UDC 004.49 DOI DOI https://doi.org/10.35546/kntu2078-4481.2024.2.32

I. O. SUPRUNENKO
Postgraduate Student at the Department of Cybersecurity
at the Faculty of Information Technologies and Systems
Cherkasy State Technological University
ORCID: 0000-0002-1188-4804

V. M. RUDNYTSKYI
Doctor of Engineering Science, Professor, Chief Researcher
State Scientific Research Institute of Armament
and Military Equipment Testing and Certification,
Professor at the Department of Cybersecurity
at the Faculty of Information Technologies and Systems
Cherkasy State Technological University
ORCID: 0000-0003-3473-7433

COMPARISON OF MESSAGE PASSING SYSTEMS IN CONTEXT OF ADAPTIVE LOGGING METHOD

Computer software is an important part of technological progress. As it becomes more and more complex and sophisticated, so does the need to protect it. Apart from typical information security aspects of integrity, availability and confidentiality, the scale and complexity of modern computer systems require a high level of control and observability.

The main goal of this research is to build upon the foundations laid by the general idea of an adaptive logging method and introduce the next iteration of its design in the form of an appropriate message passing system to be used to propagate required changes to corresponding implementation in an effective and performant manner.

Four different message passing system models are introduced, based on different technologies such as RabbitMQ message broker, communication channels in PostgreSQL database management system, general web server architecture and Linux-based process signaling interface. For each of those an overview description and graphical model is presented.

Finally, the resulting comparison is conducted, comparing aspects such as reliance on third-party software, communication medium, error surface increase and authentication related considerations. As a result, the design based on process signaling approach is determined to be the most suitable for adaptive logging method, as it does not introduce any third-party software (and as such affects error surface in a somewhat negligible manner), binds directly to an observed application, is built using low level concepts that should be present in multiple different platforms and programming languages and should be able to reuse authentication logic that is already used when accessing computational machine where observed program is executed.

Key words: *information security, debugging, control, observability, adaptive logging method, message passing systems, message broker, notification channel, Linux signals.*

І. О. СУПРУНЕНКО
аспірант кафедри кібербезпеки
факультету інформаційних технологій і систем
Черкаський державний технологічний університет
ORCID: 0000-0002-1188-4804

В. М. РУДНИЦЬКИЙ
доктор технічних наук, професор, головний науковий співробітник
Державний науково-дослідний інститут випробувань і сертифікації озброєння
та військової техніки,
професор кафедри кібербезпеки
факультету інформаційних технологій та систем
Черкаський державний технологічний університет
ORCID: 0000-0003-3473-7433

ПОРІВНЯННЯ СИСТЕМ ПЕРЕДАЧІ ПОВІДОМЛЕНЬ В КОНТЕКСТІ МЕТОДУ АДАПТИВНОГО ЛОГУВАННЯ

Комп'ютерні технології складають важливу частину технологічного прогресу. З підвищенням їх складності, стає складніше забезпечувати належний рівень їх безпеки. Окрім типових аспектів інформаційної безпеки, а саме цілісності, доступності та конфіденційності, масштаб та складність сучасних комп'ютерних систем потребує високого рівня контролю та спостережності.

Мета цього дослідження полягає в тому, щоб використати основи закладені в загальну ідею методу адаптивного логування та представити наступну ітерацію його розвитку у вигляді відповідної системи обміну повідомленнями для передачі необхідних змін відповідній імплементації ефективно та з достатнім рівнем швидкодії.

В роботі представлено чотири різні моделі систем обміну повідомленнями, що базуються на різних технологіях: меседж брокер RabbitMQ, комунікаційні канали системи управління базою даних PostgreSQL, рішення на архітектурі типу веб-сервер та Linux-орієнтованого інтерфейсу передачі міжпроцесних сигналів. Для кожного з них представлено узагальнений опис та графічну модель.

Для фінального порівняння цих рішень використані такі аспекти, як: залежність від стороннього програмного забезпечення, механізм комунікації, збільшення поверхні для помилки, а також питання аутентифікації. В результаті найбільш відповідною до вимог методу адаптивного логування визначено модель, що базується на міжпроцесній взаємодії, оскільки вона не містить залежності від сторонніх бібліотек (а тому збільшенням поверхні для помилки можна знехтувати), напряму пов'язується із програмою, за якою ведеться спостереження, побудована з використанням низькорівневих механізмів що мають бути присутні в багатьох платформах та мовах програмування, а також має можливість використовувати вже існуючий механізм аутентифікації, що використовується для доступу до обчислювальної машини, де встановлена досліджувана програма.

