The Software Genome Project: Unraveling Software Through Genetic Principles

Yueming Wu, Chengwei Liu*, Zhengzi Xu, Lyuye Zhang, Yiran Zhang, Zhiling Zhu, Yang Liu
Nanyang Technological University, Singapore

Abstract

Open-source software is crucial to modern development, but its complexity creates challenges in quality, security, and management. Current governance approaches excel at collaboration but struggle with decentralized management and security. With the rise of large language models (LLM)-based software engineering, the need for a finer-grained understanding of software composition is more urgent than ever. To address these challenges, inspired by the Human Genome Project, we treat the software source code as software DNA and propose the **Software Genome Project (SGP)**, which is geared towards the secure monitoring and exploitation of open-source software. By identifying and labeling integrated and classified code features at a fine-grained level, and effectively identifying safeguards for functional implementations and non-functional requirements at different levels of granularity, the SGP could build a comprehensive set of software genome maps to help developers and managers gain a deeper understanding of software complexity and diversity. By dissecting and summarizing functional and undesirable genes, SGP could help facilitate targeted software optimization, provide valuable insight and understanding of the entire software ecosystem, and support critical development tasks such as open source governance. SGP could also serve as a comprehensive dataset with abundant semantic labeling to enhance the training of LLMs for code. Based on these, we expect SGP to drive the evolution of software development towards more efficient, reliable, and sustainable software solutions.

CCS Concepts

• **Software and its engineering → Software design engineering**.

Keywords

Software Genes, Software Composition, OSS Governance.

ACM Reference Format:

Yueming Wu, Chengwei Liu*, Zhengzi Xu, Lyuye Zhang, Yiran Zhang, Zhiling Zhu, Yang Liu. 2024. The Software Genome Project: Unraveling Software Through Genetic Principles. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3691620.3695307

Chengwei Liu is the corresponding author.
Zhiling Zhu is also with Zhejiang University of Technology.

ASE '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1248-7/24/10…$15.00
https://doi.org/10.1145/3691620.3695307

1 Introduction

With the pervasive integration of open-source technology, open-source projects have evolved into fundamental infrastructure in contemporary software development. The expansive growth of open-source ecosystems has injected dynamism into the software landscape [1]. However, the inherent freedom and flexibility of open-source software, while fostering popularity, also introduce noteworthy challenges, necessitating robust governance [2]. For instance, as a paramount concern in real-world open-source development, security issues [3], such as vulnerabilities [4], could be easily introduced during collaborative contributions [5], while many open-source projects, driven by volunteer contributions, may lack stringent management practices [6], resulting in varying software quality and being susceptible to malicious compromises like backdoors [7]. Even worse, as the prevalent involvement of large language models in modern generative software engineering [8–10], software code could be composed by even more fragmented and complex code snippets, inadequate professional management of open-source contributions may lead to even higher risks of software supply chain risks [11], such as insufficient support [12], maintenance lapses [13], and unclear licenses [14–16], compromising the sustainability of these projects. Therefore, addressing these challenges is vital to overcoming obstacles and ensuring effective utilization of open-source software in modern development.

In response to the aforementioned challenges, we adopt a unique approach that views software code as analogous to biological DNA and take a page from the Human Genome Project (HGP)'s playbook, leading to the inception of the Software Genome Project (SGP). The HGP [17, 18], launched in the early 1990s, stands as one of the most remarkable scientific endeavors in history. Its primary aim was to decode and comprehensively map all the genes of the human species. This monumental project sought to understand human biology at the molecular level, offering insights into our genetic heritage, hereditary diseases, and the fundamental building blocks of human life. Taking a cue from the HGP, we introduce the SGP as a parallel initiative focusing on open source software. SGP's overarching goal is to achieve a comprehensive genetic mapping of software, akin to the HGP's sequencing of the human genome. In the realm of software, this translates to dissecting and annotating the complete genetic structure of code, from its most elementary components to its most intricate functionalities. By extending the analogy between software code and biological DNA, SGP aspires to provide profound insights into open source software's genetic makeup. Such insights open the door to more effective governance, maintenance, and security in the world of software development.

