 Acta Univ. Sapientiae Informatica **15,** 2 (2023) 239–266

DOI: 10.2478/ausi-2023-0016

Parallelising semantic checking in an IDE: A way toward improving profits and sustainability, while maintaining high-quality software development

Kristóf SZABADOS
Eötvös Loránd University,
Budapest, Hungary
email: SzabadosKristf@gmail.com

Abstract. After recent improvements brought the incremental compilation of large industrial test suites down to a few seconds, the first semantic checking of a project became one of the longest-running processes. As multi-core systems are now the standard, we derived a parallelisation using software engineering laws to improve the performance of semantic checking.

Our measurements show that even an outdated laptop is fast enough for daily use. The performance improvements came without performance regressions, and we can't expect additional massive benefits even from infinitely scaling Cloud resources.

Companies should utilise cheaper machines that still offer enough performance for longer. This approach can help businesses increase profits, reduce electronic waste and promote sustainability while maintaining high-quality software development practices.

Key words and phrases: parallel computing, cloud computing, semantic checking, integrated development environment, software development tools, software engineering laws, TTCN-3, performance improvement, sustainability, cost reduction, profit increase

1 Introduction

Software is ubiquitous in modern society. It helps us navigate, communicate, and manage energy resources. It drives companies, trades on markets, and supports healthcare.

As software products grow, so do their test systems. Some industrial test systems contain millions of lines of code [1, 63]. For a long time, compiling such codes for several minutes was the most time-consuming part of developers' daily work. Companies used clusters of remote servers or Cloud solutions to make the required performance available.

But, recent improvements [11] brought the incremental compilation of such systems down to a few seconds, leaving the first single-threaded semantic checking of the IDE as one of the longest-running processes. This process can still take several seconds on our industry partner's codes. Too long for an IDE that should be interactive.

However, nowadays, single-threaded execution is an unnecessary constraint, with multi-core and multi-CPU hardware readily and commercially available.

In this paper, we report on how we improved the IDE of our industry partner with parallel processing of the semantic analysis, making better use of available processing power. Our industry partner might no longer need to use Cloud or remote servers, as the laptop their employees would use to reach those services, might already be powerful enough.

We organised this paper as follows. Section 2 presents related works. Section 3 shows a technical description, and 4 is our proposal for the opportunity. Section 5 presents our measurement method, and 6 shows our measurements and observations. Section 7 their validity. Finally, 8 shows our summary, and 9 offers ideas for further research.

2 Related works

In this section, we present earlier related works. In the first group of sections, factors that serve as general requirements for our chosen solution and its general applicability: organisations intentionally design and govern the overall architecture of their products to achieve their business targets (Section 2.1), the generality and inevitability of the internal structure of large software systems (Section 2.3), unavoidability of dependency cycles (Section 2.5), all software systems evolving similar size distributions (Section 2.4).

In the second group of sections, we present how organisations use Project Management to predictably deliver the right products at the right time [21]

(Section 2.2), how all software systems evolve in a similarly predictable way (Section 2.6) to show that our chosen method permanently solves the problem.

2.1 Previous work on the impact of organisational factors on software systems

Empirical observations have identified an isomorphic relationship [71] between an organisation's communication structure and product structure, known as Conway's law [7] or Mirroring law [39]. Nagappan et al. [46] showed that organisational metrics predict failure-proneness better than code complexity, coverage, internal dependencies, churn and pre-release bug measures. This phenomenon was first recognised [39] for its significant managerial impact in 2012. By 2022 researchers observed it as likely the superior strategy used by 92% of investigated firms [6] and that alignments and "mirror breaking" in organisations are strategic to maximise business benefits [38].

Following these laws, contemporary System Architects take several environmental factors (among others: Taxation [10], Export Control [5], Geopolitics [50] and Standardisation[1]) into consideration when planning software architecture and the organisation developing it. Contemporary Project Management recommends [22] tailoring a selected development approach first for the organisation and second to the project. To ensure that individual project decisions do not threaten larger strategic goals.

2.2 Previous work on the impact of project management on software projects

Researchers have identified [68, 55, 43] that Project Management[2] techniques and processes are the critical factors contributing to project success.

Empirical observers have noted [8] that 94% of troubles and possibilities for improvement are the responsibility of management. Instead of rewarding managers for solving crises and heroes for putting out fires, companies should reward managers for preventing problems with systemic problem-solving performed with scientific rigour [17, 59, 4].

Project Management has a long history of evolving systematic practices [70] since the first use of Test Driven Development [3] by the first programmers

[1] As a form of Test-First Development, companies join to create and standardise automated tests, precisely determining required capabilities and interfaces for future products [29].

[2] Defined as "the application of knowledge, skills, tools, and techniques to project activities to meet the project requirements" [21].

in the 1940s [26, 2], Iterative and Incremental Development practices in the 1950s.

Nowadays, Professional Managers can use Agile and Lean methodologies to detect and reduce unnecessary activities before they happen. They can delay their decisions until the last responsible minute. They can direct developers to follow test-first practices to ensure the quality of new developments. Managers can use Continuous Integration to detect unapproved changes to supported requirements. Understanding, that writing programs "is only a small part of Software Engineering" ([51]).

2.3 Previous work on the dependency networks of software systems

Empirical researchers have shown that several architectural properties of software systems are scale-free like many real-life networks. Class collaboration graphs of the C++ language [45], Class, method, and package collaboration graphs of Java [20, 69], connections between the modules in TTCN-3[3] [61, 63], file inclusion graphs in C [44], and the object graph (the objects instances created at runtime) of most of the Object Oriented Programming languages in general [53], the relationships of distributed software packages [31, 28] show scale-free properties.

Taube-Schock et al. [65] showed, that approximately scale-free structures should arise for both between-module and overall connectivity from the preferential attachment-based models (like software), not as the result of poor design. Concluding, that high coupling is not avoidable, and might even be necessary for good design, contradicting previous ideas about software structure, in particular the "high cohesion/low coupling" maxim.

2.4 Previous work on the size distribution of software systems

Empirical studies have revealed that several metrics correlate to the point of redundancy[4] [40]. Measuring SLOC would be enough to obtain a landscape of the evolution of the size and complexity of FreeBSD [19]. Stating that "whatever is measured in a large scale system" the graph shows similar logarithmic distribution in most cases [1].

[3] Testing and Test Control Notation Version 3

[4] Cyclomatic Complexity, the Number of Lines of Code, Statements, Classes, Files, public APIs, and public undocumented APIs are redundant metrics, with Cyclomatic Complexity in classes and functions measuring the same subject

Empirical researchers have shown module length distribution of IBM 360/370 and PL/S code forming logarithm shape [58], Java class sizes following lognormal distribution [73], token distribution in Java code following Zipf's law [72, 74], all metrics measured on FreeBSD following lognormal and power-law distributions [19], double-Pareto distribution for five (C, C++, Java, Python, Lisp) of the top seven programming languages used in the Linux code [18] and LOC following lognormal distribution in Smart Contracts written in Solidity deployed on the Ethereum blockchain [66].

Hatton proposed [15] that the Conservation of Hartley-Shannon Information might play the same role in discrete systems as the Conservation of Energy does in physical systems, proving [16] Zipf's Law in the case of homogeneous systems and showing strong evidence for unusually long components being an inevitable by-product of the total size of the system. He validated the claims on 100 million lines of code in 7 programming languages and 24 Fortran 90 packages.

Hatton also highlighted the importance of changing software design techniques, from attempting to avoid the essentially unavoidable to mitigating its damaging effects.

2.5 Previous work on circular dependencies in software systems

Empirical studies have shown the existence of circular dependencies in several successful and known programs written in Java [49, 42, 9], C# [49], TTCN-3 [64], even in the binaries of Windows Server 2003 [75]. These studies offer empirical evidence for the understanding that if a program has enough components to support them, it is likely to have dependency cycles (with cycle sizes of 1000 classes [42] or involving a substantial part of all classes [49]).

