INHERENT QUALITY METRICS FOR CONTINUOUS SOFTWARE QUALITY ENHANCEMENT

Ning Luo and Linlin Zhang

Visual Computing Group,
Intel Asia-Pacific Research & Development Ltd, Shanghai, China

ABSTRACT.

Traditional software quality metrics based on bug number and pass rate can only provide us afterthought post validation & product release. In this paper, we propose some new inherent software quality metrics for proactive quality control during development phase, including Lines of Code (LOC#), Cyclomatic Complexity and Code Churn. In this paper, citing one ultra-large-scale software - Intel Media Driver as one example, we introduce the reason to choose those metrics, our experience on leveraging those metrics to improve the software quality, and our turn-key solution for automatic metrics data collection and analysis. We expect the identified metrics can help more researchers to form the corresponding research agendas and the experiences sharing can help following practitioners to apply similar enhancements.

KEYWORDS

Perceived Software Quality Metrics, Inherent Software Quality Metrics

1. INTRODUCTION

Traditional software quality metrics based on bug number and pass rate can only provide us afterthought post validation & product release. In this paper, we propose some new inherent software quality metrics for proactive quality control during development phase, including Lines of Code (LOC#), Cyclomatic Complexity and Code Churn. In this paper, citing one ultra-large-scale software - Intel Media Driver as one example, we introduce the reason to choose those metrics, our experience on leveraging those metrics to improve the software quality, and our turn-key solution for automatic metrics data collection and analysis.

2. GAPS ON CONVENTIONAL SOFTWARE QUALITY METRICS

Figure 1. Perceived Software Quality and Inherent Software Quality

DOI: 10.5121/ijsea.2022.13302

Traditional software quality metrics mainly comes from the project execution and can be measured by bug number, pass rate, development velocity and project execution predictability. As those metrics can be easily perceived during project execution, we would call them Perceived Driver quality.

The above perceived quality metrics usually can only provide afterthought post validation & product release, so we do need some other quality metrics for proactive quality control and measurement during development or even design phase.

Intel Media Driver is an ultra-large-scale platform software with around 3 million lines of code and supported by over 300 developers. As the bridge between Intel GPU (graphics processing unit) and the ever-changing end to end media usages, Intel media driver is designed for multiple generations' Intel GPU support on top of different OS and API. It is widely used in diverse media usages ranging from client to cloud, against different software stacks. Every year, it has over one hundred software releases for different purposes.

Based on two years' investigation against over 10000 commits for Intel Media driver, we would propose another three inherent software quality metrics: Lines of code (LOC#), code complexity and code churn.

Lines of code (LOC#) can measure the size/scale of the software and has positive correlation with the software development and maintenance effort.

Cyclomatic complexity is a quantitative measurement to the number of linearly independent code paths in one software module. It can be directly correlated to the software cohesion and the number of defects per module. Meanwhile it also decides the number of test cases required to achieve sufficient coverage for a particular module.

Code churn can demonstrate the software evolution rate and is also a good predictor of post-release defects. Code churn can also take different forms, code change breakdown to each module provides us an indicator to the coupling between different modules while effective code change percentage is a hint to the potential fire in future project execution.

The three inherent quality metrics have been used to drive the driver refactoring for Intel Media driver for more than one year. The trend for those metrics can help us better identify the potential gaps in our design and then guide us on the corresponding quality improvement.

3. METRICS DRIVEN MEDIA DRIVER QUALITY IMPROVEMENT

In intel media team, now those inherent driver quality metrics are mainly used to drive our problem-solving oriented driver refactoring. Here are several examples:

3.1. Platform Specific Code Increase

Intel Media driver suffers a lot from the big hardware change generation by generation, so we got big concern about its impact on the scale of platform specific code.

Here we can get some clue from LOC# and cyclomatic complexity data.

