Classification of commits in git repositories to find the most common bug fixes in system software (Linux kernel and cyberphysical systems projects)

Sergey Staroletov

Polzunov Altai State Technical University

STEP/online 21.08.2024

Relevant papers to read

- Analyzing hot bugs in the Linux kernel by clustering fixing commit messages (Труды института системного программирования РАН, 2023, том 35, выпуск 3, страницы 215–242), DOI: https://doi.org/10.15514/ISPRAS-2023-35(3)-16

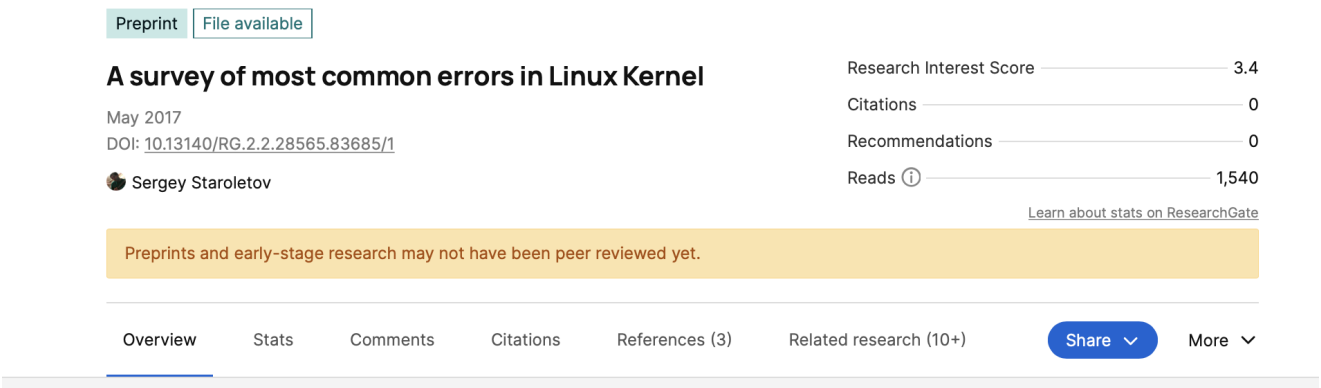
- Exploring the Taxonomy of Commits in Cyber-Physical Systems for Enhanced Error Fixes Investigation (Труды института системного программирования РАН, 2024, том 36, выпуск 2, страницы 33–46), DOI: https://doi.org/10.15514/ISPRAS-2024-36(2)-3

History of the research

- Poster session at SYRCoSE'2017 @ Innopolis

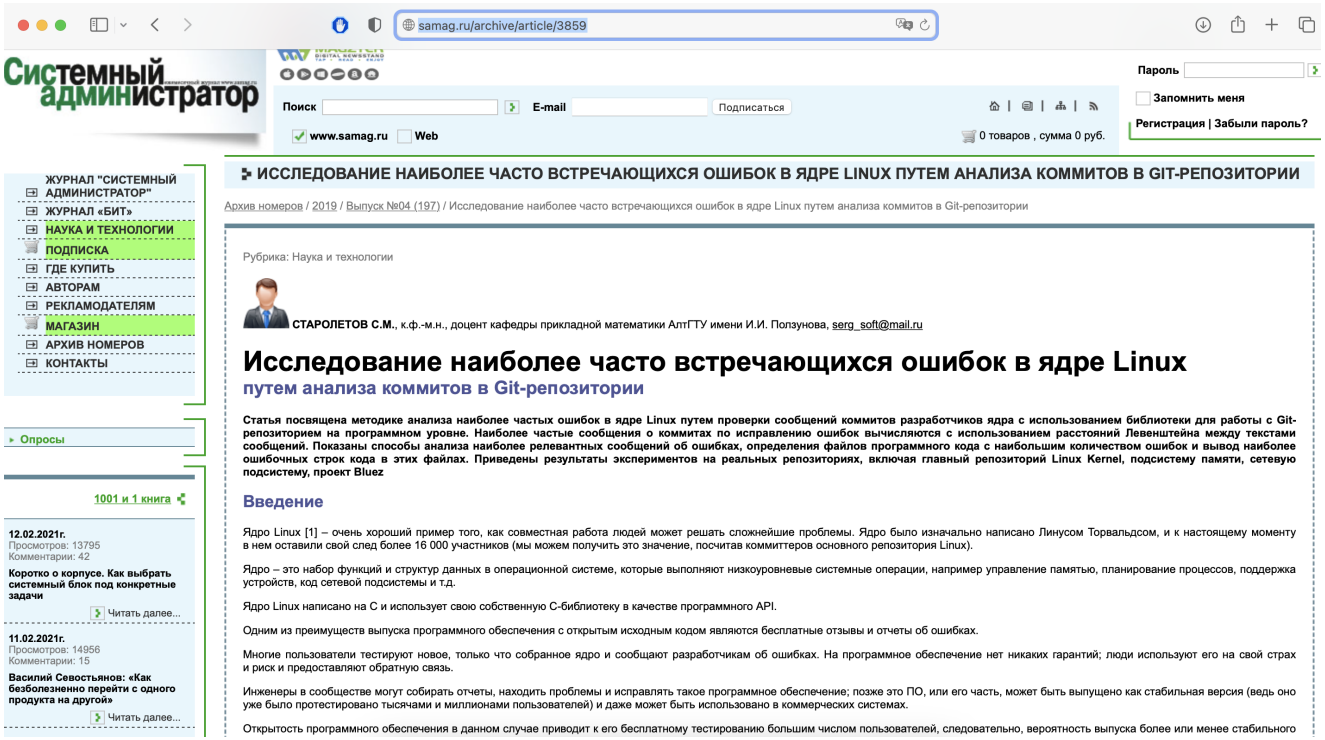
History of the research

- The work was put to ResearchGate as a preprint
- There have been over 1500 views since 2017



History of the research

- In 2019, it was published in Russian in System Administrator 04 (197) `http://samag.ru/archive/article/3859` with some new data



The purpose

- The purpose of our work is to automatically analyze commits in the git repositories to **identify the most representative bugs**.
- In this work, we mainly discuss and try **data analysis methods** for git commit messages.

Part 1 of 2

- Part1: Analyzing Linux kernel (Joint work with N. Starovoytov, N. Golovnev)

Linux and git = two super-successful projects by Linus Torvalds

Big data analysis

- In 1913, A.A. Markov used 20,000 sequential letters from A.S. Pushkin's poem as a big data to invent the Markov chains approach

Къ 200 лѣтнему юбилею закона большихъ чиселъ.