2.6 Previous work on the evolution of software systems

Since Lehman started his work on software evolution [36], showed that commercial systems have a clear linear growth [37] and published the laws of software evolution [34], this phenomenon has been attracting researchers.

There is plenty of empirical research [37, 35, 33, 24, 27, 23, 25, 54, 30, 76] in which the authors show that the laws seem to be supported by solid evidence.

Turski even showed [67] that the gross growth trends can be predicted by a mean absolute error of order 6%. Also observed by others [13, 30, 76].

Looking at the impact of outside effects on software growth, empirical researchers observed [30] that "the introduction of continuous integration, the existence of tool support for quality improvements in itself, changing the development methodologies (from waterfall to agile), changing technical and line management structure and personnel caused no measurable change in the trends of the observed Code Smells", on industrial Java and C++ projects [76], that changing architects, going open source or the organisation moving to a different building had no easily discernible effect on development.

The works performed on large open source systems [24, 13, 35, 27, 23, 25, 76] serve as observations supporting the understanding that there was no hardware, software, tooling, methodological, social, or other change at least since 1995, that would have significantly changed the development speed of large software systems already started.

3 Opportunity description

In this section, we present the opportunity in more technical terms. We describe what is already available in Titan [81], and what we were working with. We show what we can expect from large-scale scale-free systems, and how circular dependencies constrain our work.

3.1 What was already available and what we worked with

The tool of our industry partner performed syntactical analysis of source files in parallel. The complete semantic checking of the parsed modules was available, done sequentially. Parsing [62] and semantic checking [48] could be incremental.

Semantic analysis happens recursively, starting with the components of a semantic entity (including entities contained within or reachable via references), followed by itself, checking each semantic entity at most once per semantic checking.

Although the optimal order is unknown before the first semantic checking of a project, executing the process for each module in the project, in any order, will check every semantic entity exactly once.

As semantic checking was single-threaded, the implementation does not have any locks or guards against parallel access/modification on the level of the entities. Locks only protect against overlapping semantic checks on a project.

3.2 Constraints on parallel processing

Previous works have already revealed properties of software systems that can be constraints for reaching optimal parallelization:

- Large differences in module sizes (Section 2.4):
  - Uneven size distribution means uneven processing time.
  - Processing order of independent modules matters for performance.
- Scale-free distribution of dependencies (Section 2.3):
  - The maximum number of imports in a module being $\geq$ 10% of the modules, semantic checking such a module might also check all reachable modules in the same thread, sub-optimal parallelization.
  - Many modules depending on the same module creates contention waiting for its processing to finish. Large amounts of modules become available when that happens.
- Dependency cycles (Section 2.5):
  - The modules of dependency cycles can't be processed in parallel safely without the risk of introducing deadlocks, but processing a cycle as 1 unit by the same thread is sub-optimal parallelism.
  - Before checking any module from a cycle, all depend on a not yet checked module. Without deep pre-processing, it is not possible to break cycles optimally.
- Gathering more information also costs time [48]:
  - Sequential pre-processing for optimal parallelism might make the entire process last longer.
  - Only the import relations and the number of definitions/assignments in a module are known without semantic checking.

As software evolves predictably (Section 2.6) towards long-standing business goals of an infrequently changing organization (Section 2.1) and governed by people dedicated to ensuring smooth progress (Section 2.2), once some hardware becomes fast enough it will stay that way for a long time. Moore's law and the linear growth trend of software systems can ensure a practically permanent solution.

4 Our method

Our method structures the semantic checking of the list of modules into an initial sequential part, a parallel part, and a final synchronisation point.

Initial sequential part:

1. The list of modules to be processed is ordered descending according to the number of definitions/assignments inside them.
2. An executor service with a thread pool is created and filled with new runnable tasks for each module without imports.

In the parallel part:

1. When a thread has processed its module, it creates a new runnable task for each module not yet processed but having all their imported modules processed.
2. To break cycles, if the thread is the last running and it does not find any new modules to process, while there are still modules to be processed, it selects the first not yet processed module for processing.

The processing ends when all modules are processed.
Titan developers merged our changes with a bug fixed later[5].

5 Measurement methodology

In this section, we present our measurement methodology and its calibration. At the timing precision required, we had to set up a measurement methodology that let us separate the effects of our method from those of the environment.

First, we explored the limitations of the hardware[6] (5.1). Then, we generated a project that could support ideal scaling (5.2), decided on the default measurement process (5.3) and performed exploratory data analysis on 10.000 measurements to calibrate it (5.4). Finally, we used this methodology on the ideal project to establish a baseline (5.5).

[5] https://gitlab.eclipse.org/eclipse/titan/titan.EclipsePlug-ins/-/merge_requests/918, last accessed: 2023.05.15

[6] Measurements were performed on a Lenovo Legion R7000 laptop, with an AMD Ryzen™ 7 4800H 8-core 16-thread CPU (at a base Clock of 2.9GHz that can boost to 4.2GHz), 1*8 GB Kingston 3200 MHz SODIMM RAM, using an SSD, running Windows 10 Home and the 3.0.7-1 version of Cygwin, GCC 11.3.0, Eclipse 4.20.0, Java SDK 16.0.2 .

5.1 The limitations of the hardware

We explored the relevant limitations of the hardware using STREAM [41][7].

The source code was compiled with: gcc -O2 -DSTREAM_ARRAY_SIZE = 80000000 -DNTIMES=100 -fopenmp stream.c -o stream.80M.exe

We performed execution from the command line, controlling the thread count via the "OMP_NUM_THREADS" environmental variable.

Table 1: The maximum and minimum STREAM results compared to the single-threaded case and their thread count location.

	max bandwidth		min bandwidth	
Project vs test	ratio	location	ratio	location
Copy	+18.58%	4	+9.47%	22
Scale	+3.27%	2	-14.93%	18
Add	+4.34%	2	-17.93%	18
Triad	+4.59%	3	-18.18%	16

Our measurements show (Table 1) the most bandwidth available using 2-4 threads. At higher thread counts, for all but the "Copy" function, the measured memory bandwidth is below the single-threaded case. The measured values fell into a limited range (Figure 1).

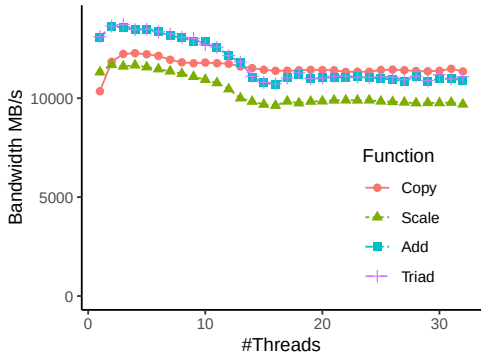


Figure 1: Memory Bandwith measured by STREAM.

[7] https://www.cs.virginia.edu/stream/FTP/Code/, version ("5.10") downloaded in May. 2022., last accessed: 2023.05.15

In practical terms, the hardware does not support increasing thread counts with enough bandwidth for memory-bound operations. The hardware also seems to suffer from some constraints or limitations, as the numbers we measured do not reach the peak transfer rate of the standard. *This laptop is a good sample for an outdated laptop, that was never meant for professional development work and cannot support theoretically ideal scaling.*

5.2 The Ideal project

To explore the parallel limits of the solution in finer detail and to calibrate the measurement process in an ideal setting, we created a synthetic project, "Ideal", by copying the RAWCodingAttributes.ttcn file from [12] 16 times[8].

In this project, all files are standalone and processable in parallel, have the same content, and need identical processing. The files are large enough[9] for the analysis to be practically measurable, and their number matches the CPU thread count.

