

The Name of the Title Is Hope

Federico Bruzzone

federico.bruzzone@unimi.it

Università degli Studi di Milano

Computer Science Department

Milan, Italy, Europe

Walter Cazzola*

cazzola@di.unimi.it

Università degli Studi di Milano

Computer Science Department

Milan, Italy, Europe

Abstract

A clear and well-documented L^AT_EX document is presented as an article formatted for publication by ACM in a conference proceedings or journal publication. Based on the “acmart” document class, this article presents and explains many of the common variations, as well as many of the formatting elements an author may use in the preparation of the documentation of their work.

CCS Concepts

• Theory of computation → Program analysis; • Software and its engineering → Compilers.

Keywords

Do, Not, Us, This, Code, Put, the, Correct, Terms, for, Your, Paper

ACM Reference Format:

Federico Bruzzone and Walter Cazzola. 2018. The Name of the Title Is Hope. In *Proceedings of 29th International Systems and Software Product Line Conference (SPLC’25)*. ACM, New York, NY, USA, 3 pages. <https://doi.org/XXXXXXX.XXXXXXX>

1 Introduction

 [Acronyms: SPL]

2 Background

In this section, we introduce some preliminary concepts that are necessary to understand the rest of the paper. We start by introducing the Rust programming language and its ownership system. Then, we introduce the concept of software product lines (SPLs) and the importance of testing SPLs. Finally, we provide an overview of centrality measures used in network analysis.

2.1 The Rust Programming Language

Rust is a systems programming language that focuses on safety, speed, and concurrency. It is designed to be memory-safe without using garbage collection. This implies that pure Rust programs are free of null pointer dereferences, double frees as well as data races. The *linear logic* [12, 13] and *linear types* [21, 28]—which force the

use of resources exactly once—inspired the *ownership* system. Rust incorporates it into its type system as relaxed form of pure linear types to ensure type soundness. The ownership system ensures that there is only one *owner* (the variable binding) for each piece of memory (a value) at any given time, and when the owner goes out of scope or is otherwise deallocated, the memory is deallocated as well. By leveraging the latter property, Rust supports user-defined destructors, enabling *resource acquisition is initialization* (RAII) pattern proposed by Stroustrup [27]. The lifetime of the owned value is determined by the scope in which the owner takes ownership. An owner can *move* (transfer) the ownership of the value to a new owner or *borrow* the value to another part of the program. By *moving* the ownership, the previous owner can no longer access the value. On the other hand, Rust support *references* that allow the owner to *borrow* the value avoiding the its invalidation. Two kind of *borrow*s are supported: *immutable* and *mutable*. Multiple *immutable borrow* can coexist, but only one *mutable borrow* can exist at a time. These restrictions allow Rust to guarantee memory safety. Furthermore, the lifetime of a reference can not outlive (exceed) the lifetime of the owner, which ensures no dangling pointers. The Rust compiler enforces all these rules at compile time also by performing *borrow checking*, preserving the runtime performance of the compiled code. Despite the notable progress in the field of safe systems programming, Rust allows **unsafe** blocks to perform low-level operations that are not safe, such as dereferencing raw pointers. In Rust, the **unsafe** keyword signifies that the responsibility for preventing undefined behavior shifts from the compiler to the programmer. This ensures that undefined behavior cannot occur in safe Rust code, as the compiler enforces strict safety guarantees in all safe contexts.

2.2 Product Families

Since the nineties, variability-rich software systems development leverages principles from product line engineering, commonly referred to as *feature-oriented programming* [25] and *software product line* [9] engineering. In this context the artifacts, which are well-defined units of software, belong to the set of artifacts $A = \{a_1, a_2, \dots, a_n\}$. In the product families the similarities and differences are characterized by a set of features $F = \{f_1, f_2, \dots, f_m\}$ where each feature $f_i \subseteq A$ is a set of artifacts. A feature is a unit that provides a piece of functionality that satisfies a requirement or represents a design decision. An artifact may be explicitly—via an annotation—or implicitly—via a dependency—associated with a feature. Given fixed i and j where $i \neq j$, it does not necessarily follow that $f_i \cap f_j = \emptyset$. A product line, or rather a family of products, is a set of products $P = \{p_1, p_2, \dots, p_k\}$ where each product $p_i \subseteq F$ is a set of features and for fixed i and j holds that

*Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SPLC’25, September 01–September 05, 2025, A Coruña, Spain

