Maintainable Production

A Model of Developer Productivity Based on Source Code Contributions

Michael Olivari
michael_olivari@gmail.com
Student Number: 11784873

August 26, 2018, 106 pages

UvA Supervisor:	dr. Ana Oprescu, a.m.oprescu@uva.nl
Host Supervisor:	dr. Magiel Bruntink, m.bruntink@sig.eu
Host organisation:	Software Improvement Group, https://sig.eu

Universiteit van Amsterdam
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Master Software Engineering
http://www.software-engineering-amsterdam.nl

Contents

Abstract

Software productivity is often seen as the key metric to a software products success. Project attributes such as project scope and roll-out estimation, personnel and resource allocation, efficiency and performance improvements, and the role of the system within the business depend on being able to measure the overall effectiveness and added business value a developer contributes over the life cycle of the system. However a truly expressive model for measuring developer productivity has eluded the engineering community time and time again, as there is no standard definition or methodology by which we can measure developer productivity. Furthermore, a single holistic measure of developer productivity is unfeasible due to the abundance of factors which influence the efficiency at which a developer adds business value to a system.

This research focuses on creating a model of productivity which expresses the interplay between software maintainability and individual developer productivity. This measure of productivity, which we name Maintainable Production, is measured as the change in system maintainability "produced" by the technical contribution in code of a developer to a software system. In this manner, we only capture a singular aspect of productivity but one that is clearly defined and is measured by a model that is robust and adaptable to the needs of a given software project while aiding in the reduction of rework and technical debt a developer incurs on the system as they contribute to it. Through observation by domain-experts, this model was found to be most beneficial in providing developers actionable feedback to move a system from unsustainable to maintainable in an efficient manner while tracking the amount to which a developer over-engineers code contributions past a point of satisfactory compliance, referred to as "gold-plated" code. This findings of this experiment provides the foundation for moving forward with a full validation of the metric.

Chapter 1

Introduction

Software productivity is often seen as the key metric to a software product's success. Various attributes within the development phase depend on being able to measure the overall effectiveness and added functionality a developer contributes over the life cycle of the system. These attributes include (but are not limited to): project scope and roll-out estimation, personnel and resource allocation, efficiency and performance improvements, and the place of the system within the business / organization.[20] Going a step further, we can say that efficient measurement of productivity aids in efficient management of a project. This means establishing the metrics needed to measure productivity is not only of high relevance to the software engineering community, but to any business which incorporates an IT solution into their organization.

However a truly expressive model for measuring developer productivity has eluded the engineering community time and time again. One issue comes from the idea of ”productivity” itself, as there is no standardized definition of productivity within the software engineering field. The selection of metrics to utilize when expressing this measure of effort to output depends on the definition chosen for productivity, which is highly subjective and varies across the industry.[15][12] Before we can move to constructing a standard model for this aspect of software development, a definition of productivity which can be clearly measured must be provided.

Rudimentary metrics such as lines of code (LOC)[31] and cost estimates (CoCoMo)[16] have been employed to measure developer effort, however these metrics are void of any relation to the quality of the code being produced from a developer. Such simple metrics on their own can be easily gamed by developers to misrepresent their true impact to the overall development of the system, further diminishing the credibility of the metric as proper measures of productivity. Apart from continuously observing a developer's work manner (soft facts) while simultaneously auditing and tracking their output (hard facts), there has yet to be an established manner in which we can express the productivity of an individual developer based solely on their technical contribution to a software project. There is also the issue of maintainability of contributions and how ”good” the quality of a developer's contribution is in ensuring that the rework needed as the system evolves is minimized.

This research focuses on creating a model of productivity which expresses the interplay between software quality and individual developer productivity. This measure of productivity, referred to henceforth as Maintainable Production (MP), is measured as the change in system maintainability ”produced” by an individual developer through their technical contributions in code to the system. By establishing a point or threshold at which the system is satisfactorily maintainable, we can measure the distance to which the developer is from this point and provide feedback which encourages or discourages developers to produce more maintainable code depending on if they have yet to reach this point or have already exceeded it. In this manner, we only capture a singular aspect of productivity but one that is clearly defined and is measured by a model that is robust and adaptable to the needs of a given software project while potentially reducing the amount of rework and technical debt a developer incurs on the system as they contribute to it. We refer to this model as the Maintainable Production Model.

1.1 Problem analysis

The software productivity problem lies in that software development does not translate directly to industrial manufacturing, where productivity measurements are trivially obtained and used to direct the development process. Software engineers are multifaceted and are required to perform a various amount of tasks, both technical and non-technical, creating a highly fragmented work style.[30] In addition to this high fragmentation, those tasks which are non-technical in nature (collaboration, analyzing documentation, searching for preexisting solutions, etc.) are highly contributory to the development of a system yet are not easily measured in the effort needed to complete them, nor the impact / added value they impart to the development of the system. This results in a holistic measure of developer productivity being near impossible to express in a single metric which can be measured via static analysis. However, by measuring clearly defined individual aspects of developer productivity, namely Maintainable Production in the case of this research, it may be possible to bring developer productivity into an actionable perspective.

1.1.1 Motivating Example: SIG and The Better Code Hub

This research was conducted in conjunction with Software Improvement Group (SIG), and it is within this organization that the case study over the application of the Maintainable Production model was conducted. SIG is an Amsterdam-based software consultancy firm whose focus is in the improvement of software systems via deep analysis of source code. While they do provide non-automated audit services to analyze an organizations software processes for areas of improvement (such as in the case of productivity), a metric of productivity which can be easily measured from source code is highly desired. Such a metric could potentially allow for faster and cheaper analysis of a system and provide quantifiable data which they can present to clients when auditing products, services, and development processes.

Currently, SIG has many products which assist them in analyzing client systems, both for the general-public and private customers. Better Code Hub (BCH)[1] is one such product offered publicly which allows developers and teams to analyze the maintainability of their software system through a collection of 10 software quality guidelines which map to the ISO 25010 standard of software quality, namely the maintainability aspect[25].

Better Code Hub offers the ability to be integrated into the development pipeline of a project, allowing for each contribution added to a software project to be analyzed automatically and in turn provides a full detailed report over the project's current maintainability according to the 10 software guidelines addressed in the tool. Better Code Hub relies solely on the data contained within a GitHub repository to conduct its analysis, so every measurement employed in the tool is automated with minimal input needed from the user.

As it stands, the current tool can successfully provide an expressive measure of the overall maintainability of a system from multiple raw measurements over the source code. The tool also provides candidate units for refactoring when applicable and displays the expected net change to the given maintainability metric of the system once these issues are resolved. This makes the tool highly useful for gauging the quality compliance of a system and the amount of future maintenance that may be required.

The BCH team at Software Improvement Group wish to extend the functionality of Better Code Hub by implementing a developer profile analyzer, which would provide a detailed view over each contributor to a given project. The intention of the developer profile is to provide actionable feedback to the developer over their quality of work done per contribution to a project, as well as some measure of productivity relating to the maintainability of the system as it evolves due to contributions from the developer. The idea would be to create a feedback loop which could potentially incentivize developers to optimize their maintainable code production up to a threshold of compliance, deemed by SIG as a "Definition of Done". This clearly defines the need for a metric of productivity which is easily automated, not easily gamed, and expresses the quality of the product delivered rather than the volume of work done over time.

1.2 Aim

Our goal is to provide a new definition of developer productivity which expresses the interplay between the capacity of a developers coding output and the quality of their code production. This new metric, referred to as Maintainable Production, will be derived from a proposed model of maintainable productivity based on automated static analysis measures of source code. The primary goal of this research is to construct a model in which we can measure the continued impact of a developer's contribution to a software system with respect to the overall maintainability of that system, with a secondary goal of constructing a feedback mechanism in which developers can review the production data that is output from this model in a manner that is actionable and easily understood.

In this iteration of the Maintainable Production model, Better Code Hub is utilized as the primary tool for measuring project maintainability. The model itself does not rely on Better Code Hub, in fact any maintainability assessment model could potentially be used to measure the quality change of a software project as contributions are made to it. Better Code Hub was chosen specifically for this iteration as it provides some specific characteristics which make it ideal, specifically the guideline based analysis which highlights individual, easy to measure aspects of code as well as a defined point of "done-ness" with respect to maintainability for each specified guideline. These aspects allow for a robust model of maintainability change to be constructed, which is a fundamental aspect to the Maintainable Production model later detailed in this research.

1.2.1 Research Questions

This research can be divided into two parts. In the first part we aim to construct the Maintainable Production model, utilizing Better Code Hub as the vehicle in which we assess a project's quality to measure how much maintainability is produced per developer's contribution. The second part of the research focuses on validating the model within a laboratory environment, in which we create a feedback mechanism for the model and test its output for comprehensibility and actionability with domain experts.

Part I

Research Question 1: By what measures can we evaluate a developer's code contribution to a project as maintainable code production?

Research Question 2: By what means can we evaluate the impact of a developer's maintainable code production to the maintainability of a given project?

Part II

Research Question 3: Can we provide feedback over maintainable code production at the commit level which is comprehensive?

3.1 Is the feedback coherent for developers?

3.2 Is the feedback actionable for developers?

1.3 Research Method

The research method we employ through this study is Action Research, specifically Technical Action Research (TAR).[39] Action Research is the process in which a researcher implements new methodology into existing strategy as it is the only way to study the affect of the changes. More formally, Action Research is "an approach in which the action researcher and a client collaborate in the diagnoses of a problem and in the development of a solution based on the diagnosis."[18]

The ultimate goal of the researcher in Technical Action Research is to "develop an artifact for use in a class of situations imagined by the researcher."[39] Our approach is artifact-driven, in the sense that we are introducing a new artifact, namely the Maintainable Production model, into an existing model, in this case Better Code Hub, in order to measure its affect in a given practice or setting. As we can not conduct a full validation of the model due to time and resource limitations, the setting in which we will conduct this research is in an internal lab environment with domain experts. The affect we wish to measure is the comprehensibility of the model from experts in maintainability and the affect of development actionability from the feedback it provides.

This can be viewed as the first step in Technical Action Research, in which the artifact is tested under ideal conditions before being scaled up to more practical application with more realistic problems. In doing so, we do not promise a full validation within this research but rather the crucial first step in the TAR process. Future work will be able to build on the findings presented through this initial laboratory testing to determine more concrete applications of the Maintainable Production model.

1.4 Contributions

The research presented in this work contributes the following:

– Methodology in which we can measure the change in software maintainability between individual contributions of a developer.

– A model for assessing the production of maintainable code.

– A mechanism for providing feedback to developers with respect to their maintainable code production.

– Insight into the manner in which developers utilize feedback from software quality analysis tools, specifically with respect to comprehensibility and actionability of that feedback.

1.5 Outline

This work is divided into 7 chapters. In this introductory chapter, Chapter 1, the research context, problem analysis, methodology overview and research methodology are stated. Chapter 2 addresses the current knowledge of our problem domain, the required background knowledge to construct the model, and related work. Chapter 3 details the construction of the Maintainable Production model and the specifics therein, along with a discussion on particular design decisions taken into account. The process over the construction of the feedback mechanism of this iteration of the Maintainable Production model along with the process in which the feedback is validated under the first TAR phase is detailed in Chapter 4. The results of the experimentation along with further discussion over the findings is presented in Chapter 5. Finally, we present our conclusion in Chapter 6, together with proposals for future work using our findings from this research.

Chapter 2

Background

In this chapter, we present all the relevant background knowledge needed for our research - the relationship between general productivity and software productivity, currently used measures of software productivity, software maintainability, and Better Code Hub.

2.1 Software Productivity

With respect to economics, productivity is the ratio between the amount of goods or services produced (output) and the resources needed to produce them (input).[9] Extending this definition to the domain of software engineering, one could define software productivity as the ratio between the software produced to the labor and resource expense of producing that functionality. While this is a suitable definition in theory, when applied in industry there is much debate over what this definition entails. This is mostly due in part to debate over what is truly intended to be produced by a developer, and in part debate over how can we effectively measure the soft factors of software development (work fragmentation, staff turnover, etc.) in conjunction with the hard facts of development (lines of code produced, functionality produced, number of commits, time-scale, etc.).

Card et al. address the motivation for finding a true measure of development productivity and the difficulties therein.[19] The position taken is that there is no single productivity measure which applies in all situations for all purposes, and emphasizes the need to define productivity measures which are appropriate for process and information needs. Card provides a breakdown of productivity into various categories:

– **Physical Productivity** - The size of the product (typical in LOC) versus effort to grow the volume (typically a time scale, e.g. Man-Hours / Person-Hours)

– **Economic Productivity** - Business value of the product (in currency) versus cost in resources to develop it (also typically a time scale, although other factors such as number of developers have been employed)

– **Functional Productivity** - Functionality provided by the software (expressed often in Function Points) versus the effort to provide that functionality (again, typically a time scale in Man-Hours / Person-Hours).

A primary focus of this research is defining an addition to this breakdown, namely Maintainable Production. Using the accepted definition of productivity, the ratio of production output divided by the resources input to produce this output, we can further define Maintainable Production as the change in maintainability of a system versus the resources expended by a developer to produce this change. More simply, this metric would measure the code production of a developer which is deemed maintainable with respect to the ISO/IEC 25010:2011 standard for software quality, specifically the maintainability aspect[25].

In order to measure the overall net change in maintainability between two snapshots, or builds, of the system, we will utilize Better Code Hub with its 10 guidelines of maintainable code. The technical effort to produce this change is measured as the percentage of the system, in LOC, which was churned between these two snapshots. In this manner we differ from the other aspects Card presents as the input is no longer a time-scale or developer count, but rather the actual code the developer is producing. In this way, the measurement can be expressed as the change in quality per single line of code a given developer produces, and can be easily automated simply through the analysis of version control data.

2.2 Common Measures of Software Productivity

There have been many proposed methods over the years employed to measure software productivity. The most common methods of measuring have been: Source Lines of Code (SLOC) volume, Constructive Cost Modeling, and Function Point Analysis. In this section, background and discussion over each of these methods is provided.

2.2.1 Source Lines of Code (SLOC)

Lines of code is an inexpressive metric when attempting to measure the functionality of a system on their own. As well as not being the main deliverable to the software user, lines of code are highly variable when it comes to the production of functionality. A software constructed with a high-level terse language such as Python or Ruby will almost always consistent of less SLOC than another software constructed in C which provides identical functionality. Additionally, a good function is the one which provides the most functionality with the least amount of lines to ensure readability and comprehension. This makes SLOC by itself a poor metric as the amount you measure does not truly reflect how much utility value is provided.

That is not to say that SLOC is never a useful metric, even with respect to functionality. According to Rosenberg, the best uses of SLOC are when it is a covariate of other metrics as it is a key predictor in overall software quality.[34] If we take it as the input in a ratio against the output of the production (whether it is functionality or quality), we can gather a measure over how efficient was the production based on the contributed code by a developer. This can prove to be a successful sub-measure of productivity to express the efficiency impact of a developer's contribution.

Nguyen et al. provide another counterargument to SLOC being a poor metric for productivity. Nguyen states that a majority of systems are homogeneous in the tech employed by the organization, so individual measures of a developer's productivity within the organization based on SLOC is fair.[31] This idea makes it the most widely used metric when measuring cost and a base unit for other software quality measures. This stance agrees with the aforementioned position that SLOC cannot stand on its own, but is useful as a covariate to other metrics.