5.3 The default process

We decided to use the closing and opening of a project for measurement, as this deletes all information the IDE knows about that project and triggers its analysis.

We created a small program to perform measurements in a loop, sleeping at the end of each iteration (eliminating its potential impact).

Settings used for the measurements:

- The measurement Eclipse was started with:

  -Xms4G -Xmx4G -XX: UseG1GC-server

- The laptop was in "Quiet Mode"[10].
- The analysis threads were started with priority 1 (lowest) and after checking each Assignment[11]/Definition[12] a "Thread.yield()" call is executed.

[8] Adding their index to both the file and module name to forego collisions.

[9] 7090 lines.

[10] Description from manufacturer: "Keep quiet by reducing your computer performance and fan speed where possible with low power consumption".

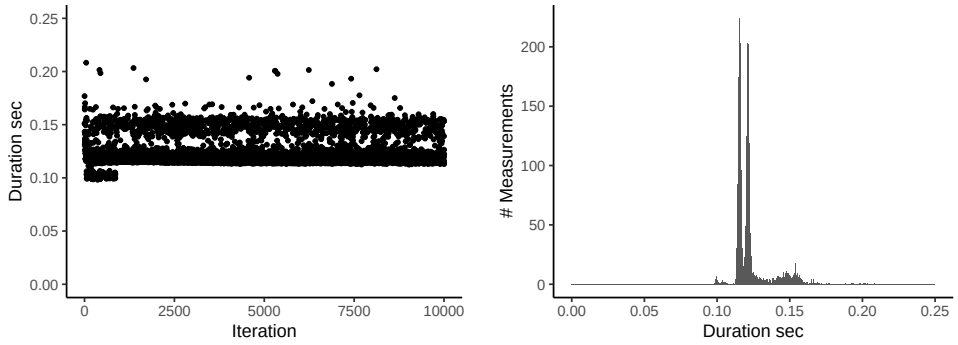
[11] https://www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf, last accessed: 2023.05.15

[12] https://www.etsi.org/deliver/etsi_es/201800_201899/20187301/04.14.01_60/es_20187301v041401p.pdf, last accessed: 2023.05.15

5.4 Calibration of the measurement process

During our measurements, we had to account for the Just-in-time compilation and optimisation of Java, the language of the IDE.

We set the helper program to perform 10.000 measurements.



(a) Direct measurements

(b) Histogram

Figure 2: The execution durations for the iterations (2(a)) and their histogram (2(b)) for a 10.000 iteration execution. For visualisation purposes, the first execution (taking 1.6 seconds) is not shown.

Our observations showed (Figure 2) that the first measurement is an outlier. The following measurements form 4 clusters using K-means clustering (means: 0.1021, 0.1158, 0.1225, 0.1536).

At this point, we could devise the routine for measurements:

1. To determine a sufficient sleep duration, perform three iterations manually with the new settings set.
2. Turn off all potential interference and restart the hardware.
3. Do three iterations manually.
4. Execute the automation code for 200 iterations and extract the last 50 measurement points ([14]).

5.5 Measurements on the Ideal project

The overall duration of analysing the project (Figure 3) drops from an average of 0.39s at 1 to 0.23s at 2, 0.147s at 4, 0.131s at 8, and 0.13s at unlimited thread counts. The maximum speed-up of 3.18 is at 10 threads. In practical terms, most performance gains happen till 4-thread parallelism. Higher thread counts

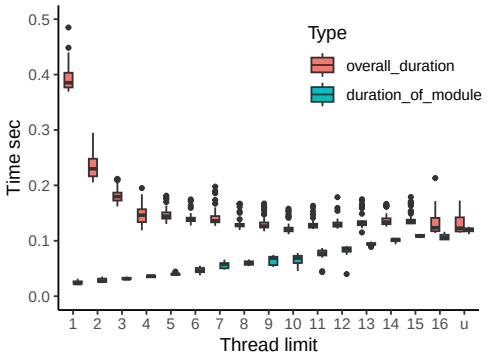


Figure 3: The overall duration for the semantic checking on the ideal project, and the single module duration from the closest to average execution, for each thread limit (u meaning unlimited).

seem to produce only marginally better results. At the same time, there is also no reason to use any smaller setting in practice.

The individual module analysis times from the closest to the average execution of that thread count revealed that the analysis duration of a given module approximately linearly depends on the thread count, indicating that the hardware is memory bandwidth limited.

6 Measurements

In this section, we present our measurements.

Section 6.1 presents that our chosen method solves the problem on large code bases and that further parallelisation is limited by the structure of the problem, not the hardware. Section 6.2 shows that there are no noticeable performance degradations in existing features.

6.1 Standardized test suites

We have analysed the behaviour of our method on all test suites created by 3GPP[13] and publicly available at `www.ttcn3.org`[14].

Used test suites:

- 3GPP UTRA UE Test Suites [80]: SSNITZ, UTRAN.

[13] 3rd Generation Partnership Project

[14] We downloaded all packages in July 2021.

- 3GPP IMS UE Test Suites [78]: IMS_EUTRA, IMS_IRAT, IMS_NR5GC, IMS_UTRAN, IMS_WLAN.
- 3GPP LTE UE Test Suites [77]: LTE, LTE_A_IRAT, LTE_IRAT.
- 3GPP 5G UE Test Suites [79]: ENDC, NR5GC, NR5GC_IRAT.

To process these test suites, we created a new "TITAN Project (Java)" with the name of the TTCN-3 project, copied the source code from the downloaded compressed files into its "src" folder, and converted all XSD files[15]. We manually reviewed all problems detected by Titan on these projects and reported the incorrect ones to the development team[16].

Table 2 presents the most important properties of these projects and table 3 shows how well the logarithmic and power-law trend lines fit the measured data for each project. Figure 4 presents the analysis durations measured for each project.

Table 2: Importation data: number of modules, layers, maximum number of imports, maximum number of being imported and lines of code

Project vs test	Nof modules	Nof layers[17]	I_{max}(project)	O_{max}(project)	LOC
SSNITZ	86	16	30	50	105.191
UTRAN	175	21	46	115	158.392
IMS_EUTRA	207	20	40	131	106.429
IMS_IRAT	207	19	40	140	182.561
IMS_NR5GC	164	20	43	116	83.858
IMS_UTRAN	128	17	35	72	108.732
IMS_WLAN	103	16	24	57	41.297
LTE	230	23	50	171	249.161
LTE_A_IRAT	266	20	42	187	219.496
LTE_IRAT	289	21	78	211	257.719
ENDC	250	22	49	190	152.194
NR5GC	201	19	47	161	130.907
NR5GC_IRAT	242	20	47	178	141.135

6.1.1 Measurement: LTE IRAT

In this section, we present our observations for the largest project processed.

[15] Using the "xsd2ttcn" utility of Titan, using the -N flag.

[16] https://gitlab.eclipse.org/eclipse/titan/titan.EclipsePlug-ins/-/issues/456, last accessed: 2023.05.15

Table 3: Trend fitting

	log r^2		power law r^2	
Project vs test	I(module)	O(module)	I(module)	O(module)
SSNITZ	0.92	0.96	0.81	0.83
UTRAN	0.97	0.82	0.84	0.92
IMS_EUTRA	0.97	0.81	0.80	0.91
IMS_IRAT	0.98	0.69	0.85	0.90
IMS_NR5GC	0.97	0.68	0.85	0.89
IMS_UTRAN	0.97	0.81	0.85	0.92
IMS_WLAN	0.95	0.90	0.84	0.90
LTE	0.94	0.87	0.78	0.87
LTE_A_IRAT	0.98	0.73	0.80	0.87
LTE_IRAT	0.97	0.78	0.78	0.87
ENDC	0.98	0.72	0.82	0.90
NR5GC	0.95	0.84	0.76	0.89
NR5GC_IRAT	0.98	0.70	0.83	0.88

The average overall duration of the analysis (Figure 5) takes 25.16s using 1 thread, 14.25s using 2 threads, 11.92s using 3 threads, 11.01s using 4 threads, 9.49s using 8 threads, 10.30s using 16 threads and 10.10s without thread limits. The overall speed-up is 2.48 without thread limits (2.64 using 8 threads).