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-XXXX-X/18/06

<https://doi.org/XXXXXXX.XXXXXXX>

$i \neq j \not\Rightarrow p_i \cap p_j = \emptyset$, meaning that the disjointness of the features is not guaranteed. A key task in SPL engineering is feature modeling, which involves creating and maintaining a *feature model*. The concept of feature model was first introduced by Kang et al. [14] in the FODA method and serves to represent the variability of a system through its features and their interdependencies. In SPLs, the feature model formalism is essential for configuring software products by defining valid feature sets, known as *configurations*. A configuration $c : F \rightarrow \{0, 1\}$ is a characteristic function over F that maps each feature to a boolean value. A feature f is considered *active* if it belongs to a configuration c such that $c(f) = 1$, otherwise it is *inactive*. The structure of a feature model implicitly captures feature dependencies by specifying mandatory, optional, alternative, and grouped features. These dependencies are often represented as parent-child relationships, where a feature can only be *active* if all its parent features are also *active*. A configuration c is deemed *valid* if and only if $\forall f_i \in F \mid c(f_i) = 1 \implies \exists p \in P \mid f_i \in p$. Given a product $p_i \in P$ we say that all products $p_j \in P$ such that $p_j \neq p_i$ are variants of p_i denoted as v_j . It is worth noting that a family of products P can theoretically contain up to $2^{|F|}$ variants, as described by Krueger [16]. This exponential growth in potential configurations has paved the way for the development of techniques to manage and test SPLs effectively [24]. Numerous approaches, recently analyzed by Agh et al. [1], have been proposed to address the challenges of testing SPLs, including product sampling [2, 18, 23] and combinatorial testing [19, 22].

2.3 Centrality Measures

Since the early days of social network analysis, *centrality measures* have been used in sociology and psychology to identify the most important nodes in a network [3, 15, 26]. Over the years, they have played a crucial role being a object of study in graph theory and network analysis [7, 10, 17]. The shared idea is to consider a *network*—which is a graph—on n nodes indexed by $i \in \{1, 2, \dots, n\}$. It is represented by an adjacency matrix $A \in \mathbb{R}^{n \times n}$, where $A_{ij} \neq 0$ if there is an edge between nodes i and j , otherwise $A_{ij} = 0$. According to Bloch et al. [4], signs and weights are not always considered by all centrality measures but could be taken into account. A centrality measure is a function $C : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}^n$, where $C_i(A)$ is a real number that represents the importance of node i in the network A . It is a cardinal invariant, meaning that an ordinal ranking of nodes can be obtained by comparing the values of $C_i(A)$ for all i . They can be classified into two main categories: *geometry-based* and *spectral-based* [5]. In the following, we introduce some of the most common centrality measures. The *degree centrality* is a measure that assigns a value to a node based on the number of edges incident to it, denoted as $d^-(i)$ for the in-degree and $d^+(i)$ for the out-degree. The *closeness centrality* introduced by Bavelas [3] could not be computed for disconnected graphs because of the potential division by zero, so it has been “patched” as follows:

$$\text{Clo}_i(A) = \frac{1}{\sum_{j=1}^n d(i, j)} \quad (1)$$

where $d(i, j)$ is the shortest path length between nodes i and j . The *harmonic centrality* [20] is the reciprocal of the sum of the shortest path lengths between a node and all other nodes: The *betweenness*

centrality [11] is a measure of the number of shortest paths that pass through a node:

$$\text{Bet}_i(A) = \sum_{s \neq i \neq t} \frac{\sigma_{st}(i)}{\sigma_{st}} \quad (2)$$

where σ_{st} is the number of shortest paths between nodes s and t and $\sigma_{st}(i)$ is the number of shortest paths between nodes s and t that pass through node i . The *eigenvector centrality* [6] is a measure that assigns a value to a node based on the centrality of its neighbors, in linear algebra terms, $\text{Eig}(A)$ is the eigenvector associated with the largest eigenvalue λ of the adjacency matrix A :

$$\lambda v = Av \quad (3)$$

The *Katz centrality* [15] considers the number of paths of different lengths that connect a node to all other nodes, in linear algebra terms:

$$k = (1 - \beta A)^{-1} \quad (4)$$

where $\mathbf{1}$ is a vector of ones and β is a dumping factor that must be less than $1/\lambda$ where λ is the dominant eigenvalue of A . The *PageRank* [8] is a variant of the Katz centrality that considers the probability of a random walker to visit a node, in linear algebra terms, it is a unique solution of the following equation:

$$p = \alpha \bar{A} + (1 - \alpha)v \quad (5)$$

where \bar{A} is l_1 -normalized adjacency matrix, v is a *preference vector* (must be a distribution), and $\alpha \in [0, 1]$ is a dumping factor.