2.2.2 Constructive Cost Modeling (CoCoMo)

Constructive Cost Modeling is a top-down approach to measuring the effort needed to produce software. This approach attempts to provide accurate predictions over the work effort along with the development time required from the development teams to produce software based on some goals. The predictions are provided at 3 levels: basic, intermediate and detailed view. It uses SLOC and a series of cost attributes as inputs for its predictions. The cost attributes include known factors such as product goals, technology utilized, project attributes and personnel attributes.

The fundamental metric used as input for the CoCoMo approach is size, namely in thousands of lines of code. As such, it carries with it much of the same pitfalls as SLOC as a metric for measuring productivity due to its lack of functionality expressiveness. Where it excels over SLOC is the approach's ability to provide estimations on the effort required to satisfy system requirements (after careful analysis).[17] Like SLOC, it can be used as a measure for input in the productivity equation but not as a substitute for productivity itself, and is further weakened due to the difficulty in automating this cost measure.

2.2.3 Function Point Analysis

In an effort to measure the amount of functionality a segment of source code produces, A.J. Albrecht created a methodology to estimate the amount of "function" that is produced from code components. Function Point Analysis measures the amount of external inputs, external outputs, files, interfaces and inquiries to be used by the software. Each parameter is assigned a complexity rating from low to high, and are weighted based on a set of fourteen general system characteristics derived from the functional requirements of the system.[11] These characteristics are used to determine the added functional value of the system and include attributes such as performance, re-usability, accessibility, and usability.

SLOC is again used in function point calculation but in conjunction with more expressive metrics such as work hours and requirement specification fulfillment of the software. This makes it a very strong measure for determining the effort needed to produce some functionality, as well as the total amount of functionality a system provides. It places function as the ultimate measure of a software system.

According to Symons, the main drawback to this "holy grail" metric of productivity is that it cannot be automated and is difficult to implement in practice.[37] It requires that the requirements of the system to be clearly analyzed and evaluated for their functional value, which changes software to software. Often it is also the case that the fulfillment of a functional requirement is judged subjectively due to too generalized requirement documentation. It is also a very time consuming process and is difficult to implement within the SDLC.

2.3 Software Maintainability

In a general sense, maintainability is defined as "the ability of an item, under stated conditions of use, to be retained in or restored to a state in which it can perform its required functions"[7], or more simply "the ability to keep in a condition of good repair or efficiency". With respect to software engineering, we can say informally that a system is maintainable when it is easy to modify the system as it evolves. This definition is satisfactory in a broad sense, but not actionable nor concrete enough to construct a model by which we measure software maintainability.

A more concrete definition is the one provided by the International Organization of Standardization (ISO) together with the International Electrotechnical Commission (IEC) which defines software maintainability under ISO 25010 as "the degree of effectiveness and efficiency with which a software product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements."[25] Furthering this definition, ISO / IEC decompose software maintainability into five sub-characteristics:

– **Modularity** - Degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.

– **Reusability** - Degree to which an asset can be used in more than one system, or in building other assets.

– **Analysability** - Degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified.

– **Modifiability** - Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality.

– **Testability** - Degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met.

While there are many other definitions of software maintainability, this is the definition we refer to in this work as it is the current official international standard for software quality. This standard itself works as an overarching definition of what software maintainability is and clearly defines the various aspects which influence maintainability, however the standard itself offers no tangible methodology or thresholds for evaluating a software's maintainability. In order to properly measure the change in maintainability a system undergoes, we need a concrete model for which we can measure these sub-characteristics of maintainability to compose some score for which we can classify the system.

2.3.1 Measuring Maintainability

At it's foundation, Better Code Hub utilizes a software analysis toolkit built on SIG's own Maintainability Model. This model utilizes risk profile analysis to assess the quality of a software system using multiple static analysis measures such as: Cyclomatic Complexity, SLOC Volume, Clone Detection, Gini-Coeffecient similarity, fan-in / fan-out measures, and code smell detection.[23] Code elements are placed in bins determined by their maintainability risk to the system, with a typical breakdown of "low risk", "medium risk", "high risk", and "very high risk" labeling. Code elements belonging to the "low risk" bin are deemed compliant by SIG with respect to the ISO 25010 standard for maintainability, while code elements in the other bins are considered of poor maintainability by SIG and should be considered for refactoring.

The thresholds for each of these bins are determined through a large (>1000) historical, domain-expert evaluated bench-marking of software projects SIG have analyzed and provided their consultancy services over, and is re-evaluated on a yearly basis. While the individual elements found in the medium, high and very high risk bins are deemed non-compliant under the Maintainability Model, the risk profile over the measured aspect takes precedence in determining overall compliance under each metric implemented in the model. This creates robustness in the model and allows for a degree of tolerance during the evaluation of the metrics utilized.

Let us look at the unit complexity metric as an example. Unit complexity is measured as the Cyclomatic Complexity (CC) contained within the unit, or rather the total number of linearly independent paths through a given unit.[28] Suppose there is a system composed of 10 units of code. 1 unit possesses a Cyclomatic Complexity of 15 while the remaining 9 units possess a CC of 10 or less. SIG's Maintainability model would place the more complex unit (complexity of 15) within the "medium" risk bin with remaining 9 units placed in the "low risk" category. From this distribution of units, a risk profile is constructed as follows:

Table 2.1: Example Cyclomatic Complexity Risk Profile of 10 Unit System

Low Risk	Medium Risk	High Risk	Very High Risk
90%	10%	0%	0%

Utilizing the compliance thresholds observed in **Table 2.2**, this system would receive a "++" rating under the Maintainability Model even though 10% of the units contained within the system are of poor maintainability with moderate severity. This showcases the nuance and tolerance of the model when assessing a given measure.

Table 2.2: Complexity Risk Profile Assessment Schema extracted from A Practical Model for Measuring Maintainability[23]

	maximum relative LOC		
rank	medium	high	very high
++	25%	0%	0%
+	30%	5%	0%
o	40%	10%	0%
-	50%	15%	5%
–	-	-	-

2.4 Better Code Hub

With the SIG Maintainability Model at its foundation along with the 10 Guidelines detailed by Visser et al.[38], Better Code Hub is able to analyze a system under the maintainability standard put forth in ISO 25010. The 10 guidelines themselves are derived from the 5 sub-characteristics which compose the definition of maintainability found within the standard, and are all measured through the risk profile analysis mentioned in **Section 2.3** of this work. The tool itself requires minimal set-up, is accessible 24 hours of the day, and provides feedback which is objective, consistent, and efficient when measuring maintainability under each guideline.

Table 2.3: BCH Guidelines and their Compliance Criteria

Guideline	**Compliance Criteria**
Write Short Units	15 LOC per Unit
Write Simple Units	4 Branching Points per Unit (McCabe Complexity of 5)
Write Code Once	No Type 1-2 Code Clones larger than 6 LOC
Keep Unit Interfaces Small	4 Parameters per Unit
Separate Concerns Into Modules	Module Volume limit of 400 LOC
Couple Architecture Components Loosely	No cycles between classes, class components are contained
Keep Architecture Components Balanced	6-12 Top-level components compose the architecture
Keep Your Codebase Small	System volume is max 200,000 LOC
Automate Tests	Test Code Volume 50% of Production Code Volume
Write Clean Code	No dead code, uncalled code No code in comments, no todo comments No Improper Exception Handling

Better Code Hub also has the ability to be integrated into the development work-flow of a given project through GitHub. This allows the tool to analyze every push, commit and pull request in any branch associated with the project to give continuous feedback over the maintainability of the project in an automated fashion. By integrating directly into GitHub's CI environment of the project, feedback is given to the development team over the current status of the project's maintainability along with a link directly to the report on BCH's website. Through BCH's UI, developers are able to inspect the affect of their changes to the system over each of the 10 guidelines, with refactoring candidates being displayed for those guidelines which are non-compliant due to an abundance offensive code elements present within the system's guideline risk profile. In this way, the measuring of a system's maintainability can be completely automated and analysis can be done at the commit-level, allowing for a fine-grained review of the maintainability as the smallest code changes are made to a system.

Chapter 3

Constructing the Maintainable Production Model

In this chapter, we explain the methodology in which we construct the Maintainable Production Model. We start by first defining the research context of the model and an overview of the underlying formula of the model, followed by a more in-depth explanation over each component and idea employed to construct the model.

3.1 Research Context

Better Code Hub (BCH) was developed by SIG to provide an easy to use, online software analysis tool for developers to receive actionable feedback over the maintainability of the projects in which they are working on. SIG's own Maintainability Model[23] is employed in BCH to perform the analysis over a given system, and a list of refactoring candidates are then returned in order to assist developers in attaining a more maintainable system. BCH in its current form can only analyze snapshots of entire systems, and does not currently have the ability to give personal feedback to developers with regards to the quality of their individual contributions to a system.

The purpose of this research is to develop a model of productivity based on the manner of change in the maintainability score provided through BCH as developers contribute to a project. The hope is that if we can track the change in maintainability against a developer's contribution, we can track a production trend of maintainable code per developer. In this way, we can provide personalized feedback to developers with a hope of improving their ability to produce higher quality code without diminishing the efficiency of development by over-engineering the code.

3.2 From Productivity to Maintainable Production

Most accepted definitions of productivity state the metric to be a measure of the ratio of goods produced to the effort / resources consumed to produce said goods, or more simply output / input[9]. As expressed in Chapter 2 of this work, we explain why this ratio is difficult to express with respect to the development of software, primarily that the goods being produced are difficult to classify. Within the scope of this research we classify a subset of productivity as the change in system maintainability per developer contribution. This allows the basic formula of productivity to be used as we are able to identify the intended production, namely the change in maintainability as reported by BCH, as well as the resources needed for production, namely the volume of code change as a percentage of the overall system. In the simplest terms, we aim to measure the efficiency of maintainable code production. Thus we have our output (or rather the numerator) in the formula, the change in the maintainability score between snapshots of a system containing a single developer's contribution.

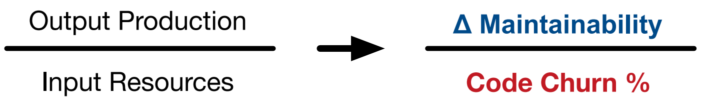


Figure 3.1: Mapping Productivity to Maintainable Production

3.3 Measuring a Developer's Contribution via Code Churn

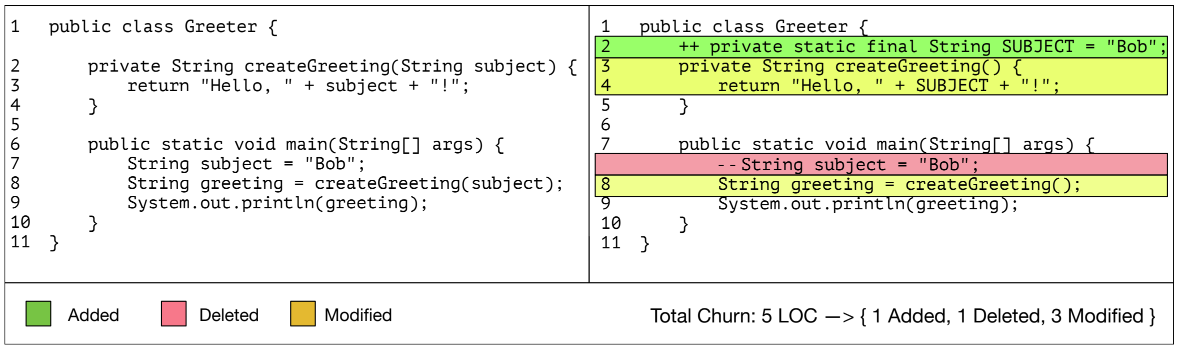
Let us first begin by identifying the resources consumed to produce this change in maintainability, namely what is the input / denominator of our formula. As we defined Maintainable Production to be the ratio of maintainability change to developer contribution we need a measure of what the amount of technical change the developer is contributing to the system. Typical productivity measures tend to use a time-based approach as the primary resource consumed, i.e. man-hours spent during development. The issue with such a measure is that it becomes very hard to determine how much actual time was spent solely on code production, maintainable or otherwise, on a per developer basis. This is primarily due to the high work fragmentation of developers which makes it difficult to holistically measure the amount of time a developer spends on a single task to produce a single output.[30]

A more fine-grained and automatable solution is to use the primary deliverable of the developer as the input, namely the amount of code change a developer introduces between analyses of the system. While this does not relate back to a timescale in the traditional sense of productivity, this in fact makes sense when we define Maintainable Production as the measure of quality code production per developer contribution which can be seen as a temporal moment in the system's evolution. In this sense, we remove the difficulty of measuring development time due to work fragmentation, specification review, communication with team members, or meetings and only view the technical contribution in code as the input. This makes sense in that the primary deliverable of the developer is indeed code, however the business value is derived from the function of the code being delivered, with maintainability being one of the primary factors that drives business value creation.[14][27] Therefore, we will use a measure of code churn, which we define in this case as the summation of the LOC added, LOC modified and LOC deleted as the input in the Maintainable Production formula.

$$LOC_{churned} = LOC_{added} + LOC_{deleted} + LOC_{modified} \tag{3.1}$$

LOC added is simply measured as the unique lines of code which were inserted into the source code of a project through a contribution. LOC deleted is then easily understood as the reverse: those lines of code which were removed from source code via the developer's contribution. LOC modified is a more nuanced measure as the metric takes into account existing code which has in-line modifications made during a contribution.

Let us consider the following example code snippet of two consecutive builds of a software system.



```
public class Greeter {

    private String createGreeting(String subject) {
        return "Hello, " + subject + "!";
    }

    public static void main(String[] args) {
        String subject = "Bob";
        String greeting = createGreeting(subject);
        System.out.println(greeting);
    }
}
```

```
public class Greeter {
    ++ private static final String SUBJECT = "Bob";
    private String createGreeting() {
        return "Hello, " + SUBJECT + "!";
    }

    public static void main(String[] args) {
        -- String subject = "Bob";
        String greeting = createGreeting();
        System.out.println(greeting);
    }
}
```

🟩 Added 🟥 Deleted 🟨 Modified

Total Churn: 5 LOC —> { 1 Added, 1 Deleted, 3 Modified }

Figure 3.2: Calculating Code Churn from Buld A (right) to Build B (left, includes contribution)

From this example, it is clear to see that a single line was added, a single line was deleted, while 3 lines of code were modified as a result of the developer's contribution to the system. This would result in an overall code churn of 5, assuming this is the only changes present in the developer's contribution.

3.3.1 Normalization of Code Churn Volume

Now that we have defined how to measure the raw volume contained within a developer's contribution we must normalize the churn as a percentage of the system. By normalizing the raw volume of the contribution's churn to the total volume of the analyzed system, we are able to mitigate issues of language dependence and are able to better interpret the overall size of the change with respect to the system. This also allows for robustness of the Maintainable Production formula, as we will better capture the relative affect of the code change on the maintainability of a system as it grows.

$$Code\ Churn\ \% = \frac{LOC_{churned}}{LOC_{system}} \tag{3.2}$$

Let us reconsider the churned example from **Figure 3.2**. Let us suppose this contribution was made on software project whose LOC_{system} is 100 LOC after including the contribution. As we calculated before, the $LOC_{churned}$ in this contribution was 5 LOC. This effectively means the developer churned 5% of the system as a result of their contribution.

