# Automatically Characterising the Purpose of GitHub Repositories

By Wilson Blaikie, Supervised by Christoph Treude

README.md

## Dyno: Runtime polymorphism done right

build error

### DISCLAIMER

At this point, this library is experimental and it is a pure curiosity. No stability of interface or quality of implementation is guaranteed. Use at your own risks.

### Overview

**Dyno** solves the problem of runtime polymorphism better than vanilla C++ does. It provides a way to define interfaces that can be fulfilled non-intrusively, and it provides a fully customizable way of storing polymorphic objects and dispaching to virtual methods. It does not require inheritance, heap allocation or leaving the comfortable world of value semantics, and it can do so while outperforming vanilla C++.

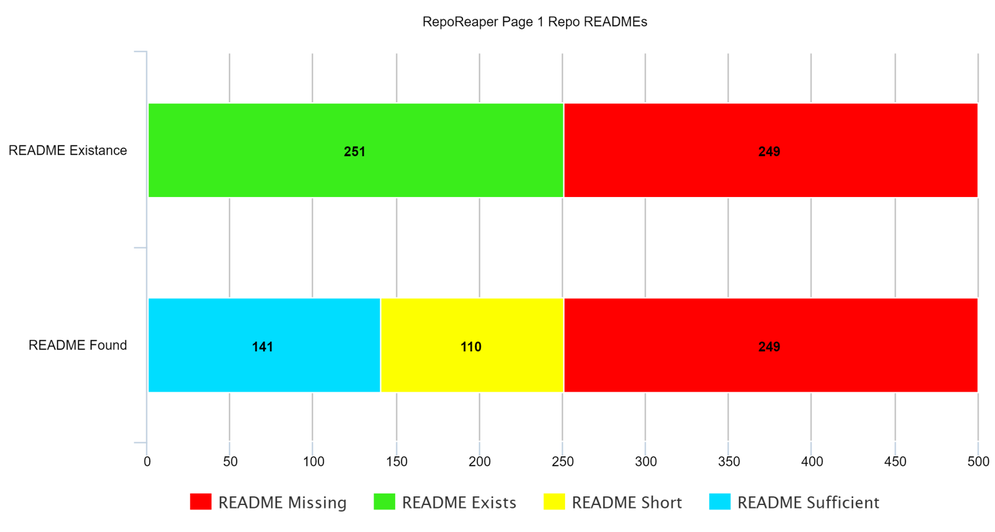
**Dyno** is pure-library implementation of what's also known as Rust trait objects, Go interfaces, Haskell type classes, and virtual concepts. Under the hood, it uses a C++ technique known as type erasure, which is the idea behind `std::any`, `std::function` and many other useful types.

```
#include <dyno.hpp>
#include <iostream>
using namespace dyno::literals;

// Define the interface of something that can be drawn
struct Drawable : decltype(dyno::requires_(
  "draw"_s = dyno::method<void (std::ostream&) const>
)) { };

// Define how concrete types can fulfill that interface
template <typename T>
auto const dyno::default_concept_map<Drawable, T> = dyno::make_concept_map(
  "draw"_s = [](T const& self, std::ostream& out) { self.draw(out); }
);
```

## Generated Summaries:

**Naive Sentence:** dyno runtime polymorphism done right
disclaimer
at this point, this library is experimental and it is a pure curiosity. no stability of interface or quality of implementation is guaranteed. use at your own risks. overview
dyno solves the problem of runtime polymorphism better than vanilla c++ does.
**Single README:** dyno solves the problem of runtime polymorphism better than vanilla c++
dyno is pure-library implementation of what's also known as rust trait objects, go interfaces, haskell type classes, and virtual concepts.
under the hood, it uses a c++ technique known as type erasure, which is the idea behind  and many other useful types.
**Multi-README:** for this, it includes the most popular data types, type classes and abstractions such as  to empower users to write pure fp apps and libraries built atop higher order abstractions. felix dynamix was initially developed as boost.mixin( but is now a separate library, that doesn't depend on the boost libraries collection.badge.release

## README Presence Sample




## Introduction

- The sample GitHub repository to the left[1] can be difficult to understand for developers not entirely familiar with the all of the terminology used.
- This section represents less than half of the entire README's contents.
- A tool that could summarise GitHub README's would be an invaluable tool for any developer looking to utilise an external library.
- Using TextRank[2] document sentences can be ranked in importance

## Methodology

- The RepoReapers[3] database of repositories is used as test data
- 2 automatic summarisation methods are proposed
  Single README Summary: Summarises the contents of the repository's README file
  Multi-README Summary: Finds "similar" repositories based on GitHub topics & keyword extraction,
- These methods were then compared with a naïve method of extracting the first 4 sentences of the repository & a manual method where a human summarises the repository
- These summaries are then scored in a blind test by an independent party

## Key Findings

- While Human Summarisation performed perfectly in all cases, Single README & Naïve Sentence techniques trailed closely behind with modal ratings of 4/5
- Using TextRank, no statistical difference could be determined between the performance of Singe README Summarisation & Naïve Sentence Summarisation
- This infers that many repositories contain most of their information in the first few sentence, reading further does not dramatically increase understanding of repository
- Many repositories do not include README files at all, those who do suffer from having very short README files that cannot be summarised
- The current implementation of Multi-README Summarisation does not perform as well as other techniques, there is evidence to suggest the differences in README structures can affect the performance of this method. Normalising README contents may pose a solution to this

## Conclusion

- A tool has been developed that provides summaries for GitHub repositories
- While the tool does not perform to the same standard as a human summary, most repositories with more than 250 characters of non-formatted plaintext are successfully summarised
- This opens up future work in summarisation of technical documents as well as summarisation of README's using similar repositories
- A benchmark for performance comparison of summarisation methods has also been developed
- Investigation into supervised learning techniques could also increase the effectiveness of the methods

## Score Distribution per Method


References
[1] ldionne,"Dyno", https://github.com/ldionne/dyno
[2] R. Mihalcea and P. Tarau, “Textrank: Bringing order into texts,” 07 2004.
[3] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan,
“Curating github for engineered software projects” 01 2016.

A repository of the project is available here: