hierarchical decomposer. Figure 31 shows the organizational decomposer.

The modularity violation demonstrator can discriminate between hardware driven and software driven changes. Figure 32 shows the co-change analyzer. This component calculates how modules in different architecture perspectives change together in the maintenance history. In particular, by taking the output of concept learner as the input to this module, one can separate software changes and hardware changes. The user inputs a score threshold to implement the separation thresholds as discussed in the analysis of OpenWrt.

Finally, the user can use the visualizer to analyze modularity violations using one of the modular views and the co-change relationship using component shown in Figure 33. The measurements, such as cross-module co-change, inner-module co-change, and value for change will be displayed on the dialog box on the bottom. The graphical views of different architecture perspectives will be generated as a .png/.jpeg file separately as shown in Figure 34.

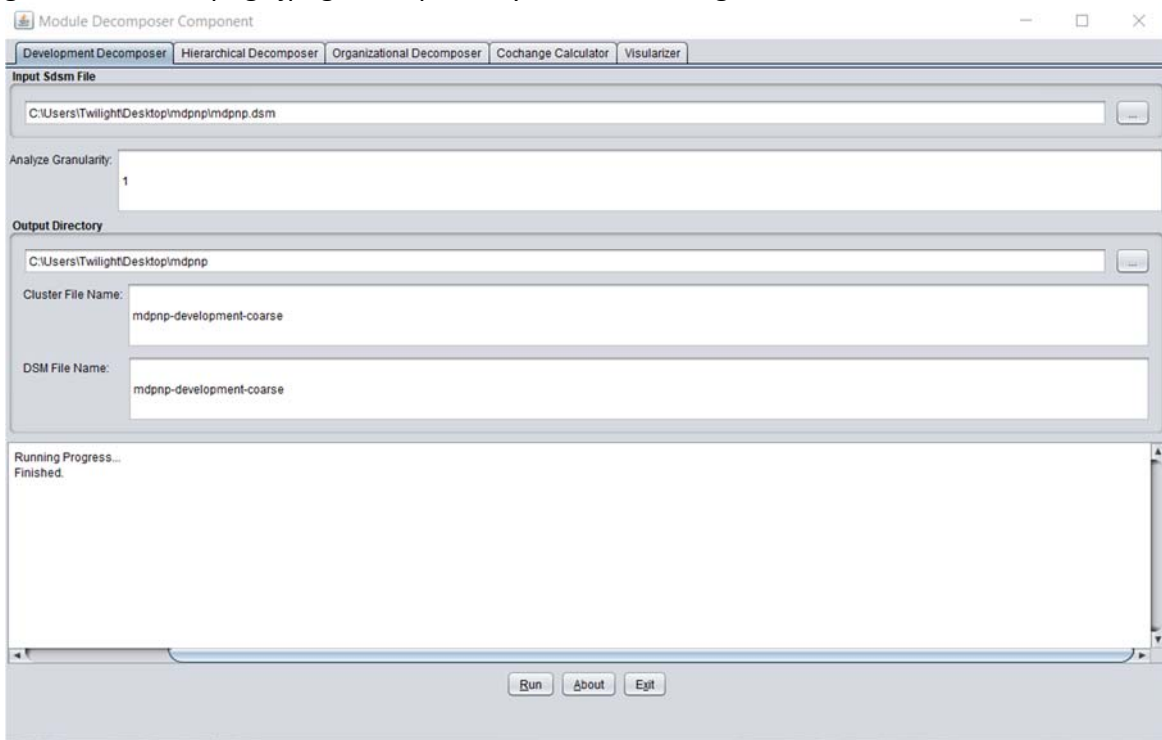


Figure 29 - Module Decomposer Component-Development

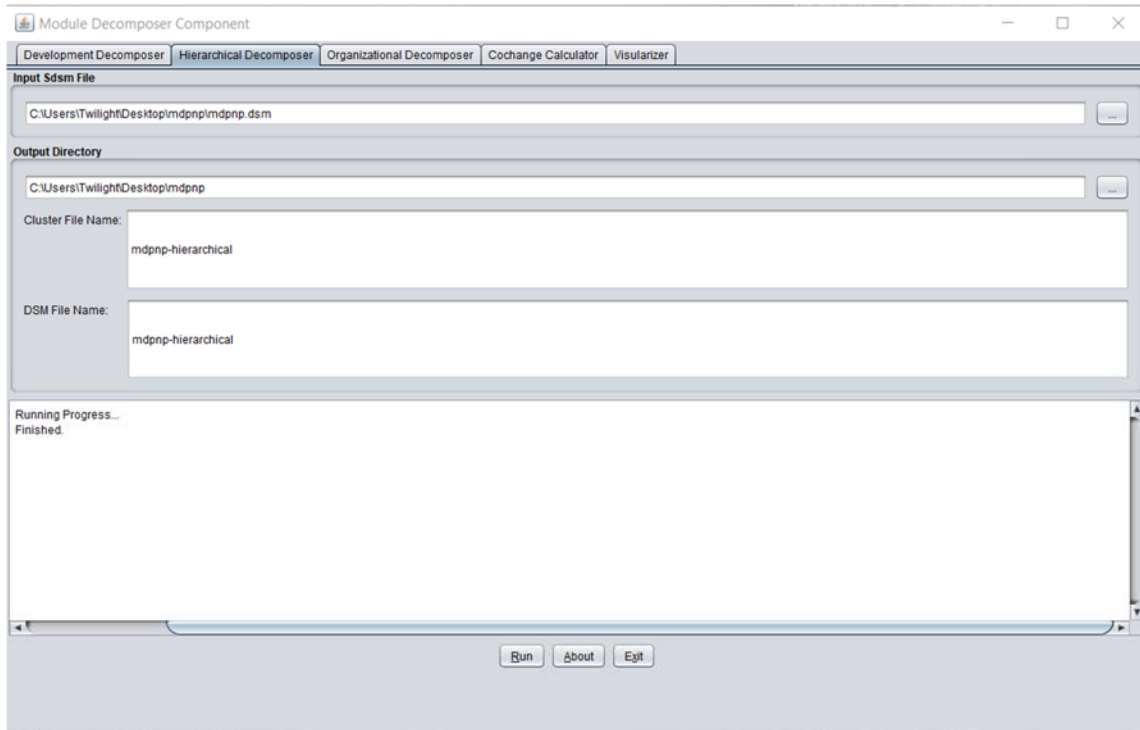


Figure 30 - Module Decomposer Component-Hierarchical

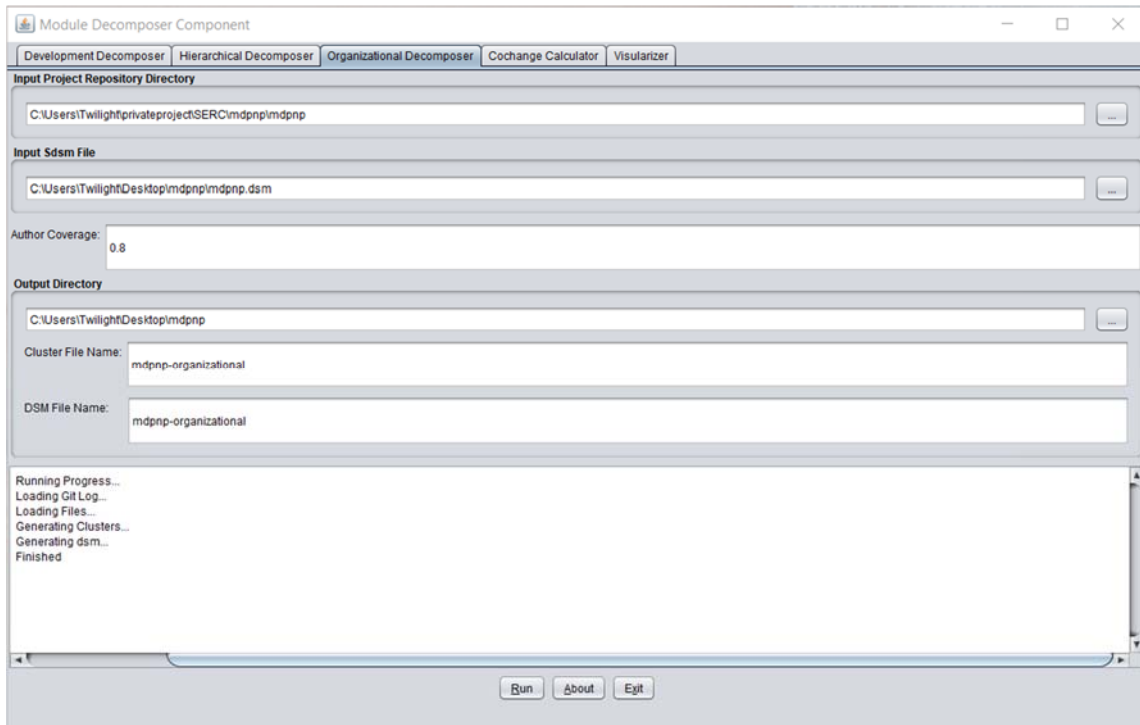