What becomes quickly evident with this measure is how it scales as a system grows larger. Previous work has shown that the median commit size is roughly 20 LOC.[22][32] Suppose a typical commit is made to a system with a total volume of 100,000 LOC. The contained churn from the contribution would show roughly 0.0002% of change within the system due to this commit. By itself, this value only demonstrates how small of a technical change occurred to the system, but when used as the input value in the Maintainable Production formula, the total volume acts as a boosting agent to the $\Delta Maintainability$. Consider the following transformation of the formula for Maintainable Production:

$$\frac{\Delta Maintainability}{\left(\frac{LOC_{churned}}{LOC_{system}}\right)} \longrightarrow \frac{(\Delta Maintainability * LOC_{system})}{LOC_{churned}} \tag{3.3}$$

With this transformation, it is clear to see the effect of the system's volume on the $\Delta Maintainability$. This allows the formula to perform robustly by measuring the relative change in maintainability as the system grows. As the contained churn is inversely proportional with the overall volume, it is easier to have larger technical changes on smaller systems, and the reverse holding true for larger systems. However, if the purpose is to measure the average change per developer contribution regardless, this measure must scale with respect to the system. Therefore, we impose direct proportionality on the change in maintainability with the size of the analyzed system including the contributed code.

3.4 Measuring Change in Maintainability

As we've identified the input of the Maintainable Production formula, we must now define how to measure the change in maintainability. As introduced in the introduction of this work, we will be utilizing Better Code Hub to evaluate the overall maintainability of the system. Each of the 10 guidelines BCH is composed of individually measure some attribute of a software system's maintainability. As described in **Section 2.3.1** of this work, each guideline defines an optimal risk profile as its point of satisfactory compliance. Risk profiles are constructed by grouping code elements into labeled risk bins via a risk grading schema per guideline.

Compliance under a selected guideline is determined as a comparison between the real risk profile of the given system against this optimal risk profile, with each labeled risk bin in the optimal risk profile defining a threshold for compliance under the specific guideline. The thresholds per guideline define a single vector of compliance, which can be seen as a single point in a multi-dimensional space of which the guideline is composed.

Table 3.1: Guideline Metrics and Compliance Thresholds and Risk Schema, extracted from Building Maintainable Software [38]

Guideline	**Metric**	**Risk Schema**	**Compliance Thresholds**
Write Short Units	Unit Volume (in LOC)	**Low**: <= 15 **Medium**: 16 - 30 **High**: 31 - 60 **Very High**: > 60	**Low**: 56.3% **Medium**: 43.7% **High**: 22.3% **Very High**: 6.9%
Write Simple Units	McCabe Complexity (MC)	**Low**: < 5 MC **Medium**: 6 - 10 MC **High**: 11 - 25 MC **Very High**: > 25 MC	**Low**: 74.8% **Medium**: 25.2% **High**: 10.0% **Very High**: 1.5%
Write Code Once	Code Duplication	% Duplicated Code % Unique	**Duplicated**: 4.6% **Unique**: 95.4%
Keep Unit Interfaces Small	Unit Parameters	**Low**: <= 2 **Medium**: 3 - 4 **High**: 5 - 6 **Very High**: > 6	**Low**: 86.2% **Medium**: 13.8% **High**: 2.7% **Very High**: 0.7%
Separate Concerns Into Modules	Incoming Calls	**Low**: <= 10 Calls **Medium**: 11- 20 Calls **High**: 21 - 50 Calls **Very High**: > 50 Calls	**Low**: No Constraint **Medium**: 21.6% **High**: 13.8% **Very High**: 6.6%
Couple Architecture Components Loosely	External Dependency	% Hidden Code % Interface Code	**Hidden**: 85.8% **Interface**: 14.2%
Keep Architecture Components Balanced	Component Number and Size	Component Amount Size Uniformity	**Amount:** 2 - 12 **Uniformity**: < 0.71*
Keep Your Codebase Small	System Volume	Volume in Man-Years	**Volume:** < 20 Man-Years
Automate Tests	Test Volume vs Production Volume	% Test LOC against Production LOC	**Test LOC**: >= 50% Production
Write Clean Code	Clean Code vs Code Smells	% Clean Code % Code Smells	**Clean code**: 99% **Code Smells**: 1%

3.4.1 Pre-selection of Guidelines

While each of Better Code Hub's 10 guidelines measure some aspect of software maintainability as defined by ISO 25010, the scope and depth at which the measure is done varies per guideline. In fact, three distinct guideline categories can be identified based on the scope, or granularity of the measure, in which the guideline measures the system.

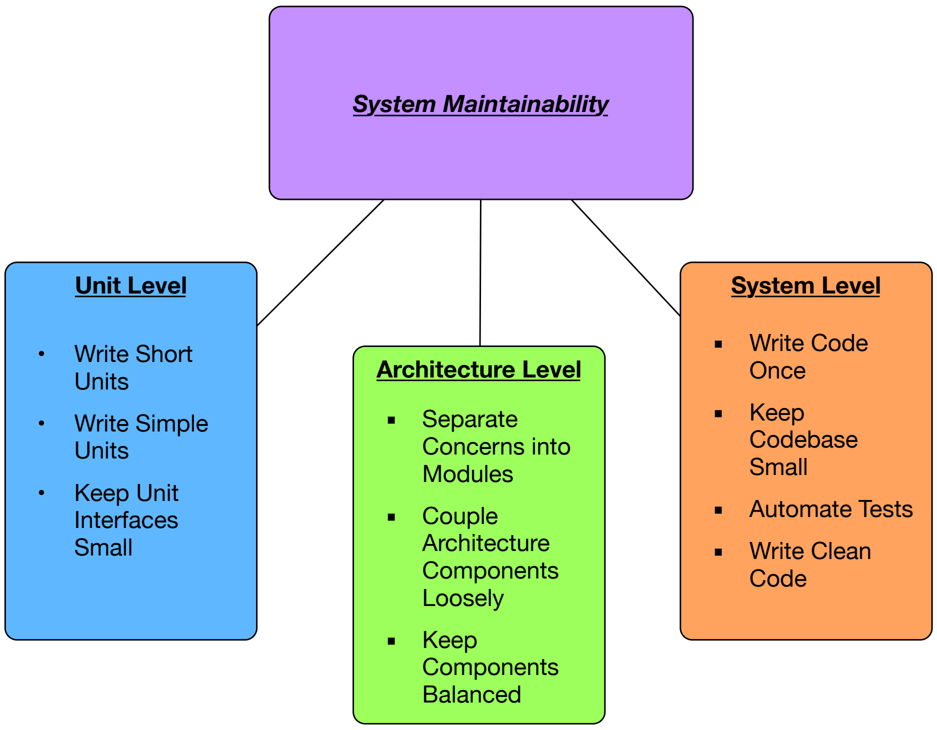


Figure 3.3: Guideline Categories by Scope

Scope is important in this sense as the Maintainable Production model measures the maintainability impact of individual contributions from a developer at the commit level. From a software versioning perspective, commits represent the smallest changes to a system built in a collaborative fashion. In this context, the overall affect to maintainability will be found within those guidelines that measure at the finest granularity. Coarse-grained measures will likely go unchanged when measuring at the commit level under typical development, in which a given commit touches a single file in 95% of cases and consists of roughly 20 LOC.[22]

The guidelines *Write Short Units, Write Simple Units* and *Keep Unit Interfaces Small* build risk profiles based on the risk categorization of individual units so naturally these guidelines possess a Unit-Level scope. We define a unit as the smallest collection of code lines which can be executed independently, i.e. a method, constructor or function in Java or other C-based languages. As units represent the smallest building block of a software system, these guidelines are accepted for use in the Maintainable Production model.

Within the Architecture-Level category are the guidelines which are concerned with the overall system structure and the interaction between the individual components of the system. A component is defined as a top-level division of a system, in which a grouping of related modules reside, with modules being grouping of related units i.e. a class, interface or enum in C-based languages. Here, the guidelines *Separate Concerns into Modules, Couple Architecture Components Loosely* and *Keep Components Balanced* construct risk profiles based on the risk categorization of components and component modules, which is a coarser-grained scoping as opposed to that found in the Unit Level measures. As these guidelines do not fit the fine-grained approach of the Maintainable Production model, we will not be including these measures for the purpose of this research. The reasoning for this exclusion is the assumption that a single commit from a developer will not alter these guidelines to any degree of significance and could ultimately skew the model's output if included.

The more general System Level guidelines employ volume measures, in LOC, of specific system attributes and properties. While the guideline *Keep Codebase Small* and *Automate Tests* simply measure either total LOC of the system to determine compliance based on the volume of code employed by the system in the former or the ratio of test code to production code in the latter, *Write Code Once* and *Write Clean Code* build risk profiles associated with the amount of code as a percentage of total production code relating to some system property, namely the ratio of duplicated code to total production code for the former and the ratio of code smells to clean production code in the latter. As each of these measures offer the fine-grained measure needed for the MP Model, as individual LOC is finer than even the Unit-Level guidelines, we accept these guidelines as well.

After filtering based on the scope at which the guidelines measure attributes of maintainability, we must also consider the robustness, or rather the susceptibility to extreme change to compliance, of the measures involved per accepted guideline. In the case of the Unit Level guidelines, the risk profile assessment as described in **Chapter 2.3** of this work showcases the tolerant nature of these guidelines in that there is tolerance for some level of offending code as defined by the compliance thresholds associated with a given guideline.

However, the volume-based measures found in the System Level guidelines do not afford the same attribute of tolerance. While *Write Code Once* and *Write Clean Code* both employ similar, simpler risk profiles to those found in the Unit Level guidelines, *Keep Your Codebase Small* only records a binary check if the system volume is smaller than 20 Man-Years (roughly 200 000 LOC) as its measure of compliance. Likewise, *Automate Tests* measures only if the *amount* of test code is equal to or greater than 50% of production code in the system, without an actual measure over the test coverage of the code base. Both of these guidelines measure compliance in a purely binary fashion. The primary issue here is the ease in which a developer could misrepresent their true affect on maintainability under these two volume-based guidelines.

As an example, let's say there is a Build A of a system which has reached satisfactory compliance under each selected guideline and currently has a volume of 199,995 LOC. Now, suppose a developer contributes a new feature to the system contained within a 20 LOC commit which maintained compliance under each of the guidelines except *Keep Your Codebase Small* as it exceeding the hard cut threshold of 20 Man-Years. This would in turn result in a less than optimal change to the overall maintainability due to this developer's contribution, simply because the system was close to the threshold of compliance. It would not be fair to penalize the developer for this commit, as it was typical in size and maintained compliance otherwise.

As a further example, let us consider a similar system that is compliant under all guidelines except *Automate Tests* due to possessing only enough test code volume that is equivalent 40% of the total production code. As the *Automate Tests* guideline measures pure volume rather than linked coverage of tests units to production units, this means the developer can simply contribute additional, non-functioning or overly verbose code lines in the tests to satisfy this measure. This is not an accurate reflection on the amount of maintainability was generated through this developer's contribution, however they would be rewarded if the model includes such a measure. **Figure 3.4** showcases the selection process and final guidelines to be included.

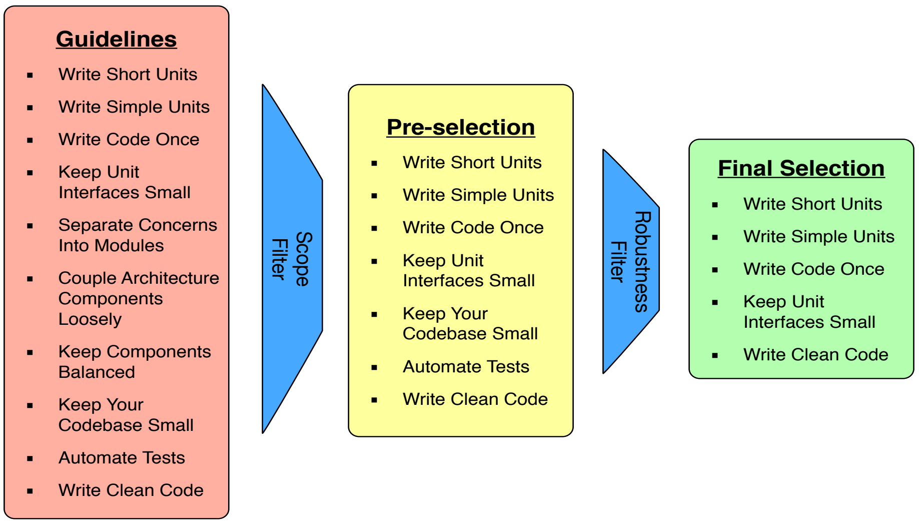


Figure 3.4: Guideline Selection Process

3.4.2 Measuring Compliance Distance

Now that we have a selection of guidelines with the appropriate granularity and robustness, the next step in the construction of the maintainability model is developing a process for measuring the change in maintainability between two builds of a given system. As each guideline provides a clearly defined point of satisfactory compliance, we can determine how far a given system is from being compliant / being past the point of satisfactory compliance by measuring the distance *to* this optimal risk profile point *from* the observed risk profile per guideline. A distance of zero (0) indicates the system has met it's compliance needs and is maintainable and future-proof under the given guideline, defined as the "Definition of Done" (DoD) for the observed guideline.

Table 3.2: Selected Guidelines' Compliance Vectors (Risk Profile percentages converted to decimals)

Guideline	**Definition of Done / Compliance Vector**
Write Short Units	[0.563, 0.437, 0.223, 0.069]
Write Simple Units	[0.748, 0.252, 0.10, 0.015]
Write Code Once	[0.954, 0.046]
Keep Unit Interfaces Small	[0.862, 0.138, 0.027, 0.007]
Write Clean Code	[0.99, 0.01]

An important aspect to note when determining compliance is the nature of each threshold. The first value in the vector, for example 0.563 for *Write Short Units* or 0.954 for *Write Code Once*, is the "low risk" category which contains the percentage of code elements which are deemed maintainable through BCH's analysis. This is a minimum threshold, meaning a system can only be compliant under this guideline if the percentage of "low risk" code elements observed is greater than or equal to this threshold. Conversely, the remaining values in the compliance vector behave as maximum thresholds, in that each bin either corresponds to a single amount of non-compliant code elements, namely *Write Code Once* and *Write Clean Code*, or the "medium", "high", and "very high" risk categories of code elements with varying severity, namely *Write Short Units, Write Simple Units* and *Keep Unit Interfaces Small.*

Additionally, for those guidelines with variable risk severity thresholds, the thresholds themselves are cumulative starting from the most severe (very high risk) to least severe (medium risk). This means that every very high risk code element also is classified under the high risk category as well as the medium risk category.

$$\begin{aligned}
&T_{VeryHigh} = T_{VeryHigh} \\
&T_{High} = T_{High} + T_{VeryHigh} \\
&T_{Medium} = T_{Medium} + T_{High} + T_{VeryHigh} \\
&T_{Low} = T_{Low}
\end{aligned} \tag{3.4}$$

While it is possible to convert these values to non-cumulative measures, for the sake of measuring the distance to compliance there is no difference if the values are cumulative or non-cumulative and would require additional processing to BCH's analysis output so for the purpose of this research we leave them as they are.