The SGP, influenced by the transformative HGP, presents a comprehensive approach to revolutionize the open-source software landscape. Comprising five main phases, it harnesses the insights from human genetics to comprehend software code intricacies.

① Beginning with the construction of a comprehensive software genome and intelligence genome, this initiative paves the way for robust security analyses and defenses. ② After collecting the genome, we apply detailed annotation to discover software genes, the functionality of genes, and the functionality of intelligence genome. ③ The subsequent phases delve deeper into genome analysis, offering insights into software gene relationships, families, and evolution. ④ These knowledge is further leveraged to facilitate comprehensive software composition analysis, akin to genetic testing, and software maintenance, similar to genetic engineering. ⑤ Ultimately, the SGP serves as a catalyst for open source governance, fostering real-time security, compatibility, and compliance in the dynamic world of open source software. Overall, the SGP, inspired by the concept of DNA in genetics, undertakes a systematic and comprehensive approach to enhance open-source software security. By treating software code as genetic material and adapting genetic principles, we can revolutionize the understanding, evaluation, and protection of open-source software.

To pioneer the understanding and derive actionable solutions of SGP, in this paper, we first introduce the basic concepts of the Human Genome Project, and based on that, we propose the first analogy from concepts of software composition from different granularity to HGP concepts, and derive the similarities and differences between HGP and SGP, to discuss the actionability, potential roadmap, and future impact of SGP.

2 Background

In this part, we introduce the concepts in human genome and the Human Genome Project (HGP). Drawing inspiration from the HGP, we establish a mapping relationship between DNA and software, leading to the proposed pathway to the Software Genome Project.

2.1 Central Dogma of Molecular Biology

The Central Dogma of Molecular Biology [19, 20], a fundamental concept in genetics, describes the flow of genetic information within a biological system. This process begins with the transcription of DNA into messenger RNA (mRNA). The mRNA then undergoes translation to form proteins, which are crucial for various cellular functions. This directional flow of information – from DNA to RNA to protein – is pivotal in understanding how genetic information is expressed in living organisms. In a similar vein, in computer science, the transformation of source code into executable programs mirrors aspects of the Central Dogma. Source code, much like DNA, contains the original instructions or information. Compiler acts like mRNA, converting this source code into a executable program, just as mRNA translates genetic information into proteins. This executable program is what computers execute to perform specific tasks. Additionally, the replication of source code to create clone code in software development is analogous to the replication of DNA in biological systems, where the genetic code is duplicated to ensure continuity and fidelity of information across generations. This parallel between the Central Dogma of Molecular Biology and computer programming highlights a fascinating intersection of concepts across biology and computer science.

Based on the mapping relationship established by the Central Dogma of Molecular Biology, we can draw analogies from biology in three aspects as shwon in Figure 1: source code to DNA, compiler to mRNA, and executable program to protein. This comprehensive

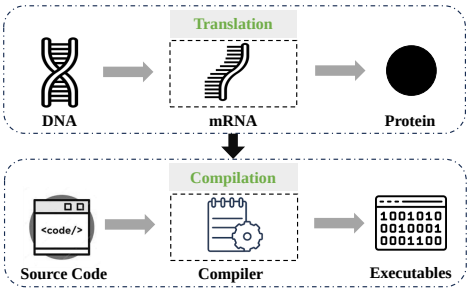

Figure 1: Central dogma of molecular biology and the process of source code to executables

approach aims to leverage biological insights for advancing our understanding and practices in software development and governance. In this paper, we first primarily focus on the detailed analysis of source code (software DNA), with the objective of applying insights from the well-established field of DNA development to enhance the understanding of software composition.

