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

Figure 1: Seattle Mariners at Spring Training, 2010.

Abstract

A clear and well-documented LaTeX 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, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

*Corresponding author.

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

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

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 in graph theory, which are necessary to understand the approach we propose in this paper.

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* [1, 2] and *linear types* [3, 6]—which force the use of resources exactly once—inspired the ownership system in Rust. Rust incorporates the *ownership* system 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 [4]. 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 invalidation of the owner. Two kind of *borrows* are supported: *immutable* and *mutable*. Multiple *immutable borrow* can

coexist, but only one *mutable borrow* can exist at a time. The *borrow checker* enforces these rules at compile 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, thus avoiding a runtime overhead. 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 Software Product Lines

2.3 Centrality Measures

Acknowledgments

BF [TODO: [5]]

References

[1] Jean-Yves Girard. 1987. Linear logic. *Theoretical computer science* 50, 1 (1987), 1–101.
[2] Jean-Yves Girard, Yves Lafont, and Laurent Regnier. 1995. *Advances in linear logic*. Vol. 222. Cambridge University Press.
[3] Martin Odersky. 1992. Observers for linear types. In *European Symposium on Programming*. Springer, 390–407.
[4] Bjarne Stroustrup. 1994. *The design and evolution of C++*. Pearson Education India.
[5] Test. 0. Test. *Test* 0 (0), 0.
[6] 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

BF [TODO]

A.2 Part Two

BF [TODO]

B Appendix 2

C Part One

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