Ключові слова: *інформаційна безпека, debugging, контроль, спостережність, метод адаптивного логування, системи обміну повідомленнями, канал для нотифікацій, Linux сигнали.*

Formulation of the problem

Information technologies play a huge part in everyday lives: from being able to read news online to managing multi server international financial transactions. And it is also growing constantly: according to "Digital 2023: Global Overview Report" [1] the number of people using mobile phones at the beginning of 2023 was estimated to be 5,44 billion (which is around 68% of total population) and the number of unique mobile users has increased by 168 million new users compared to previous year. Same can be stated about the Internet as one of the most widely used technologies in the world: the number of total Internet users in October 2022 was 5,07 billion, but as of January 2023 it was already around 5,16 billion.

But not only mobile and Internet technologies are affected, different other aspects of human lives become increasingly computerized. And in order to deal with such high demand, computer systems become more and more complex, grow to a scale never seen before and as a result – face new issues, threats and dangers. Those might be quite different, from algorithmic or communicational complexity to malicious actors seeking ways to compromise and harm other people. As such it is extremely important to have software operating with appropriate levels of information security, protecting users and their data. During COVID-19 pandemic it became evident that new challenges in the physical world require corresponding changes in technology and a prime example is the usage of virtual private network (VPN) technologies during initial outbreaks. As lots of people were forced to move to remote workspaces and work from home, it became critical to protect their communications amidst the growing number of cybercriminals trying to leverage pandemic and deal some serious damage. VPNs were used as a solution that would protect confidentiality of the exchange and the demand was so high, that some enterprises expanded from having 8000 daily users to 80000 relying on VPN every day [2].

But among all information security aspects, not only integrity, availability and confidentiality require taking appropriate protective measures and spending a sufficient amount of resources improving those, the same is also true for the aspect of control over the information system. It is typically expressed in a form of observability and it is equally important to know why things happen the way they do (in such huge complex systems like online banking applications), but also how can one tell whether system behaves as expected and if not – what is the fastest way to resolve the issue and restore required functionality. This research is focused on some aspects of observability in digital systems, with practical solutions described as a part of a more global software solution aimed to solve several control related drawbacks.

Analysis of recent research and publications

There exist different approaches aimed at bringing more control into software systems. One of those is called "software testing" and can be formally described as "the process of evaluating and verifying that a software product or application does what it's supposed to do" [3]. Using this approach makes software products more predictable and reliable, aiding in tracking bugs and issues, but as it deals mainly with predefined setups and conditions, requires appropriate degree of design and planning. Even though the process of writing tests might be something that developers don't exactly enjoy, Guilherme and Vincenzi [4] showed that it is possible to utilize software, such as ChatGPT, delegating the need of writing some simple testing cases to it. It should be noted that introducing new code, that in theory should make other code better, has some drawbacks. The study conducted by Peruma and Newman [5] showed that test files themselves might contain bugs, which in result affects software quality. Their result data showed that developers usually introduce fixes to functionality related files and testing related ones in separate commits, which is certainly a potential area of errors during development. Some of the test related issues can be solved using specific software, for example, "tsDetect" tool introduced by Peruma et al. [6], which detects possible issues in Java based test files with stated precision of 97%. Still not every issue can be accounted for using some automated solution. And as Ardic and Zaidman [7] conclude in their analysis of educational efforts related to testing, even though about a half of all analyzed curricula had a dedicated course about it,

there is still need for more knowledge in areas such as creating acceptance, security and performance oriented tests, which would require more expertise from developers.

In order to diagnose performance and runtime of a software system, a technique called monitoring is often used. Monitoring can be described as the task of assessing the health of a system and is accomplished by collecting predefined metrics during runtime execution of a software program, analyzing those afterwards [8]. It focuses on very general characteristics of a computer, such as central processing unit (CPU) utilization, memory usage, amount and throughput of disk operations, network traffic, etc. As a result, a wide variety of different software solutions can be monitored. Research conducted by Wang et al. [9] showed that it is possible to utilize natural language processing in order to run diagnostics and predict possible defects by extracting key data from the monitoring results. This can aid greatly in reducing human error and making routine tasks less irritating. Even areas as complex as Internet-of-Things are known to make use of monitoring and, as shown by Ma et al. [10], and complex software monitoring and early warning systems can be designed to keep track of system's performance.