Module "RRC_MeasurementUG" takes the longest to analyse, with 6.78s using 1 thread, 7.76s using 2 threads, 8.07s using 3 threads, 8.39s using 4 threads, 8.24s using 8 threads, 8.78s using 16 threads and 8.35s without thread limits, a slowdown of approx. 23%.

With unlimited threads, the longest to analyse path contains the modules "LTE_IRAT_Testsuite" with 0.03s, "RRC_MeasurementUG" with 8.35s, "EUTRA_Measurement_Templates" with 0.90s and "EUTRA_RRC_Templates" with 0.13s to analyse. 9.41s of the 10.10s, or approx. 93% of analysing the project with unlimited threads. *Although this is not the critical path, it still shows that no amount of increase in parallel processing threads/CPU cores alone would be able to decrease the processing time further significantly.*

The duration values for analysing a given module in descending order (Figure 6) are similar to the length values of modules plotted in descending order. With the exponential trend lines fitting to all thread limit cases with at least r^2 of 0.97 and power trend lines between 0.84 and 0.88.

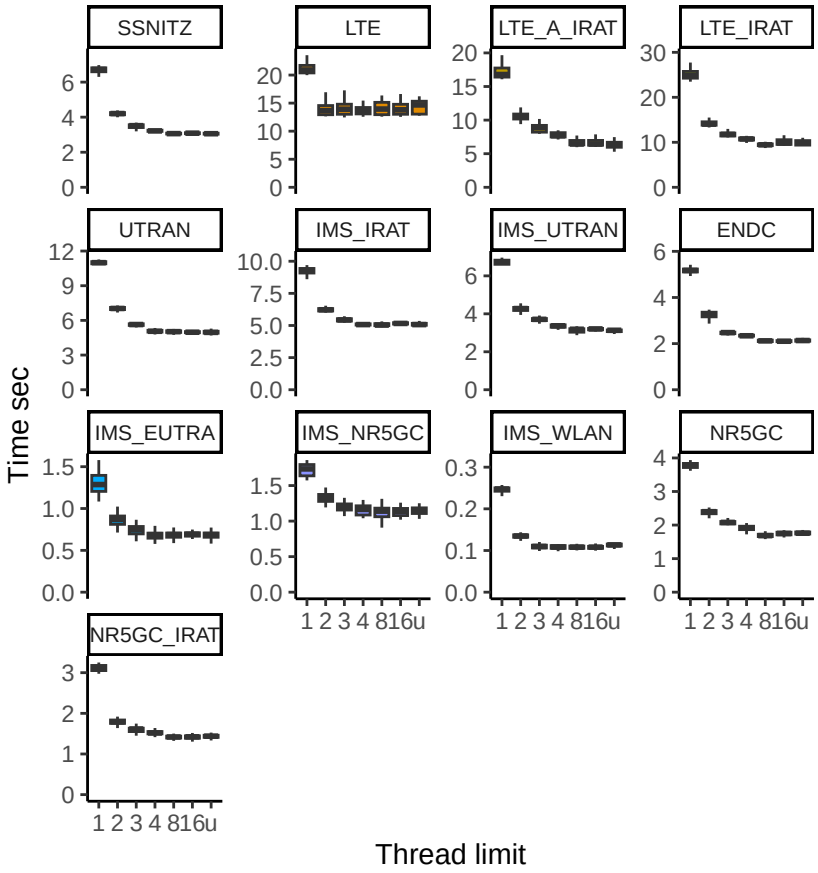
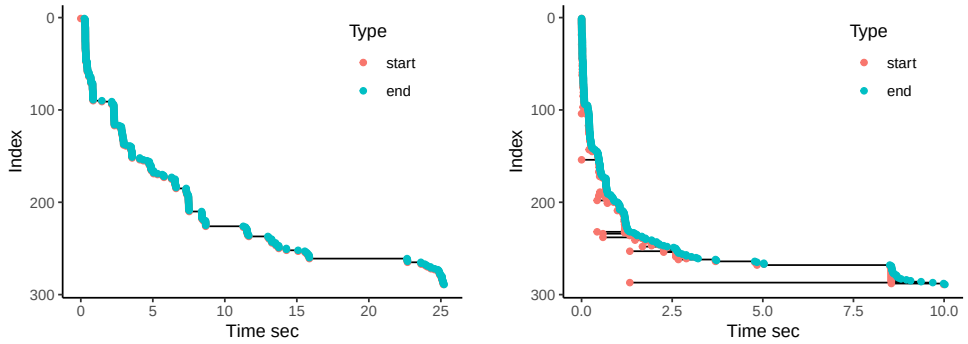


Figure 4: The overall duration, for each thread limit (u meaning unlimited) and each measured standardised test suite.

Without thread limits (Figure 7(b)), the maximum number of active threads[18] is 23. *Indicating that at most* 23 *threads can work in parallel, additional hardware capacity can not be utilised.*

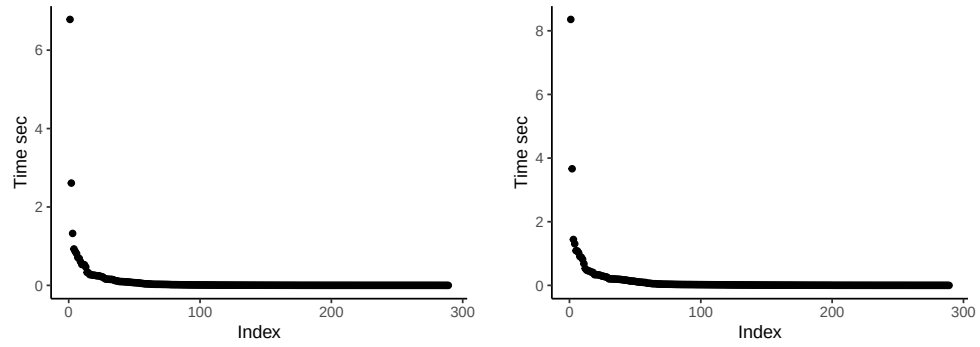
With thread limits set (Figure 7(a)) the number of active threads below the actual thread limit is 0 for 1, 2 for 2, 7 for 3, 14 for 4, 121 for 8 and 270 for 16 thread limit. *At small thread limits, parallel processing operates at or near*

[18]In our measurements we call active threads, the software threads that are actively running at the time of measurement. Where the measuring is done in the "Runnable" object's "run()" function is called, right before it starts to analyse its module.



(a) 1 thread limit (b) Unlimited thread count

Figure 5: Execution trace for the LTE IRAT modules, when the analysis is limited to 1 threaded processing 5(a) and when there is no thread limit (5(b)).