2.2 Human Genome Project

Based on the genetic concepts, we give a brief description of the Human Genome Project (HGP) [17, 18] and then clarify the related terms to software genes. The HGP was a major global scientific effort initiated in 1987 and formally launched in 1990. It aimed to decode the 3 billion base pairs within human chromosomes, creating a comprehensive map of the human genome, including both coding and non-coding sequences. This collaborative endeavor, involving scientists from various nations, culminated in April 2003. The HGP is often considered as one of the most significant scientific projects in history, akin to the Manhattan Atomic Bomb Project [21] and the Apollo Project [22]. The outcomes of the HGP are invaluable, serving as a foundational resource for medicine, biology, and genetics, aiding in the study of hereditary diseases, personalized medicine, pharmaceutical development, and gene-environment interactions. The basic terms used in Human Genome [23] are as follows:

- **Base** [24]: A base is a chemical unit that makes up DNA or RNA molecules. In DNA, there are four types of bases: adenine (A), cytosine (C), guanine (G), and thymine (T). They form base pairs (A-T and C-G) and are essential for encoding genetic information.
- **Codon** [25]: A codon is a sequence of three consecutive bases in DNA or RNA. It represents the basic unit of the genetic code, specifying an amino acid in protein synthesis. Codons guide protein synthesis by determining amino acid sequences, and there are 64 unique codons, including three stop codons that terminate protein synthesis.
- **DNA Fragment** [26]: A DNA fragment is a part of the DNA molecule that can be a coding region or a non-coding region. A coding region is a gene that encodes proteins, which is essential component of an organism's functions. A non-coding region, while not directly encoding proteins, can also play a crucial role in gene regulation and other cellular processes.
- **Chromosome** [27]: A chromosome is a linear structure made of DNA and associated proteins. It contains a long DNA molecule, tightly coiled to ensure orderly genetic information transmission. They have an organized arrangement of genes and non-coding regions, ensuring accurate gene expression and inheritance.

- **Human**: A human typically contains 46 chromosomes, organized into 23 pairs. Males have one pair of sex chromosomes, comprising one X and one Y chromosome (XY), while females have one pair of X chromosomes (XX).
- **Human Genome**: The human genome refers to the complete collection of genetic information within the human body. The DNA differences between different individuals account for a relatively small portion of the entire human genome. The human genome is an extensive molecular database, consisting of billions of base pairs that form the genetic code of human beings.

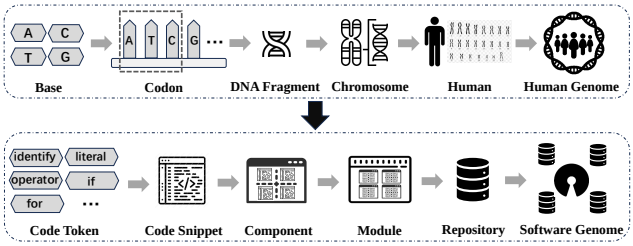

Figure 2: Mapping relationship between DNA and open-source software

In conclusion, the human genome, in its entirety, is the collective repository of all genetic information contained within human beings. This vast genetic archive includes not only the complete set of genes encoded in DNA, but also extends to non-coding regions, regulatory elements and additional DNA sequences that are intricately linked to heritable traits and biological functions. In essence, the genome serves as the genetic blueprint within an organism, meticulously dictating its growth, development, functionality and distinctive characteristics. Genes, embedded within the genome, represent specific fragments of DNA with the essential function of encoding proteins. Genomic research is emerging as an indispensable endeavour, providing profound insights into the genetic make-up of organisms, evolution, health and disease. It reveals the intricate interplay between genes and how they collectively influence the characteristics and behaviour of an organism.

3 Software Genome Project

Inspired by human genome, we adopt a perspective that treats software code as akin to biological DNA. In this part, we first introduce several terms used in software genome.

- **Code Token**: A code token is the fundamental building block of software, similar to how a base is a fundamental unit in DNA. Code tokens include elements like *Identifier*, *Keywords*, and *Operator*, and they form the basic syntax of a programming language.
- **Code Snippet**: A code snippet is a sequence of code tokens, just as a codon is a sequence of three consecutive bases. Code snippets represent a basic unit of the software's functionality, specifying a specific operation or function in the program.
- **Component**: A component in software is a larger unit that comprises multiple code snippets. It can include functionalities such as functions, classes, or files. Components can be either coding regions, which directly contribute to the software's functionalities, or non-coding regions that support software maintenance.
- **Module**: A module in software is an organized structure that contains multiple components for organizing and isolating functionalities. Modules aim to enhance code organization, promote reusability, and facilitate maintenance in software projects, akin to chromosomes in genetics, which help maintain order and transmit genetic information.

