Introducing Debtgrep, a Tool for Fighting Technical Debt in Base Station Software

Svante Arvedahl
Ericsson AB BNEW DNEP 4G5G
Stockholm, Sweden
svante.arvedahl@ericsson.com

ABSTRACT

One of the biggest challenges in a large software development project is to manage technical debt. For example, if an API is replaced by a newer version and the old one is deprecated, there is a big risk that a substantial technical debt is incurred. The time where two versions of an API can co-exist can be quite long, and in a big organization like Ericsson 4G5G Baseband there can be a big challenge to prevent developers from continuing to use deprecated API's. At Ericsson 4G5G Baseband we invented the tool *debtgrep* to be incorporated in our Continuous Integration machinery. The tool will prevent any new code to be added using deprecated API's, or other configured keywords. The tool has proven very useful in the development of a new product based on a new software architecture. Debtgrep is used for deprecated API's, enforcing design rules, and enforcing architecture rules. The paper describes the behavior and configuration possibilities of the tool, and how it is used at Ericsson 4G5G Baseband.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**;
• **Computer systems organization** → Embedded software;

KEYWORDS

technical debt, static analysis, software cost

ACM Reference Format:
Svante Arvedahl. 2018. Introducing Debtgrep, a Tool for Fighting Technical Debt in Base Station Software. In *TechDebt '18: TechDebt '18: International Conference on Technical Debt , May 27–28, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3194164.3194183

1 INTRODUCTION

The 4G LTE baseband software program at Ericsson is developed at various development sites spanning multiple continents. With a large number of developers and about 8,000,000 lines of code it is one of the bigger software projects in the telecom industry. Moreover we have a software product line that needs to support multiple product and hardware variants. A software project easily accumulates technical debt and need arises to pay interest on the debt, in line with the technical debt expression coined by W. Cunningham [1]. There is much to save in terms of R&D headcount, server capacity, development infrastructure, feature time to market, etc., if you succeed well in managing technical debt.

As the code base has grown in size, and the number of developers and sites have grown in numbers, the challenge in communicating design rules, best practices, API deprecation, platform deprecation etc., has increased, and is facing similar challenges as ultra-large-scale (ULS) systems, as explained by L. Northrop in e.g. her ICSE 2013 keynote [3].

To mitigate the communication problem, and prevent debt that is not intentional (using a term from technical debt research [2]) from growing we developed the tool *debtgrep*. Debtgrep will issue warnings and errors whenever a forbidden character sequence is used in source code, textual model artifacts, or whatever textual artifacts the design base consists of.

2 DEBTGREP CONFIGURATION

To explain how debtgrep works we start with the configuration. The debtgrep config file contains keywords that are deprecated or in other ways not recommended, or even strictly forbidden in the source code; hereafter referred to as forbidden keywords. Listing 1 contains an example from deprecation in libavcodec [4]. The *KEYWORDS* section contains the forbidden keywords as a perl regex pattern. The *MESSAGE* section defines a descriptive message that should provide an explanation to the developer of what the problem is, and what the counter action is. The *TARGETS* section contains a list of file paths that should be searched, specified as perl regex patterns. The *SKIP* section contains file paths that should not be searched. Any file matching the TARGETS patterns will be searched for the keywords, unless it matches an expression in the SKIP section. For every keyword match in the targeted files an error will be output. The *EXCEPTIONS* section contains file paths for which a warning should be output instead of an error.

Listing 1: Debtgrep Keyword Configuration

```
- KEYWORDS:
    - coded_frame
  MESSAGE: 'coded_frame' is deprecated. Use the quality
       factor packet side data instead
  TARGETS:
    - \.[ch]
  EXCEPTIONS:
    - foo/src/foo\.c
    - foo/src/bar\.c
  SKIP:
    -/oldLibrary/
```


TechDebt '18, May 27–28, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5713-5/18/05.
https://doi.org/10.1145/3194164.3194183

3 API DEPRECATION

Debtgrep was first developed to simplify the retirement of deprecated API's. Normal C compilers provide the ability to issue deprecation warnings when declarations annotated as deprecated are used. If the usage of the deprecated API is extensive a compilation will yield a large amount of warnings. There is a big risk that developers don't even notice that new deprecation warnings are generated from new code submissions. With the exception mechanism in debtgrep we can, by means of exception expressions, configure the legacy code to issue warnings for the deprecated usage, while new files added in new submissions will generate errors. To facilitate continuous integration we demand that all code is free from errors. Gating delivery tests, using debtgrep, assure that it is not possible to deliver code violating the API deprecation.

