

МОДЕЛЮВАННЯ ЗВ'ЯЗНОСТІ ВИХІДНОГО КОДУ В СИМУЛЯЦІЯХ ПРОЦЕСУ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Модульна організація вихідного коду програмних систем, низька зв'язність між модулями та високе зчеплення коду всередині них вже більше як 45 років вважаються базовими принципами проєктування програмних продуктів для суттєвого покращення продуктивності розробки, надійності системи та зменшення вартості її подальшого обслуговування [1]. Проте питання досягнення оптимальної модульності як в монолітних системах, так і в мікросервісній архітектурі все ще активно вивчається науковцями різних країн [2; 3; 4; 5; 6]. Одним із способів дослідження впливу різних топологій модульності на продуктивність розробки є симуляція процесу розробки програмного забезпечення на достатньо високому рівні деталізації, який може врахувати вплив зв'язності між модулями на відповідні аспекти процесу розробки.

На даний момент існує багато методів оцінки зв'язності коду як на рівні окремих компонентів, так і на рівні файлів з кодом, а також окремих класів та функцій. В контексті симуляції процесу розробки програмного забезпечення для моделювання зв'язності абстрактних одиниць коду між собою доцільно використати рівень модулів (компонентів, бібліотек), припустивши, що в результаті проєктування та попередньої розробки програмної системи, код в таких модулях має високе зчеплення між собою та низьку зв'язність з іншими модулями.

Враховуючи те, що зв'язність модулів не обмежуються лише прямими залежностями одних від інших, але також включає транзитивні та неявні залежності, так само як і причинно-наслідкові зв'язки [7], то найбільш доцільною для моделювання зв'язності модулів в симуляціях процесу розробки програмного забезпечення буде метрика еволюційної зв'язності, що також використовується для передбачення майбутніх змін коду в пов'язаних модулях [8; 9]. Параметри даної метрики вираховуються шляхом аналізу історії змін коду для виявлення зв'язків між модулями, коли декілька модулів змінюються одночасно в одній транзакції. В методах передбачення майбутніх змін використовуються обидва параметри метрики: “support” (підтримка) та “confidence” (довіра), що розраховуються за формулами 1 та 2, проте для моделювання зв'язності модулів в симуляціях достатньо використання лише параметру “confidence”, що відображає імовірність того, що при внесенні змін в модуль А, модуль В також має бути модифікований.

$$support(A \rightarrow B) = \frac{count(A \cup B)}{N}, \quad (1)$$

де $count(A \cup B)$ – кількість транзакцій зміни коду, в яких одночасно модифікуються модулі А і В, N – загальна кількість транзакцій.

$$confidence(A \rightarrow B) = \frac{count(A \cup B)}{count(A)}, \quad (2)$$

де $count(A \cup B)$ – кількість транзакцій зміни коду, в яких одночасно модифікуються модулі А і В, $count(A)$ – загальна кількість транзакцій, в яких змінюється модуль А.

Використання історичної частоти модифікацій певного модуля в якості імовірності внесення в нього змін на кожній ітерації симуляції не є доцільним, оскільки дана частота також включає в себе вплив зв'язності з іншими модулями, що має моделюватись окремо. Для врахування даних зв'язків та визначення незалежної імовірності внесення змін в модуль на певній симуляційній ітерації, можна скористатись формулою 3.

$$P_k = \frac{count(M_k)}{\sum_i count(M_i)}, \quad (3)$$

де P_k – імовірність вибору k -го модуля для внесення змін в поточній транзакції, $count(M_k)$ – кількість транзакцій, в яких змінюється k -ий модуль.

Отже, відповідно до розподілу імовірностей, обчислених за допомогою формули 3, на кожній ітерації процесу симуляції має обиратись один початковий модуль для внесення основних змін, а також всі пов'язані з ним модулі згідно відповідним асоціативним імовірностям, розрахованим за формулою 2.

Викладений матеріал пропонує підхід до моделювання зв'язності лише статичної топології модулів, проте в процесі еволюції програмної системи, сила зв'язності модулів змінюється з часом, створюються нові та видаляються старі модулі, декілька модулів можуть об'єднуватись в один для підвищення зчеплення коду, або один модуль ділиться на декілька для зменшення зв'язності з іншими модулями. Моделювання перелічених аспектів потребує додаткового вивчення, як і подальше дослідження різних типових топологій зв'язності модулів та кількісних параметрів їх зв'язності.

Список використаних джерел

1. Edward Yourdon and Larry L. Constantine. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design (1st. ed.). Prentice-Hall, Inc., USA. 1979.
2. H. Vural and M. Koyuncu, "Does Domain-Driven Design Lead to Finding the Optimal Modularity of a Microservice?," in *IEEE Access*. Vol. 9. pp. 32721-32733. 2021. DOI: 10.1109/ACCESS.2021.3060895
3. Panichella, Sebastiano, et al. Structural Coupling for Microservices. *arXiv*, 1. 2021. DOI: 10.48550/ARXIV.2103.04674
4. Zhong, C., Zhang, H., Li, C., Huang, H. and Feitosa, D. On measuring coupling between microservices. *Journal of Systems and Software*, 200. 2023. p.111670. DOI: 10.1016/j.jss.2023.111670
5. Apolinário, D.R., de França, B.B. A method for monitoring the coupling evolution of microservice-based architectures. *J Braz Comput Soc* 27 (17). 2021. DOI: 10.1186/s13173-021-00120-y
6. Fregnan, Enrico, Tobias Baum, Fabio Palomba, and Alberto Bacchelli. A Survey on Software Coupling Relations and Tools. *Information and Software Technology*. Elsevier BV. 2019. DOI: 10.1016/j.infsof.2018.11.008
7. James Koppel and Daniel Jackson. Demystifying dependence. In *Proceedings of the 2020 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2020)*. Association for Computing Machinery, New York, NY, USA. 2020. pp. 48–64. DOI: 10.1145/3426428.3426916
8. T. Rolfsnes, S. Di Alesio, R. Behjati, L. Moonen and D. W. Binkley. Generalizing the Analysis of Evolutionary Coupling for Software Change Impact Analysis. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Osaka, Japan. 2016. pp. 201-212. DOI: 10.1109/SANER.2016.101
9. Mondal, Manishankar, Banani Roy, Chanchal K. Roy, and Kevin A. Schneider. Historank: History-based ranking of co-change candidates. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2020. pp. 240-250. DOI: 10.1109/SANER48275.2020.9054869

Науковий керівник: Антонюк Д.С., к. пед. н., доцент.