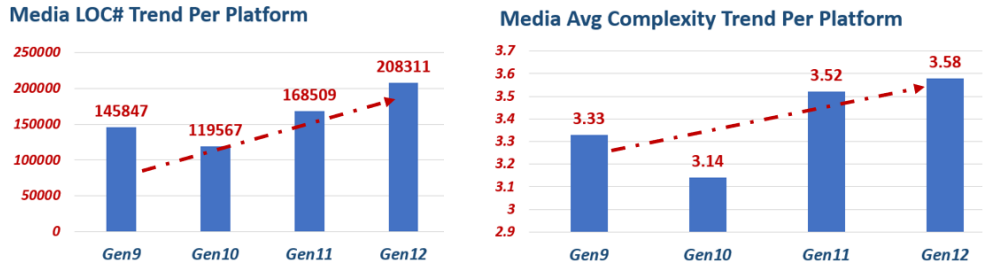

Figure 2. Media Driver LOC# & Cyclomatic Complexity Trend for different generations (before refactoring)

As can be shown by Figure 2, both LOC# and cyclomatic complexity show an upward tendency generation by generation for platform specific code which means bigger and bigger maintenance effort for our future platforms.

To solve the gap on LOC#, we refactored our driver to better decouple the domain specific logic (mainly related to industry codec standard) from hardware specific logic (mainly related to the Intel GPU implementation details) so that more domain specific logic can be better reused between different platforms. In that way, we can greatly shrink the platform specific lines of code by 55%.

To solve the gap on Cyclomatic Complexity, we start to enforce stricter policy for cyclomatic complexity control during our refactoring and intentionally deliver some refactoring to those functions with cyclomatic complexity above the threshold (>10). So currently the cyclomatic complexity for platform specific code has been greatly reduced by three times.

3.2. Coupling between OS Agonistic layer and OS Dependent Layer

Intel media driver need support multiple OSes, so the coupling between OS agnostic layer (HAL, hardware abstraction layer) and OS dependent layer (MOS, media OS, the media driver UMD interface to KMD& hardware) is another focus to us, especially after we found several customer escapes are all related to some MOS changes.

Coupling level can be extracted from the code churn data.

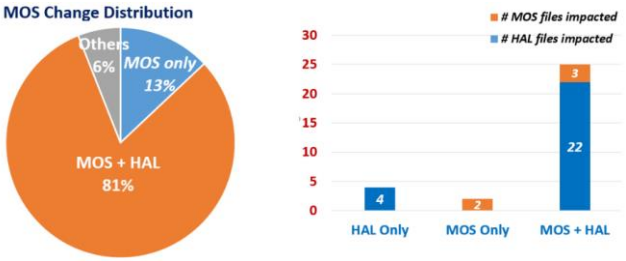

Figure 3. One-year Code Churn Analysis for Media Driver OS dependent Layer (MOS) (before refactoring)

As can be shown in figures 3, more than 80% MOS changes in last year need touch HAL simultaneously while when the code change including both HAL and MOS, it tends to be much

larger than the HAL only and MOS only one. All those data proved the tight coupling between our OS agnostic layer and OS dependent layer.

To solve the problem, we added another shim layer to centralize those HAL to MOS accesses. In that way, now the reference count to OS dependent MOS in OS agonistic HAL is dramatically reduced by 99%, so we tend to believe the OS agonistic layer and OS dependent layer are basically decoupled.