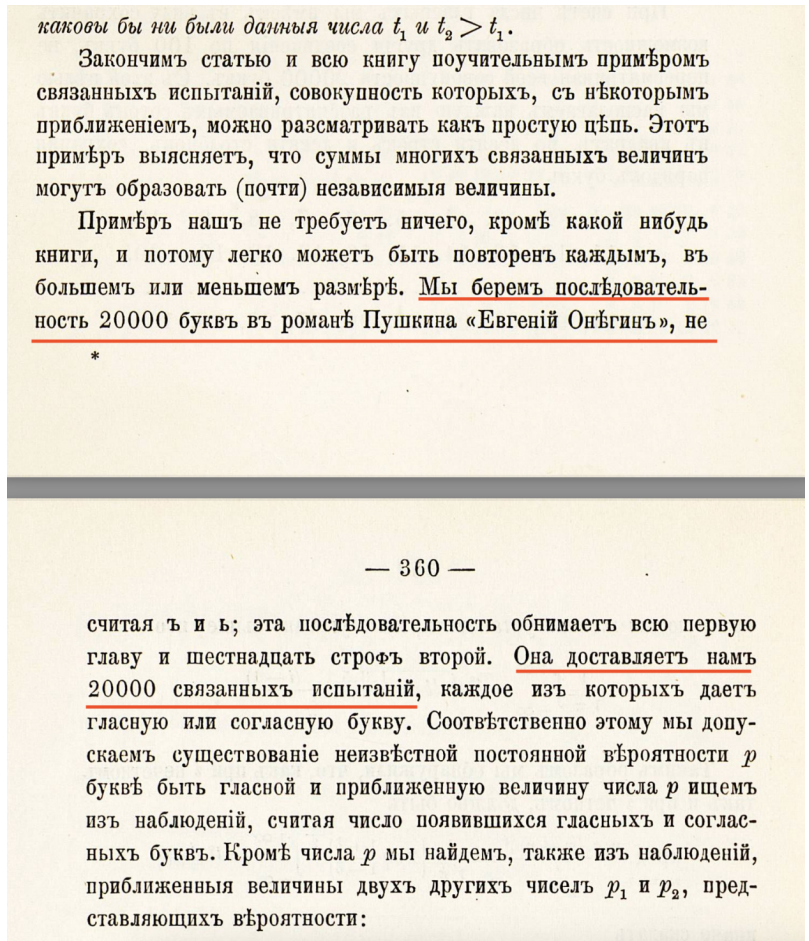
ИСЧИСЛЕНІЕ ВѢРОЯТНОСТЕЙ.

А. А. Марковъ.

ТРЕТЬЕ ИЗДАНІЕ,

пересмотрѣнное и значительно дополненное.

Съ портретомъ Якова Бернулли.



каковы бы ни были данныя числа t_1 *и* $t_2 > t_1$.

Закончимъ статью и всю книгу поучительнымъ примѣромъ связанныхъ испытаній, совокупность которыхъ, съ нѣкоторымъ приближеніемъ, можно разсматривать какъ простую цѣпь. Этотъ примѣръ выясняетъ, что суммы многихъ связанныхъ величинъ могутъ образовать (почти) независимыя величины.

Примѣръ нашъ не требуетъ ничего, кромѣ какой нибудь книги, и потому легко можетъ быть повторенъ каждымъ, въ большемъ или меньшемъ размѣрѣ. Мы беремъ послѣдовательность 20000 буквъ въ романѣ Пушкина «Евгеній Онѣгинъ», не

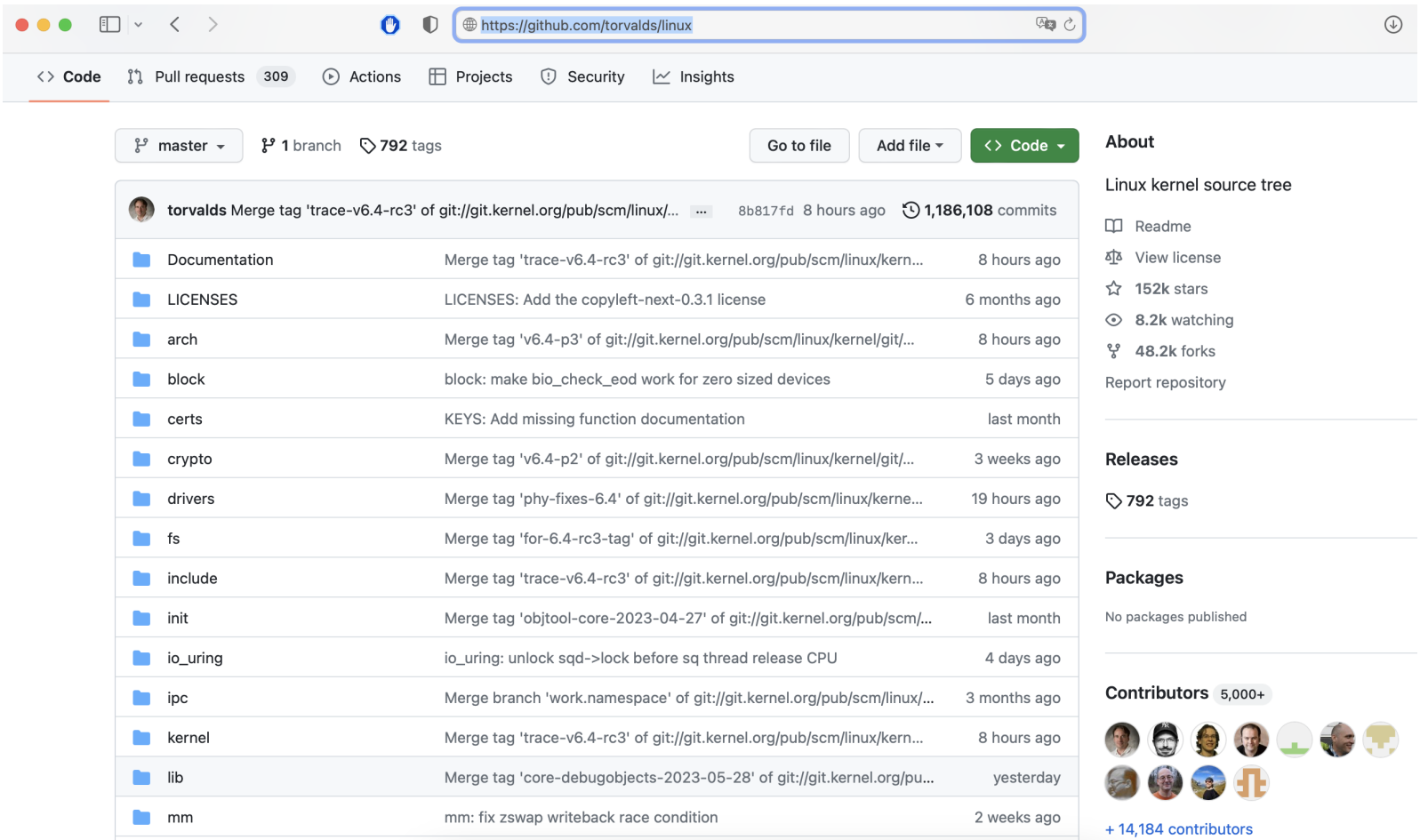
*

— 360 —

считая ъ и ь; эта послѣдовательность обнимаетъ всю первую главу и шестнадцать строфъ второй. Она доставляетъ намъ 20000 связанныхъ испытаній, каждое изъ которыхъ даетъ гласную или согласную букву. Соотвѣтственно этому мы допускаемъ существованіе неизвѣстной постоянной вѣроятности p буквѣ быть гласной и приближенную величину числа p ищемъ изъ наблюденій, считая число появившихся гласныхъ и согласныхъ буквъ. Кромѣ числа p мы найдемъ, также изъ наблюденій, приближенныя величины двухъ другихъ чиселъ p_1 и p_2, представляющихъ вѣроятности:

Big data analysis

- Up to 2023, Linux has more than 1,000,000 commits (listed on github.com/torvalds/linux)

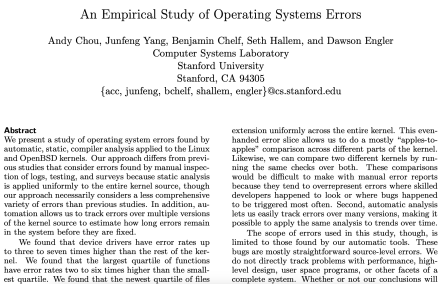


Big data analysis for Linux

- Such an operating system belongs to a class of system software that provides abstractions for accessing hardware from client code and the ability for such code to work cooperatively.
- With the increase in the number of developers in the world, an ever smaller percentage of them are capable of developing system program code, and therefore the development of system code is not popular.
- However, there is a large amount of data circulating in system software environments that can be analyzed by today's popular data analysis methods.

