



Exploring the Taxonomy of Commits in Cyber-Physical Systems for Enhanced Error Fixes Investigation

Nikita Starovoytov
Applied mathematics department
Polzunov Altai State Technical University
Barnaul, Russia
0009-0007-0242-0198 

Sergey Staroletov
Applied mathematics department
Polzunov Altai State Technical University
Barnaul, Russia
0000-0001-5183-9736 

Abstract—Cyber-physical systems are a symbiosis of multi-level control systems that take into account the physical aspects of the functioning of target objects. Errors in such systems can be associated both with incorrect organization of the code and operation of the hardware, as well as with an incorrect understanding of physical laws and their numerical approximation. Continuing our previous work, we apply technologies for analyzing commits in git repositories of some well-known cyber-physical systems, followed by classification of messages from developers. As a result, we discuss the identified strong keywords and generalized fix messages that can reveal the main classes of bugs in these projects. The results of the work can be used in training and consulting on errors and vulnerabilities in complex systems.

Index Terms—clustering, fixing commits, errors classification, cyber-physical systems

I. INTRODUCTION

Modern software development environments are crucial for improving the efficiency and quality of software development processes. One such valuable tool is the git version control system [1], which is widely used among developers and provides a detailed change history in its repositories. Each change to the code is documented with a message written in natural language by the developer, making it easier to track the evolution of the software.

This work seeks to leverage data analysis methods to identify common types of errors in the source code of software used in cyber-physical systems. By analyzing the messages in git repositories, automated methods can be used to detect patterns of errors fixes for the code. This approach can help developers identify recurring issues and take preventive measures to improve software quality.

By conducting such analysis, software development teams can streamline their processes, reduce the occurrence of errors, and enhance the overall quality of the software being developed. By understanding the common types of errors and their main causes, developers can implement targeted solutions to address these issues and prevent them from occurring in future projects to ensure that software for cyber-physical systems is robust and reliable.

The rest of the paper has the following structure. In Section II, we discuss the background on commits to version control systems and cyber-physical systems. Section III is about the related works on errors detection in cyber-physical systems.

Section IV is devoted to the implementation plan. In Section V, we discuss the results we got by running our software to analyze some known repositories of cyber-physical systems. In Conclusion, we point out the generalization of found errors types.

II. BACKGROUND

A. The git version control system

A version control system is a system that helps manage software development. It tracks changes, provides multiple version control of files, and allows multiple developers to collaborate [2]. The git version control system was created by Linus Torvalds to manage the development of the Linux kernel. Projects in the git system are called repositories. A git repository contains a collection of files and folders associated with a project, along with a history of changes to each file. A file history is a list of changes at a specific moment called commits. They can be organized into multiple development lines called branches. Because git is a distributed version control system, anyone with a copy of the repository can access the entire project's codebase and its history. Thus, by analyzing the repository one can track the progress of project development.

Each commit contains the following tracking information [3]: (1) a snapshot of the current state of the files in the repository at the time the changes were committed, a commit stores all the changes that have been made to the files at the time of the commit; (2) author of the commit; (3) the date the commit was created; (4) a commit message written by the developer in natural language that describes the essence of the changes made in this commit; (5) hash sum; and (6) parent commits.

B. Commit classification

The three most common reasons for making changes to the code by introducing a commit, are [4], [5]:

- 1) Adding new functionality: this refers to the process of introducing new features to the software application. It could involve creating new modules, implementing new algorithms, or integrating third-party libraries.
- 2) Improving code quality: this reason includes making changes to the code to enhance its maintainability,

readability, and efficiency. It could involve refactoring code to follow coding standards, optimizing algorithms or improving documentation.

- 3) Bug fixes: they involve identifying and resolving issues in the code that affect the functionality of the software application.

While bug fixes and adding new functionality are closely related reasons for code changes, it is essential to accurately classify commits in a version control system to distinguish between the two categories. To address this, some research focused on identifying key features that differentiate bug fixes from adding new functionality in commits. By analyzing the patterns, code changes, and contextual information associated with each type of commit, a commit classifier were made [6]. In our work we use almost the same ideas.

C. Cyber-physical systems and errors in it

Edward Lee's works on modeling cyber-physical systems [7], [8] provide in-depth exploration of the fundamental concepts underlying these complex systems. Lee points out the importance of a precise definition of cyber-physical systems, highlighting their evolution from the earlier cybernetic systems studied in control theory. One key aspect that sets modern cyber-physical systems apart is their reliance on diverse sensor data, which necessitates a distributed approach to operation. These systems not only interact with their environment but also have direct impacts on human lives, underscoring the need for accurate modeling to ensure their safe and effective functioning.