3.3. Reduce the ineffective code change percentage

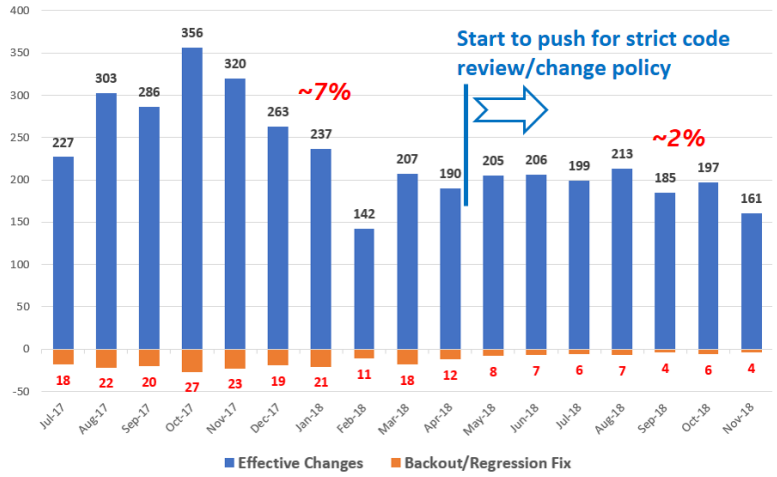

Figure 4. Media Drier Effective Commit Percentage Trend in last 18 months

Ineffective commit percentage shows the percentage of the passive commits (backout, regression fix, etc) inside our total commits. It is an important indicator to the quality of our code change and as well as the developer self-check.

As can be shown by figures 4, in Q4'17 and Q1'18, the ineffective commit percentage for Intel media driver used to be up to 7% which will impact not only the driver quality but also the development efficiency.

So we try to make some difference with stricter code change and review policy by requiring all code changes to be less than 500 lines and it does help to improve our code change & code review quality, now the ineffective commit percentage has been greatly reduced to ~2%.

So far, the above improvement on inherent quality metrics still cannot be directly reflected in our perceived quality metrics (bug number, pass rate and development velocity, etc), but we believe finally all inherent quality metrics improvements will and must contribute to the perceived quality.

So now we are working on one enhanced quality tracking system with solid connection between inherent quality metrics and perceived quality metrics which will help us better leverage those inherent quality metrics for future driver quality improvement.

4. TURN-KEY SOLUTION TO TRACK THE INHERENT DRIVER QUALITY METRICS

To better track the inherent quality metrics trend, we need on one turn-key solution for automatic metrics data capturing and analysis which should be automated, visualized and cross system.

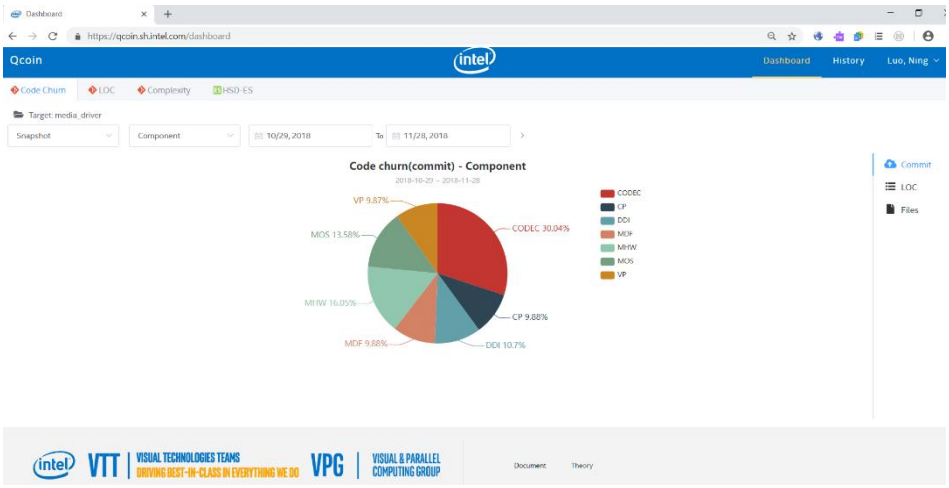

Figure 5. Visualized Dashboard for Media Driver Quality Metrics

Figure 5 above shows Visualized Dashboard for Media Driver Quality Metrics we have now. It can regularly poll the metrics data from our code repository system (Git/Github), deliver the corresponding post-analysis, and then provide the user one visualized dashboard for both snapshot and trend, plus the metrics breakdown by sub-components, platforms and OSes.