Figure 31 - Module Decomposer Component-Organizational

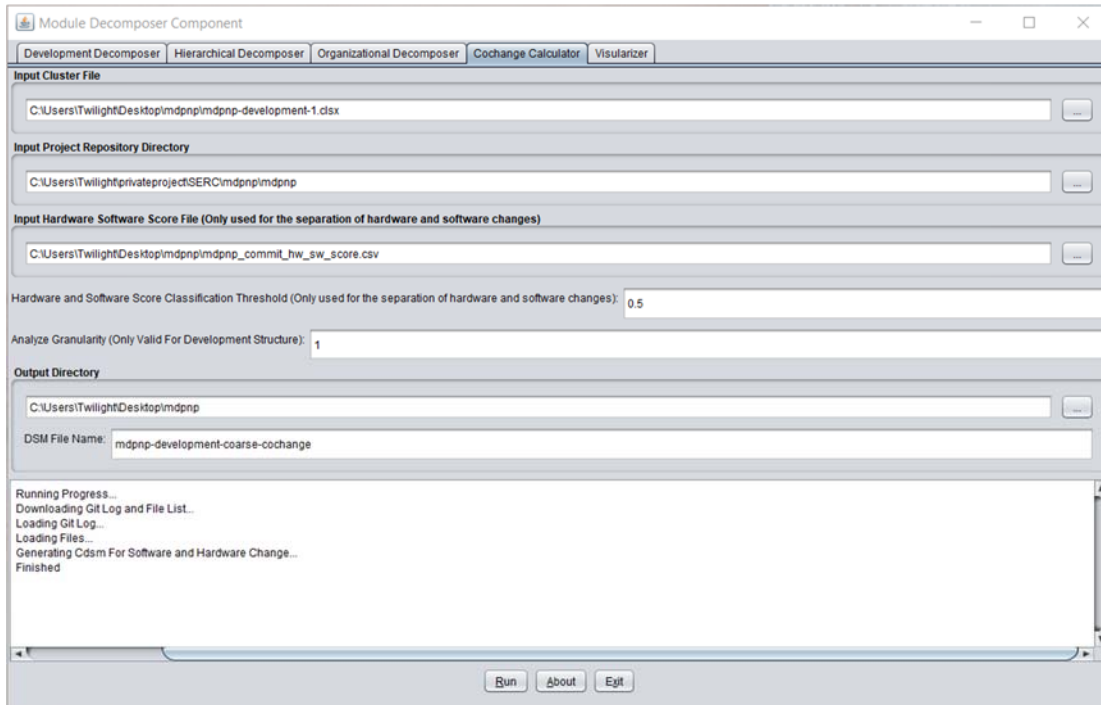


Figure 32 - Co-change Calculator that Separate Hardware and Software Changes

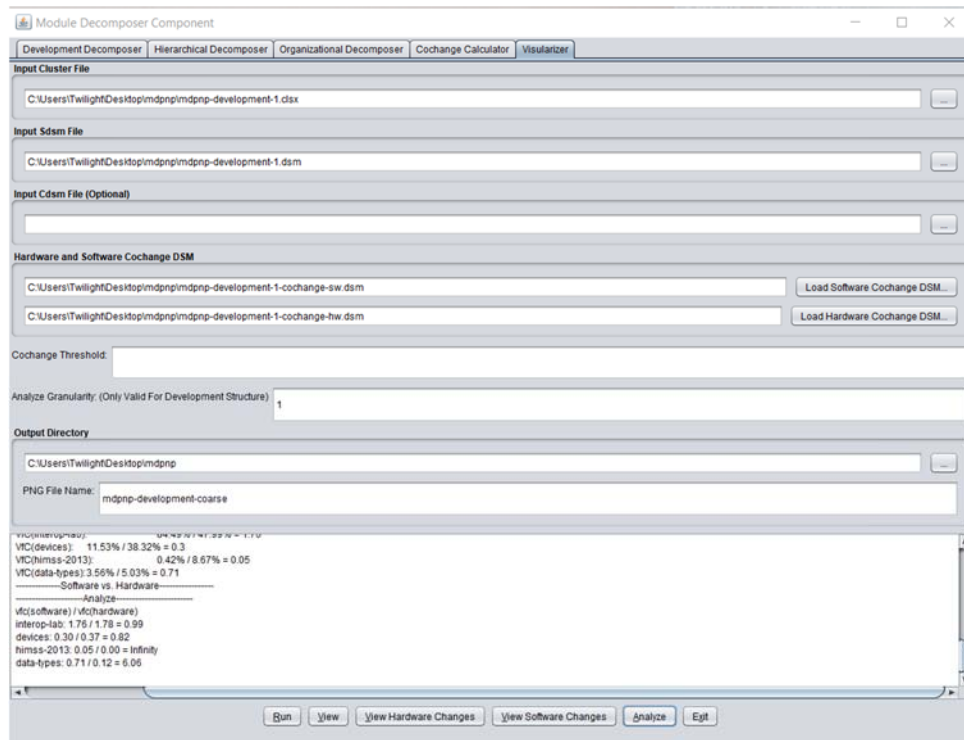


Figure 33 - Modularity Violation Visualizer/Analyzer

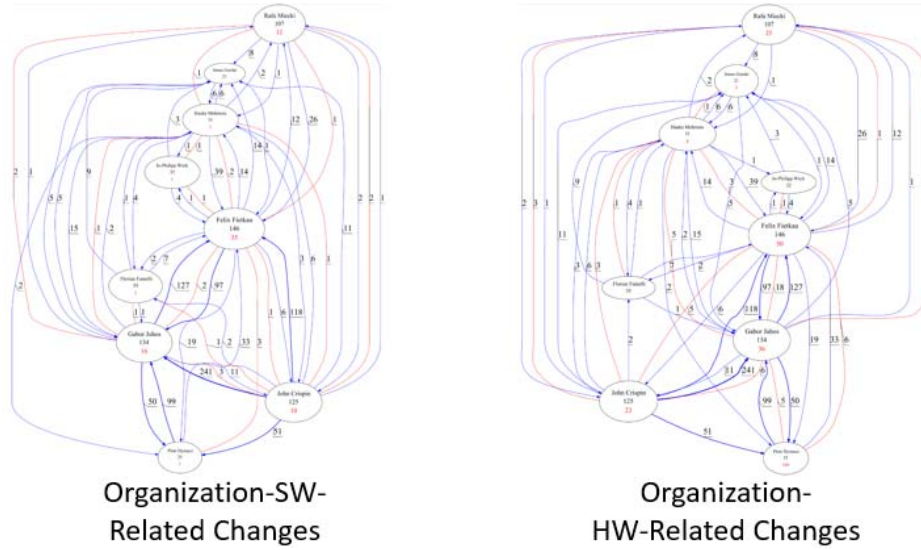


Figure 34 - Examples of modular views generated by the demonstrator tool

5 DISCUSSION

In this section, we summarize the answers to the three research questions based on our study results and discuss the implications.

Different modular decomposition criteria help to reveal the system architecture from different perspectives. The development modular view reveals how the developers treat the system as separate components in the development process. The cross modules changes (i.e. modularity violations) help to pin-point coupling across components in the development process. The hierarchical modular view reveals how the system could be implemented as sequential tasks. Although both projects can be treated as several sequential modules based on their structural design, the actual maintenance of the systems require a large amount of co-changes among sequential modules. The organizational modular view reveals how the systems are developed by different contributors/vendors. The associated modularity