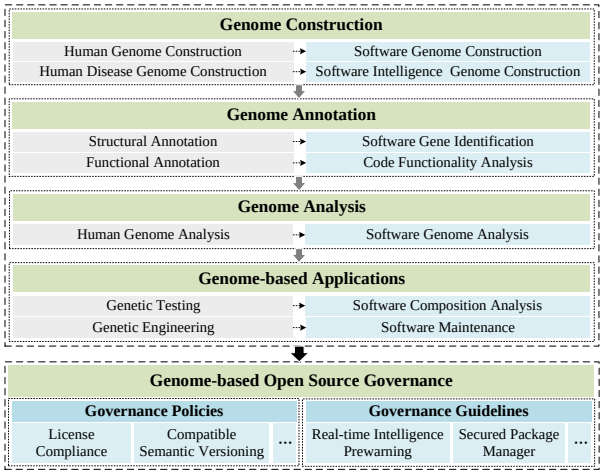

Figure 3: Framework of the software genome project

- **Repository**: A repository, similar to a human being, is a complete entity. It encompasses all the modules, components, and code snippets that make up the functional aspects of the software. A repository is designed to serve specific purposes and functions, just as a human being has a specific genetic makeup and characteristics.
- **Software Genome**: The software genome is a metaphorical term used to describe the complete collection of code within a software system. It serves as a comprehensive reference of software code, akin to the human genome.

Figure 2 shows the mapping relationship between DNA and software. We can see that an individual carries multiple chromosome, each comprising coding and non-coding segments. Segments of DNA that encode proteins are referred to as genes. A DNA fragment consists of multiple codons, with each codon made up of three bases. Looking at software architecture, a software repository is a composition of various modules, each module housing several components. Components, in turn, contain multiple code snippets, and each snippet's purpose is achieved through different code token combinations. When viewing a software as a human, all of its code constitutes its genetic material, and the collective code of all open-source software in the open source community constitutes what we refer to as the software genome. Building on this analogy, we introduce the concept of the Software Genome Project (SGP), aiming to apply a HGP-like approach to the software community.

As depicted in Figure 3, the SGP comprises five major stages:

❶ **Genome Construction**. The scope of this initiative extends beyond the mere construction of the software genome, it also involves the assembly of a software intelligence genome, drawing parallels with the human disease genome. This comprehensive dataset equips us to conduct exhaustive security analyses and enact robust software defenses. In this stage, the SGP aims to answer the questions: what kind of objectives in source code should be considered as genes? what is the scope of software genes that compose the software world? what kind of software genes are there to threatening real-world software?

❷ **Genome Annotation**. Subsequently, SGP delves into structural annotation to identify software genes that hold paramount significance within the open-source community, meriting ongoing maintenance. Moreover, it also perform functional annotation to enable us to comprehend the roles and functionalities of all software genes and the software intelligence genome. This is a fundamental step to interpret the semantics of software genes, based on which, large-scale tasks, such as training codeLLMs with semantic-enhanced data, finer-grained migrations, can be further conducted.
❸ **Genome Analysis**. Armed with this knowledge, we proceed to the analysis of gene clusters, delving into aspects such as dependency cohesion, software architecture, portrait, and evolution. Based on these, we could have deeper understanding on the formation, collaboration, semantic aggregation, as well as evolution of software tech stacks to guide future developement.
❹ **Genome-based Applications**. SGP also incorporates the knowledge acquired through genome analysis into existing techniques to enhance the effectiveness of software composition analysis and software maintenance, akin to the principles of genetic testing and genetic engineering.
❺ **Genome-based Open Source Governance**. Moreover, unlike the natural evolution of biological genes, we got the chance to participate and guide the evolution of software ecosystems. Our ultimate goal of SGP is to chart a course for open-source governance, ensuring the development of a robust, real-time, secured, and effective open-source community.