Errors in Linux by static analyzing

- In the pioneering work[1] and then in[2], static analyzers were used to automatically check for potential errors in the Linux kernel code based on a given configuration over different kernels.
- Classes of errors were defined as predefined messages of a static analyzer, and graphs of the evolution of errors over time and for different subsystems were presented.
- Specifically, drivers have been found to be 3-7 times more error prone than other components.



An Empirical Study of Operating Systems Errors

Andy Chou, Junfeng Yang, Benjamin Chelf, Seth Hallem, and Dawson Engler
Computer Systems Laboratory
Stanford University
Stanford, CA 94305
{acc, junfeng, bchelf, shallem, engler}@cs.stanford.edu

Abstract

We present a study of operating system errors found by automatic, static, compiler analysis applied to the Linux and OpenBSD kernels. Our approach differs from previous studies that consider errors found by manual inspection of logs, testing, and surveys because static analysis is applied uniformly to the entire kernel source, though our approach necessarily considers a less comprehensive variety of errors than previous studies. In addition, automation allows us to track errors over multiple versions of the kernel source to estimate how long errors remain in the system before they are fixed.

We found that device drivers have error rates up to three to seven times higher than the rest of the kernel. We found that the largest quartile of functions have error rates two to six times higher than the smallest quartile. We found that the newest quartile of files extension uniformly across the entire kernel. This even-handed error slice allows us to do a mostly "apples-to-apples" comparison across different parts of the kernel. Likewise, we can compare two different kernels by running the same checks over both. These comparisons would be difficult to make with manual error reports because they tend to overrepresent errors where skilled developers happened to look or where bugs happened to be triggered most often. Second, automatic analysis lets us easily track errors over many versions, making it possible to apply the same analysis to trends over time.

The scope of errors used in this study, though, is limited to those found by our automatic tools. These bugs are mostly straightforward source-level errors. We do not directly track problems with performance, high-level design, user space programs, or other facets of a complete system. Whether or not our conclusions will

[1] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler, "An empirical study of operating systems errors," in Proceedings of the eighteenth ACM symposium on Operating systems principles, 2001, pp. 73–88.

[2] N. Palix, G. Thomas, S. Saha, C. Calve's, J. Lawall, and G. Muller, "Faults in Linux: Ten years later," in Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems, 2011, pp. 305–318.

Errors in Linux by error grouping

- In work[3], an analysis of typical errors is made in the drivers of the Linux operating system.
- Here the concept of a typical error is introduced. It is specific to a large number of drivers (for example, resource leaks, incorrect use of locks), while a non-typical error is domain-specific for a particular driver.
- The authors manually analyzed the changes during the transition from one kernel version to another and compiled tables of the distribution of errors by classes. It was also found that drivers make 85% of all errors in the kernel.
- The paper[4] continues this work, summarizes various statistics on changes in the kernel and concludes that about 40% of changes between stable versions of the kernel are fixes of typical errors.

---

[3] V. Mutilin, E. Novikov, and A. Khoroshilov, "Analysis of typical errors in Linux OS drivers (in Russian)," Proceedings of the Institute for System Programming of the Russian Academy of Sciences, vol. 22, pp. 349–374, 2012.

[4] E. M. Novikov, "Evolution of the Linux OS kernel (in Russian),"Proceedings of the Institute for System Programming of the Russian Academy of Sciences, vol. 29, no. 2, pp. 77–96, 2017.

Errors in Linux by bug types

- The work[5] is devoted to the study of 5741 Linux kernel bug reports, which were analyzed according to the description, comments and attached files from the Linux kernel bug tracker `bugzilla.kernel.org`.
- Bugs are classified into fast-reproducible (Bohrbug), difficult-to-reproduce (Mandelbug) or context-dependent, and are also defined categories from which the bug context depends, that is, errors with memory, not freed resources, etc.
- At the same time, the authors built a network based on the Linux call graph, with the help of which they track the impact of the functions affected in bug reports by counting various metrics.

[5] 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.

Errors in Linux by compiling kernels

- Researchers present the results[6] of compiling 42,060 kernels with all warnings enabled.
- As a result of the analysis of 400,000 warnings, they classified by type and distribution by kernel subsystems and identified drivers as the most vulnerable portion of the kernel.

[6] J. Melo, E. Flesborg, C. Brabrand, and A. Wasowski, "A quantitative analysis of variability warnings in Linux," in Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems, 2016, pp. 3–8.

Is a patch bug-fixing or not?

- The research[7] is separately devoted to determining whether a patch to the kernel is a bug fix or not.
- The authors note that simple analysis based on commit messages does not always lead to results and propose a model that uses two classification algorithms:
  - Learning from Positive and Unlabeled Examples;
  - Support Vector Machine.
  - It also uses features extracted from the commit diff.

[7] 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.

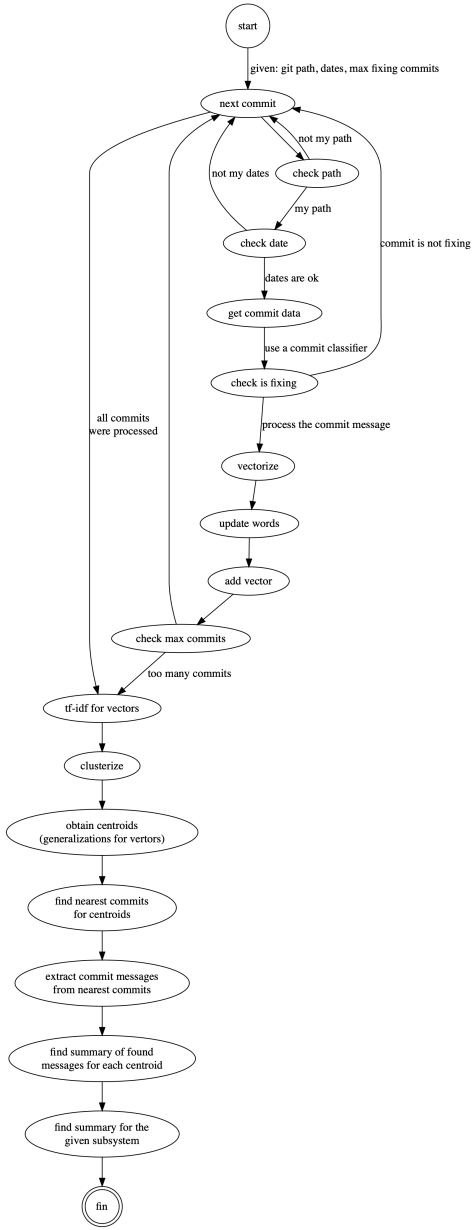
Related work survey results

Summarizing the above on Linux bug analysis, it can be seen that:

1. bugs in drivers are the most common;
2. different methods are used for classification, these are static analysis, build logs and patch analysis;
3. a lot of huge manual work has been done but the results may now be considered no longer relevant (the code is constantly changing).