As each guideline's DoD point is a multi-dimensional vector of compliance thresholds, ranging from 2 dimensions (*Write Code Once, Write Clean Code*) and 4 dimensions (*Write Short Units, Write Simple Units, Keep Unit Interfaces Small*), we can utilize the Euclidean distance between the observed risk profile of each build of the system to the DoD vector per guideline. Euclidean distance represents the magnitude of the straight line or "as the crow flies" distance between two points in a common space.[21] The following equation details the formula of the Euclidean distance between a give point a and another point b. The points a and b are representative of individual vector points within a common space, each consisting n number of dimensions.

$$\begin{aligned} D(a,b) &= \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + ... + (b_n - a_n)^2} \\ &= \sqrt{\sum_{i=1}^{n} (b_i - a_i)^2} \end{aligned} \tag{3.5}$$

In the context of measuring the distance to compliance, a represents the compliance vector defined by a selected guideline while b represents the vector of the observed risk profile obtained after analysis. The number of dimensions corresponds to the dimensionality of the risk profile assessment per guideline, as observed in **Table 3.2**. As an example, let us consider a system which was just analyzed under the *Write Short Units*. We want to measure the distance to compliance under this guideline, and we have an observed risk profile of [0.400, 0.500, 0.050, 0.050] which will be used as input a in the Euclidean distance equation. The compliance vector under this guideline is [0.563, 0.437, 0.223, 0.069], and will be used as input b. The dimensionality n of the common space is 4.

$$\begin{aligned} D(a,b) &= \sqrt{(0.400 - 0.563)^2 + (0.500 - 0.437)^2 + (0.050 - 0.223)^2 + (0.050 - 0.069)^2} \\ &= \sqrt{(-0.163)^2 + (0.063)^2 + (-0.173)^2 + (-0.019)^2} \\ &= \sqrt{(0.026569) + (0.003969) + (0.029929) + (0.000361)} \\ &= \sqrt{0.060828} \\ &= 0.24663 \end{aligned} \tag{3.6}$$

Now let us consider a second build, one representing the same system with an addition contributed by a developer. Analyzing the same *Write Short Units* guideline, let's say BCH outputs an observed risk profile of [0.500, 0.450, 0.050, 0.000] as our a input. With the same compliance vector as input b and the same dimensionality n, we can calculate the compliance distance for this second build as follows:

$$\begin{aligned} D(a,b) &= \sqrt{(0.500 - 0.563)^2 + (0.450 - 0.437)^2 + (0.050 - 0.223)^2 + (0.000 - 0.069)^2} \\ &= 0.19705 \end{aligned} \tag{3.7}$$

By simply comparing this observed risk profile with that found in the previous example, we see that the both the "medium" risk and "very high" risk values lowered while the "low" risk value increased. This would suggest an overall increase of maintainability under this guideline, and in fact we can confirm that by the new lower distance of 0.19705 as compared to the previous distance of 0.24663. In order to determine precisely how much closer to compliance, or rather how much maintainability change accrued due to this new contribution, we simply measure the distance delta from the original build, p, to the new build containing the contribution, q.

$$\begin{aligned} \Delta D(p,q) &= D(p) - D(q) \\ &= 0.24663 - 0.19705 \\ &= 0.04958 \end{aligned} \tag{3.8}$$

From this example, we can clearly quantify the increase in maintainability for the given guideline due to the developer's contribution.

3.4.3 Determining Compliance and the Effect of Gold-Plating

Determining compliance over a guideline occurs simply by comparing the observed risk profile of a system against the compliance thresholds of the chosen guideline. Let us again consider the two builds in which we measured the Distance Delta for the guideline *Write Short Units*. The observed risk profile from the original build of the system was [0.400, 0.500, 0.050, 0.050], while the observed risk profile of the build containing the new contribution from the developer was [0.500, 0.450, 0.050, 0.000] and the compliance vector of the guideline is [0.563, 0.437, 0.223, 0.069]. By comparing each value found in the observed risk profile against the respective compliance threshold found in the compliance vector for *Write Short Units* for each build, we can determine if the build is compliant under the guideline. Again, we have to take into account the nature of each compliance threshold, i.e. is it a maximum or minimum threshold. The following algorithm demonstrates the rule-based approach to determining compliance:

Algorithm 1 Determining Guideline Compliance

```
1: function isGuidelineCompliant(ObservedRisks, ComplianceThresholds)
2:     observedLowRisk ← ObservedRisks[0]
3:     lowRiskThreshold ← ComplianceThresholds[0]
4:     if $observedLowRisk < lowRiskThreshold$ then return false
5:     $i \leftarrow 1$
6:     for $i <$ length of $ObservedRisks$ do
7:         if $ObservedRisks[i] > ComplianceThresholds[i]$ then return false
8:         $i \leftarrow i + 1$
9:     return true
```

The compliance of a given build determines the manner in which the Distance of the build to its DoD is processed. The previous example of the Distance Delta was calculated with two builds which were non-compliant. In this scenario, calculation of the distance delta remains unchanged as it fits the natural intuition of moving toward or away from compliance.

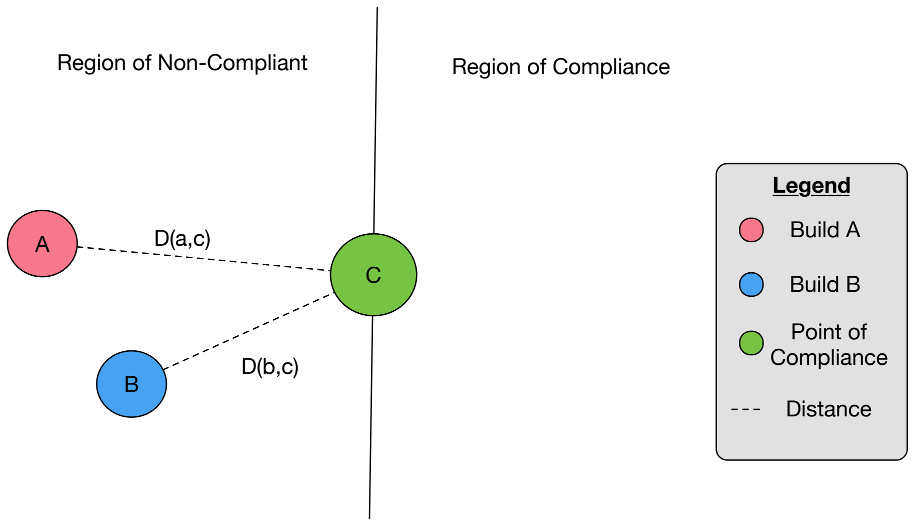


Figure 3.5: Change in Maintainability: Non-compliance to Non-compliance

If we assume *Build A* is the original build while Build B is the new build containing a developer's contribution, then $D(a, c) - D(b, c)$ evaluate to the relative change in maintainability for the given guideline, demonstrated here as a positive increase. Likewise, if we assume the reverse, then $D(b, c) - D(a, c)$ demonstrates a negative increase as it moves further from the point of compliance. In the even the distances are equal, $D(a, c) - D(b, c)$ evaluates to 0 indicating no change in maintainability as would be expected.

Let us consider another example in which the system moves from a non-compliant build, *Build A*, to a compliant build, *Build B*.

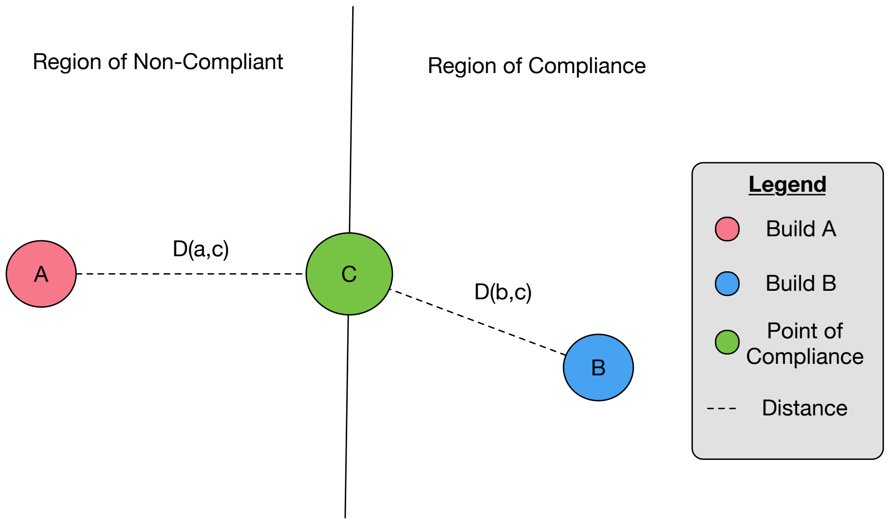


Figure 3.6: Change in Maintainability: Non-compliance to Compliance

What becomes evident in this scenario is the manner in which distance within the compliance region is evaluated the same as distance in the non-compliant region. Utilizing the original version of the Distance Delta equation, when observing the above simplified illustration of a system moving from *Build A* to *Build B* the calculation of $D(a, c) - D(b, c)$ would result in a substantially small value for the Distance Delta. In fact, if we assume the distances are of equal magnitude, then the calculation of $D(a, c) - D(b, c)$ would result in 0, falsely indicating no change in maintainability when in fact it is a complete flip in compliance which naturally is a large change in maintainability. In this scenario, we actually want to measure the entire span of the change in the guideline. This means altering not the Distance Delta equation itself, but rather the signage of the observed distances to reflect their relative compliance. We choose to equate distances for a compliant build as negative in value to non-compliant guidelines as a means of enforcing compliance polarity. This simply allows for the Distance Delta equation to calculate the total spanned distance over a guideline which experiences compliance switch over successive builds.

Let us assume p is a non-compliant build under the *Write Short Units* guideline while q is compliant. Then the Distance Delta equation becomes:

$$\begin{aligned} \Delta D(p, q) &= D(p) - (-D(q)) \\ &= D(p) + D(q) \end{aligned} \tag{3.9}$$

As an attempt to demonstrate how this alteration to the Distance Delta fits the intuition of the maintainability change caused by a compliance switch over a given guideline, let us consider system build-pair, *Build A* and *Build B*, with distance magnitudes of 0.4 and 0.6 respectively. *Build A* is a non-compliant build under the *Write Short Units* guideline, while *Build B* is in fact compliant, therefore the guideline has experienced a compliance switch. Under the original Distance Delta equation, the change in maintainability under this guideline would evaluate to $D(a, c) - D(b, c) = -0.2$, a negative change to maintainability when in fact the guideline went from non-compliant to compliant over the successive builds. However, with the revised Distance Delta calculation, the change would evaluate to $D(a, c) - (-D(b, c)) = 1.0$, indicating a relatively large increase in maintainability over this guideline which matches expectations. Likewise, if the system were to move in the exact opposite direction, *Build B* to *Build A*, this would also be reflected accurately under the revised Distance Delta equation. Under the original equation, the Distance Delta would evaluate as $D(b, c) - D(a, c) = 0.2$, which indicates a small positive increase in maintainability, however this is incorrect. Under the revised equation, $(-D(b,c)) - D(a,c) = -1.0$, indicating a relatively large decrease to overall maintainability which again matches intuition.

Let us now consider a final scenario, in which we encounter a system with two analyzed builds, *Build A* and *Build B*, both of which are compliant under a given guideline. In this scenario, we are most interested in detecting situations in which a developer is "leaking" productivity due to over-engineering aspects of the system which are already at a satisfactory state of compliance. We refer to this manner of development as "gold-plating", or rather the process of improving software maintainability past the point of diminishing returns, defined through Better Code Hub as the "Definition of Done" per guideline. In this context, a "gold-plated" contribution is measured in the same vein as deteriorating non-compliant code in that a penalty is incurred in the Distance Delta measure as you are moving further beyond the original point of compliance. Consider the example illustrated below in **Figure 3.7**.

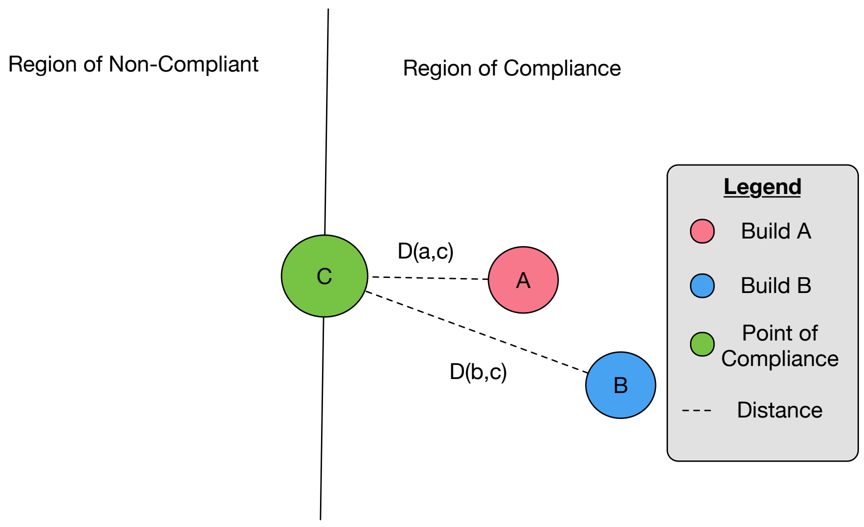


Figure 3.7: Change in Maintainability: Compliance to Compliance

Suppose a contribution is made to the system constructing again two builds, the original *Build A* and the newly contributed *Build B*. Let us say again that Build A has a distance to compliance of $D(a,c) = 0.4$, while Build B has a distance of $D(b,c) = 0.6$. Utilizing the original Distance Delta measure, we see that $D(a,c) - D(b,c) = -0.2$ while the real change shows an increase to the maintainability of this guideline. However with respect to gold-plating, this matches intuition as both builds were already past the point of satisfactory compliance, thus overall productivity suffered from this contribution as it was unnecessary from the maintainability perspective. In this context, the original Distance Delta is again relevant as a means of capturing the interplay developer productivity and system maintainability.

An interesting situation happens when moving in the opposite direction, however. Consider the same builds, A and B, with the same distance measures, $D(a,c) = 0.4$ and $D(b,c) = 0.6$. If the system moves from *Build B* to *Build A* due to a developer's contribution, intuitively we observe a degradation to the guideline's maintainability. However, according to the original Distance Delta equation, $D(b,c) - D(a,c) = 0.2$ indicates an overall positive affect to the guideline's maintainability. This is not a desired result, as do not wish to reward development which degrades compliant guidelines, but only penalize those which improve the guideline's past a satisfactory point of compliance.

In this scenario, we take the design decision to nullify the result of the Distance Delta and return 0. The rationale behind this decision is that priority is given to the fact that the guideline has maintained compliance, which is the ultimate goal of the developer in this situation. Therefore, $D(b,c) - D(a,c)$ will evaluate to 0 in order to provide feedback to the developer which is neutral regarding this contribution's affect on the guideline. This concludes all scenarios in which the Delta Distance equation is utilized in measuring maintainability change over a guideline.