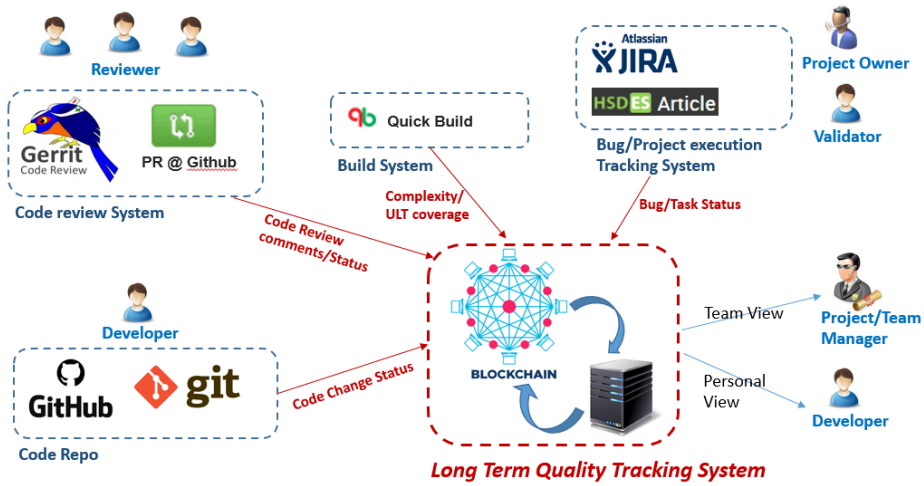

Figure 6. Long Term Quality Tracking System based on Cross-system Data Mining

Next step, we plan to set up a long-term quality tracking system against cross system data mining between code repository (git/github), code review system(gerrit), build system (Quick build) and Bug system (Jira), as can be shown by Figure 6 above. It can help us build the solid connection between inherent quality metrics and perceived quality metrics and directly demonstrate how the

inherent quality improvement affect the bug trend / development velocity during project execution.

5. CONCLUSION

Based on our two years' investigation, we believe Inherent Software Quality Metrics, in the form of lines of code, cyclomatic complexity and code churn, is a beneficial supplement to the traditional Perceived Quality Metrics like pass rate and bug number.

For intel media driver, the inherent driver quality metrics has been used to drive our problem-solving oriented driver refactoring for more than one year and got great achievements. Next step we need build the solid connection between inherent Quality metrics and perceived quality metrics to ensure the improvement on the former can finally benefit to the latter. We expect the identified metrics can help more researchers to form the corresponding research agendas and the experiences sharing can help following practitioners to apply similar enhancements.

ACKNOWLEDGEMENTS

Thanks to all colleagues working on refactoring for continuous software delivery and competency improvement. Appreciate your hard work to turn all our good designs into the reality.

REFERENCES

[1] Wikipedia, "Lines of Code", https://en.wikipedia.org/wiki/Source_lines_of_code
[2] Wikipedia, "Cyclomatic Complexity", https://en.wikipedia.org/wiki/Cyclomatic_complexity
[3] "Code churn". Available: https://codescene.io/docs/guides/technical/code-churn.html
[4] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, "A case study to Modern Code Review", available: https://sback.it/publications/icse2018seip.pdf
[5] Luo Ning, Xiong Andy, Towards Maintainable Platform Software - Delivery Cost Control in Continuous Software Development, SEA 2022 (The 11th International Conference on Software Engineering and Applications)
[6] Luo Ning, Xiong Andy, Enhanced Software Design for Boosted Continuous Software Delivery, International Journal of Software Engineering & Applications (IJSEA)

AUTHORS

Ning Luo is the senior software architect at Intel. His research interests include software requirements and architecture, continuous delivery, DevOps, and software product lines.

Linlin Zhang is a senior software engineer at Intel.