4 Discussion

We discuss the insights from the similarities and differences between the software genes and the biological genes.

Similarity. ① **Species & Domains.** The genetic information (inner factor) and environment (outer factor) decide the functioning of both software and biological individuals. Similar to Human genes, source code, the genes of software, is composed of tokens of code. Common patterns of token compositions could also be interpreted as meaningful fragments (i.e., genes) when they are expressed to external manifestation, and these manifestations also distinguish individuals as different categories and species, leading to the diversity of software. ② **Survival of Fitness.** The environment also fosters the metamorphosis of software genes, including not only the competition but also the corporation. Software genes that are more adaptable to the environment would be gradually more competitive than less fitted ones (e.g., the iteration of technique stacks towards user requirements), and new combinations of software genes could also enjoy a renaissance when new opportunities arise. ③ **Engineering**. Engineering can intervene in the genes of both software and creatures for maintenance and improvement. For instance, genes could be edited to prevent genetic disease, like fixing patches for bugs and code smells.

Difference. ① **Reproductive Isolation vs. Technique Stack.** The evolution of the biological genes could lead to reproductive isolation between species, which blocks the communication of genes from different species. While software domains do not isolate vertical clusters, genes and functionalities (e.g., technique stacks) could still be referred to and transferred among domains. ② **Food Chain vs. Supply Chain.** In the biological world, cooperation and competition are established on the basic relationship, i.e., the food chain among species, which unites the biosphere and fosters its stability. While in the world of software, the fundamental relationship is no longer the predator-prey relationship but upstream and downstream supply chain relationships. The downstream software consumes the upstream ones, while such a supply-chain relationship would facilitate the prosperity of both upstream and downstream software (double-win) instead of threatening the existence of prey species, and then, further influencing the predator species (double-lose when unbalanced) in the biological world. ③ **Nature Evolution vs. Hands of God.** In the biological realm, genes naturally mutate over time, allowing species to adapt through natural selection for long-term survival. Conversely, in the software world, software is created to meet human requirements and community support, shaping the evolution of techniques and the prosperity of software domains. Unlike manipulating biological genes, it is much easier to manipulate software genes and guide the community to devote to specific domains of software (i.e., open source governance and regulations), like the hands of god for the biosphere.

The SGP seeks to engender a comprehensive and nuanced understanding of software creation, encompassing not only the structural composition and functional aspects of code but also delving into the methodologies underlying its development and maintenance. Drawing an analogy with human genetics and inspired by the Human Genome Project, the SGP endeavors to transpose concepts, methodologies, measurement techniques, and governance strategies from the realm of biological sciences to elucidate the intricacies of "software genes", thereby paving the way for enhanced future governance in this domain. Additionally, given the rapid iteration of software in contrast to the slower evolutionary pace of biological genes, it is anticipated that insights gleaned from the SGP may reciprocally inform and enrich the fields of biological genetic engineering and biosphere governance.

5 Conclusion

This paper draws inspiration from the Human Genome Project to provide a Software Genome Project to understanding software composition, likening it to human genetic formation. This analogy is used to develop strategies for improving software maintenance and guiding the evolution of the software ecosystem. We focus on analyzing code snippets to understand software formation, identify coding knowledge diversity, and assess the ecosystem's capacity. The insights gained from this software genome analysis can illuminate the software ecosystem's evolution and provide valuable reference points for understanding other evolving subjects.

Acknowledgments

We would thank the anonymous reviewers for their insightful comments to improve the quality of the paper. This work is supported by the National Research Foundation, Singapore, and the Cyber Security Agency under its National Cybersecurity R&D Programme (NCRP25-P04-TAICeN), the National Research Foundation Singapore and DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2-RP-2020-019), and NRF Investigatorship NRF-NRFI06-2020-0001. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Cyber Security Agency of Singapore.

References

[1] Oscar Franco-Bedoya, David Ameller, Dolors Costal, and Xavier Franch. Open source software ecosystems: A systematic mapping. *Information and software technology*, 91:160–185, 2017.