However, automatic classification by analyzing commits in git repositories has not been applied yet.

General approach



Helpers

- Levenshtein distance[8] (very easy method to find a closest string)

$$
\begin{aligned}
L(s_1, s_2) := \forall i \in (0..|s_1|) : d_{i,0} := i + 1; \\
\forall j \in (0..|s_2|) : d_{0,j} := j + 1; \\
\forall i \in (1..|s_1|) : \\
(\forall j \in (1..|s_2|) : cost := (s_1[i-1] = s_2[j-1])?0 : 1 \\
d_{i,j} := min(min(d_{i-1,j} + 1, d_{i,j-1}), d_{i-1,j-1} + cost); \\
d_{|s_1|,|s_2|}) \quad (1)
\end{aligned}
$$

[8] V.I.Levenshtein,"Binary codes with correction of dropouts, insertions and character substitutions (in Russian)," in Reports of the Academy of Sciences, vol. 163, no. 4. Russian Academy of Sciences, 1965, pp. 845–848.

Helpers

- Phrase vectorization, "bag of words", cosine distance

$$D(s_1, s_2) := D((w_{1,1}, w_{1,2}, .., w_{1,n}), (w_{2,1}, w_{2,2}, .., w_{2,n})) = \frac{\sum_{i=1}^{n} w_{1,i} \cdot w_{2,i}}{\sqrt{\sum_{i=1}^{n} w_{1,i}^2 \cdot \sum_{i=1}^{n} w_{2,i}^2}} \quad (2)$$

In this case, permutations of words in a string will not change anything.

Helpers

- TF-IDF for searching strings with "strong components"or relevant words/tokens:

  If we denote n_w as the number of occurrences of the word w into a commit message $m \in M$, and n_w as the total number of words in the document, and $|M|$ as the total number of messages, then:

$$\text{tf-idf}(w, m, M) := \text{tf}(w, m) \times \text{idf}(w, M) = \frac{n_w}{\sum_k n_k} \times \log \frac{|M|}{|\{ m_i \in M \mid w \in d_i \}|} \quad (3)$$

Helpers

- Lemmatization or lemma normalization for obtain the stem word form for each word.
- The use of StanfordCoreNLP API

```
fix     fix     $N$
fix     fix     $V+0$
fixed   fix     $V+ed$
fixed   fix     $V+en$
fixes   fix     $N+s$
fixes   fix     $N+s$
fixes   fix     $V+s$
fixing  fix     $V+ing$
```

General Vectors (Linux kernel)

Vector #1: [cpu, period, grace, rcu, event, callback, commit, probe, function, state, check, structure, rcu_node, stall, file]

> *Fix day-one dyntick-idle stall-warning bug*
> *rcu: Suppress more involved false-positive preempted-task splats*
> *rcu: Accelerate grace period if last non-dynticked CPU*
> *rcu/segcblist: Prevent useless GP start if no CBs to accelerate*
> *Go dyntick-idle more quickly if CPU has serviced current grace period*

These fixes address the very important RCU subsystem in Linux, which provides ways to non-blockingly synchronize concurrent entities. However, there are problems in the form of potential unfinished waiting or inefficiency in its implementation, since processors in modern computing systems can go into energy-efficient hibernation, which leads to bugs in the RCU implementation for the tasks running on them (problems with the waiting period or grace period).

General Vectors (Linux kernel)

Vector #2: [buffer, kernel, ring, warning, doc, function, page, parameter, iterator, read, type, member, resource, trace, tracepoint]

> *ring-buffer: Fix kernel-doc*
> *ring-buffer: Always reset iterator to reader page*
> *resource/docs: Fix new kernel-doc warnings*
> *seccomp: fix kernel-doc function name warning*
> *rcu: Fix a kernel-doc warnings for "count"*

These fixes refer to corrections to code documentation, which are done in the form of comments embedded in the code. The kernel-doc tool collects these comments and checks their completeness. The comments here refer to the ring buffer, which can be used to implement efficient network applications.

General Vectors (Linux kernel)

Vector #3: [module, panic, patch, build, state, cpu, error, message, new, unloaded, code, function, kernel, notifier, list, load_module]

> *module: Ensure a module state is set accordingly during module coming cleanup code*
> *livepatch: Fix subtle race with coming and going modules*
> *debug: track and print last unloaded module in the oops trace*
> *Kprobes: Reference count the modules when probed on it*
> *debug: show being-loaded/being-unloaded indicator for modules*

These fixes concern errors when loading and unloading kernel modules, more precisely during their live loading, when the already loaded code is replaced in a running system. This is possible using the function tracing approach.

General Vectors (Linux kernel)

Vector #4: [tracer, function, graph, ret_stack, task, tracing, option, new, callback, code, ftrace, return, add, boot, buffer]

> *tracing/function-graph-tracer: drop the kernel_text_address check*
> *function-graph: allow unregistering twice*
> *tracing: Move mmio tracer config up with the other tracers*
> *function-graph: move initialization of new tasks up in fork*
> *tracing: Add ftrace events for graph tracer*

These fixes concern the actual implementation of tracing and working with the function call graph.

General Vectors (Linux kernel)

Vector #5: [timer, base, cpu, target, clk, idle, code, interval, posix, task, tick, jiffy, case, race, trace]

> *posix-timers: Fix full dynticks CPUs kick on timer rescheduling*
> *timers: Use proper base migration in add_timer_on()*
> *timer/trace: Improve timer tracing*
> *posix-cpu-timers: Unbreak timer rearming*
> *hrtimer: Preserve timer state in remove_hrtimer()*

These fixes are aimed at fixing the kernel code for implementing timers according to the POSIX standard (see for example a discussion on its userspace interface), namely errors during recharging (when changing the response time of already set timers), which entails working with related processes that may be located on temporarily retired processors.

General Vectors (Linux kernel)

Vector #6: [lock, ftrace, error, kernel, rlock, trace, deadlock, incompatible, type, possible, comparison, lockdep, info, irq, timekeeping]

> *timekeeping: Avoid possible deadlock from clock_was_set_delayed*
> *sched/core: Make dl_b->lock IRQ safe*
> *timekeeping: Fix HRTICK related deadlock from ntp lock changes*
> *cpu/hotplug: Drop the device lock on error*
> *pid: fix lockdep deadlock warning due to ucount_lock*

These fixes are related to the internal kernel elapsed time measurement subsystem and associated incorrect locking in the implementation.

General Vectors (Linux kernel)

Vector #7: [console, printk, boot, message, list, line, early, srcu, add, time, tracepoint, use, patch, problem, console_lock]

> *console: prevent registered consoles from dumping old kernel message over again*
> *CON_CONSDEV bit not set correctly on last console*
> *printk: don't prefer unsuited consoles on registration*
> *Revert "printk: Block console kthreads when direct printing will be required"*
> *console: allow to retain boot console via boot option keep*

This series of fixes is devoted to kernel diagnostic messages and their output via tty consoles.

General Vectors (Linux kernel)

Vector #8: [lockdep, lock, patch, time, code, cross, performance, release, run, second, boot, case, kernel, bug, counter]

> *lockdep: spin_lock_nest_lock(), checkpatch fixes*
> *lockdep, bug: Exclude TAINT_FIRMWARE_WORKAROUND from disabling lockdep*
> *lockdep: more robust lockdep_map init sequence*
> *locking/lockdep: Add a boot parameter allowing unwind in cross-release and disable it by default*
> *tracing: use raw spinlocks for trace_vprintk*

These fixes are related to the work of the lockdep deadlock prevention tool in the kernel and the work of the checkpatch tool to check the formal requirements of the patches associated with it.