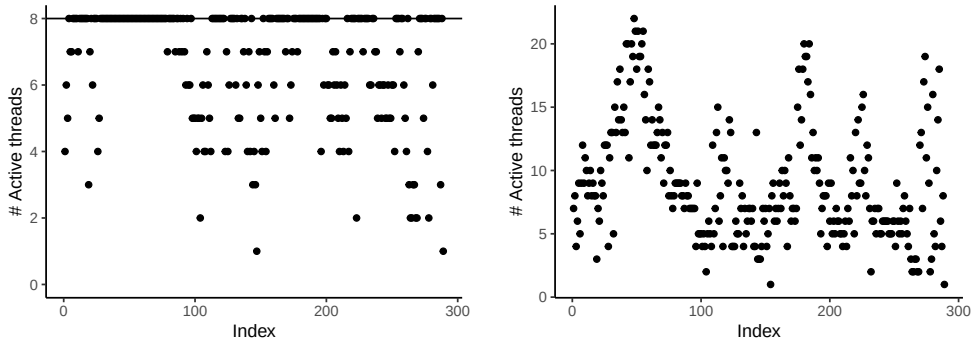


(a) 1 thread limit (b) Unlimited thread count

Figure 6: The time it takes to analyse a given module, in descending order, when there is a 1 thread limit (6(a)) and when the number of threads is not limited (6(b)), while checking LTE IRAT.

*maximum capacity. At high thread limits (*8 *and* 16 *measured), the structure of the dependency graph and the runtime processing is the stronger restriction.*

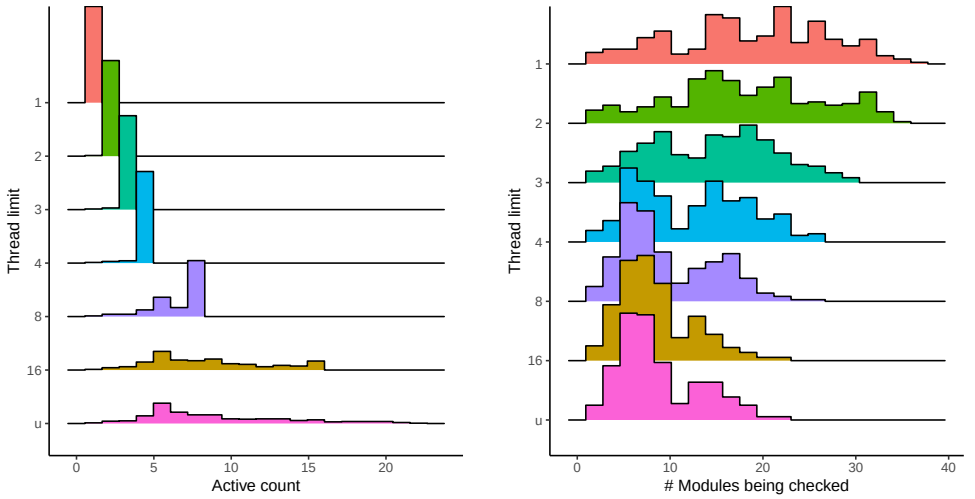
The system was underutilised in 41.9% of the 8 thread limit and in 93% of the 16 thread limit measurement points. The increase in the thread count limit comes with a smoother spread for the number of active threads (Figure 8(a)).



(a) 8 thread limit

(b) Unlimited thread count

Figure 7: The number of active threads at the time of measurement, when there is an 8 thread limit (7(a)) and when the number of threads is not limited (7(b)) while checking LTE IRAT.

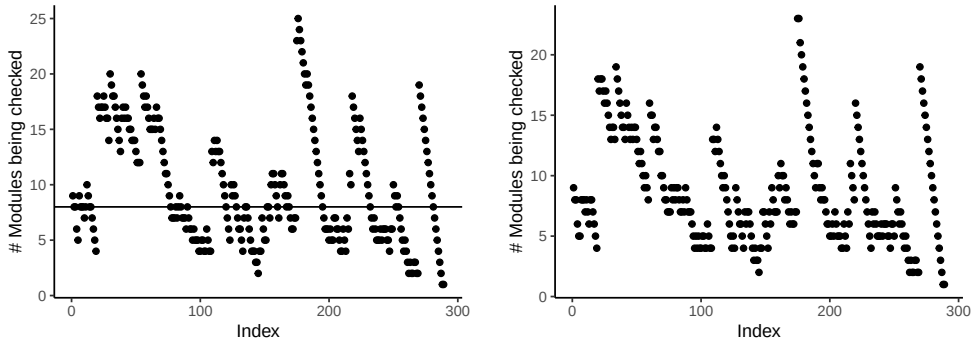


(a) Active thread count

(b) Available for checking

Figure 8: The histograms for the active thread count (8(a)) and the number of modules available for checking (8(b)) numbers measured, for each thread limit, while checking LTE IRAT.

We also measured the number of modules being checked or available for checking (Figure 9). The maximum values are 36 for 1, 35 for 2, 29 for 3, 26 for 4, 25 for 8 and 23 for 16 thread limit and 23 when there is no thread limit.



(a) 8 thread limit

(b) Unlimited thread count

Figure 9: The number of modules available for checking at the time of measurement, when there is an 8 thread limit (9(a)) and when the number of threads is not limited (9(b)) while checking LTE IRAT.

A higher thread count limit means more modules processable in parallel and a shorter spread of modules available for processing at any time (Figure 8(b)).

6.2 Impact on existing features

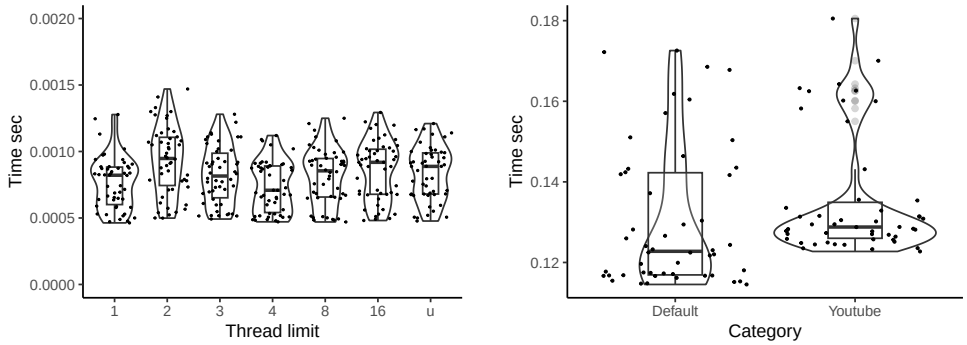
6.2.1 Impact on Titan's existing tests

The Eclipse Plug-ins of Titan we extended have tests for approx. 20.000 syntactic and semantic markers (warnings and errors together). We frequently executed these tests to ensure that existing detections were not changed. *The tests found the exact same issues at the expected locations, texts and severities.*

6.2.2 Impact on incremental parsing

We repeated the measurement described in [62] to prove that the incremental syntax checking is not affected negatively, inserting 203 spaces at the end of a line in a file of the Ideal project and using the last 50 measurements.

Every measured execution time of the syntax analysis (figure 10(a)) was below $1.47 * 10^{-3}$ seconds, going as low as approx. $5 * 10^{-4}$ seconds. *These values, for all thread counts, are similar to the published values indicating that the performance of incremental syntax checking has not regressed.*



(a) Incremental parsing comparisons

(b) YouTube test comparisons

Figure 10: Comparisons of incremental syntax checking duration, for different thread limits (10(a)) and the overall duration in Default mode, unlimited threads, with and without YouTube running in the background (10(b)).

6.2.3 YouTube test

IDEs have an upper limit on the hardware capacity they are allowed to use. For a good user experience, IDEs must not overload the developer's machine to the point where background music playback is negatively affected.

To prove that the new version still respects this upper boundary, we have re-run the measurements on the Ideal project, without thread limits, without blocking external interfaces (WiFi) and playing a video[19] in 1080p60HD quality via the Internet in the background.

During the measurement memory usage was approx. 95%, GPU utilisation approx. 30%. During the analysis phases the CPU was utilised at 100%[20], increasing speed to 3.50 - 3.88 GHz (falling back to approx. 2.15 GHz in-between these phases).

Playing a video in the background created (Figure 10(b)) a difference of 0.0068s in the means. Subjectively, we have not noticed any difference in the music during measurements.