Presenting main material

Similar solution to monitoring using predefined metrics is software logging which can be seen as a process of describing program execution using small chunks of textual data called "logs". Generally logging is implemented using "severity only" approach, where each chunk of data is coupled with a small string literal value describing so-called "severity" of a log invocation, which defines how critical this log message is (is it an error and requires immediate attention, or is it just some informational data and does not require urgent processing) [11]. An improvement of this approach is presented in [12] and is called "adaptive logging approach". The formalization of it can be described using three main components. First part is an adaptive logging function (1) and its main distinctive feature is that it adds a third argument, which serves as a description-like primitive helping to identify particular log invocation:

$$f_{log\ adp} = f(S_{ev}, M, T_{incl}) \tag{1}$$

where *Sev* – severity of current log invocation, M – message, T_{incl} – set of tags that describe current invocation.

The second component is method (2), that allows to both initialize and reinitialize current configuration for an adaptive logging method:

$$f_{init} = f(S_{ev}, C) \tag{2}$$

where S_{ev} – the lowest severity level runtime should report, C – special configuration object that maps tags to their corresponding action, include or exclude.

Finally, last component is a configuration object C (3) and it is basically what allows to compare and decide whether particular log invocation matches current logging setup:

$$C = T_{11}^{mod}\ \&\&\ T_{12}^{mod}\ \&\&\ \ldots\ ||\ T_{21}^{mod}\ \&\&\ T_{22}^{mod}\ \&\&\ \ldots\ ||\ \ldots \tag{3}$$

where T_ij – particular tag, mod (modifier) – "include" or "exclude", && – equivalent of logical "AND" operator, || – equivalent of logical "OR" operator.

In order for this approach to become adaptive and flexible enough to match runtime requirements, that might change "on the fly", any software solution utilizing adaptive logging method requires some sort of message passing mechanism. Several models describing typical setups with their corresponding message passing technology are described in this work and then comparison of those is conducted, outlining their advantages and drawbacks.

One typical solution to use when there is a need to exchange messages is a message broker, which is basically a piece of software that is responsible for receiving and delivering messages from producers to consumers. RabbitMQ [13] is an example of a message broker and it functions based on concepts such as producers, queues and consumers. Producer connects to a RabbitMQ instance and sends a message that is later forwarded to a consumer by the broker itself (Figure 1).

With this setup an external producer can initiate reconfiguration and all necessary parameters can come using payloads of the messages in the message broker's queue.

A bit simpler (but still dependent on separate services) solution can be to utilize notification channel functionality of PostgreSQL object-relational database system [14]. It is similar to queues in RabbitMQ in that it also has concepts of producers and consumers, but it's somewhat simpler. Basically, a database server acts as a centralized communication point, where all interested parties can subscribe to messages in a particular channel (using LISTEN clause) and anyone wanting to pass some information can use NOTIFY clause (with optional payload if needed) in order to pass it to all live subscribers. Compared to all the features that a typical message broker might have, like acknowledgment of message processing, retry logic if the consumer is busy, timeouts, etc. – PostgreSQL gives less functionality, but it should suffice for the needs of adaptive logging configuration override. Schematic model is presented in Figure 2.

While this looks similar to the model with message broker, main differences are that it's more lightweight (as it lacks some functionality that RabbitMQ provides) and that it is a perfect fit for projects that already use this database system, as it does not require addition of new software to the codebase.