Now that we have defined the rules in which an individual guideline's maintainability is measured over successive builds of a system, we must devise a method of aggregating each observed Distance Delta to derive an overall Maintainability Delta for a given system. The most intuitive manner is simply a summation of all selected guidelines' Distance Deltas to use in the Maintainable Production equation.

$$\begin{aligned}\Delta Maintainability(p,q) =& \Delta D_{ShortUnits}(p,q) + \Delta D_{SimpleUnits}(p,q) + \Delta D_{WriteCodeOnce}(p,q) \\ &+ \Delta D_{ShortUnitInterfaces}(p,q) + \Delta D_{WriteCleanCode}(p,q)\end{aligned} \tag{3.10}$$

By using this Maintainability Delta as the numerator, or rather value of production, in the Maintainable Production formula, we can now read the formula as the "average change in maintainability produced by a single contribution" for a given developer. In doing so we now have a full definition of the Maintainable Production metric.

$$\frac{\Delta Maintainability}{Code\ Churn\ \%} \longrightarrow \frac{(\sum_{i=1}^{n} \Delta D(x,c)_i * LOC_{system})}{LOC_{churned}} \tag{3.11}$$

3.5 Building a Benchmark of Maintainable Production

Now that we have a defined formula for measuring the Maintainable Production of a given contribution to a system, we must create an appropriate benchmark of the metric over real development data to understand that statistical significance of the measure. This also allows us to construct a classification methodology depending on the distribution of Maintainable Values over the benchmark analysis. The data over which this iteration of the MP Benchmark naturally comes from a snapshot of historical data from Better Code Hub.

3.5.1 Project Selection

We began by selecting the top analyzed 50 projects in Better Code Hub, as well as the Better Code Hub project itself. The specific criteria for project selection is as follows:

- **Development Integration** - Projects must have had the BCH continuous integration webhook enabled over GitHub to ensure each contribution to the project received an analysis result.
- **Number of Analyses** - The project should have been analyzed a minimum of 50 times.
- **Latest Release** - The project must have been analyzed under the most recent release of Better Code Hub for completeness of data.
- **Open-Source / SIG Accessible** - Not all measures utilized in the Maintainable Production formula are captured by BCH analysis itself, specifically LOC metrics for calculating churn. For these measures, we utilize GitHub's API to mine the selected project to measure these aspects. Unfortunately, without proper authentication credentials we are unable to collect data over closed-source projects outside of SIG's access rights.

Using this criteria, filtering on the 50 projects resulted in 39 usable system in the benchmark analysis, 40 in total including Better Code Hub. Of the 11 filtered projects, 9 were inaccessible closed-source projects while 2 were projects which had been deleted within the week the benchmark analysis was being conducted.

3.5.2 Commit Selection

After obtaining the list of 40 software systems came the process of selecting usable commit data from each project. Commits represent individual builds of the system at a given time. Each project underwent a sampling of the 50 most recent analyses performed by Better Code Hub, allowing for a potential of 2000 contributions to be measured and benchmarked. The following commit selection criteria was enforced to ensure analyzed commits were compatible through the Maintainable Production formula:

- **Commit-Pair Analyses** - Each potential commit as well as its former, or ”parent”, commit must be analyzed under Better Code Hub. This is to ensure the calculation of the Maintainability Delta is possible.
- **Successive Commits** - A commit-pair must have a distance no greater than 1 in the version control tree. This is to ensure the appropriate granularity of the Maintainable Production measure as the ultimate goal is to measure the impact of a single contribution to the system. Aggregation of commits is possible, and possibly desired for future work, however this aspect is not the focus of this research.
- **Commit Volume** - Analyzed commits must have a volume greater than 1 LOC to ensure compatibility with the Maintainable Production equation. Commits without churn are therefore filtered out.

After applying the commit selection criteria over the sampled commits from each project, we were left with a total of 824 usable commits out of the 2000 (41.2% retention). Data collected from commits included: author, volume, commit SHA, parent commit SHA, and LOC change data. The overall churn each commit introduced was then measured and stored to by used in the maintainable production measurement. The SHA values of both the selected commit and its parent commit were used to map the commits back to the analysis results stored in BCH to calculate the Maintainability Delta over the commit-pair. Better Code Hub also provides the analyzed total volume of the system in the analysis results, which was used to scale the resulting Maintainable Production value to the system size as a means of normalization.

3.5.3 Benchmark Analysis

Plotting of the commit data measured under the MP equation, we observe the following distribution of Maintainable Production.

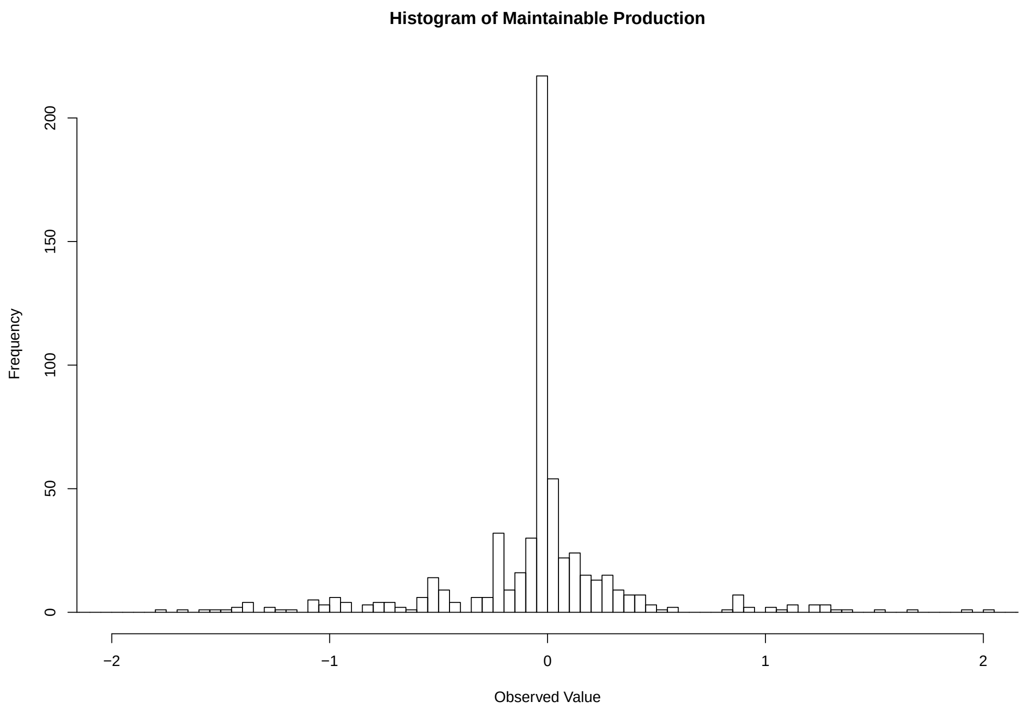


Figure 3.8: Histogram of Commits Measured via Maintainable Production (Excluding Extreme Outliers

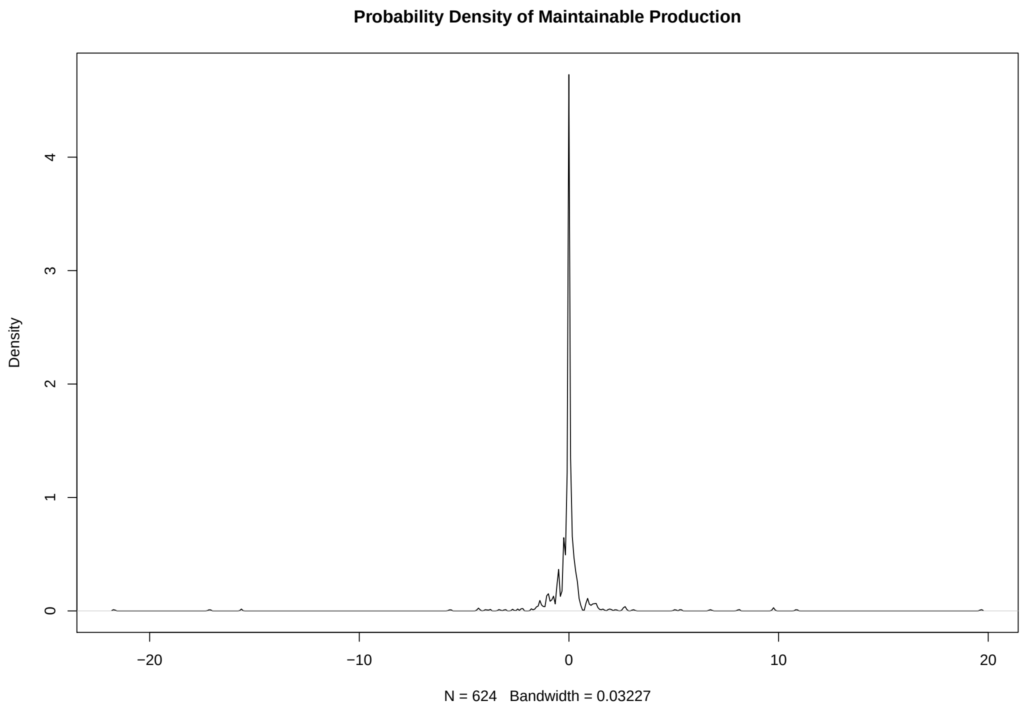


Figure 3.9: Probability Density of Maintainable Production

Table 3.3: Maintainable Production Summary

Attribute	Value
Maximum	19.6701
Minimum	-20.7111
Mean	-0.0415
Median	0
Standard Deviation	1.8664
Kurtosis	70.8835
Skewness	-1.8329

Observing this distribution of the Maintainable Production value over the commit data, we see a strongly leptokurtic normal distribution with a minor left-skewing. This distribution shows most single commits produce very little change in maintainability as analyzed through BCH, with a median measure of 0. As a means of understanding how the Maintainable Production was evaluated over this sampling, let us consider the distributions of the Code Churn and Maintainability Delta over the commit data.

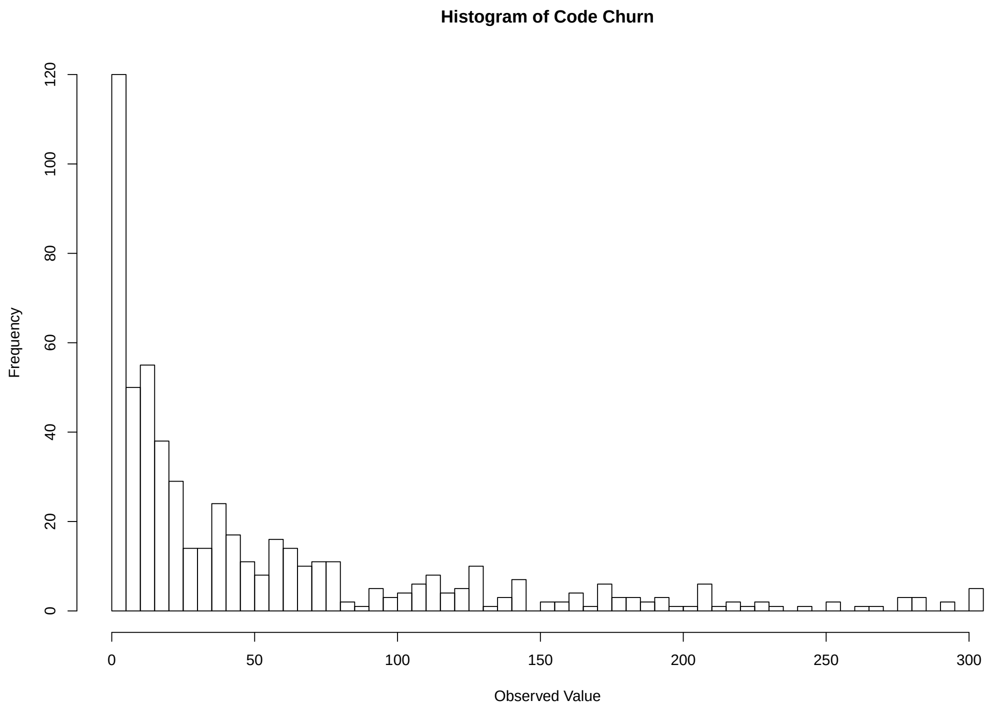


Figure 3.10: Code Churn Distribution Over Commit Data

Table 3.4: Code Churn Summary

Attribute	Value
Maximum	303.0
Minimum	1.0
Mean	53.82
Median	24.00
Standard Deviation	67.8488
Kurtosis	2.7885
Skewness	1.8064

Here we observe the distribution of code churn over the commit data. From our sampling, we see a median churn of 24 LOC which is slightly higher than the findings of previous work which analyzed the average size of a contribution.[22][24][32] Most interesting is the mean commits size of roughly 54 LOC, due in most part to a relatively high amount of large commits (> 50 LOC). As the divisor of the MP formula, a larger commit size diminishes the effective change in maintainability, thus the smaller the commit with the larger Maintainability Delta returns the highest Maintainable Production value. Of course the MP value is also scaled to the total size of the system, however as the system volume acts as a boosting agent this would only increase the magnitude of the value. This does not then fully explain why the majority of commits have a Maintainable Production of 0. In order to understand this phenomena better, we observe the following distribution of the Maintainability Delta over the commit data.

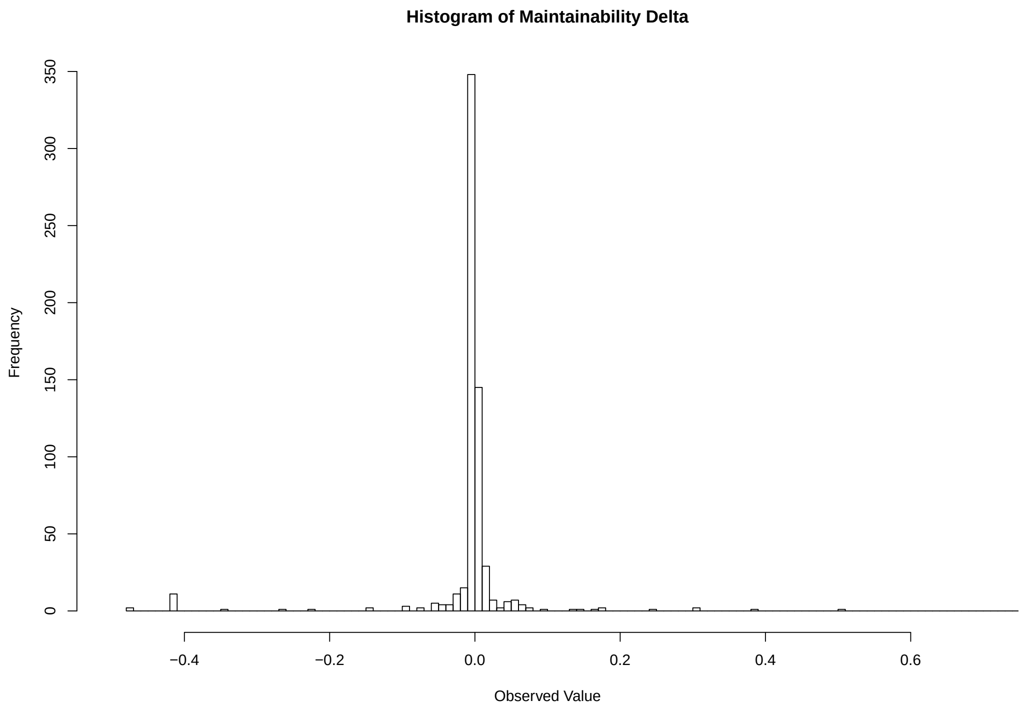


Figure 3.11: Maintainability Delta Distribution Over Commit Data

Table 3.5: Maintainability Delta Summary

Attribute	**Value**
Maximum	0.9391
Minimum	-0.4723
Mean	-0.0048
Median	0.0
Standard Deviation	0.0843
Kurtosis	39.4813
Skewness	0.2332

Here we see how little the maintainability delta varies over the commit data. Again we have a strongly leptokurtic normal distribution with a very minor right skewing. The distribution is centered around 0, indicating again that the majority of individual commits do not make a significant change to a given system. As the numerator in the Maintainable Production equation, this is the single biggest factor in determining the overall value of Maintainable Production. No matter the amount of churn contained within a commit, if the maintainability change is 0 then we observe a 0 value for the maintainable production.