Raj Rajkumar emphasizes the interconnected nature of cyber-physical systems [9], notes how these systems combine physical processes with communication networks and computing capabilities. Helen Gill expands the definition of cyber-physical systems to include human factors, taking into account the role of human operators in these systems [10]. Janos Sztipanovits highlighting the interdisciplinary nature of cyber-physical systems, pointing how these systems bring together expertise from various fields such as computer science, engineering, and mathematics [11].

Errors in cyber-physical systems can be categorized into various types [12] such as design errors, implementation errors, communication errors, timing errors, and environmental errors. These errors can result in safety, system failures, performance, and security vulnerabilities.

Addressing errors in cyber-physical systems presents numerous challenges [13] due to the intricate nature of system interactions, real-time constraints, resource limitations, and the necessity for interdisciplinary expertise. To enhance error detection and mitigation strategies, researchers are investigating advancements in formal methods, model-based design, runtime monitoring, anomaly detection, and machine learning techniques [14].

D. Examples of cyber-physical systems to analyze

In this subsection, we describe some notable projects from our past experience and from the list of projects on the topic "cyber-physical systems" on GitHub, which we chose to analyze error fixes in their repositories.

The Ardupilot project [15] can be considered as a cyber-physical system. Ardupilot is an open-source autopilot software suite that enables autonomous control of drones,

unmanned aerial vehicles (UAVs), and other robotic systems. It integrates physical components (such as sensors, actuators, and communication modules, see the example of architectural modeling in [16]) with computational elements (software algorithms, control logic) to enable real-time monitoring, control, and navigation of the vehicle.

The Scada-LTS project [17] can also be understood as a cyber-physical system. SCADA (Supervisory Control and Data Acquisition) systems are used to monitor and control industrial processes, infrastructure systems, and other complex systems that involve the interaction between physical components and digital control systems. The Scada-LTS project is an open-source SCADA software system provides monitoring, control, monitoring, and data acquisition capabilities for various industrial applications.

The Modelica project [18], [19] is related to the cyber-physical system topic. Modelica is an open-source, object-oriented modeling language used to model complex cyber-physical systems that involve the interaction between physical components and digital control systems. In a cyber-physical system context, Modelica can be used to create models that represent the behavior of physical components such as mechanical systems, electrical systems, thermal systems, and more. These physical component models can then be integrated with control algorithms and other software components to create a comprehensive model of the entire system.

The KeYmaeraX project [20], [21] relates to the promising topic of proving the correctness of cyber-physical systems. KeYmaeraX is a theorem prover for hybrid systems, which are systems that exhibit both continuous dynamics (physical processes) and discrete dynamics (digital control algorithms). Cyber-physical systems often fall into the category of hybrid systems. KeYmaeraX supports the verification of safety-critical properties, such as collision avoidance, stability guarantees, reachability analysis, and dynamic logic specifications [22]–[24], which are essential for ensuring the reliable operation of cyber-physical systems.

III. RELATED WORK

To search for errors and vulnerabilities in cyber-physical systems, both static and dynamic methods are applicable. Since software for cyber-physical systems is a set of executable programs (possibly running on different nodes), general methods of analysis, testing and verification specific to distributed software are applicable to it. For such systems, monitoring is preferable, when the system is running and its current state can be obtained and compared with the expected one. The work [25] describes a review of this kind of monitoring for given requirements in the form of specifications. Specific signal analysis methods are applicable for different classes of such systems [26]. For real-time systems, this kind of monitoring can check parameters such as input rate, scheduling delay, and processing time [27]. For systems dependent on network interactions, methods for analyzing network vulnerabilities [28] are applicable. It has long been discussed that static code analysis techniques are applicable to such systems [29]. Although for special domains, special approaches must be used to generate benchmarks [30]. As theoretical research, first of all, methods for constructing behavioral models of such systems and