violations reveal the potential points of vendor lock-in. The implication is that different stakeholders should focus on different system architectures to identify modularity violations that are most relevant to their concerns.

The concept learner can effectively separate hardware concepts from software concepts in OpenWrt based on a learned distance metric. However, it was not as effective for MD PnP. The scores of hardware concepts and software concepts largely overlapped. We believe this result is due to 1) the volume of training data: OpenWrt has significantly larger amount of well-organized documentation than MD PnP; 2) the change messages in OpenWrt are in better quality than the change messages in MD PnP: about 30% of change messages in MD PnP do not contain sufficient information to distinguish software vs. hardware triggered changes; and 3) the MD PnP contains more complicated domain information than OpenWrt: the concepts in MD PnP may not be just in either software or hardware related. The implication is that a concept learner should not

simply treat a change as either software related or hardware related. There could be a more nuanced information structure in certain problem domains such as the medical domain relevant to MD PnP.

In OpenWrt, most (about 70%) software changes are triggered by hardware related concepts. In comparison, only 16% software changes are triggered by hardware related concepts in MD PnP. In OpenWrt, hardware triggered modularity violations have very consistent root causes: adding support for new hardware devices, such as chips. Software triggered modularity violations are related to project management, bug fixes, and general house-keeping activities. In MD PnP, 48% software changes are triggered by software related concepts and 16% software changes are triggered by hardware related concepts, such as adding new devices, device drivers/adapters, or fixing device related bugs.

6 CONCLUSIONS AND FUTURE WORK

While the results of RT-205 did successfully identify and measure hardware driven modularity violations in cyber-physical systems, it is important to emphasize that analysis techniques developed are retrospective in nature. We found modularity problems that have already occurred. While this information can be useful when re-architecting a system or determining how to subcontract system maintenance, what would be more useful to DoD are leading indicators that can be assessed early enough in the acquisition process to prevent modularity violations from being created in the first place. This work was a first step toward that goal, as it is necessary to identify and diagnose modularity violations in real systems to identify and validate leading indicators.

Based on our recent observations, we hypothesize that the modularity violations are caused by a misalignment between the conceptual models used to architect and design a cyber-physical system and the actual system architecture perspectives as the system is being built. For example, there may be misalignments between the structure of the organization or the domain conceptual model and the physical architecture of the system, and these misalignments lead to latent connections among nominally separate physical modules. While this hypothesis is conceptually appealing and reflects prior assertions such as Conway's Law, the techniques developed during RT-205 create the opportunity to actually test this hypothesis empirically. If successful, the results of these tests would lead to analysis techniques and metrics that would serve as leading indicators of modularity violations. We recognize that there are already techniques for traversing early stage systems engineering artifacts to search for inconsistencies. Instead, our goal is to identify which of these detected inconsistencies are likely to lead to modularity violations versus those that are more manageable.