General Vectors (Linux kernel)

Vector #9: [kernel, inline, bpf, event, type, btf, pid, trace, number, buffer, foo, lock, rip, code, cat]

> *bpf: prevent decl_tag from being referenced in func_proto*
> *tracing: Free buffers when a used dynamic event is removed*
> *coredump: fix crash when umh is disabled*
> *tracing: Fix memory leak in eprobe_register()*
> *tracing: Check return value of __create_val_fields() before using its result*

The fixes are related to BPF integration into the kernel and tracing, which were detected by the Syzkaller tool. Since the tool reports contain listings with the same keywords (register dump, call stack), they were detected as similar vectors.

General Vectors (Linux kernel)

Vector #10: [error, return, code, function, value, failure, case, cgroup, file, negative, ret, add, userspace, bpf, caller]

> *cred: add missing return error code when set_cred_ucounts() failed*
> *rcutorture: Fix error return code in rcu_perf_init()*
> *bpf: Fix error return code in map_lookup_and_delete_elem()*
> *ftrace: Deal with error return code of the ftrace_process_locs() function*
> *genirq/timings: Fix error return code in irq_timings_test_irqs()*

This series of fixes included fixes for the "fix error return code" error in various parts of the kernel, including the BPF and RCU torture functions.

The result of analysis of whole Linux kernel

- In general, based on this key subsystem of the Linux kernel, we can conclude that most of the problems found and corrected were associated with incorrect operation of multiprocessor concurrent systems due to incorrect processing of all scenarios in the control flow, which involve accurate processing in conditions of variability of resources such as processors, pages memory, etc.
- That is, the handling of unexpected situations was not carried out completely correctly.
- The key kernel components mentioned were the RCU subsystem, swap management, timing, BPF and tracing.
- The code identified problems were related with the correct processing of return codes.

The result of analysis of Linux drivers (/linux/drivers)

- Fixes for kernel drivers are distinguished by the presence of a large number of identical changes ("serial patches").
- Essentially, some change to the API is made and then the code for a large number of drivers that depend on that API should be changed[9].
- Such changes can be described in the form of so-called semantic patches and applied to a given set of files and also attempted to be generalized from a set of source code files[10].

[9] Lawall J., Muller G. Automating Program Transformation with Coccinelle. In Proc. of NASA Formal Methods Symposium, pp. 71-87, 2022.

[10] Serrano L., Nguyen V.A., Thung F., Jiang L., Lo D., Lawall J., Muller G. SPINFER: Inferring Semantic Patches for the Linux Kernel. In Proc. of 2020 USENIX Annual Technical Conference (USENIX ATC 20), pp. 235-248, 2020.

The result of analysis of Linux Memory management (/linux/mm)

- Modern changes in the Linux memory subsystem relate to virtual memory, large memory blocks and various tools for memory monitoring and debugging.
- Working with memory is so complex that without additional tools that are provided by the kernel, debugging and optimization are not possible.

The result of analysis of Linux Scheduling (/linux/kernel/sched)

- The fixes in the scheduling subsystem are mainly devoted to improving deadline, fair and realtime scheduling for groups, correct work with idle cpu state, calculation of deadlines and updating time counters, and work in NUMA systems.

The result of analysis of Linux IRQ (/linux/kernel/irq)

- As a result, we can say that the main fixes for IRQ in recent years have been devoted to code refactoring, improving work with IRQ affinity, providing some functions for export to other subsystems, and improving IRQ state management to support disabling interrupts and interrupt context transfer to other processors[11].

[11] Some more information on IRQ modeling: Staroletov, S. (2023, December). Towards Model Checking Linux Interrupts Behavior for SMP Systems. In 2023 Ivannikov Ispras Open Conference (ISPRAS) (pp. 150-156). IEEE.

The result of analysis of Linux x86 (/linux/arch/x86)

- Fixes for this historical part of the kernel are devoted to code reorganization, addressing known types of vulnerabilities, better support for virtualization, playing around with quirks for specific hardware, as well as providing support for legacy systems and subsystems.

The result of analysis of Linux ARM64 (/linux/arch/arm64)

- It turned out that a huge number of fixes for the ARM platform are devoted primarily to additions to device trees. There are a huge number of hardware configurations for the platform, and such configurations are described through a device tree in order to be accessible later programmatically by loading the appropriate drivers.
- Thus, for ARM64, the defining fixes are the reorganization of the device tree set, reorganization of the code, synchronization of access to the state of internal registers and operations with virtual dynamic shared objects.

The result of analysis of Linux ARM64 (/linux/arch/arm64)

- It turned out that a huge number of fixes for the ARM platform are devoted primarily to additions to device trees. There are a huge number of hardware configurations for the platform, and such configurations are described through a device tree in order to be accessible later programmatically by loading the appropriate drivers.
- Thus, for ARM64, the defining fixes are the reorganization of the device tree set, reorganization of the code, synchronization of access to the state of internal registers and operations with virtual dynamic shared objects.

Conclusion about Linux kernel analysis

- Our analysis yielded satisfactory results, presenting the main vectors (in the form of keywords) and example messages for each class of fix.
- We also conducted a manual summary analysis for each class, revealing typical fixes for each subsystem and the automated tools used to detect them.
- We referenced these tools in our article and concluded that developers must study them when working with the corresponding subsystem.
- By selecting specific dates and parts of the repository, it is possible to identify errors and gain valuable insights into their prevalence.
- However, there is always room for improvement in any analysis, and our current implementation could benefit from enhancements such as refining the detection of fixing commits, improving the commit coupling thresholds for hierarchical clustering, better normalization and removal of stop words, and developing methods for automatically describing each class of errors found.

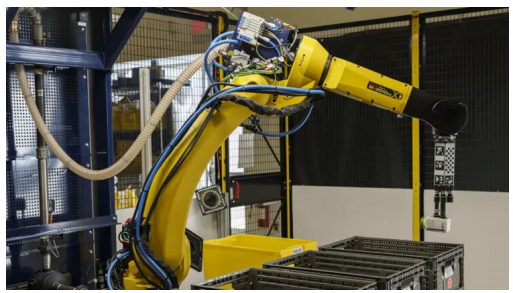
Part 2 of 2

- Part2: Analyzing repositories of cyber-physical systems (Joint work with N. Starovoytov)

Cyber-physical systems

- Cyber-physical systems (CPS) are symbiotic multi-level control systems that take into account the physical aspects of object operation. CPS integrate software components, hardware, and the physical environment, such as machinery, sensors, and actuators. These systems are actively applied in various domains including automotive industry, medical equipment, manufacturing processes, and smart cities, enabling the integration of digital and physical worlds to address complex tasks of automation, control, and monitoring.





Analyzed products

- Ardupilot[12] – an open-source autopilot software suite that enables autonomous control of drones, unmanned aerial vehicles, and other robotic systems.
- Scada-LTS[13] – is used to monitor and control industrial processes, infrastructure systems, and other complex systems that involve the interaction between physical components and digital control systems.
- Modelica[14] – Modelica is an open-source, object- oriented modeling language that is used to model complex cyber- physical systems that involve the interaction between physical components and digital control systems.
- KeYmaeraX[15] – theorem prover for hybrid systems, which are systems that exhibit both continuous dynamics (physical processes) and discrete dynamics (digital control algorithms).