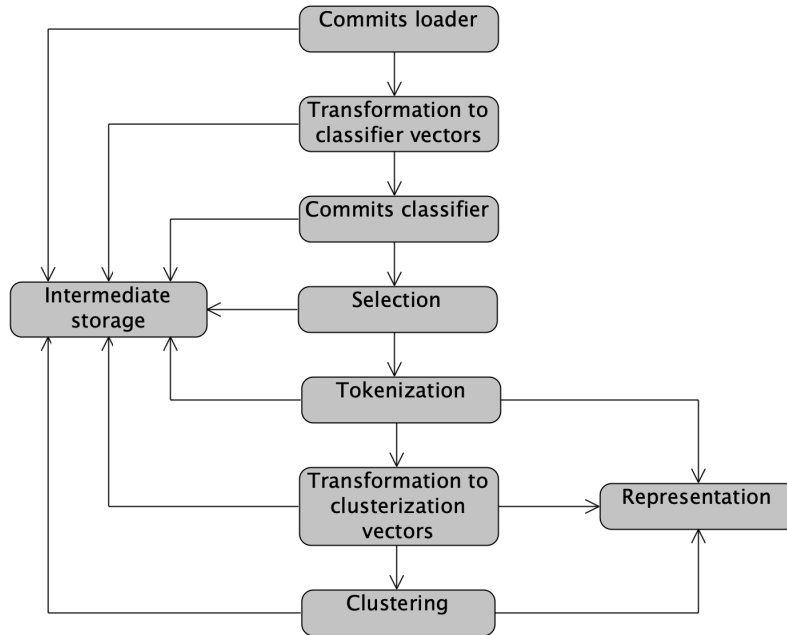


Fig. 1. Solution diagram

testing them [31], as well as logical calculus systems for modeling vulnerabilities [32] are known. Such models are decoupled from real code, which entails the use of abstraction and a potential loss of adequacy, but they can potentially test situations that are difficult to reproduce.

Empirical work by Tan et al. [33] focuses on conducting a comprehensive analysis of bugs in open-source software, with a specific emphasis on the Linux kernel. In the case of Linux, bugs are categorized based on their respective subsystems, such as core, driver, network, FS, arch, or other. The study utilizes BugZilla’s message text for various open-source components, employing vectorization techniques for automatic bug classification. In a similar vein, Xiao et al. [34] proceed with a detailed examination of 5741 Linux kernel bug reports. Their analysis delves into bug descriptions, comments, and attached files from the Linux kernel bug tracker, BugZilla. The bugs are further classified as fast-reproducible (Bohrbug), difficult-to-reproduce (Mandelbug), or context-dependent. Additionally, the researchers define specific categories to which bugs can be attributed, such as memory errors or unfreed resources, based on their contextual dependencies. In the present work, we are going to do such kind of analysis. By focusing on cyber-physical systems rather than Linux kernels, we can uncover specific issues and vulnerabilities that are relevant to this particular domain. Utilizing git repositories as the source for messages provides a wealth of data that can offer insight into the development and maintenance of these systems. By differentiating between fixing commit messages, we can pinpoint common errors and identify patterns that may indicate underlying issues within the codebase.

IV. ON THE IMPLEMENTATION

The internal flow and main modules of our solution are presented in Fig. 1. The ideas to implement the approach are presented in our previous paper [35]. To repeat, this is

getting the text of commits, identifying commits that commit, representing its text in vector form using a “bag of words”, working with increasing the role of significant words, and further clustering such vectors into generalizing centroids with the ability to restore close messages from commits for each of the found centroids. Let us just describe the logical modules.

The **Intermediate storage** is a module necessary to be able to interrupt the progress of the program and replace other modules without completely losing the result of the work, it can be presented in the form of a small database management system.

The **Commits loader** loads commit information from the git system. This module is necessary because calculating file changes can be a costly operation, and preloading information allows it to be done only once.

The **Transformation to classifier vectors** module obtains commits in a vector representation. For conversion, both data from the commit information in natural language and in a programming language are used. Priority in classification is given to control flow graph representation of the commit diff. The vector data is used only for classification and is not used further in commit clustering.

The **Commit classifier module** determines whether a certain commit is a bug fix.

The **Selection** module makes a selection from the general pool of commits that will be needed for a specific study. For example, the comment belongs to a specific domain and time period.

The **Tokenization** module converts a commit message to a list of keywords. This process involves removing technical information, converting words to a basic form, and removing unimportant words.

The **Transformation to clusterization vectors** module converts a list of keywords to numeric vectors, and also generates a dictionary for the reverse arrangement of vectors.

The **Clustering module** defines a certain cluster for a specific commit or assigns it to an outlier (does not assign it to any of the clusters). Works with the **Distance calculation module** which calculates the matrix of distances between all vectors.

The **Representation** module collects data from different modules, organizes them and presents them in a user-friendly format.