While the analysis shows a statistically meaningful difference, in practical terms, we did not see a significant, real-world performance difference.

[19] https://www.youtube.com/watch?v=A-aSaw7JfB8, last accessed: 2023.05.15

[20] The built-in "Task Manager" application

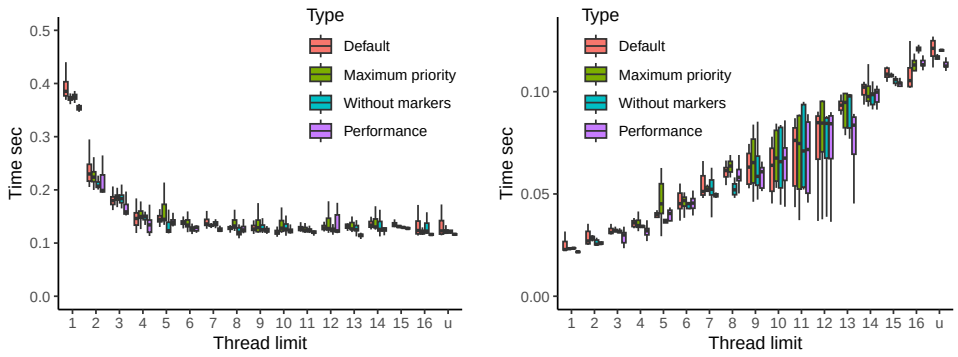
6.3 Additional research directions covered

In this section, we present additional research directions we have covered.

"Maximum priority": We maximised the priority of our application at the cost of other applications. The analysis threads executed at maximum priority (10) "Thread.yield()" was not called after checking each Assignment/Definition.

"Without markers": We removed the crucial functionality of reporting errors to eliminate a potential lock contention, where the Eclipse platform stores markers in an internal database. Here we have commented out the body of the "Location.reportProblem" and all functions in the "ParserMarkerSupport" class with "createOnTheFly" in their name.

"Performance": We overclocked the hardware at the cost of higher noise and power consumption. We have set the laptop in "Performance Mode"[21].



(a) Overall duration

(b) Single module duration

Figure 11: The overall duration (11(a)) and the single module duration (11(b)) for each experiment, for each thread limit (u meaning unlimited).

Our analysis shows (Table 4):

1. "Maximum priority" is mostly faster with a small margin.
2. "Without markers" is clearly faster, but would lose core functionality.
3. "Performance" is the fastest, but to handle the increased heat, the laptop became too loud for office usage.

[21] Description from manufacturer: "Boost your computer performance with higher fan speed and power consumption".

Threads	Max. Priority	Without Markers	Performance
1	0.01577	0.01373	0.03663
2	0.00686	0.02121	0.02332
3	-0.003821	-0.00304	0.018593
4	-0.00654	-0.002403	0.01335
5	-0.00817	0.02175	0.00702
6	0.00300	0.011101	0.009106
7	0.00237	0.000804	0.01143
8	-0.00286	0.00909	0.00364
9	-0.00121	0.00206	0.00413
10	-0.00776	-0.00702	-0.002307
11	0.00317	0.00312	0.00614
12	-0.000437	0.004698	0.005903
13	0.00452	0.005544	0.01673
14	0.002477	0.00746	0.006955
15	0.00262	0.004450	0.006402
16	0.00361	0.00149	0.00664
u	0.000897	0.000424	0.00646

(a) muDiff

Threads	Max.Priority	Without Markers	Performance
1	100	100	100
2	93.8	100	100
3	7.67	15.3	100
4	4.09	23.2	100
5	3.667	100	100
6	95.93	100	100
7	99.7	78.3	100
8	1.151	100	99.94
9	31.19	96.0	100
10	0	0.002	0.188
11	99.9	100	100
12	46.08	100	100
13	100	100	100
14	98.1	100	100
15	99.9	100	100
16	98.5	76.5	100
u	67.0	57.5	100

(b) $>compval

Table 4: Results of comparing the default measurements to a direction, showing the difference in the means (4(a)) and the percentage of the posterior probability mass above the comparison value 0 (4(b)).

While the analysis showed statistically meaningful improvements, no experiment offered practical real-world performance improvements (Figure 11). We consider these research directions closed.

7 Threats to validity

This study might suffer from the usual threats to external validity. There might be limits to generalising our results beyond our settings (the programming language used and possible industry-specific effects). We can only claim the validity of our results for programming languages and code bases which demonstrate the properties discussed in Section 2. Further research could investigate these properties for other languages and validate if our claim also holds for them.

One specific threat to generalization might come from our measurement performed only on a single laptop. To address this threat, we point out that this laptop was already outdated during the measurements and never meant for professional development work. In the paper, we showed how this laptop is already fast enough for daily work and how the structure of the problem would not benefit from more parallel resources, demonstrating how companies could already save on hardware costs.

8 Summary

IDE performance was a pain point for developers for a long time [22]. In this paper, we present our work on improving this situation by parallelizing the semantic checking phase of an industrial IDE.

We have presented earlier works on the structure of software systems, which serve as general requirements and prove the general applicability of our chosen method. We also presented works on the evolution tendencies of software systems to show that our chosen method permanently solves the problem.

We have shown that the new version improves performance on real-life projects, utilising contemporary hardware better without performance regression in other parts of the system. We showed that the structure of the problem limits better utilising all parallel hardware resources. We can not expect additional benefits even from infinitely scaling Cloud resources.

Our measurements showed that even outdated laptop hardware, not aimed at professional development work, is now strong enough to support working on large open-source test systems. From the perspective of performance only, our results make it hard to justify investing in Cloud resources or remote servers to provide developers with a performing IDE. Companies should optimise their development costs and sustainability efforts [52] by utilising weaker/cheaper machines that still offer enough performance.

Our results could also inspire simplification in future IDEs, making pre-built indices obsolete and user interfaces simpler.

The functionality we developed is available in open source [81] as part of the `Titan` toolset.

9 Further work

Further research in improving performance could target: finer-grain locking, keeping the work on "warm cores" [32], running our Java code directly on bare metal [56, 57], and further optimisation of a Java Just-In-Time compiler [60].

Other research could target determining the most cost-efficient hardware configuration for a given project, recommend code management and coding styles that reduce processing times [47].

[22] https://blog.jetbrains.com/kotlin/2021/06/kotlin-ide-performance/#Reworkedplatform,plugin,andcompilerAPIforcodehighlighting

10 Acknowledgements

The authors would like to thank the DUCN Software Technology unit of Ericsson AB, Sweden for the financial support of this research and the Test Competence Center of Ericsson Hungary for providing access to their in-house tools. We would also like to thank Izabella Ingrid Farkas for her help invaluable to our measurements, and Peter Verhas for his feedback on this article.