3.6 Discussion

In this chapter, we identified the core metrics we will measure to derive the Maintainable Production of a given contribution by a developer. While the observed distribution of Maintainable Production over the commit data could potentially indicate a negative result of the metric, this in fact matches previous work done by Hattori et al. that showed that individual commits by themselves tend to be small with minimal impact on the system.[22] This would suggest future work is needed to observe the value of Maintainable Production over an aggregation of commits, perhaps with respect to a Pull / Merge Request, in which the branch being considered for merge is measured against the current production version of the system.

While most commits generally produce very little if any change in maintainability, we observe a distribution of values which is normal in nature and provides the foundation for which we can rank a given contribution when compared against this benchmark. By defining relevant thresholds within this distribution, we can measure and then classify a contribution's Maintainable Production depending on where the contribution falls within the observed range of values. This ranking scheme would then allow for the development of a feedback mechanism to provide developers an overview of their maintainability change through source code contributions. We will explore this topic further in **Chapter 4** of this work.

Revisiting the research questions posed in the introductory chapter of this work, **Section 1.2.1**, we asked "By what measures can we evaluate a developer's code contribution to a project as maintainable code production?" As we have now identified and defined the core metrics of Maintainable Production, we are able to answer **Research Question 1** by defining the developer's maintainable code production as the rate of maintainability change, measured through a subset of BCH's guidelines, incurred on the system as the result of the percentage of the system that was changed, measured as the $LOC_{churned} \div LOC_{system}$. Moving forward, we will investigate how we can use the benchmark distribution as a means of constructing a feedback mechanism for the Maintainable Production of a given contribution.

3.7 Threats to Validity

- We measure maintainability purely through Better Code Hub. Any bug or fault in its implementation would result in inaccurate measure of Maintainable Production.
- We use a relatively small sampling in our benchmark analysis. This is primarily due to data access issues from specific GitHub repositories as well as ongoing revisions to Better Code Hub's data collection methodology.
- Human error could have potentially be introduced by the researcher when processing the data.

Chapter 4

Constructing a Feedback Mechanism Within Better Code Hub

Now that we've constructed the Maintainable Production Model and observed how this value is distributed over real commit data from a large sampling of systems under active development, we must determine a manner in which we can present the output of this model to the developer's. In this case, we will construct an addition to Better Code Hub in the form of a report over the Maintainable Production of a given commit, referred to in Better Code Hub as Commit Quality Impact Report.

This report will be used to provide developer's with an analysis over their individual contributions to the system by providing a rank of Maintainable Production as well as a breakdown of the statistics used to calculate this rank. Statistics over the relative churn of the commit as a percent of the system size will be included in this report, as well as statistics over the change in the individual guidelines incurred through the developer's contribution. Short explanations over each of these elements is provided in the form of definitions of particular terminology used within the report. The motivation of building this feedback mechanism is to answer the research question, "Can we provide feedback over maintainable code production at the commit level which is comprehensible?"

4.1 Identifying Production Classifications

Through the benchmark analysis, we were able to observe the distribution of Maintainable Production over a large sampling of commit data taken from 40 software systems in active development. While this distribution showed the majority of single commits produce little change in the maintainability of a given system, the normal distribution of values allows for us to determine clear divisions of these values as a means of identifying individual classifications of commits. By doing so, we are able to measure a new contribution's Maintainable Production and then rank the commit according to where it falls within this distribution.

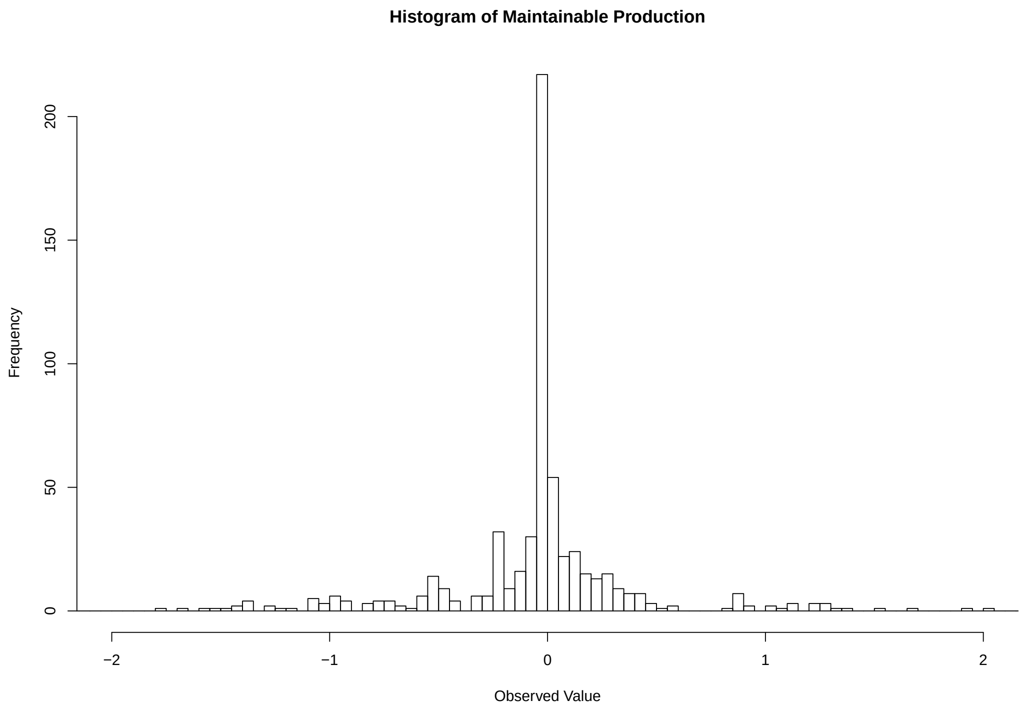


Figure 4.1: Distribution of Maintainable Production Over 40 Software Systems

Returning to the distribution of values, we determine thresholds via a distribution split following a 5-30-30-30-5 schema, which effectively splits the distribution over the 5%, 35%, 65%, 95% percentiles. This schema was chosen as a means of staying consistent with SIG's star-system of rating software projects through their consultancy and it follows roughly the 3-sigma rule of observing normally distributed values.[26] Splitting the distribution by this schema also allows for the identification of 5 clear categories of values, allowing for us to employ the well known letter grade scheme utilized in the United States grading system.[5] It is important to note that this was a clear design decision made in particular for this iteration of the feedback mechanism. In fact, any split of the distribution can be used to define any number of specific categories of Maintainable Production. We chose this schema as it fits with commonly used grading conventions and its usage in industry-leading software consultancy.

Applying this schema to the the Maintainable Production benchmark distribution, we observe the following Maintainable Production thresholds:

Table 4.1: Observed Threshold Values

Percentile	**Threshold Value**	**Threshold Label**
5th	–1.0778	"Worst"
35th	-0.0059	"Poor"
65th	0.0051	"Average"
95th	0.9933	"Good"

Table 4.2: Maintainable Production Value Grading Scheme

Category Condition	Category Grade
Observed Value < Worst Threshold	F
Worst Threshold <= Observed Value < Poor Threshold	D
Poor Threshold <= Observed Value < Average Threshold	C
Average Threshold <= Observed Value < Good Threshold	B
Observed Value > Good Threshold	A

Comparing the Maintainable Production output of an analyzed commit against the defined thresholds, we are able to provide a letter grade to each commit analyzed under Better Code Hub. This allows us to provide personal feedback to developers over each individual commit they make to a system, and if measured over time could produce a trend of their Maintainable Production as sort of maintainability grade point average. The expectation is that developers will be incentivized to receive a higher grade and continue to improve their maintainability habits until there are considered an "A" ranked developer. This grade then marks the first step in providing a detailed report over the developer's Maintainable Production at the commit level.

4.2 Building the Production Report

While a single grade of Maintainable Production for a given commit can be used for tracking maintainability change over time, it provides little feedback on how to proceed going forward with development and which individual aspects contributed to calculating this grade. As an attempt of improving the actionability and ensuring developer comprehension a Maintainable Production report feature was developed within Better Code Hub. This feature includes a custom tag-line built into Better Code Hub's existing feedback details through its guideline reports which includes navigation to the full Maintainable Production report, referred to in Better Code Hub as the Commit Quality Impact Report (CQI report). Included in the CQI report is the Maintainable Production rank of the commit as described in the previous section, referred to in the report as the Commit Quality Impact Rank, as well as statistics over the code churn and feedback over the individual guideline changes used to calculate the Maintainable Production.

4.2.1 Delta Reports in BCH

Better Code Hub offers the ability for developers to integrate its anaylsis service into the development flow of a software project via a GitHub webhook, a special HTTP callback to BCH which triggers an analysis any time a contribution is made to the software project through a git Push Commit or Pull Request. With this webhook enabled, this analysis produces a specialized report produced through BCH's UI called a Delta Report. This Delta Report is different from the traditional report BCH generates through manual analysis in that it attempts to show the change in maintainability per guideline through particular UI elements, namely a colored bar representing the analyzed risk profile and an up or down arrow to indicate positive or negative change on the guideline, respectively.

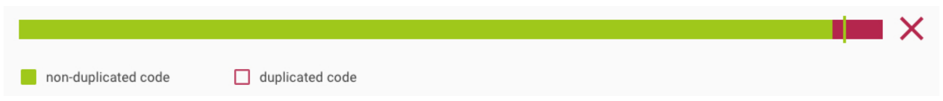
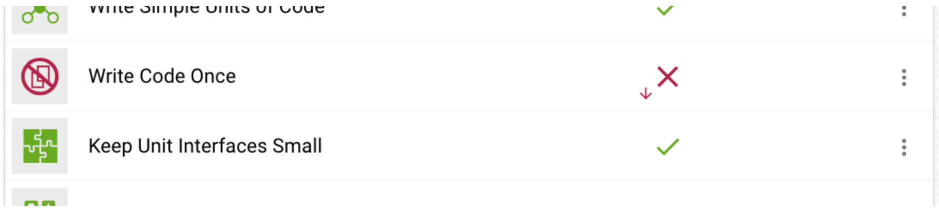


Figure 4.2: Visual Representation of the Observed Risk Profile. The colored bars represent the volume of the risk profile bins relative to each other. The vertical line on the bar depicts the compliance threshold.

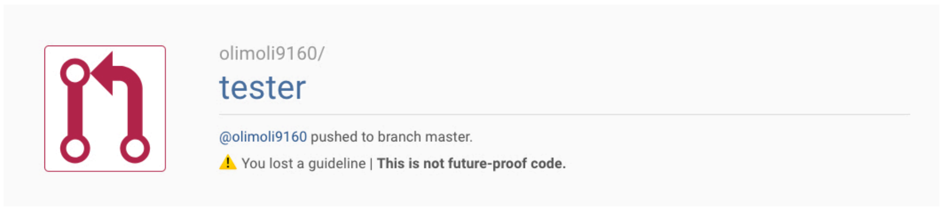

Write Code Once

Keep Unit Interfaces Small

Figure 4.3: Compliance Change is depicted by an upward or downward arrow on the guideline check. This example shows that the *Write Code Once* guideline deteriorated as a result of the commit that generated this Delta Report.

While this visual representation can be useful in getting an understanding of the manner in which a guideline changes due to a commit, a singular value in which the developer understands the magnitude of the change is absent. It is unclear by how much the commit has affected the maintainability under this guideline as we cannot see the original risk profile of the system prior to this commit without accessing a previous report. In the following section (**Section 4.2.2**), we will demonstrate how to use the Distance Deltas generated by the Maintainable Production Model to provide more detailed information regarding how large of a change occurs in the subset of guidelines used in the model.

In addition to the guideline change visuals, a tag-line comment is included in the header of the Delta Report to indicate "at a glance" what was the affect of the commit on the guidelines. This tag-line is also sent to GitHub to be reported in the CI checks of the software system being contributed to as a means of informing the developer of the simplified status of their contribution without the need to visit Better Code Hub's UI. Delta Reports are accessed solely through the "Details" button in the GitHub CI check.

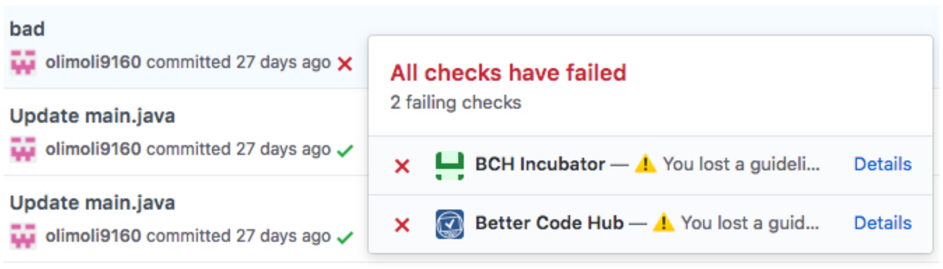

olimoli9160/

tester

@olimoli9160 pushed to branch master.

⚠ You lost a guideline | **This is not future-proof code.**

Figure 4.4: The header of a Delta Report including the tag-line comment "You lost a guideline".


bad

olimoli9160 committed 27 days ago ✗

Update main.java

olimoli9160 committed 27 days ago ✓

Update main.java

olimoli9160 committed 27 days ago ✓

All checks have failed

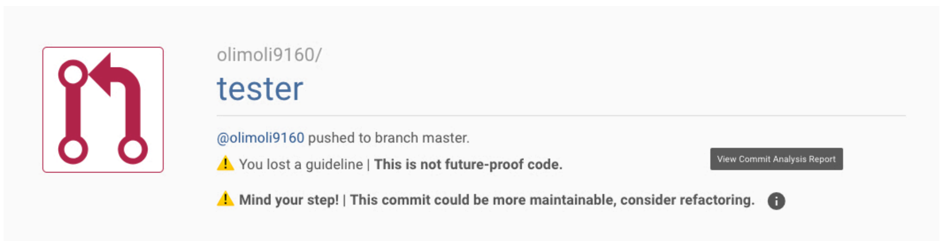
2 failing checks

✗ **BCH Incubator** — ⚠ You lost a guideli... Details

✗ **Better Code Hub** — ⚠ You lost a guid... Details

Figure 4.5: Better Code Hub CI integration in GitHub, including tag-line comment from same commit as **Figure 4.4**.