V. RESULTS OF ANALYSIS OF REPOSITORIES WITH CYBER-PHYSICAL SYSTEMS PROJECTS

In this section, we show the results of the analysis of the git repositories of the specified projects by our software. First, we demonstrate the generalized vectors found, with the words there arranged in descending order of weight using tf-idf. Next, for each vector, we show examples of the most relevant messages from commits that match this vector.

A. The Ardupilot project

Vector #1: ['apvehicle', 'compiler', 'float', 'override', 'keyword', 'old', 'airspeed', 'double', 'gc', 'initialisation', 'serial', 'avoid', 'wrapper', 'older', 'return']

```
AP_PID: compiler warnings: apply is_zero(float)
APMrover2: compiler warnings: apply is_zero(float) or is_equal(float)
Plane: compiler warnings: apply is_zero(float) or is_equal(float)
AP_Compass: compiler warnings: apply is_zero(float) or is_equal(float)
AntennaTracker: compiler warnings: apply is_zero(float) or is_equal(float)
```

This vector represents a fix for the compiler warning when working with floating-point values.

Vector #2: ['hwdef', 'binding', 'copter', 'register', 'match', 'changes', 'update', 'sub', 'optional', 'recent', 'upstream', 'manual', 'i2c', 'avoid', 'lua']

```
AP_HAL_ChibiOS: hwdef add support for Networking
hwdef: add hwdef for SDMODELH7V1
AP_HAL_ChibiOS: update truenav hwdef
AP_HAL_ChibiOS: hwdef for Flywoo F405 Pro
AP_HAL: move defaulting of HAL_DSHOT_ALARM into hwdef
```

This vector represents an improvement to the hardware definition for hardware abstraction components.

Vector #3: ['hil', 'initial', 'wrapper', 'implementation', 'roll', 'tailsitter', 'pitch', 'eliminate', 'transition', 'ap', 'attitude', 'call', 'vtol', 'timing', 'creation']

```
ACM: fixed HIL build again
ArduPlane: remove HIL support
AP_Compass: remove HIL support
Blimp: remove HIL support
GCS_MAVLink: remove HIL support
```

This vector deals with support for the Hardware-in-the-loop simulator, which was gradually removed from the project.

Vector #4: ['define', 'separate', 'send', 'patch', 'stability', 'simple', 'hold', 'alt', 'ability', 'able', 'absolute', 'abstraction', 'ac', 'accel', 'acceleration']

```
AP_RCProtocol: add separate define for AP_RC_PROTOCOL_PPMSUM_ENABLED
AP_RCProtocol: add separate define for AP_RC_PROTOCOL_ST24_ENABLED
AP_RCProtocol: add separate define for AP_RC_PROTOCOL_DSM_ENABLED
AP_RCProtocol: add separate define for AP_RC_PROTOCOL_SUMD_ENABLED
AP_RCProtocol: add separate define for AP_RC_PROTOCOL_IBUS_ENABLED
```

This vector deals with define macros in code.

Vector #5: ['fence', 'without', 'register', 'attitude', 'pointer', 'must', 'collective', 'avoid', 'quad', 'horizontal', 'next', 'tradheli', 'heli', 'term', 'structure']

```
AC_Fence: add polygon fence check to check_destination_within_fence
AC_Avoid: add support for stopping at polygon fence
AC_Fence: add support for polygon fences
AP_OSD: Add fence indicator panel
Rover: add fence support
```

This vector deals with fixes for obstacle avoidance functions.

Vector #6: ['location', 'adjust', 'vector3f', 'require', 'accepts', 'send', 'home', 'packet', 'specify', 'original', 'unify', 'circle', 'mount', 'support', 'usage']

```
autotest: fixed buildlogs location for *.BIN
AP_Math: move line_path_proportion to Location
ArduPlane: use past_interval_finish_line and line_path_proportion from Location
Tweaks to fix Loiter
Changed save location to int32
added some filtering and smoothing
APMrover2: use past_interval_finish_line and line_path_proportion from Location
```

This vector deals with fixes for location objects.

Vector #7: ['scheduler', 'enum', 'old', 'ability', 'able', 'absolute', 'abstraction', 'ac', 'accel', 'acceleration', 'accelerometer', 'accept', 'accepts', 'access', 'accessor']

```
Rover: windvane update called from scheduler at 20hz
AP_NavEKF: Scheduler Improvements
Plane: remove update_events scheduler shim
AP_HAL_AVR: Scheduler extensions implemented
```

Copter: remove shims used in scheduler

This vector deals with fixes for the scheduler.