References

[1] N. Bartha, Scalability on it projects, *Master's thesis*, Eötvös Loránd University, 2016. ⇒240, 242

[2] K. Beck, Why does Kent Beck refer to the "rediscovery" of test-driven development? what's the history of test-driven development before Kent Beck's rediscovery?, https://www.quora.com/Why-does-Kent-Beck-refer-to-the-rediscovery-of-test-driven-development-Whats-the-history-of-test-driven-development-before-Kent-Becks-rediscovery, Last acc.: 2023.05.15. ⇒242

[3] K. Beck, *Test Driven Development. By Example (Addison-Wesley Signature)*, Addison-Wesley Longman, Amsterdam, 2002. ⇒241

[4] R. Bohn, Stop fighting fires, *HBR*, 78:83–91, 07 2000. ⇒241

[5] M. Choudaray and M. Cheng, *Export Control. In Open Source Law, Policy and Practice*, Oxford University Press, 10 2022. ⇒241

[6] L. J. Colfer and C. Y. Baldwin, The mirroring hypothesis: Theory, evidence and exceptions, *IRPN: Innovation & Organizational Behavior (Topic)*, 2016. ⇒241

[7] M. E. Conway, How do committees invent, *Datamation*, 1967. ⇒241

[8] W. E. Deming, Out of the Crisis, *volume 1 of MIT Press Books*. The MIT Press, 12.2000. ⇒241

[9] J. Dietrich, C. McCartin, E. Tempero, and S. M. A. Shah, Barriers to modularity - an empirical study to assess the potential for modularisation of Java programs. *In Research into Practice – Reality and Gaps*, pages 135–150, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ⇒243

[10] M. Dorner, M. Capraro, O. Treidler, T.-E. Kunz, D. Šmite, E. Zabardast, D. Mendez, and K. Wnuk, Taxing collaborative software engineering, 2023. ⇒241

[11] I. I. Farkas, K. Szabados, and A. Kovács, Improving productivity in large scale testing at the compiler level by changing the intermediate language from C++ to Java, *Acta Univ. Sapientiae Informatica*, **13,** 1 (2021) 134–179. ⇒240

[12] I. I. Farkas, K. Szabados, and A. Kovács, *Regression test data*, http://compalg.inf.elte.hu/ attila/materials/RegressionTestSmall_20190724.zip, 2019. ⇒248

[13] J. Fernandez-Ramil, D. Izquierdo-Cortazar, and T. Mens, What does it take to develop a million lines of open source code?, *In Open Source Ecosystems: Diverse Communities Interacting*, volume 299, pages 170–184, 06 2009. ⇒243, 244

[14] A. Georges, D. Buytaert, L. Eeckhout, Statistically Rigorous Java Performance Evaluation, *In Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications*, OOPSLA '07, pages 57–76, New York, NY, USA, 2007. ACM. ⇒249

[15] L. Hatton, Conservation of information: Software's hidden clockwork?, *IEEE Trans. Softw. Eng.*, **40,** 5 (2014) 450–460. ⇒243

[16] L. Hatton and G. Warr, Strong evidence of an information-theoretical conservation principle linking all discrete systems,*R. Soc. O. Sci.*, **6,** 10 (2019) 191101 ⇒243

[17] R. Hayes, Why Japanese factories work, *HBR*, **59,** 1 (1981) 56–66. ⇒241

[18] I. Herraiz, D. Germán, and A. E. Hassan, On the distribution of source code file sizes, *In International Conference on Software and Data Technologies*, v. 2, p. 5–14, 01 2011. ⇒243

[19] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles, Towards a theoretical model for software growth, *In Fourth International Workshop on Mining Software Repositories* (MSR'07:ICSE Workshops 2007), p. 21–21, 2007. ⇒242, 243

[20] D. Hyland-Wood, D. Carrington, and S. Kaplan, Scale-free nature of java software package, class and method collaboration graphs, *In Proceedings of the 5th International Symposium on Empirical Software Engineering*, 2006. ⇒242

[21] P. M. Institute, *A guide to the Project Management Body of Knowledge* (PMBOK guide), PMI, Newton Square, PA, 6th edition, 2017. ⇒240, 241

[22] P. M. Institute, *A guide to the Project Management Body of Knowledge* (PMBOK guide), PMI, Newton Square, PA, 7th edition, 2021. ⇒241

[23] A. Israeli and D. Feitelson, The Linux kernel as a case study in software evolution, *J. Syst. Softw.*, 83:485–501, 03 2010. ⇒243, 244

[24] C. Izurieta and J. Bieman, The evolution of FreeBSD and Linux, *In Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, ISESE '06, p. 204–211, NY, USA, 2006. ACM. ⇒243, 244

[25] K. Johari and A. Kaur, Effect of Software Evolution on Software Metrics: An Open Source Case Study, *SIGSOFT Softw. Eng. N.*, 36(5):1–8, 09.2011. ⇒243, 244

[26] T. Joosse, December 1945: The ENIAC Computer Runs its First, Top-Secret Program, https://www.aps.org/publications/apsnews/202212/history.cfm, 2022. Last accessed: 2023.05.15. ⇒242

[27] C. Kemerer and S. Slaughter, An empirical approach to studying software evolution, *IEEE Trans. Softw. Eng.*, 25(4):493–509, 1999. ⇒243, 244

[28] G. Kohring, Complex dependencies in large software systems, *Advances in Complex Systems*, 12, 11 2011. ⇒242

[29] A. Kovács and K. Szabados, Advanced TTCN-3 Test Suite validation with Titan, *In 9th International Conference on Applied Informatics*, p. 273–281, 2015. ⇒241

[30] A. Kovács and K. Szabados, Internal quality evolution of a large test system–an industrial study, *Acta Univ. Sapientiae*, 8(2):216–240, 2016. ⇒243, 244

[31] N. LaBelle and E. Wallingford, Inter-package dependency networks in open-source software, *CoRR*, cs.SE/0411096, 2004. ⇒242

[32] J. Lawall, H. Chhaya-Shailesh, J.-P. Lozi, B. Lepers, W. Zwaenepoel, and G. Muller, Os scheduling with nest: Keeping tasks close together on warm cores, *In Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22, page 368–383, New York, NY, USA, 2022. ACM. ⇒260

[33] M. J. Lawrence, An examination of evolution dynamics, *In Proceedings of the 6th International Conference on Software Engineering*, ICSE '82, page 188–196, Washington, DC, USA, 1982. IEEE CS Press. ⇒243

[34] M. Lehman and J. Fernandez-Ramil, Rules and tools for software evolution planning and management, *ASE*, 11:15–44, 01 2001. ⇒243

[35] M. Lehman, D. Perry, and J. Ramil, On evidence supporting the feast hypothesis and the laws of software evolution, *In Proc Fifth Int. Software Metrics Symposium. Metrics (Cat. No.98TB100262)*, pages 84–88, 1998. ⇒243, 244

[36] M. Lehman and J. Ramil, Towards a theory of software evolution - and its practical impact, *In Proc. Int. Symposium on Principles of Software Evolution*, pages 2–11, 2000. ⇒243

[37] M. M. Lehman and J. F. Ramil, Evolution in software and related areas, *In Proceedings of the 4th International Workshop on Principles of Software Evolution*, IWPSE '01, page 1–16, New York, NY, USA, 2001. ACM. ⇒243

[38] E. Leo, Breaking mirror for the customers: The demand-side contingencies of the mirroring hypothesis, *Cont. Man. Res.*, 18:35–65, Mar. 2022. ⇒241

[39] A. MacCormack, C. Baldwin, and J. Rusnak, Exploring the duality between product and organizational architectures: A test of the "mirroring" hypothesis, *Research Policy*, 41(8):1309–1324, 2012. ⇒241

[40] M. A. Mamun, C. Berger, and J. Hansson, Effects of measurements on correlations of software code metrics, *Empir. Softw. Eng.*, 24, 08 2019. ⇒242

[41] J. McCalpin, Memory bandwidth and machine balance in high performance computers, *IEEE Technical Committee on Computer Architecture Newsletter*, pages 19–25, 12 1995. ⇒247

[42] H. Melton and E. Tempero, An empirical study of cycles among classes in Java, *Empir. Softw. Eng.*, 12(4):389–415, Aug. 2007. ⇒243

[43] S. Moradi, K. Kähkönen, and K. Aaltonen, From past to present- the development of project success research, *J. Mod. Proj.*, 8(1), Apr. 2022. ⇒241

[44] A. Moura, Y. Lai, and A. Motter, Signatures of small-world and scale-free properties in large computer programs, *Phys. Rev. E*, 68(2), 2003. ⇒242

[45] C. R. Myers, Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs, *Phys. Rev. E*, 68(4), 2003. ⇒242