[12] https://github.com/ArduPilot/ardupilot
[13] https://github.com/SCADA-LTS/Scada-LTS
[14] https://github.com/modelica/ModelicaStandardLibrary
[15] https://github.com/LS-Lab/KeYmaeraX-release

Errors in cyber-physical systems

- In the book «Cyber-Physical Systems»[16] authors propose various types of errors in cyber-physical systems can, such as design errors, implementation errors and etc. These errors can result in safety, system failures, performance, and security vulnerabilities.
- The research «Cyber-physical systems: a computational perspective»[17] presents numerous challenges due to the intricate nature of system interactions, real-time constraints, resource limitations, and the necessity for interdisciplinary expertise.
- Authors of «Deep learning-based anomaly detection in cyber-physical systems»[18] research investigative to advancements in formal methods, model-based design, runtime monitoring, anomaly detection, and machine learning technique.

---

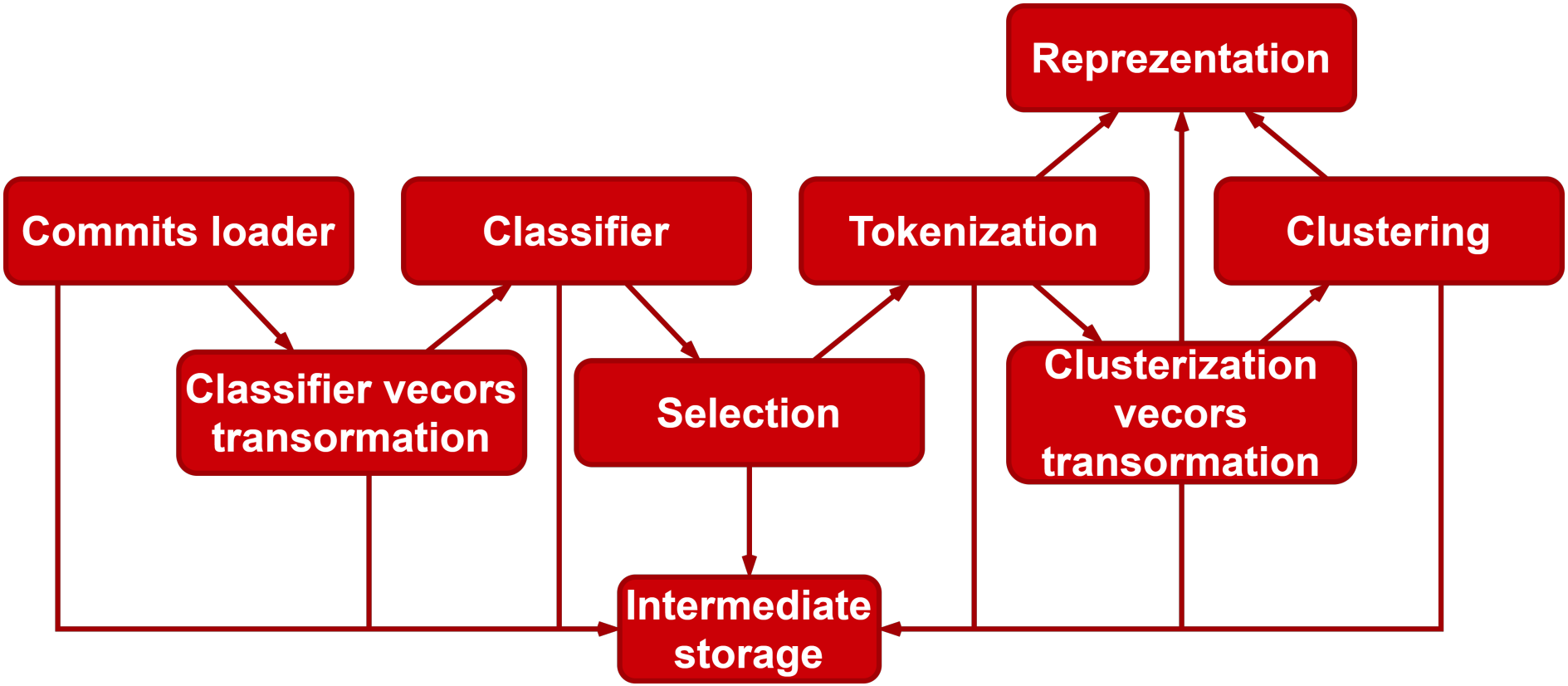
[16] G. M. Siddesh, G. C. Deka, K. G. Srinivasa, and L. M. Patnaik, Cyber-physical systems: a computational perspective. CRC press, 2015.

[17] 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.

[18] 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, 202

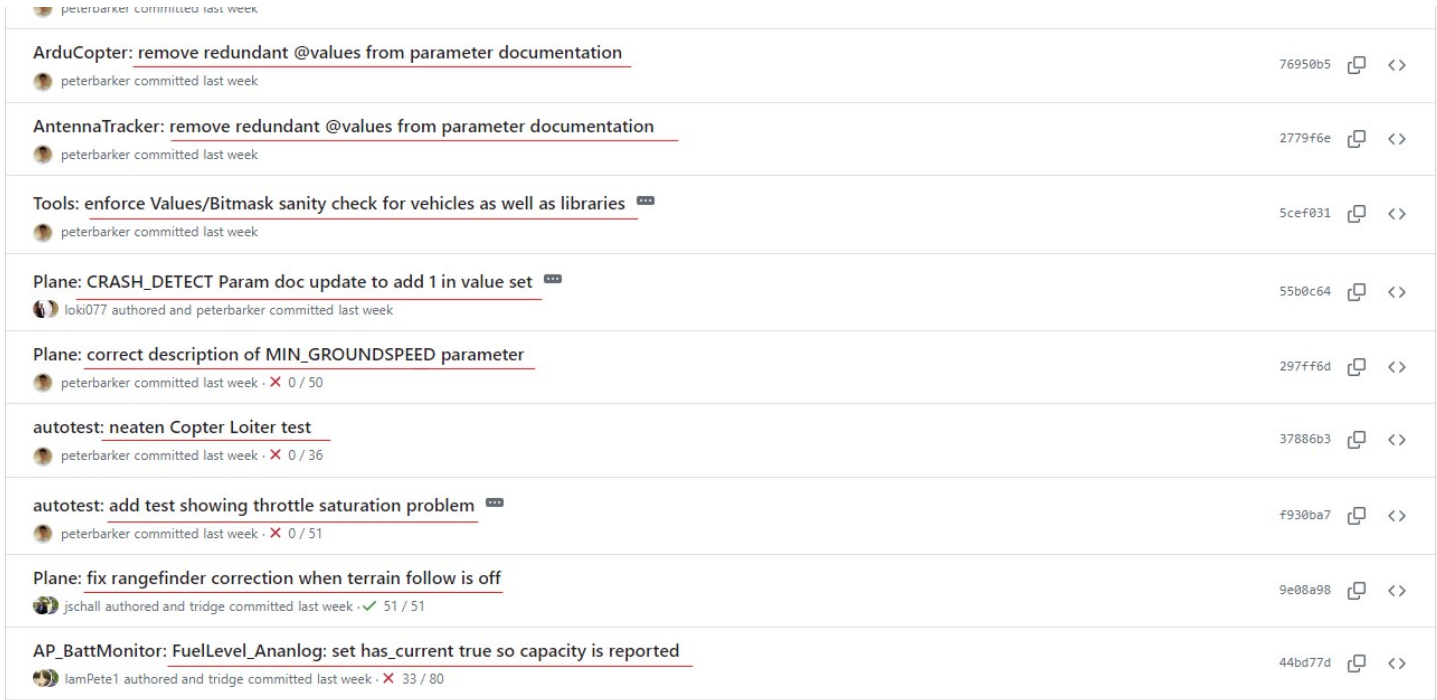
General implementation

Processing of commits analysis with machine learning methods:



Data source

- The analyzed repositories very rarely have a description body. Therefore, only headers are used as data.
- The word before the colon is omitted to prevent the formation of clusters around the names of files and modules.