Vector #8: ['macro', 'rewrite', 'frontend', 'flag', 'ability', 'able', 'absolute', 'abstraction', 'ac', 'accel', 'acceleration', 'accelerometer', 'accept', 'accepts', 'access']

all: use CLASS_NO_COPY() macro

Copter: Obey RANGEFINDER_ENABLED,
AUTOTUNE_ENABLED and AC_TERRAIN build
macros

AP_HAL_SMACCM: fix to goofed
PPM_MAX_CHANNELS macro

HAL_ChibiOS: use EXPECT_DELAY() macro

AP_InertialSensor: use EXPECT_DELAY()
macro

This vector deals with various code fixes in macros
(specific to C/C++ projects).

Vector #9: ['script', 'reset', 'last', 'install', 'readme', 'folder', 'dronecan', 'style', 'multiple', 'optflow', 'byte', 'iofirmware', 'lua', 'jsbsim', 'enumeration']

AP_HAL_ChibiOS: fix script for HerePro

AP_HAL_F4Light: fixed some support
scripts

AP_Scripting: add arming check for failed
scripts

HAL_ChibiOS: moved to generated loader
script

AP_Scripting: add checksum of running and
loaded scripts with arming check

This vector deals with fixes for built-in scripts.

Vector #10: ['quadplane', 'adjust', 'ability', 'able', 'absolute', 'abstraction', 'ac', 'accel', 'acceleration', 'accelerometer', 'accept', 'accepts', 'access', 'accessor', 'accessors']

Plane: Quadplane: use uint16_t for
output_motor_mask

Plane: added QTUN logging for quadplane

AC_WPNav: converted to use AP_AHRS_View

Plane: Quadplane remove THR_MIN_PWM and
THR_MAX_PWM

Plane: allow for NAV_LOITER_UNLIM and
NAV_LOITER_TIME in quadplane

This vector deals with fixes for working with QuadPlane,
which is a combined fixed wing and MultiCopter aircraft.

B. The Scada-LTS project

Vector

#1 ['settimeout', 'state', 'commonjs', 'pointpropertiesapi', 'eventtextrender', 'translate', 'interface', 'messagesms', 'localization', 'validate', 'copy', 'plcalarmsdao', 'messages', 'adapter', 'miscdwrdoLongPoll']

#2111 Fixed using setTimeout in common.js
- added multi thread tests:

```
MiscDwrDoLongPollAlarmsMultiThreadTest  
, MiscDwrDoLongPollMultiThreadTest;  
corrected MiscDwr.doLongPoll, without  
copy state
```

#2111 Fixed using setTimeout in common.js
- use setInterval

#2111 Fixed using setTimeout in common.js
3 - setTimeout functionality ported
to java

#2111 Fixed using setTimeout in common.js
- validate uiPerformance, added 'Very
high' option (1000ms)

#2111 Fixed using setTimeout in common.js
- corrected refresh points with
statistics, for request that run
longer than the Interval Time

This vector is a fix for the timer function - executing code
by timeout in JS.

Vector #2: ['patch', 'vue', 'views', 'tests', 'ability', 'able', 'abstract', 'abstractbeforeafterworkitem', 'accept', 'access', 'acceptable', 'acknowledge', 'acl', 'action', 'active']

patch removed newline

patch MangoTextContent.java

Refactor Vue Unit Tests #1533

#1641 ADD [NotFinished] VirtualDataPoint
Vue

#1894 Viewscaching - corrected

This vector represents patches for views (that is,
representing the states of objects on the screen).

Vector #3: ['correction', 'commit', 'display', 'partial', 'isalive2', 'viewdao', 'clean', 'eventdetectorapi', 'not', 'eventlist', 'component', 'number', 'ability', 'able', 'abstract']

Correction of commit number display.
#1502

#2051 Visual corrections (component
shrink)

EventDetectorAPI corrections #1532

#2051 Add corrections to the IsAlive2

SLTS-40 Add correction to ViewDAO

This vector represents various corrections associated with
the display of information and events in the system.

Vector

#4: ['slts13', 'testdao', 'annotation', 'cleanup', 'scriptdao', 'direct', 'rewrite', 'navigation', 'link', 'controller', 'ability', 'able', 'abstract', 'abstractbeforeafterworkitem', 'accept']

SLTS-13 Added FlexProjectRowMapper and
remove @SuppressWarnings annotation

SLTS-13 move expectedException to TestDAO

SLTS-13 Rewrite ScriptDAO

SLTS-13 controller's clean-up

SLTS-132 Implemented direct link
navigation.

This vector represents fixes for DAO (data access object).