[46] N. Nagappan, B. Murphy, V. Basili, and N. Nagappan, The influence of organizational structure on software quality: An empirical case study, *Technical Report MSR-TR-2008-11*, Microsoft Research, January 2008. ⇒241

[47] G. Nagy and Z. Porkoláb, Performance issues with implicit resolution in scala, *In Proceedings of the 10th International Conference on Applied Informatics*, pages 211–223, 01 2018. ⇒260

[48] P. Olah, Szemantikus elemzés gyorsítása TTCN-3 környezetben, *Master's thesis*, Eötvös Loránd University, 2016. ⇒244, 245

[49] T. D. Oyetoyan, R. Conradi, and D. S. Cruzes, Criticality of defects in cyclic dependent components, *In 2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation* (SCAM), pages 21–30, 2013. ⇒243

[50] A. Pannier, *Software power: The economic and geopolitical implications of open source software*, 2022. ⇒241

[51] D. L. Parnas, Structured programming: A minor part of software engineering, *Information Processing Letters*, 88(1):53–58, 2003. ⇒242

[52] Z. Porkoláb, Save the Earth, Program in C++!, *In 2022 IEEE 16th Int. Scientific Conf. on Informatics* (Informatics), p. 11–12. IEEE, 2022. ⇒260

[53] A. Potanin, J. Noble, M. Frean, and R. Biddle, Scale-free geometry in OO programs, *Commun. ACM*, 48(5):99–103, May 2005. ⇒242

[54] R. Potvin and J. Levenberg, Why Google stores billions of lines of code in a single repository, *Commun. ACM*, 59(7):78–87, Jun 2016. ⇒243

[55] S. Pretorius, H. Steyn, and T. Bond-Barnard, The relationship between project management maturity and project success,*J. Mod. Proj.*,10:219–231, 2023. ⇒241

[56] W. Puffitsch and M. Schoeberl, PicoJava-II in an FPGA, *In Proceedings of the 5th International Workshop on Java Technologies for Real-Time and Embedded Systems*, JTRES '07, page 213–221, New York, NY, USA, 2007. ACM. ⇒260

[57] D. Simon, C. Cifuentes, D. Cleal, J. Daniels, and D. White, Java™ on the bare metal of wireless sensor devices: the squawk Java virtual machine, *In Proceedings of the 2nd international conference on Virtual execution environments*, pp. 78–88, 2006. ⇒260

[58] C. P. Smith, A software science analysis of programming size, *In Proceedings of the ACM 1980 Annual Conference*, page 179–185, 1980. ACM. ⇒243

[59] S. Spear and H. Bowen, Decoding the DNA of the Toyota Production System, *HBR*, 77, 09 1999. ⇒241

[60] T. Suganuma, T. Yasue, M. Kawahito, H. Komatsu, and T. Nakatani, Design and Evaluation of Dynamic Optimizations for a Java Just-in-Time Compiler, *ACM Trans. Program. Lang. Syst.*, 27(4):732–785, Jul 2005. ⇒260

[61] K. Szabados, Structural Analysis of Large TTCN-3 Projects, *In Proceedings of the 21st IFIP WG 6.1 International Conference on Testing of Software and Communication Systems and 9th International FATES Workshop*, TESTCOM '09/FATES '09, page 241–246, Berlin, Heidelberg, 2009. Springer-Verlag. ⇒242

[62] K. Szabados, Creating an efficient and incremental IDE for TTCN-3, *In 10th Joint Conference on Mathematics and Computer Science, Cluj-Napoca*, In Studia Universitatis Babes-Bolyai, Informatica, volume 60, pages 5–18, 2015. ⇒244, 256

[63] K. Szabados, Quality Aspects of TTCN-3 Based Test Systems, *PhD thesis*, Eötvös Loránd University, 11 2017. ⇒240, 242

[64] K. Szabados, A. Kovács, G. Jenei, and D. Góbor, Titanium: Visualization of TTCN-3 system architecture, *In 2016 IEEE International Conference on Automation, Quality and Testing, Robotics* (AQTR), pages 1–5, 2016. ⇒243

[65] C. Taube-Schock, R. J. Walker, and I. H. Witten, Can we avoid high coupling?, *In Proceedings of the 25th European Conference on Object-Oriented Programming*, ECOOP'11, page 204–228, Berlin, Heidelberg, 2011. Springer-Verlag. ⇒242

[66] R. Tonelli, G. A. Pierro, M. Ortu, and G. Destefanis, Smart contracts software metrics: A first study, *PLoS ONE*, 18, 01 2023. ⇒243

[67] W. Turski, The reference model for smooth growth of software systems revisited, *IEEE Trans. Softw. Eng.*, 28(8):814–815, 2002. ⇒243

[68] J. Varajão, R. P. Marques, and A. Trigo, Project management processes – impact on the success of information systems projects, *Inf.*, 33(2):421–436, 2022. ⇒241

[69] L. Šubelj and M. Bajec, Software systems through complex networks science: Review, analysis and applications, *In Proceedings of the First International Workshop on Software Mining*, p. 9–16, NY, USA, 2012. ACM. ⇒242

[70] A. Whiteley, J. Pollack, and P. Matous, The origins of agile and iterative methods, *J. Mod. Proj.*, pages 20–29, 02 2021. ⇒241

[71] E. Yourdon and L. L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Yourdon, 1978. ⇒241

[72] H. Zhang, Exploring Regularity in Source Code: Software Science and Zipf's Law, *In 15th Working Conference on Reverse Engineering*, 101–110, 2008. ⇒243

[73] H. Zhang and H. B. K. Tan, An Empirical Study of Class Sizes for Large Java Systems, *In 14th Asia-Pacific Software Engineering Conference* (APSEC'07), pages 230–237, 2007. ⇒243

[74] H. Zhang, H. B. K. Tan, and M. Marchesi, The Distribution of Program Sizes and Its Implications: An Eclipse Case Study, *CoRR*, abs/0905.2288, 2009. ⇒243

[75] T. Zimmermann and N. Nagappan, Predicting Subsystem Failures using Dependency Graph Complexities, *In The 18th IEEE Int. Symp. on Soft. Rel.* (ISSRE '07), pages 227–236, 2007. ⇒243

[76] A. Zsiga, Termelékenységi trendek, minták elemzése szoftverfejlesztési projektekben, *Master's thesis*, Eötvös Loránd University, 2019. ⇒243, 244

[77] ∗ ∗ ∗, Evolved universal terrestrial radio access (e-utra) and evolved packet core (epc); user equipment (ue) conformance specification; part 3: Abstract test suite (ats), ftp://ftp.3gpp.org/Specs/archive/36_series/36.523-3/36523-3-g90.zip. Last accessed: 2023.05.15. ⇒251

[78] ∗ ∗ ∗, Internet protocol (ip) multimedia call control protocol based on session initiation protocol (sip) and session description protocol (sdp); user equipment (ue) conformance specification; part 3: Abstract test suites (ats), ftp://ftp.3gpp.org/Specs/archive/34_series/34.229-3/34229-3-g20.zip. Last accessed: 2023.05.15. ⇒251

[79] ∗ ∗ ∗, Technical specification group radio access network; 5gs; user equipment (ue) conformance specification; part 3: Protocol test suites, fftp://ftp.3gpp.org/Specs/archive/38_series/38.523-3/38523-3-g20.zip. Last accessed: 2023.05.15. ⇒251

[80] ∗∗∗, User equipment (ue) conformance specification; part 3: Abstract test suites, ftp://ftp.3gpp.org/Specs/archive/34_series/34.123-3/34123-3-g20.zip. Last accessed: 2023.05.15. ⇒250

[81] ∗∗∗, Titan, https://projects.eclipse.org/projects/tools.titan, 2020. Last accessed: January 2020. ⇒244, 260

Received: September 19, 2023 • Revised: October 25, 2023