It is within this specialized analysis report that we include the CQI report detailing the commit's Maintainable Production characteristics. Access to the CQI report is made via an additionally tag-line comment created within the header of the analysis report. This comment behaves similarly to the original analysis tag-line in that it provides a single sentence to inform the developer over the nature of the changes to the system's maintainability as it relates to productivity. Included in this additional tag-line comment is an information button which toggles a new window containing the specifics of the CQI report.


olimoli9160/
tester

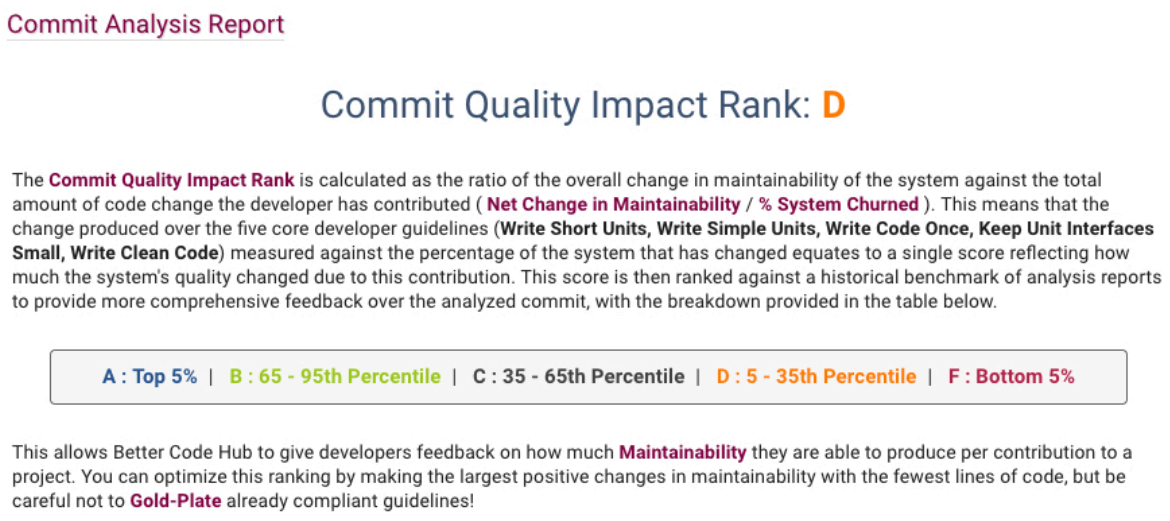
@olimoli9160 pushed to branch master.
⚠ You lost a guideline | **This is not future-proof code.**
⚠ **Mind your step! | This commit could be more maintainable, consider refactoring.**

Figure 4.6: The header of the Delta Report including the additional tag-line comment "Mind your step! — This commit could be more maintainable, consider refactoring." along with CQI report access button.

With this addition, we can now construct an additional report over the Maintainable Production characteristics of the commit that can be viewed in tandem with the original analysis results from Better Code Hub.

4.2.2 Constructing the CQI Report

As a means of improving the comprehensibility of the Maintainable Production grade, or CQI rank as it is referred to within this experimental version of Better Code Hub, we have chosen to include the original formula used to generate the grade within the report, specifically $\Delta Maintainablility \div Code\ Churn\ \%$, along with a short explanation over how the CQI rank was calculated and which guidelines contributed to the rank given. The intention of this explanation is to ensure transparency over the metric and to improve justification of the given rank. The grading schema is also supplied here as a means of helping the developer understand how their contribution compares against the commit-data measured in the benchmark analysis.


Commit Analysis Report

Commit Quality Impact Rank: **D**

The **Commit Quality Impact Rank** is calculated as the ratio of the overall change in maintainability of the system against the total amount of code change the developer has contributed (**Net Change in Maintainability** / **% System Churned**). This means that the change produced over the five core developer guidelines (**Write Short Units, Write Simple Units, Write Code Once, Keep Unit Interfaces Small, Write Clean Code**) measured against the percentage of the system that has changed equates to a single score reflecting how much the system's quality changed due to this contribution. This score is then ranked against a historical benchmark of analysis reports to provide more comprehensive feedback over the analyzed commit, with the breakdown provided in the table below.

A : Top 5% | **B : 65 - 95th Percentile** | **C : 35 - 65th Percentile** | **D : 5 - 35th Percentile** | **F : Bottom 5%**

This allows Better Code Hub to give developers feedback on how much **Maintainability** they are able to produce per contribution to a project. You can optimize this ranking by making the largest positive changes in maintainability with the fewest lines of code, but be careful not to **Gold-Plate** already compliant guidelines!

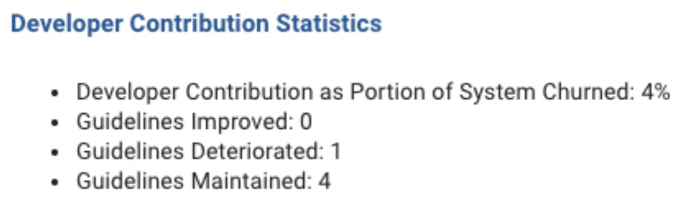
Figure 4.7: CQI Report Header of Commit

In addition to explanation of the CQI Rank, we provide a breakdown of the elements in the formula which contributed to this rank, namely the $\Delta Maintainability$ and the $Code\ Churn\ \%$.

Let us first look at the $Code\ Churn\ \%$ information presented in the report. We have chosen to keep this relatively simplistic by reporting only the real value of code churn as a percentage of the system. In the report, this value is referred to as the $\%SystemChurned$ and explain it in the CQI report header as the "percentage of the system that has changed". Within the actual statistics reported to the developer, we refer to this value again as the "Developer Contribution as Portion of System Churned." The actual calculation of this measure is not provided in this report.

In addition to this value, we include the first information regarding change in the BCH guidelines in 3 categories: improved, deteriorated and maintained. Improved and deteriorated are self-explanatory;

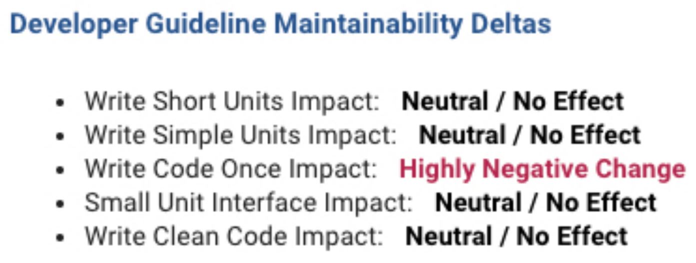
they simply report the total amount of guidelines that were either improved or degraded due to the developer's contribution to the system. Maintained simply refers to those guidelines which experienced no change, therefore their maintainability score was maintained through the developer's contribution.


Developer Contribution Statistics

- Developer Contribution as Portion of System Churned: 4%
- Guidelines Improved: 0
- Guidelines Deteriorated: 1
- Guidelines Maintained: 4

Figure 4.8: Example of Code Churn % details as presented in the CQI Report

Simply stating the number of guidelines that improved, deteriorated or were maintained is not an improvement over what Better Code Hub reports by default. Utilizing the maintainability deltas generated by the Maintainable Production Model, changes to the individual guidelines measured under the model are reported using as a classification scheme with a value range of *Highly Negative Change, Negative Change, Neutral / No Effect, Positive Change,* and *Highly Positive Change.*


Developer Guideline Maintainability Deltas

- Write Short Units Impact: **Neutral / No Effect**
- Write Simple Units Impact: **Neutral / No Effect**
- Write Code Once Impact: **Highly Negative Change**
- Small Unit Interface Impact: **Neutral / No Effect**
- Write Clean Code Impact: **Neutral / No Effect**

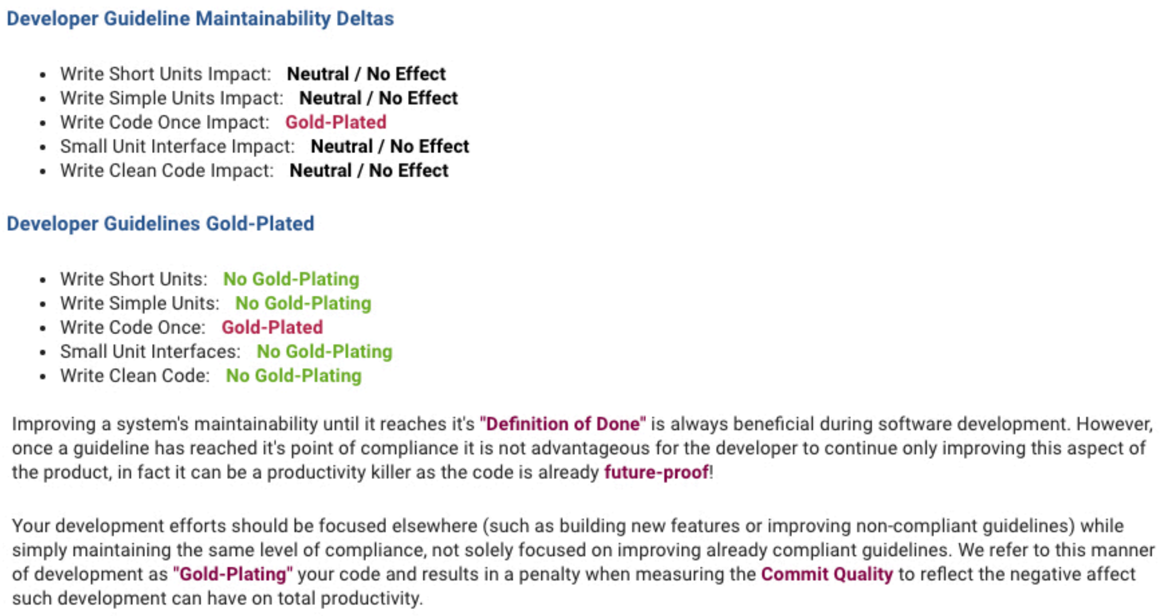
Figure 4.9: Example of Guideline Details as presented in the CQI Report

This classification range was obtained by utilizing the same 5-30-30-30-5 scheme used to classify the Maintainable Production value from the commit data per guideline used in the model. Thus, a Highly Negative Change in a given guideline is categorized as a guideline change found in the bottom 5% of the benchmarked commits, while a Highly Positive Change is categorized as belonging to the top 5% of changes found in the benchmark data. We chose to categorize the individual changes of the guidelines against the benchmarked commits rather than on the theoretical maximum and minimums of the guideline distance as a means of keeping it consistent with the measures used to categorize the maintainable production, however this was a specific design decision for this iteration of the CQI report.

Table 4.3: Guideline Maintainability Delta Threshold Values

Guideline	5th Percentile	35th Percentile	65th Percentile	95th Percentile
Short Units	-0.03264	-0.00004	0.00001	0.02098
Simple Units	-0.00849	-0.00007	0.00001	0.00546
Code Once	-0.00164	-0.00001	0.00001	0.00195
Unit Interface	-0.00077	-0.00001	0.00001	0.00761
Clean Code	-0.00028	-0.00001	0.00001	0.00042

The final aspect of the Maintainable Production Model we address in the CQI report is the "gold-plating" detection per guideline. This feature is particularly important within the context of this feedback as it directly addresses the relationship between the continuous improvement of maintainability and overall productivity of the developer. Due to the manner in which gold-plating negatively affects the overall calculation of the CQI rank, we chose to provide a more in-depth explanation for this particular aspect in the CQI report as it is a fundamental aspect developer's would need to understand to get the most out of the Maintainable Production feedback. In addition to this explanation, we make it explicit which guidelines are "gold-plated" against those which are not. "Gold-plating" is also reflected in the Maintainability Delta feedback to indicate a negative change to the particular guideline as a means of demonstrating to the developer the negative affect gold-plating has in the calculation of the CQI rank.


Developer Guideline Maintainability Deltas

- Write Short Units Impact: **Neutral / No Effect**
- Write Simple Units Impact: **Neutral / No Effect**
- Write Code Once Impact: **Gold-Plated**
- Small Unit Interface Impact: **Neutral / No Effect**
- Write Clean Code Impact: **Neutral / No Effect**

Developer Guidelines Gold-Plated

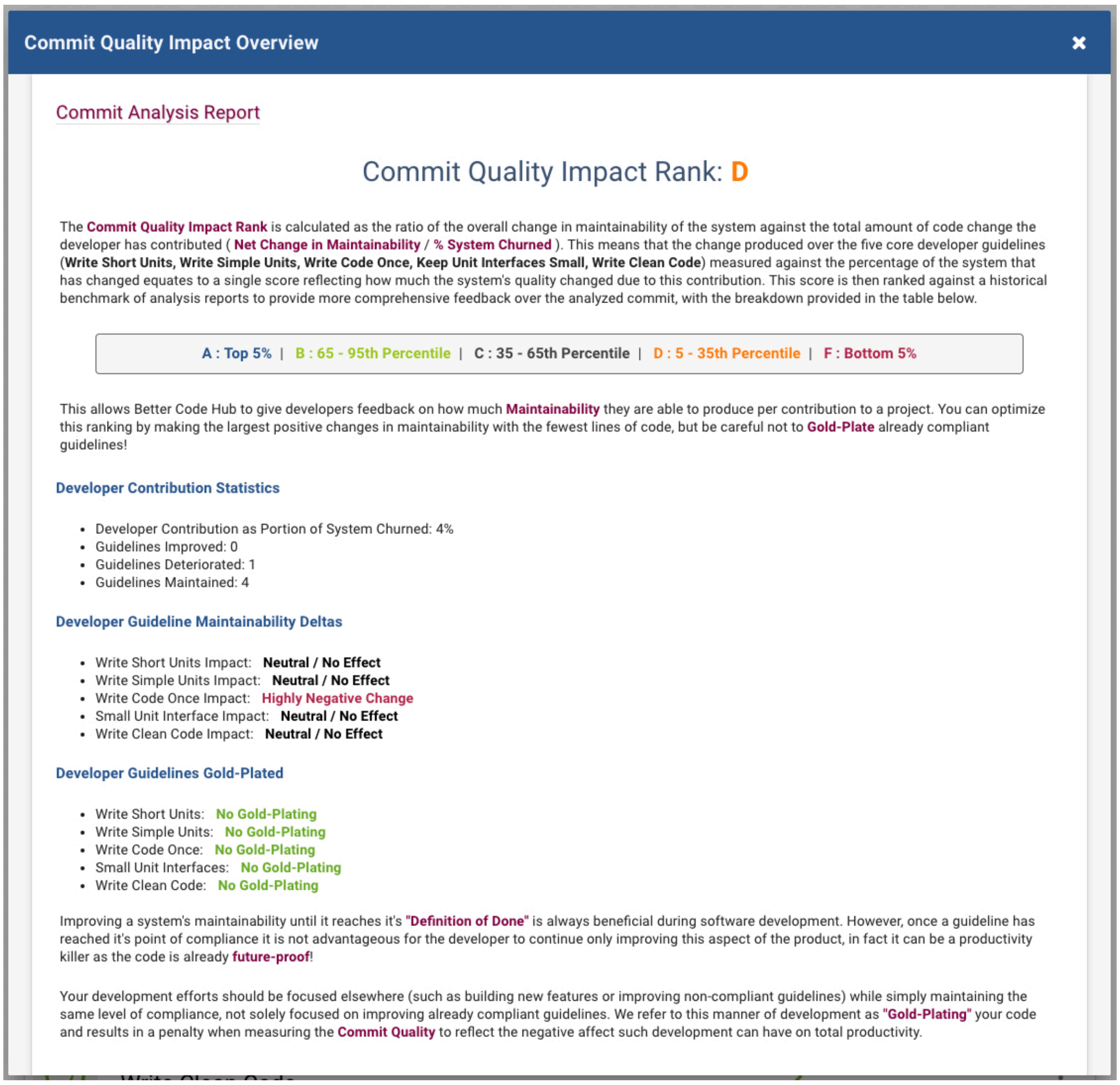
- Write Short Units: **No Gold-Plating**
- Write Simple Units: **No Gold-Plating**
- Write Code Once: **Gold-Plated**
- Small Unit Interfaces: **No Gold-Plating**
- Write Clean Code: **No Gold-Plating**

Improving a system's maintainability until it reaches it's **"Definition of Done"** is always beneficial during software development. However, once a guideline has reached it's point of compliance it is not advantageous for the developer to continue only improving this aspect of the product, in fact it can be a productivity killer as the code is already **future-proof**!