Vector #5: ['log4j', 'pointdetails', 'slts34', 'logger', 'ability', 'able', 'abstract', 'abstract before after workitem', 'accept', 'access', 'acceptable', 'acknowledge', 'acl', 'action', 'active']

#1982 - log4j update
#1982 - log4j update
SLTS-97 Add logger for log4j. Correct
PointValueService.
WORKSAVE New PointDetails ideas
SLTS-34 Add DataSourceServiceTest

This vector represents logging fixes using log4j (log for Java).

C. The Modelica project

Vector #1: ['proper', 'word', 'format', 'use', 'number', 'leakage', 'accordingly', 'individual', 'usersguide', 'magnetic', 'electrical', 'work', 'ha', 'implementation', '09']

Use proper number formats
Proper word
Use proper number formats
Proper word

SymmetricPolyphaseWinding: moved
individual leakage from magnetic to
electrical implementation: it works!
UsersGuide has to be adapted accordingly.

This vector represents fixes for formatting messages with the correct number format.

Vector #2: ['unify', 'time', 'ccr', 'event', 'enhancement', 'cleanup', 'drive', 'controller', 'some', 'powerconverters', 'partial', 'interface']

refs #1627: Fix detection of (scaled)
time events
refs #1627: Fix detection of scaled time
events
CCR: corrected error in setState_ps

This vector represents fixes for working with time events.

Vector #3: ['electricaldigital', 'correction', 'elementary', 'grid', 'fit', 'mechanicsmultibody', 'layout', 'better', 'info', 'svn', 'revision', 'function']

#799 solved for Electrical.Digital
errors fixed in Electrical.Digital
errors fixed in Electrical.Digital
corrections according to #994 in
Electrical.Spice3.Internal.Fet
Correction of info of function Matrices.
realSchur

This vector represents bug fixes in the Electrical.Digital component (this library contains packages to model digital electronic systems based on combinational and sequential logic).

Vector #4: ['due', 'picture', 'complexblocks', 'modification', 'docu', 'referenceair', 'referencemoistair', 'mass', 'energy', 'balance', 'some', 'static']

Comments added due to #1475
modifications due to ComplexBlocks
due to #407 bugs 2, 3, 4, 5, 7 fixed (
docu, pictures)
Some changes due to renaming of
ReferenceMoistAir and ReferenceAir.

medium.preferredMediumStates is now false
if both mass and energy balances are
static
Attempt to fix issue #3236

This vector represents error fixes for complex blocks with images.

Vector #5: ['dynamicselect', 'boolean', 'comparison', 'real', 'individual', 'stray', 'implementation', '09', '150', '2dtable', '3rdparty', '64bit', 'abbreviation']

Avoid Boolean comparison with Real in
DynamicSelect (#2862)
Avoid Boolean comparison with Real in
DynamicSelect (#2879)
corrected implementation of individual
stray permeances

This vector represents fixes for proper comparison with real (floating-point) values.

D. The KeYmaeraX project

Vector #1: ['implement', 'syntactic', 'derivative', 'index', 'skeleton', 'skolemization', 'counting', 'magic', 'less', '20150824', '20160308', '20160601', '20160802', '20160816', '2sided']

implement GetPathAll
implement the CreateProblemRequest
implement BranchRoot
implement skolemization
implement more of skeleton

This vector represents the addition of various functionality to the project in the form of large commits, for example, the implementation of skolemization (reduction to Skolem normal form is an approach for removing existential quantifiers from formal logic statements).

Vector #2: ['package', 'private', 'change', 'moves', 'datastructures', 'firstintegrals', 'fol', 'replaceall', 'tooltips', 'dwplus', 'compilability', 'link', 'thanks', 'cert', 'sos']

packages
Change package
Change package for compilability.
merge tons of code from kaisar package to
experiments package, get bot working
again
Move strategic AxiomIndex to package
btactics

This vector represents fixes for rebuilding packages in the logical organization of the architecture.

Vector #3: ['solve', 'axiominfo', 'nilpotent', 'search', 'feedback', 'anyarg', 'anything', 'axiombase', 'output', 'rescue', 'diffsolve', 'theme', 'choice', 'consistently', 'speedup']

Nilpotent solve (preliminary)
Nilpotent solve speedup
Re-unification to solve $p(.) \sim > . \geq 0$, $p(.) \sim > 2 \geq 0$ issues as in

[x' :=2]x' >=0 <-> 2 >=0 against [' :=]
 Nilpotent solve: dW only when provable
 key/recursor in AxiomInfo

This vector represents fixes for Nilpotent solve (for solving algebraic structures).