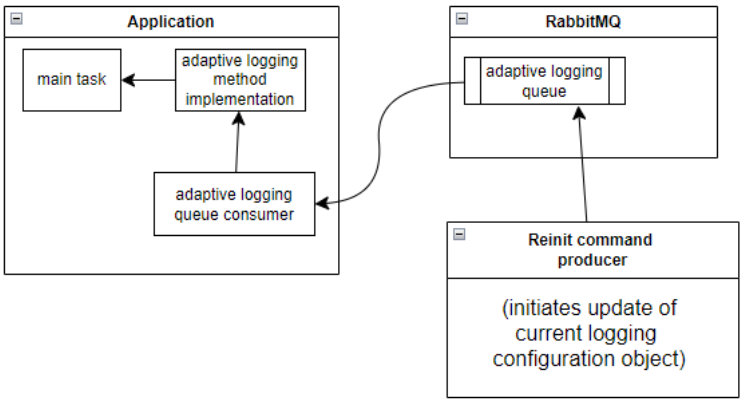

Fig. 1. Message passing model – RabbitMQ

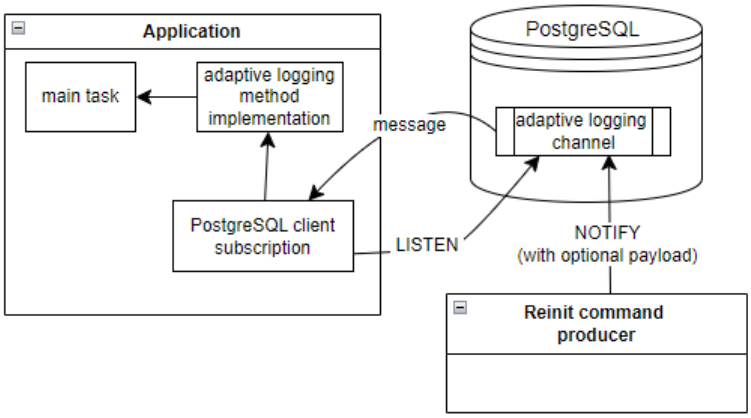

Fig. 2. Message passing model – PostgreSQL

Both models rely on some third party software to be implemented. This is fine if a project already relies on a required solution, but that would make the applicability of the adaptive logging method more limited than it should be. As the main requirement for propagation of reinitialization call is only to somehow trigger method (2) with new parameters, it is also possible to utilize a web server technology (those are pretty common in different programming languages and platforms) by subscribing to a particular web route in the same way as subscribing to queue or channel works. This design is much more flexible and portable, as it relies on a general concept rather than on a particular implementation. It is also more lightweight and as such, starting a small server is not as critical as with previous solutions. Figure 3 shows general scheme of this approach:

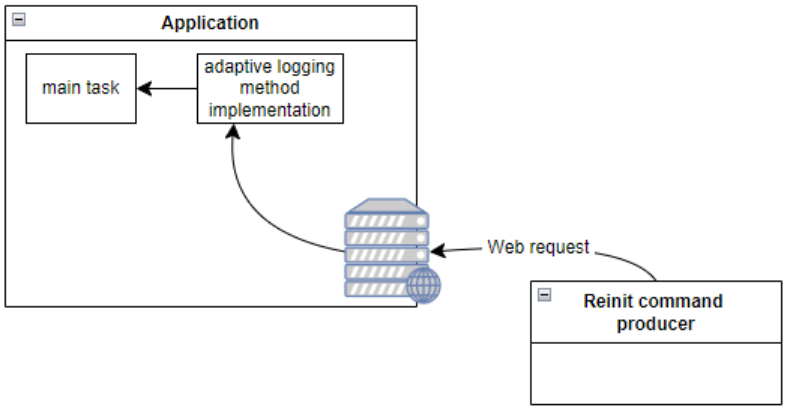

Fig. 3. Message passing model – Web server

Here the connection is more direct, going right to the application, which makes it more coupled than previous solutions, but that should not be a big problem as required communication is relatively limited and also client-server text-based information exchange is still pretty permissive.

For simplicity, the last model is described in terms of Linux-based systems with emphasis on process signaling mechanism [15]. In its very basic form, process signals function similarly to other message passing technologies presented previously, which means that it is possible to subscribe to some predefined signal and do some action after it occurs. Most of those signals are very specific and often have behavior connected to them by default (like with SIGTERM or SIGKILL). But there are also several mechanisms that can be used for user-defined signaling: real-time signals (defined by the macros SIGRTMIN and SIGRTMAX) and two user-defined signals (SIGUSR1 and SIGUSR2). Emitting these generally should not conflict with any default behaviors and gives a solution with lower overhead, compared to previous three, and decreased error surface as there is no third party dependency and no new components are introduced just to use a fraction of their capabilities. Figure 4 shows the corresponding model:

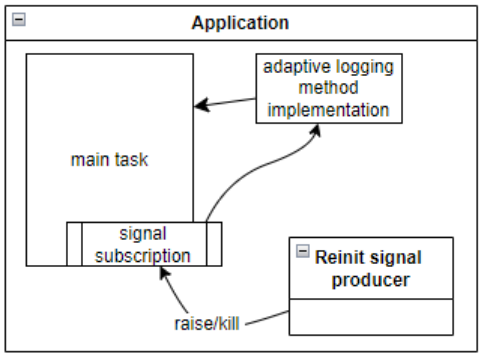

Fig. 4. Message passing model – process signals

Some important differences that should be emphasized right away are that with this setup reinitialization logic gets enclosed near the main task and there is no payload connected to a signal (but that can be solved differently).