3 Related Work

[\[BF\] \[Nella sezione 4.4 di \[1\] \]](#) [\[BF\] \[Guardare le precedenti literature review \]](#)

Acknowledgments

[\[BF\] \[TODO \]](#)

References

- [1] Halimeh Agh, Aidin Azamnouri, and Stefan Wagner. 2024. Software product line testing: a systematic literature review. *Empirical Software Engineering* 29, 6 (2024), 146.
- [2] Mustafa Al-Hajjaji, Thomas Thüm, Malte Lochau, Jens Meinicke, and Gunter Saake. 2019. Effective product-line testing using similarity-based product prioritization. *Software & Systems Modeling* 18 (2019), 499–521.
- [3] Alex Bavelas. 1950. Communication patterns in task-oriented groups. *The journal of the acoustical society of America* 22, 6 (1950), 725–730.
- [4] Francis Bloch, Matthew O Jackson, and Pietro Tebaldi. 2023. Centrality measures in networks. *Social Choice and Welfare* 61, 2 (2023), 413–453.
- [5] Paolo Boldi and Sebastiano Vigna. 2014. Axioms for centrality. *Internet Mathematics* 10, 3–4 (2014), 222–262.
- [6] Phillip Bonacich. 1972. Factoring and weighting approaches to status scores and clique identification. *Journal of mathematical sociology* 2, 1 (1972), 113–120.
- [7] Stephen P Borgatti. 2005. Centrality and network flow. *Social networks* 27, 1 (2005), 55–71.
- [8] Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems* 20 (April 1998), 107–117.
- [9] Paul Clements and Linda Northrop. 2001. *Software Product Lines: Practices and Patterns*. Addison-Wesley.
- [10] Kousik Das, Sovan Samanta, and Madhumangal Pal. 2018. Study on centrality measures in social networks: a survey. *Social network analysis and mining* 8 (2018), 1–11.
- [11] LC Freeman. 1977. A set of measures of centrality based on betweenness. *Sociometry* (1977).
- [12] Jean-Yves Girard. 1987. Linear logic. *Theoretical computer science* 50, 1 (1987), 1–101.
- [13] Jean-Yves Girard, Yves Lafont, and Laurent Regnier. 1995. *Advances in linear logic*. Vol. 222. Cambridge University Press.

- [14] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-21. Carnegie Mellon University, Pittsburgh, Pennsylvania, USA.
- [15] Leo Katz. 1953. A new status index derived from sociometric analysis. *Psychometrika* 18, 1 (1953), 39–43.
- [16] Charles W. Krueger. 2006. New Methods in Software Product Line Practice. *Commun. ACM* 49, 12 (Dec. 2006), 37–40.
- [17] Andrea Landherr, Bettina Friedl, and Julia Heidemann. 2010. A critical review of centrality measures in social networks. *Business & Information Systems Engineering* 2 (2010), 371–385.
- [18] Jihyun Lee and Sunmyung Hwang. 2019. Combinatorial Test Design Using Design-Time Decisions for Variability. *International Journal of Software Engineering and Knowledge Engineering* 29, 08 (2019), 1141–1158.
- [19] Malte Lochau, Sebastian Oster, Ursula Goltz, and Andy Schürr. 2012. Model-based pairwise testing for feature interaction coverage in software product line engineering. *Software Quality Journal* 20 (2012), 567–604.
- [20] Massimo Marchiori and Vito Latora. 2000. Harmony in the small-world. *Physica A: Statistical Mechanics and its Applications* 285, 3-4 (2000), 539–546.
- [21] Martin Odersky. 1992. Observers for linear types. In *European Symposium on Programming*. Springer, 390–407.
- [22] Sebastian Oster, Florian Markert, and Philipp Ritter. 2010. Automated incremental pairwise testing of software product lines. In *International Conference on Software Product Lines*. Springer, 196–210.
- [23] Sachin Patel, Priya Gupta, and Vipul Shah. 2013. Combinatorial interaction testing with multi-perspective feature models. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 321–330.
- [24] Klaus Pohl and Andreas Metzger. 2006. Variability Management in Software Product Line Engineering. In *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)*, Leon J. Osterwell, H. Dieter Rombach, and Mary Lou Soffa (Eds.). ACM, Shanghai, China, 1049–1050.
- [25] Christian Prehofer. 1997. Feature-Oriented Programming: A Fresh Look at Objects. In *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97) (Lecture Notes in Computer Science 1241)*, Mehmet Akşit and Satoshi Matsuoka (Eds.). Springer, Helsinki, Finland, 419–443.
- [26] John R Seeley. 1949. The net of reciprocal influence. a problem in treating sociometric data. *Canadian Journal of Experimental Psychology* 3 (1949), 234.
- [27] Bjarne Stroustrup. 1994. *The design and evolution of C++*. Pearson Education India.
- [28] Philip Wadler. 1990. Linear types can change the world!. In *Programming concepts and methods*, Vol. 3. Citeseer, 5.

A Appendix 1

A.1 Part One

 [TODO]

A.2 Part Two

 [TODO]

B Appendix 2

C Part One

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009