Vector

#4: ['unification', 'derive', 'bidirectional', 'axindex', 'imply', 'monomial', 'projection', 'init', 'sign', 'bit', 'variation', 'dot', 'dbx', 'match', 'dotterm']

Colored-dots unification

Improve unification a bit further

Unification support for projection

Unification match: 0-indexed colored dots

DiffHelper derive fix (sign error)

This vector represents fixes for the unification algorithms [36] in the theorem prover.

VI. DISCUSSION AND CONCLUSION

The results of the clusterer can be considered successful only for the Ardupilot project. For the rest, there are problems with obtaining both the required number of arbitrarily representative clusters, and with the quality of the found vectors, so that they actually represent fixes of repeating types of errors. We see the main problems here in the organization of development of the selected projects, when developers write uninformative messages about commits. In addition, task management suffers: code fixes are committed to solve large problems at once, rather than small changes with a clear justification. The final problem we see is the classification of commits into adding functionality and fixes: the classifier is based on the code and messages of Linux kernel commits and for other repositories, as it turned out, it does not work entirely correctly.

As for the processed commits, in the Ardupilot project the main errors were specific to the software domain for unmanned vehicles, such as issues of location processing, obstacle avoidance, abstraction from hardware and scheduling. For SCADA systems, errors specific to Java applications and MVC architecture were found, which is not surprising, because such projects visualize monitoring data of cyber-physical systems. For the Modelica system, the Electrical.digital subsystem was found, which is prone to errors, as well as other issues there are related to the conversion of numeric floating-point values. Finally, in the theorem prover for cyber-physical systems KeYmaeraX (with not a good organization of commits that does not reveal the full complexity of development), the main logical subsystems for proving cyber-physical models in which there were many changes in the code were found: skolemization, nilpotent solve and unification.

The final advantages of our solution are the ability to easily evaluate what is being done in a given repository (with a sufficiently large number of well-organized commits) and what errors were corrected in order to learn from examples of the development of this kind of systems with increased reliability requirements.

REFERENCES

- [1] *git*. [Online]. Available: <https://git-scm.com>
- [2] S. Otte, "Version control systems." 2009. [Online]. Available: <https://api.semanticscholar.org/CorpusID:7541013>
- [3] S. Chacon and B. Straub, *Pro git*. Springer Nature, 2014.
- [4] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: how misclassification impacts bug prediction," in *2013 35th international conference on software engineering (ICSE)*. IEEE, 2013, pp. 392–401.
- [5] H. Kagdi, M. L. Collard, and J. I. Maletic, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution," *Journal of software maintenance and evolution: Research and practice*, vol. 19, no. 2, pp. 77–131, 2007.
- [6] Y. Tian, J. Lawall, and D. Lo, "Identifying Linux bug fixing patches," in *2012 34th international conference on software engineering (ICSE)*. IEEE, 2012, pp. 386–396.
- [7] 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.
- [8] —, *Plato and the nerd: The creative partnership of humans and technology*. MIT Press, 2017.
- [9] R. Rajkumar, D. De Niz, and M. Klein, *Cyber-physical systems*. Addison-Wesley Professional, 2016.
- [10] H. Gill, "From vision to reality: cyber-physical systems," in *HCSS national workshop on new research directions for high confidence transportation CPS: automotive, aviation, and rail*. Austin USA, 2008, pp. 1–29.
- [11] J. Sztipanovits, X. Koutsoukos, G. Karsai, N. Kottenstette, P. Antsaklis, V. Gupta, B. Goodwine, J. Baras, and S. Wang, "Toward a science of cyber-physical system integration," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 29–44, 2011.
- [12] G. M. Siddesh, G. C. Deka, K. G. Srinivasa, and L. M. Patnaik, *Cyber-physical systems: a computational perspective*. CRC Press, 2015.
- [13] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: the next computing revolution," in *Proceedings of the 47th design automation conference*, 2010, pp. 731–736.
- [14] Y. Luo, Y. Xiao, L. Cheng, G. Peng, and D. Yao, "Deep learning-based anomaly detection in cyber-physical systems: Progress and opportunities," *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–36, 2021.
- [15] *ArduPilot Project*. [Online]. Available: <https://github.com/ArduPilot/ardupilot>
- [16] S. Staroletov, "Architectural software-hardware co-modeling a real-world cyber-physical system: Arduino-based ardupilot case," in *2021 30th Conference of Open Innovations Association FRUCT*. IEEE, 2021, pp. 267–278.
- [17] *Scada-LTS*. [Online]. Available: <https://github.com/SCADA-LTS/Scada-LTS>
- [18] *Modelica Standard Library*. [Online]. Available: <https://github.com/modeledica/ModelicaStandardLibrary>
- [19] P. Fritzon, *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*. John Wiley & Sons, 2014.
- [20] *KeYmaera X Theorem Prover for Hybrid Systems*. [Online]. Available: <https://github.com/LS-Lab/KeYmaeraX-release>
- [21] N. Fulton, S. Mitsch, J.-D. Quesel, M. Völpl, and A. Platzer, "KeYmaera x: An axiomatic tactical theorem prover for hybrid systems," in *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*. Springer, 2015, pp. 527–538.
- [22] J.-D. Quesel, S. Mitsch, S. Loos, N. Aréchiga, and A. Platzer, "How to model and prove hybrid systems with keymaera: a tutorial on safety," *International Journal on Software Tools for Technology Transfer*, vol. 18, no. 1, pp. 67–91, 2016.
- [23] S. Staroletov, "Automatic proving of stability of the cyber-physical systems in the sense of Lyapunov with KeYmaera," in *2021 28th Conference of Open Innovations Association (FRUCT)*. IEEE, 2021, pp. 431–438.
- [24] S. Staroletov, H. Schulte, T. Baar, I. Konyukhov, N. Shilov, A. Rozov, T. Liakh, and V. Zyubin, "Modeling and verification using different notations for CPSs: The one-water-tank case study," in *2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS)*. IEEE, 2021, pp. 485–488.
- [25] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan, "Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications," *Lectures on Runtime Verification: Introductory and Advanced Topics*, pp. 135–175, 2018.
- [26] J. Morgan and G. E. O'Donnell, "Cyber physical process monitoring systems," *Journal of Intelligent Manufacturing*, vol. 29, no. 6, pp. 1317–1328, 2018.