4 EXAMPLE USAGE OF DEBTGREP

As soon as we had deployed debtgrep for preventing deprecated API usage, we realized the big potential of the tool to prevent other kinds of mostly invisible [2] technical debt. Our code base consists of C/C++ and assembly code, but also UML models, configuration files in proprietary format, xml, perl, python and various other formats. The fact that the tool is language independent, and all configuration is done by regular expressions, makes it very powerful. At Ericsson 4G5G Baseband we currently use debtgrep for:

- Dependency directions: making sure that lower layers contain no references to higher layers according to the architecture rules. This is acheived by enumerating excerpts of forbidden file paths in the test for the lower layer.
- Software product line rules: making sure software variants contain no illegal references to other variants. When variant specific code is located in variant named directories, this can be achieved by enumerating forbidden variants.
- Managing lint suppressions: making sure important lint checks are not disabled in the source code. This is achieved by searching for lint style suppressions.
- Debug trace verbosity levels: making sure high intensity debug traces are not put on low frequent verbosity levels. This is very application dependent, but in our case debtgrep searches for parts of debug trace macro names.
- Interface integrity: making sure that private interfaces are not used as public interfaces. This is achieved by configuring debtgrep to search for interface paths that do not comply with the naming conventions for public interfaces.
- Restricting API's: making sure unrecommended library API's are not used. If a library function is not recommended to use debtgrep is configured to search for its name.
- Configuration options: making sure unrecommended configuration is not used. Debtgrep can catch both configuration in text files, and command line options from tool invocations in scripts.
- Naming conventions: making sure e.g. variable names have the right upper/lower case style according to the design rules.

5 PROJECT EXPERIENCE

Although developers sometimes express frustration over that the debtgrep tool caught an issue in their git commit, preventing them from delivery, the overall acceptance in the organization has been positive. The tool was introduced early 2015 and has now been in use in our large scale software projects for three years. It is hard to measure the payback, but we estimate that we save at least 5 full-time developer positions, in development/maintenance effort with debtgrep. The importance of the tool is growing, which is an important indicator of the tool's merit. We have relied heavily on the tool in finding architecture rule violations, when we have introduced a new software architecture for a new product portfolio. The exception configuration also provides:

- opportunity to track the amount of the technical debt, possibly grouped by product domain,
- the ability to use different developer teams to extinguish different kinds of debt, since the debt expressions are already grouped by KEYWORDS, or
- use domains teams to extinguish the debt in one product domain.

6 RELATED WORK

The tool Code Climate contains a grep function that was introduced in April 2017 [5]. It has similar *KEYWORDS*, *TARGETS* and *MESSAGE* configuration as debtgrep. It lacks the *EXCEPTIONS* and *SKIP* configuration, but it adds a severity classification.

Using *todo* comments and *grep* for tracking technical debt has been suggested in a blog by P. Bourgau [6], also in April 2017.

7 CONCLUSIONS

In this paper we introduce debtgrep, a tool to prevent dependency violations, violation of naming conventions, usage of deprecated API's, and other kinds of mostly invisible technical debt, from growing. We have given some specific examples of use cases for debtgrep, and presented the experience from Ericsson 4G5G Baseband.

ACKNOWLEDGMENTS

The author would like to thank Alexandru Gosoiu and Anders Ruberg at Ericsson AB BNEW DNEP 4G5G for taking lead in the implementation of debtgrep, at that time named *depgrep*.

REFERENCES

[1] Cunningham, W. "The WyCash Portfolio Management System", *OOPSLA'92 Experience Report.*

[2] Kruchten et al. "Technical Debt in Software Development: from Metaphor to Theory Report on the Third International Workshop on Managing Technical Debt", *ACM SIGSOFT Software Engineering Notes, pp. 36-38, September 2012, Volume 37, Number 5.*

[3] Northrop, L. "Does Scale Really Matter? Ultra-Large-Scale Systems Seven Years after the Study (Keynote)" in *35th Int'l Conference on Software Engineering (ICSE 2013)*, San Francisco, CA, USA, 2013.

[4] GPAC issue on Github, *build warning: in newer versions of libavcodec, AVCodecContext.coded_frame has become deprecated*, https://github.com/gpac/gpac/issues/782, Accessed: 2018-02-10.

[5] Diggs, G. "Let's Talk About Grep!", *Code Climate Blog*, April 11, 2017, https://codeclimate.com/changelog/58ecfa297705a149790008b2, Accessed: 2018-02-11.

[6] Bourgau, P. "A Seamless Way to Keep Track of Technical Debt in Your Source Code", *Philippe Bourgau's blog*, April 12, 2017, http://philippe.bourgau.net/a-seamless-way-to-keep-track-of-technical-debt-in-your-source-code, Accessed: 2018-02-11.