Repository management issue

- Incomplete and uninformative commit messages make it difficult to understand the development history and trace the causes of problems. In the repositories being scanned, these commits appear too often.

Height DF value issue

Poorly maintained repositories have the problem of high DF values in dictionaries, indicating a lack of diversity in the tokens used and making it difficult to isolate key aspects of the code under study. What causes some problems:

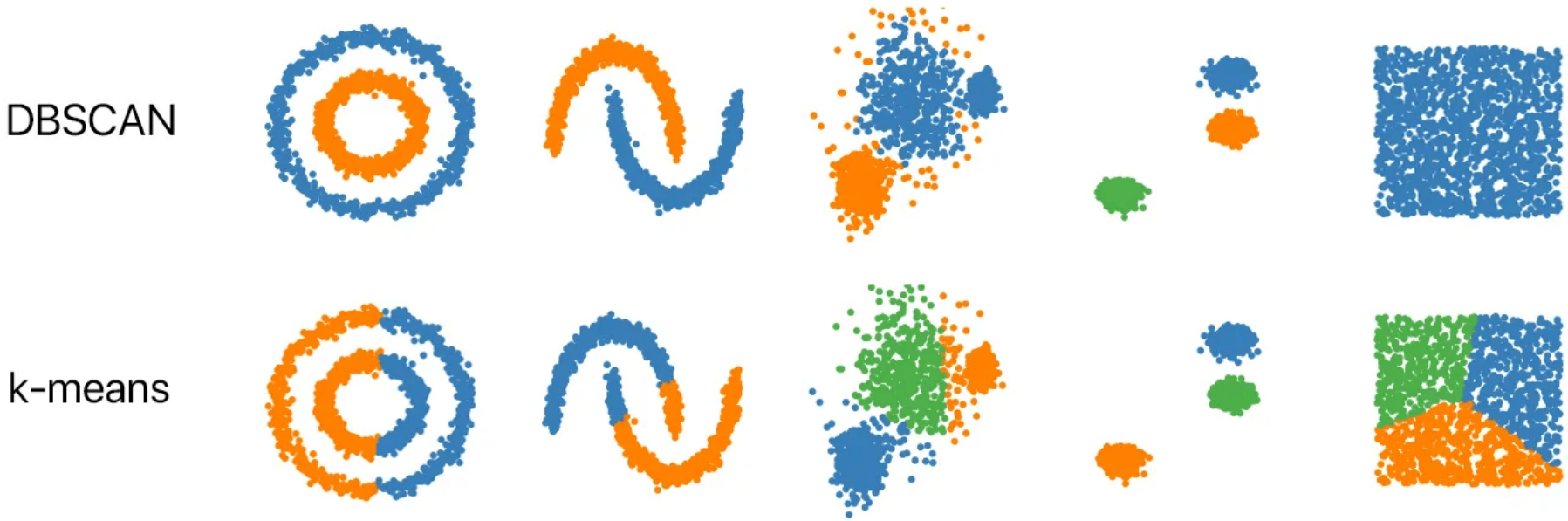
- Decreased accuracy of keyword extraction
- Increased noise in the data
- Difficulty in representation of results

LSA (Latent Semantic Analysis) implementation

- **Dimensionality Reduction** LSA helps reduce the dimensionality of the commit data, allowing for more efficient analysis
- **Semantic Similarity** by capturing semantic relationships between terms, LSA enables the identification of semantically similar commits, even if they use different terminology or wording
- **Noise Reduction** LSA helps filter out noise and irrelevant information from the commit data, focusing on the most significant semantic features and improving the quality of analysis results.

Clustering method

- Switching to clustering with the DBSCAN method can significantly improve the commit analysis process due to its ability to detect clusters with different densities and shapes. This method also automatically determines the number of clusters and is less affected to outliers, making it an ideal choice for working with datasets of different sizes and characteristics.



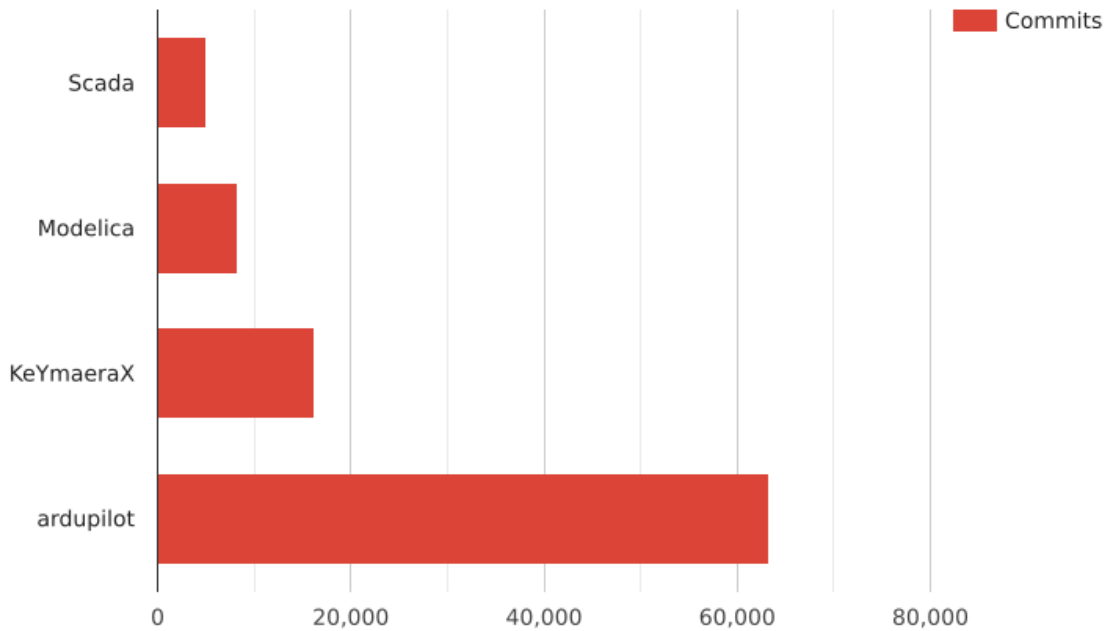
Parameter selection

- Small repositories are particularly sensitive to clustering parameters that require careful manual tuning. Their manual selection becomes critical to achieve optimal results. This involves experimenting with different parameter values and parameter combinations to find the most appropriate ones for a particular dataset.

Issue \ Parameter	count LSA units	min cluster size	eps
large cluster	increase	no change	increase
large outlier	decrease	decrease	decrease
large count clusters	increase	increase	increase
low relevance	decrease	increase	decrease

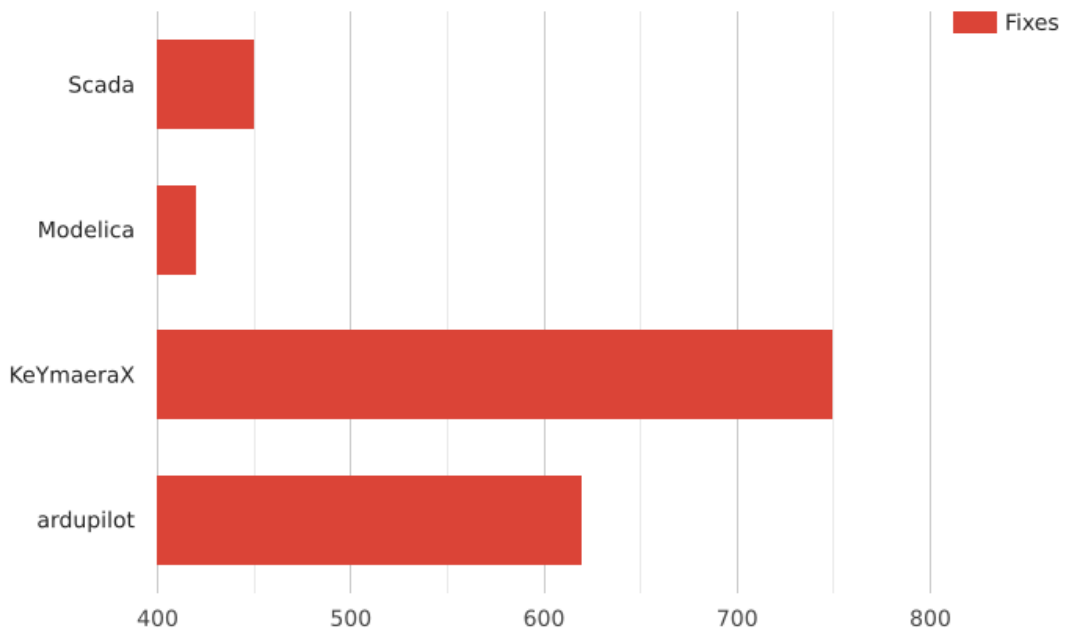
Size repository issue

- A few commits in cyber-physical system repositories limits the data available for analysis, making it difficult to identify patterns and long-term trends. Possible reasons for the low commit frequency include smaller development teams, less frequent updates, or more extensive testing and validation processes before committing changes.



Relevance commit count

- A classifier which was trained on one repository may perform poorly on others due to differences in coding practices, commit message conventions, and types of errors encountered. These differences reduce the size of a relevance defined sample of fixes and clustered data.



Possible solutions of the samples problem

To solve the problem of weak descriptions and small number of commits, several solutions are envisaged:

- Analyzing relationships between commits
  - Tracking code changes over time
  - Identifying patterns based on commit sequences
- Augmenting data with machine code
  - Extracting semantic tree data
  - Retrieving information about function names and variables
- The use of metainformation
  - Analyzing meta-information such as commit author, time and related files
  - Combining different data sources to get a more complete image

Summary for repository issues

- All the analyzed repositories have problems in one way or another in clearly recognizing commit clusters.

Issue \ Repository	Scada	Modelica	KeYmaeraX	ardupilot
weak description	+	+	+	+
low commit count	+	+		
fix classification		+		+
low relevance	+	+	+	

Summary for repository issues

- For each repository, its own set of parameters has been determined that give the optimal result in terms of the number and proportions of clusters.
- The fewer problems there are in the repository, the more strict the parameters can be specified.

parameters / Issue	Scada	Modelica	KeYmaeraX	ardupilot
min cluster size	5	5	10	15
eps	0.23	0.23	0.38	0.45
LSA units	100	100	200	300

Ardupilot clusters examples

	floating-point values fixes. 'apvehicle', 'compiler', 'float', 'override', 'keyword', 'old', 'airspeed'
1	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) $AP_Compass : compilerwarnings : applyis\_zero(float)oris\_equal(float)$ AntennaTracker: compiler warnings: apply is_zero(float) or is_equal(float)
	improvement to the hardware definition for hardware abstraction level 'hwdef', 'binding', 'copter', 'register', 'match', 'changes
2	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

Ardupilot clusters examples

	support the Hardware-in-the-loop simulator. 'hil', 'initial', 'wrapper', 'implementation', 'roll', 'tailsitter'
3	ACM: fixed HIL build again ArduPlane: remove HIL support AP_Compass: remove HIL support Blimp: remove HIL support GCS_MAVLink: remove HIL support
	define macros in code. 'define', 'separate', 'send', 'patch', 'stability', 'simple'
4	add separate define for AP_RCPROTOCOL_PPMSUM_ENABLED add separate define for AP_RCPROTOCOL_ST24_ENABLED add separate define for AP_RCPROTOCOL_DSM_ENABLED add separate define for AP_RCPROTOCOL_SUMD_ENABLED add separate define for AP_RCPROTOCOL_IBUS_ENABLEDf

Ardupilot clusters examples

	fixes for location objects. 'location', 'adjust', 'vector3f', 'require', 'accepts'
5	autotest: fixed buildlogs location for *.BIN AP_Math: move line_path_proportion to Location added some filtering and smoothing Tweaks to fix Loiter Changed save location to int32
	fixes for obstacle avoidance functions. 'fence', 'without', 'register', 'attitude', 'pointer'
6	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

Scada clusters examples

	fixes for DAO (data access object). 'slts13', 'testdao', 'annotation', 'cleanup', 'scriptdao'
1	Added FlexProjectRowMapper and remove @SuppressWarning annotation move expectedException to TestDAO Rewrite ScriptDAO controller's clean-up Implemented direct link navigation
	patches for views (that is, representing the states of objects on the screen) 'correction', 'commit', 'display', 'partial', 'isalive2', 'viewdao
2	Correction of commit number display. 1502 2051 Visual corrections (component shrink) EventDetectorAPI corrections 153 2051 Add corrections to the IsAlive2 SLTS-40 Add correction to ViewDAO

Modelica clusters examples

1	fixes for working with time events 'unify', 'time', 'ccr', 'event', 'enhancement'
	refs 1627: Fix detection of (scaled) time events refs 1627: Fix detection of scaled time events CCR: corrected error in setState_ps
2	fixes for complex blocks with images 'due', 'picture', 'complexblocks', 'modification', 'docu'
	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 Attempt to fix issue 3236

KeYmaeraX clusters examples

	addition of various functionalityto the project 'implement', 'syntactic', 'derivative', 'index'
1	implement GetPathAll implement the CreateProblemRequest implement BranchRoot implement skolemization implement more of skeleton
	fixes for the unification algorithms 'unification', 'derive', 'bidirectional', 'axindex', 'imply'
2	Colored-dots unification Improve unification a bit further Unification support for projection Unification match: 0-indexed colored dots DiffHelper derive fix (sign error)

Cluster analyzing results

- Ardupilot project has the main errors that are specific to the software domain for unmanned vehicles, such as issues of location processing, obstacle avoidance, abstraction from hardware and scheduling.
- For the SCADA system, errors specific to Java applications and MVC architecture were found, which is not surprising, because such projects visualize monitoring data of cyber-physical systems.
- In the KeYmaeraX, a theorem prover for cyber-physical systems, 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.

Conclusion for cyber-physical systems analysis

- The clustering results 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.
- The main problems to analyze in the organization of development of the selected projects, when developers write uninformative messages about commits.
- The advantage of our solution is the ability to easily evaluate what is being done in a given repository and what errors were corrected, in order to learn from examples of the development of this kind of systems with increased reliability requirements.