Translator

Parth Thosani
North Carolina State University
Raleigh, North Carolina, USA
pthosan@ncsu.edu

Nisarg Shah
North Carolina State University
Raleigh, North Carolina, USA
nsshah5@ncsu.edu

Neha Agarwal
North Carolina State University
Raleigh, North Carolina, USA
nagarwa3@ncsu.edu

Simran Bosamiya
North Carolina State University
Raleigh, North Carolina, USA
sbosami@ncsu.edu

Parth Jinturkar
North Carolina State University
Raleigh, North Carolina, USA
pjintur@ncsu.edu

ABSTRACT

A linguistic translator is a tool that will convert a particular text or words into the desired language. We see that commonly we need to lookup to Google to find the translation of our desired texts. Our concept of translator is not restricted to just translation of words into a language but also finding the synonyms, pronouns of it for better understanding and use. Our project will also talk about the issues that users might face while watching a video but not in their desired language. Our aim is to convert the speech to text and later into the preferred language given by the user. We bundle and resolve these issues and put it in the form of a Chrome extension so that user might be able to get the desired language within seconds!

KEYWORDS

Translator, Chrome Extension, Best practices, Software Engineering

ACM Reference Format:
Parth Thosani, Nisarg Shah, Neha Agarwal, Simran Bosamiya, and Parth Jinturkar. 2018. Translator. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/1122445.1122456

1 SHORT RELEASE CYCLES

With short release cycles, new code is immediately integrated into a stable release which makes it possible to bring in even fundamental changes with minimal disruption. It eases the developers to have flexibility of testing their premature code and being able to fix any issues or bugs in the code with the next version release. In our project we have practices implementing the short release cycles. With every new feature added, a commit was made, thus aligning with short release cycles practice of Linux Kernel. Test cases were routinely executed and a commit was made with every new feature added, issues were posted frequently and resolved.

2 ZERO INTERNAL BOUNDARIES

Zero internal boundaries enables anyone to make any changes in any part of the project. This practice ensures that problems are fixed where they originate rather than making way for multiple workarounds. It also gives the developers a wider view of the project as a whole. Issues were reported by any team member who noticed it and an issue was created wherein the discussions followed and later it was worked upon. We have used a virtual environment to ensure everyone is using the same package. Initially, all the features were distributed among the 5 contributors to this project. Later, as and when an issue or an improvement was found by a contributor, it was worked upon by that contributor thus member working across multiple places in the code base.

3 THE NO-REGRESSION RULE

A No-regression rule means that if a code base works on a specific setting, all the subsequent code merges must work there too. Doing short releases and maintaining a consensus-oriented model is one step closer to a no regression rule. This rule tells us that any changes or upgrades made to the kernel should not break the system which allows for development of new feature with the stability maintained. The files CONTRIBUTING.md lists coding standards and lots of tips on how to extend the system without screwing things up.

4 CONSENSUS ORIENTED MODEL

The consensus-oriented model states that a proposed change cannot be integrated into the code base as long as a respected developer is opposed to it i.e. it needs a prior approval before it can be accepted as a new addition to the project. Different issues were being assigned to each and every member accordingly and after approval of all the members in the team it was committed and the issue was closed. All the discussions were being carried out in the comments section of the issue, and were moderated by other members before the issue was closed. We all used a virtual environment to ensure that all the contributors are using the same package. Since, all contributors worked on different parts of the project simultaneously, the config file (manifest.json) was updated as and when required by that contributor.

5 DISTRIBUTED DEVELOPMENT MODEL

Initially, all the changes made to the Linux Kernel's code base were sent to a single person for review and integration. However, this soon proved to be cumbersome, as it is impossible for a single person to be able to keep up with something as complex and varied as an operating system kernel. This means assigning different portions of the kernel (such as networking, wireless, device drivers, etc.) to different individuals, based on their familiarity with the area in turn enabling seamless code review and integration across the thousands of areas in the kernel without any compromise in kernel stability. All the features were distributed among 5 contributors according to their expertise in that area. The contributor who worked on a particular feature committed that.

6 TOOLS MATTER

Kernel development struggled to scale until the advent of the Bit-Keeper. The switch to Git brought about another leap forward. Without the right tools, a project like the kernel would simply be unable to function without collapsing under its own weight. We all used a virtual environment to ensure that all the contributors are using the same package. We have used Git as our version control system to maintain the project and newer features. Visual Studio Code was used as the integrated development environment throughout the project.

7 CORPORATE PARTICIPATION IN THE PROCESS

We would not have the fast-moving project described here without it. But it is also important that no single company dominates kernel development. While any company can improve the kernel for its specific needs, no company can drive development in directions that hurt the others or restrict what the kernel can do.This ensures that the project is available and can be extended by other companies. Our project is available on Github and can be extended by anyone who is interested. We have provided documentation support for developers ensuring clear understanding in order to help them contribute.

REFERENCES

[1] UNESCO, (2012). Why Language Matters for the Millennium Development Goals, United Nations Educational, Scientific and Cultural Organisation Bangkok Asia and Pacific Regional Bureau for Education. ISBN 978-92-9223-387-7

[2] United Nations General Assembly, (2012). Study on the role of languages and culture in the promotion and protection of the rights and identity of indigenous peoples, Expert Mechanism on the Rights of Indigenous Peoples, Fifth Session, A/HRC/EMRIP/2012/3.

[3]https://cloud.google.com/translate/docs