With these four main models described, the comparison should focus on several important aspects, such as: reliance on third-party software, communication medium, authentication (this is a very important aspect as it is critical not to introduce any new vulnerabilities while adding capabilities aimed to help with debugging), error surface increase (whether proposed solution adds new potential “pain points” for developers to be aware of). Results of this comparison are presented in Table 1.

Table 1

	RabbitMQ	**PostgreSQL**	**Web-server**	**Process signals**
Reliance on third-party software	Yes	Yes	Maybe (some platforms and languages have built-in solutions)	No
Communication medium	Network	Network	Network	Local
Authentication	Provided by RabbitMQ	Provided by PostgreSQL	Has to be added manually	Shared with application
Error surface increase	Considerable (broker requires separate maintenance)	Considerable (RDBMS requires separate maintenance)	Minor (relatively small addition to existing codebase)	Negligible (works based on native mechanisms)

Conclusions

Software systems take a huge part in human lives and as they become more and more complex to satisfy corresponding needs, it is increasingly important to take care of aspects related to information security. As recent experience with COVID-19 shows, the demand for secure and protected products can increase very rapidly. And not only in terms of confidentiality, integrity and availability, but also with a high degree of control and observability. This work presented a further improvement of the “adaptive logging method” and demonstrated 4 possible solutions to the issue of effective and performant message passing process, based on message broker technology, capabilities of a PostgreSQL database management system, general web-server architecture and using process signaling approach in Linux-based systems.

The comparison of those 4 is then presented with emphasis on things such as reliance on third-party software, authentication considerations, required communication medium and increase of error surface. As a result, process signaling approach appears to be the most suited for the requirements of adaptive logging method: it does not introduce any new dependencies, functions based on local communication (completely eliminating network related issues), increase

in error surface is rather negligible as it uses native capability of an operating system and the authentication related concerns are handled using procedures that would already be in place and shared with the main application (such as establishing secure shell connection in order to configure application on a remote host). Given these advantages the number of applications that should be able to use adaptive logging should remain relatively high. One important limitation that should be mentioned is that process signaling does not give a developer a standard way to pass any payloads together with notification call, but as all of the communication is local by design this could be solved, for example, by reading from a local file.

As for further research, topics, such as developing a more complex logging behavior, that would allow to change log statement contents during program execution, and comparing the impact of general severity-based logging on program's execution flow and resource consumption with the one that is observed while using adaptive logging method, seem to have a lot of scientific potential.

Bibliography

1. Digital 2023: Global Overview Report – DataReportal – Global Digital Insights. URL: https://datareportal.com/reports/digital-2023-global-overview-report (дата звернення: 20.04.2024).

2. Abhijith M. S., Senthilvadivu K. IMPACT OF VPN TECHNOLOGY ON IT INDUSTRY DURING COVID-19 PANDEMIC. International Journal of Engineering Applied Sciences and Technology. 2020. Vol. 5, Issue 5, P. 152–157. https://doi.org/10.33564/ijeast.2020.v05i05.027.

3. What Is Software Testing? | IBM. URL: https://www.ibm.com/topics/software-testing (дата звернення: 20.04.2024).

4. Guilherme V., Vincenzi A. An initial investigation of ChatGPT unit test generation capability. In Proceedings of the 8th Brazilian Symposium on Systematic and Automated Software Testing. 2023. P. 15–24. https://doi.org/10.1145/3624032.3624035.

5. Peruma A., Newman C. D. On the Distribution of "Simple Stupid Bugs" in Unit Test Files: An Exploratory Study. IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), Madrid, Spain. 2021. P. 525–529. https://doi.org/10.1109/MSR52588.2021.00067.

6. Peruma A. et al. TsDetect: an open source test smells detection tool. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2020. P. 1650–1654. https://doi.org/10.1145/ 3368089.3417921.

7. Ardic B., Zaidman A. Hey Teachers, Teach Those Kids Some Software Testing. IEEE/ACM 5th International Workshop on Software Engineering Education for the Next Generation (SEENG), Melbourne, Australia. 2023. P. 9–16. https://doi.org/10.1109/SEENG59157.2023.00007.

8. Observability vs. monitoring: What's the difference? URL: https://www.ibm.com/blog/ observability-vs-monitoring/ (дата звернення: 20.04.2024).

9. Wang J., Liu B.J., He W., Xue J.K., Han X.Y. Research on computer application software monitoring data processing technology based on NLP. The 10th International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering. 2021. Vol. 1043. https://doi.org/10.1088/ 1757-899X/1043/3/032021.

10. Ma H., Pljonkin A., Singh P.K. Design and implementation of Internet-of-Things software monitoring and early warning system based on nonlinear technology. Nonlinear Engineering. 2022. Vol. 11, no. 1. P. 355–363. https://doi.org/10.1515/nleng-2022-0036.

11. RFC 5424 – The Syslog Protocol. Gerhards, R. Adiscon GmbH, 2009.

12. Супруненко І.О., Рудницький В.М. Адаптивний підхід до логування як новий вимір спостережності за прикладним програмним забезпеченням. VII Міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології", м. Кропивницький, 1 листопада 2023. С. 45–46.

13. RabbitMQ tutorial – "Hello World!" | RabbitMQ. URL: https://www.rabbitmq.com/ tutorials/tutorial-one-javascript (дата звернення: 21.04.2024).

14. PostgreSQL: Documentation: 16: 34.9. Asynchronous Notification. URL: https://www.postgresql.org/docs/current/libpq-notify.html (дата звернення: 21.04.2024).

15. Signal(7) – Linux manual page. URL: https://man7.org/linux/man-pages/man7/signal.7.html (дата звернення: 21.04.2024).

References

1. Digital 2023: Global Overview Report – DataReportal – Global Digital Insights. Retrieved from: https://datareportal.com/reports/digital-2023-global-overview-report (accessed 20.04.2024).

2. Abhijith M. S., Senthilvadivu K. (2020). IMPACT OF VPN TECHNOLOGY ON IT INDUSTRY DURING COVID-19 PANDEMIC. International Journal of Engineering Applied Sciences and Technology, Vol. 5, Issue 5, pp. 152–157. https://doi.org/10.33564/ijeast.2020.v05i05.027.

3. What Is Software Testing? | IBM. Retrieved from: https://www.ibm.com/topics/software-testing (accessed 20.04.2024).

4. Guilherme V., Vincenzi A. (2023). An initial investigation of ChatGPT unit test generation capability. In Proceedings of the 8th Brazilian Symposium on Systematic and Automated Software Testing (SAST '23). Association for Computing Machinery, New York, NY, USA, 15–24. https://doi.org/10.1145/3624032.3624035.

5. Peruma A., Newman C. D. (2021). On the Distribution of "Simple Stupid Bugs" in Unit Test Files: An Exploratory Study. IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), Madrid, Spain. P. 525–529. https://doi.org/10.1109/MSR52588.2021.00067.

6. Peruma A., Almalki K., Newman C. D., Mkaouer M. W., Ouni A., Palomba F. (2020). TsDetect: an open source test smells detection tool. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. P. 1650-1654. https://doi.org/10.1145/3368089.3417921.

7. Ardic B., Zaidman A. (2023). Hey Teachers, Teach Those Kids Some Software Testing. IEEE/ACM 5th International Workshop on Software Engineering Education for the Next Generation (SEENG), Melbourne, Australia. P. 9–16. https://doi.org/10.1109/SEENG59157.2023.00007.

8. Observability vs. monitoring: What's the difference? Retrieved from https://www.ibm.com/blog/observability-vs-monitoring/ (accessed 20.04.2024).

9. Wang J., Liu B.J., He W., Xue J.K., Han X.Y. (2021). Research on computer application software monitoring data processing technology based on NLP. The 10th International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering. Vol. 1043. https://doi.org/10.1088/1757-899X/1043/3/032021.

10. Ma H., Pljonkin A., Singh P.K. (2022). Design and implementation of Internet-of-Things software monitoring and early warning system based on nonlinear technology. Nonlinear Engineering. Vol. 11, no. 1, P. 355–363. https://doi.org/10.1515/nleng-2022-0036.

11. Gerhards, R. (2009). RFC 5424 – The Syslog Protocol. Adiscon GmbH.

12. Suprunenko I., Rudnytskyi V. (2023). Adpative logging method as a new observability dimension in software. Information security and computer technologies: materials of VII international scientific and practical conference, KNTU, pp. 45–46.

13. RabbitMQ tutorial – "Hello World!" | RabbitMQ. Retrieved from: https://www.rabbitmq.com/tutorials/tutorial-one-javascript (accessed 21.04.2024).

14. PostgreSQL: Documentation: 16: 34.9. Asynchronous Notification. Retrieved from: https://www.postgresql.org/docs/current/libpq-notify.html (accessed 21.04.2024).

15. Signal(7) – Linux manual page. Retrieved from: https://man7.org/linux/man-pages/man7/signal.7.html (accessed 21.04.2024).