Thus, our next research question to investigate is: Do discrepancies between the domain conceptual structure and different system architecture perspectives predict or provide early warning to modularity violations in CPS? In future work, we plan to answer this question via repeated experiments across a diverse set of cyber physical systems.

7 LIST OF PUBLICATIONS RESULTED

Xiao, L., Pennock, M. J., Cardoso, J. L. F. P. & Wang, X. A Case Study on Modularity Violations in Cyber Physical Systems, in revision for *Systems Engineering*, submitted April 3, 2019.

Xiao, L. & Pennock, M.J. (2018), Identifying and Measuring Modularity Violations in Cyber-Physical Systems, *CESUN 2018*.

8 REFERENCES

- [1] R. Mo, Y. Cai, K. R. and L. Xiao, "Hotspot patterns: The formal definition and automatic detection of architecture smells," in *Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on*, 2015.
- [2] P. Derler, E. A. Lee and A. S. Vincentelli, "Modeling cyber–physical systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13-28, 2012.
- [3] E. A. Lee, "Cyber physical systems: Design challenges," in *Object oriented real-time distributed computing (isorc), 2008 11th ieee international symposium on*, 2008.
- [4] E. A. Lee, "The past, present and future of cyber-physical systems: A focus on models," *Sensors*, vol. 15, no. 3, pp. 4837-4869, 2015.
- [5] P. González-Nalda, I. Etxeberria-Agiriano and I. Calvo, "Flexible, modular, standard, free and affordable model for CPS control applied to mobile robotics," in *Information Systems and Technologies (CISTI), 2015 10th Iberian Conference on*, 2015.
- [6] E. Jantunen, U. Zurutuza, L. L. Ferreira and P. Varga, "Optimising maintenance: What are the expectations for cyber physical systems," in *Emerging Ideas and Trends in Engineering of Cyber-Physical Systems (EITEC), 2016 3rd International Workshop on*, 2016.
- [7] C. Y. Baldwin and K. B. Clark, *Design rules: The power of modularity (Vol. 1)*, MIT press, 2000.
- [8] R. Baheti and H. Gill, "Cyber-physical systems," *The impact of control technology*, vol. 12, pp. 161-166, 2011.
- [9] I. Malavolta, H. Muccini and M. Sharaf, "A preliminary study on architecting cyber-physical systems," in *Proceedings of the 2015 European Conference on Software Architecture Workshops*, 2015.
- [10] A. Rajhans, S. Cheng, B. Schmerl, D. Garlan, B. H. Krogh, C. Agbi and A. Bhawe, "An architectural approach to the design and analysis of cyber-physical systems," in *Electronic Communications of the EASST*, 2009.

- [11] K. Sinha, *Structural complexity and its implications for design of cyber-physical systems*, Doctoral dissertation, Massachusetts Institute of Technology, 2014.
- [12] C. Cristalli, S. Boria, D. Massa, L. Lattanzi and E. Concettoni, "A Cyber-Physical System approach for the design of a modular Smart Robotic Cell," in *Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE*, 2016.
- [13] S. Wong, Y. Cai, M. Kim and M. Dalton, "Detecting software modularity violations," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011.
- [14] R. Mo, Y. Cai, R. Kazman, L. Xiao and Q. Feng, "Decoupling level: a new metric for architectural maintenance," in *Proceedings of the 38th International Conference on Software Engineering*, 2016.
- [15] L. Xiao, Y. Cai and R. Kazman, "Titan: A toolset that connects software architecture with quality analysis," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014.
- [16] R. Schwanke, L. Xiao and Y. Cai, "Measuring architecture quality by structure plus history analysis," in *Proceedings of the 2013 International Conference on Software Engineering*, 2013.
- [17] L. Xiao, Y. Cai, R. Kazman, R. Mo and Q. Feng, "Identifying and quantifying architectural debt," in *Proceedings of the 38th International Conference on Software Engineering*, 2016.