- [27] M. Canizo, A. Conde, S. Charramendieta, R. Minon, R. G. Cid-Fuentes, and E. Onieva, "Implementation of a large-scale platform for cyber-physical system real-time monitoring," *IEEE Access*, vol. 7, pp. 52 455–52 466, 2019.
- [28] Y. Ashibani and Q. H. Mahmoud, "Cyber physical systems security: Analysis, challenges and solutions," *Computers & Security*, vol. 68, pp. 81–97, 2017.
- [29] E. A. Lee, "Cyber physical systems: Design challenges," in *2008 11th IEEE international symposium on object and component-oriented real-time distributed computing (ISORC)*. IEEE, 2008, pp. 363–369.
- [30] C. Eichler, P. Wägemann, and W. Schröder-Preikschat, "Genee: A benchmark generator for static analysis tools of energy-constrained cyber-physical systems," in *Proceedings of the 2nd Workshop on Benchmarking Cyber-Physical Systems and Internet of Things*, 2019, pp. 1–6.
- [31] T. Fabarisov, N. Yusupova, K. Ding, A. Morozov, and K. Janschek, "Model-based stochastic error propagation analysis for cyber-physical systems," *Acta Polytechnica Hungarica*, vol. 17, no. 8, pp. 15–28, 2020.
- [32] R. Lanotte, M. Merro, A. Munteanu, and L. Viganò, "A formal approach to physics-based attacks in cyber-physical systems," *ACM Transactions on Privacy and Security (TOPS)*, vol. 23, no. 1, pp. 1–41, 2020.
- [33] L. Tan, C. Liu, Z. Li, X. Wang, Y. Zhou, and C. Zhai, "Bug characteristics in open source software," *Empirical software engineering*, vol. 19, pp. 1665–1705, 2014.
- [34] G. Xiao, Z. Zheng, B. Yin, K. S. Trivedi, X. Du, and K.-Y. Cai, "An empirical study of fault triggers in the Linux operating system: An evolutionary perspective," *IEEE Transactions on Reliability*, vol. 68, no. 4, pp. 1356–1383, 2019.
- [35] S. Staroletov, N. Starovoytov, and N. Golovnev, "Analyzing hot bugs in the Linux kernel by clustering fixing commit messages," *Proceedings of the Institute for System Programming of RAS*, vol. 35, no. 3, 2023.
- [36] K. Hoder and A. Voronkov, "Comparing unification algorithms in first-order theorem proving," in *KI 2009: Advances in Artificial Intelligence: 32nd Annual German Conference on AI, Paderborn, Germany, September 15-18, 2009. Proceedings 32*. Springer, 2009, pp. 435–443.