[2] Pros and cons of open-source software. https://www.vspry.com/pros-and-cons-of-open-source-software/, 2023.

[3] Christian Payne. On the security of open source software. *Information systems journal*, 12(1):61–78, 2002.

[4] Scott A. Hissam, Daniel Plakosh, and C Weinstock. Trust and vulnerability in open source software. *IEE Proceedings-Software*, 149(1):47–51, 2002.

[5] Andrew Meneely and Laurie Williams. Secure open source collaboration: an empirical study of linus' law. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 453–462, 2009.

[6] Paul Kavanagh. *Open source software: Implementation and management*. Elsevier, 2004.

[7] State of open source security 2023. https://snyk.io/reports/open-source-security/, 2023.

[8] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*, pages 1–7, 2022.

[9] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. Large language models for software engineering: Survey and open problems. *arXiv preprint arXiv:2310.03533*, 2023.

[10] Yuan Huang, Yinan Chen, Xiangping Chen, Junqi Chen, Rui Peng, Zhicao Tang, Jinbo Huang, Furen Xu, and Zibin Zheng. Generative software engineering. *arXiv preprint arXiv:2403.02583*, 2024.

[11] Threats associated with llm and generative ai: Safeguarding enterprise open-source practices | by aishield | medium. https://boschaishield.medium.com/threats-associated-with-llm-and-generative-ai-safeguarding-enterprise-open-source-practices-c6ffae621934, 2024. (Accessed on 06/17/2024).

[12] The open source sustainability crisis - open path by chad whitacre. https://openpath.chadwhitacre.com/2024/the-open-source-sustainability-crisis/, 2024. (Accessed on 06/17/2024).

[13] Ido Morag, Peter Chemweno, Liliane Pintelon, and Mohammad Sheikhalishahi. Identifying the causes of human error in maintenance work in developing countries. *International Journal of Industrial Ergonomics*, 68:222–230, 2018.

[14] Tao Liu, Chengwei Liu, Tianwei Liu, He Wang, Gaofei Wu, Yang Liu, and Yuqing Zhang. Catch the butterfly: Peeking into the terms and conflicts among spdx licenses. *arXiv preprint arXiv:2401.10636*, 2024.

[15] Jiaqi Wu, Lingfeng Bao, Xiaohu Yang, Xin Xia, and Xing Hu. A large-scale empirical study of open source license usage: Practices and challenges. 2024.

[16] Sihan Xu, Ya Gao, Lingling Fan, Zheli Liu, Yang Liu, and Hua Ji. Lidetector: License incompatibility detection for open source software. *ACM Transactions on Software Engineering and Methodology*, 32(1):1–28, 2023.

[17] The human genome project. https://www.genome.gov/human-genome-project, 2023.

[18] Maynard V Olson. The human genome project. *Proceedings of the National Academy of Sciences*, 90(10):4338–4344, 1993.

[19] Matthew Cobb. 60 years ago, francis crick changed the logic of biology. *PLoS biology*, 15(9):e2003243, 2017.

[20] Francis H Crick. On protein synthesis. In *Symp Soc Exp Biol*, volume 12, page 8, 1958.

[21] Francis George Gosling. *The Manhattan Project: making the atomic bomb*. Diane Publishing, 1999.

[22] The apollo program. https://www.nasa.gov/the-apollo-program/, 2023.

[23] Elizabeth Pennisi. The human genome, 2001.

[24] Nucleobase - wikipedia. https://en.wikipedia.org/wiki/Nucleobase, 2024. (Accessed on 06/17/2024).

[25] Dna and rna codon tables - wikipedia. https://en.wikipedia.org/wiki/DNA_and_RNA_codon_tables, 2024. (Accessed on 06/17/2024).

[26] Dna fragmentation - wikipedia. https://en.wikipedia.org/wiki/DNA_fragmentation, 2024. (Accessed on 06/17/2024).

[27] Chromosome - wikipedia. https://en.wikipedia.org/wiki/Chromosome, 2024. (Accessed on 06/17/2024).