Your development efforts should be focused elsewhere (such as building new features or improving non-compliant guidelines) while simply maintaining the same level of compliance, not solely focused on improving already compliant guidelines. We refer to this manner of development as **"Gold-Plating"** your code and results in a penalty when measuring the **Commit Quality** to reflect the negative affect such development can have on total productivity.

Figure 4.10: An example of the "Gold-plating" details as presented in the CQI report.

With the addition of the Gold-Plating details, the CQI report includes an overview of all aspects which influence the Maintainable Production calculation for a given commit. Some aspects are intentionally left vague to avoid "holding the hand" of the developer and to hide implementation details. With this feedback mechanism in place, we can further investigate the comprehensibility of the Maintainable Production Model as well as the improvement in actionability over the default version of Better Code Hub.



Commit Quality Impact Overview

Commit Analysis Report

Commit Quality Impact Rank: **D**

The **Commit Quality Impact Rank** is calculated as the ratio of the overall change in maintainability of the system against the total amount of code change the developer has contributed (**Net Change in Maintainability** / **% System Churned**). This means that the change produced over the five core developer guidelines (**Write Short Units, Write Simple Units, Write Code Once, Keep Unit Interfaces Small, Write Clean Code**) measured against the percentage of the system that has changed equates to a single score reflecting how much the system's quality changed due to this contribution. This score is then ranked against a historical benchmark of analysis reports to provide more comprehensive feedback over the analyzed commit, with the breakdown provided in the table below.

A : Top 5% | **B : 65 - 95th Percentile** | **C : 35 - 65th Percentile** | **D : 5 - 35th Percentile** | **F : Bottom 5%**

This allows Better Code Hub to give developers feedback on how much **Maintainability** they are able to produce per contribution to a project. You can optimize this ranking by making the largest positive changes in maintainability with the fewest lines of code, but be careful not to **Gold-Plate** already compliant guidelines!

Developer Contribution Statistics

- Developer Contribution as Portion of System Churned: 4%
- Guidelines Improved: 0
- Guidelines Deteriorated: 1
- Guidelines Maintained: 4

Developer Guideline Maintainability Deltas

- Write Short Units Impact: **Neutral / No Effect**
- Write Simple Units Impact: **Neutral / No Effect**
- Write Code Once Impact: **Highly Negative Change**
- Small Unit Interface Impact: **Neutral / No Effect**
- Write Clean Code Impact: **Neutral / No Effect**

Developer Guidelines Gold-Plated

- Write Short Units: **No Gold-Plating**
- Write Simple Units: **No Gold-Plating**
- Write Code Once: **No Gold-Plating**
- Small Unit Interfaces: **No Gold-Plating**
- Write Clean Code: **No Gold-Plating**

Improving a system's maintainability until it reaches it's **"Definition of Done"** is always beneficial during software development. However, once a guideline has reached it's point of compliance it is not advantageous for the developer to continue only improving this aspect of the product, in fact it can be a productivity killer as the code is already **future-proof**!

Your development efforts should be focused elsewhere (such as building new features or improving non-compliant guidelines) while simply maintaining the same level of compliance, not solely focused on improving already compliant guidelines. We refer to this manner of development as **"Gold-Plating"** your code and results in a penalty when measuring the **Commit Quality** to reflect the negative affect such development can have on total productivity.

Figure 4.11: An example of a complete CQI report.

4.2.3 Technical Summary

As the CQI report is not an existing feature in Better Code Hub and required a non-trivial amount of engineering to construct, we provide a brief summary of the technologies utilized in the creation of the CQI report and high-level view of the architectural modifications made within Better Code Hub.

Better Code Hub itself is a web-based application built in the Angular 5 JavaScript framework[3], with the client-side of the application written in TypeScript.[10] The back-end and data processing domain of BCH is written in Java 9[6] utilizing the SpringBoot[13] framework, with data storage handled through a Mongo database.[8] A tailored version of SIG's own software analysis toolkit (SAT), which includes their custom parsing and tokenization framework, is included in the application pipeline as an external resource. The output of the custom SAT is then analyzed through a specialized guideline processing external component, which is used to calculate the analysis results and compliance per guideline.

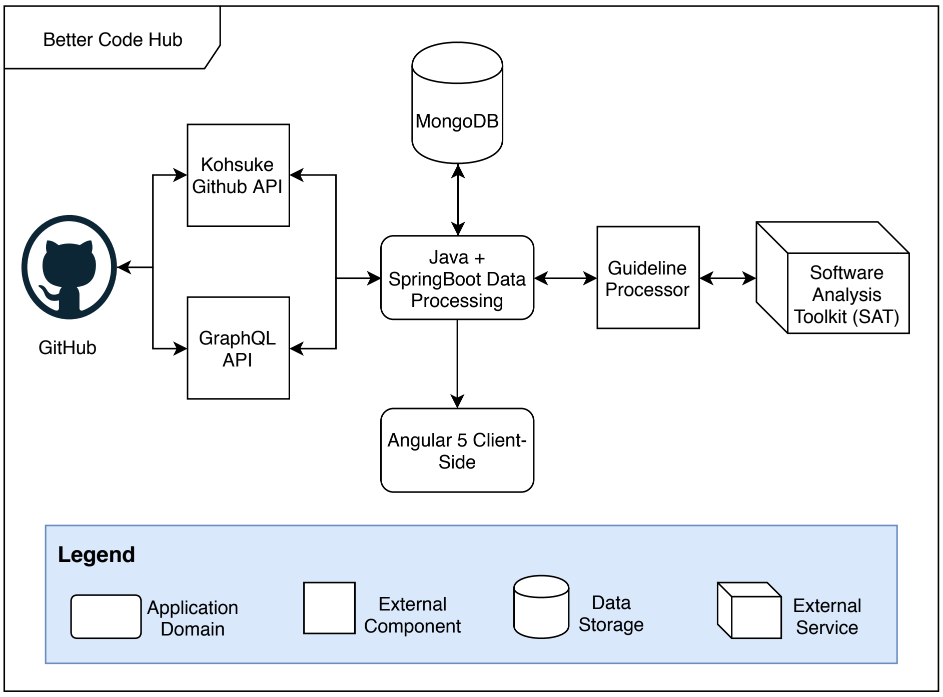


Figure 4.12: Simplified Overview of the BCH Application Pipeline

The construction of the CQI report consisted of engineering in both the client-side and server-side domains of BCH. In the application's back-end, the Maintainable Production algorithm was built into the analysis report generation resource, specifically within the resource for generating a Delta Report from two consecutive BCH analysis result entities: the original report results and the new report results containing a change made through a developer's contribution. Here we introduced a new service entity for processing the two results with respect to the metrics needed to calculate the $\Delta Maintainability$. We also utilized both the standard GraphQL GitHub API[4] as well as an open-source, object-oriented GitHub API written in Java[2] as a means of querying project source code to apply the metrics for measuring code churn. Once complete with processing and collecting of the necessary measures to calculate the contribution's Maintainable Production, along with the calculated MP value itself, a data-model of the production report was constructed as a means of storing the data within the application's Mongo database.

Within the front-end of the application, we created a mirrored TypeScript production report data-model of the Java data-model discuss previously. These data-models are constructed as a means of persisting the actual data to be displayed in the CQI report through the full application. This allows for simple data-binding of the necessary entity values to be displayed in HTML. Within the UI of Better Code Hub, we created a unique Angular dialog based on existing entities as the vehicle for displaying CQI reports. From here, a simple HTML template was created for displaying the necessary information in the dialog, with the real values displayed through Angular's efficient data-binding properties. The custom CSS theme of BCH was used to create a consistent appearance of the CQI report with the remainder of the application, with alterations made where necessary to ensure readability. **Figure 4.13** shows a simplified overview of the modifications made within the specified components in BCH.

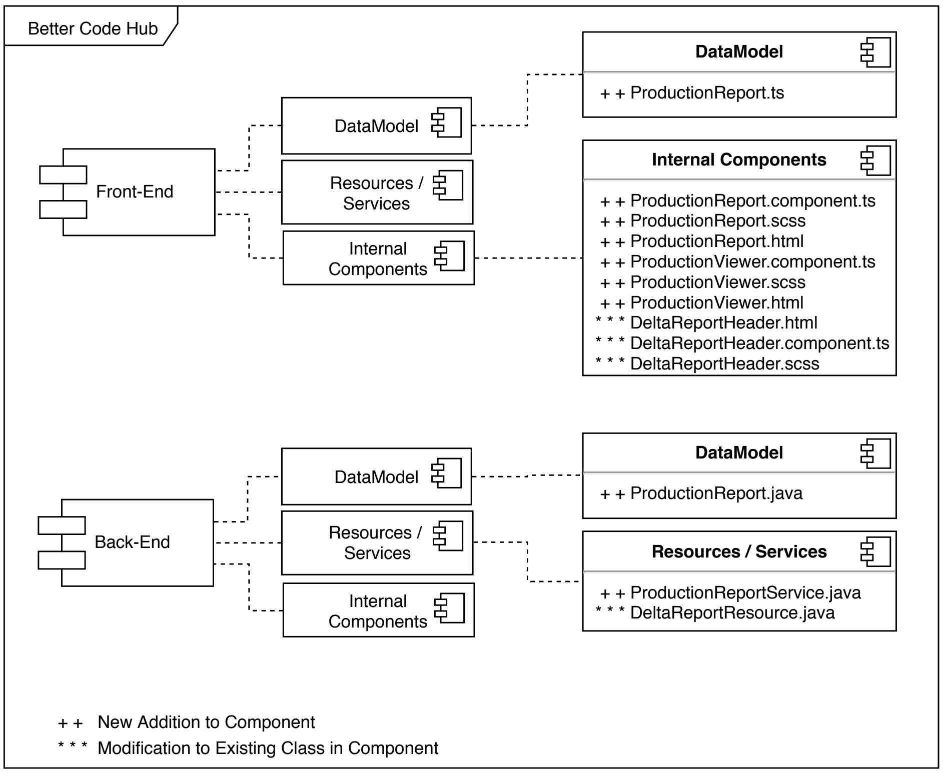

Better Code Hub
Front-End
DataModel
Resources / Services
Internal Components
DataModel
+ + ProductionReport.ts
Internal Components
+ + ProductionReport.component.ts
+ + ProductionReport.scss
+ + ProductionReport.html
+ + ProductionViewer.component.ts
+ + ProductionViewer.scss
+ + ProductionViewer.html
* * * DeltaReportHeader.html
* * * DeltaReportHeader.component.ts
* * * DeltaReportHeader.scss
Back-End
DataModel
Resources / Services
Internal Components
DataModel
+ + ProductionReport.java
Resources / Services
+ + ProductionReportService.java
* * * DeltaReportResource.java
+ + New Addition to Component
* * * Modification to Existing Class in Component

Figure 4.13: Component Additions and Modifications in BCH for CQI Report Feature

In total, 9 file additions were made with modifications to 4 existing files to accommodate the new CQI Report feature in BCH. These changes spread across 6 components, with 2 new internal components being added to the front-end of the application, namely the Production Report component and the Production Viewer Component.

4.3 Discussion

By utilizing the 5-30-30-30-5 schema for splitting the Maintainable Production distribution into quantiles for which we could apply the A-F grading schema, we are able to rank the impact of a developer's contribution with respect to the maintainability of the system. Thus we have answered **Research Question 2**, "By what means can we evaluate the impact of a developer's maintainable code production?", as the observed CQI Rank of a developer's contribution as it compares to the benchmark analysis of Maintainable Production.

The construction of the CQI Report feature realizes the goal of creating a feedback mechanism in which we can inform developers over the MP Model output generated through their contribution of the system. This feedback attempts to breakdown the underlying metrics used in the calculation of the Maintainable Production of a contribution by providing details over the scaled maintainability impact per guideline, the effect of gold-plating, and the churn introduced within the contribution as a percentage of the system. In the following chapter, we will investigate the extent to which this feedback is comprehensible to domain-experts as a means of answering **Research Question 3**, "Can we provide feedback over maintainable code production at the commit level which is 1) coherent and 2) actionable for developers?"

Chapter 5

Evaluating the Maintainable Production Feedback

There are many different perspectives and approaches taken to validate a software metric, as there is currently no standard empirical or theoretical methodology of validation in the software engineering field.[35][36] Typically speaking, validation is done theoretically using properties of the selected measure itself. Technical Action Research purposes that validation be done in "cycles", in which an experiment is tested in a hierarchy of environments, from a small lab setting to more robust "real-world" settings.[39] Meneely et al. performed a literature review over 20 metric validation papers, in which they were able to identify 47 unique criteria to be used depending on the philosophy chosen by which to validate a metric.[29] These 47 criteria were then grouped into individual, and often overlapping, categories to represent the diversity of perspectives which could be taken to overcome this complex issue.

In the case of the Maintainable Production Model and the underlying metric described in this work, this complexity of validation is also observed. In order to properly validate this metric under the given research method, a full TAR cyclic approach would be needed in which we rounds of validation is performed in different settings, beginning in an internal laboratory environment to a "field" test utilizing A-B testing methodologies with actual users of BCH. With the time and resources available for this research, it was not feasible to test the Maintainable Production Model in an external environment in the "field". This likely would provide the most actionable results for this research going forward, however we chose to limit the scope of the validation in this research to an internal laboratory environment following the TAR protocol. In this manner, we have a more controlled observation of the effective use of the metric as well as direct feedback from the domain-expert users.

The next issue is the selection of the validation criteria by which we will test the metric. This is dependent on the intended purpose of the metric's employment, which we for now state as a simple feedback mechanism for developer's to improve their software development habits with regards to maintainable code production. For this purpose, we intend partially observe subset of the validation criteria detailed in the Meneely et al. literature review, in particular the actionability and definition validity aspects. Actionability is defined as the "ability for the software manager [or practitioner] to make an empirically informed decision from the status of the software product's status" as reported by the metric.[29] Definition validity, as defined by Roche[33], is the the extent to which a metric's definition and construct is "clear" and "unambiguous", which we will refer to further as the comprehensibility of the metric. We will test these aspects of the metric through an formal interview process with software maintainability domain-experts within SIG.

It is important to state that then intention is not to fully validate the metric in all scenarios, but only to validate how understandable the metric is and how actionable is the feedback for in the context of a laboratory environment with domain-experts of software maintainability. This says nothing of the statistical correctness or soundness of the metric, nor should it be seen as a validation for developers in a general-sense. These are attributes reserved for exploration in future work due in-feasibility in the scope of this research.

5.1 Interview Protocol

The first step in beginning the interview process is defining a protocol for interviewing. We defined a set of 21 questions split over 4 categories: interviewee demographics, ISO 25010 definition of software maintainability, usage of Better Code Hub, and finally the experience with the Maintainable Production feature in the experimental branch of BCH. Furthermore, questions are categorized as scaled and non-scaled questions, with scaled questions indicating an expected response from a selection of predetermined responses based on the Likert Scale, with non-scaled questions allowing for an open and free response from the interviewee. Scaled questions were allowed to also be answered openly, however interpretation would be based on the selected pre-determined reply from the scale provided.

- **Interviewee Demographics** - Used to gain insight into the background and experience of the interviewee. Questions include queries over the development experience, current role, education level, and type of software projects the interviewee has experienced over their professional career.
- **Definition of Software Maintainability** - Scaled questions to determine the extent to which the interviewee understands and agrees with the ISO 25010 standard.
- **Usage of Better Code Hub** - Scaled and non-scaled questions to determine the extent to which the interviewee agrees with the feedback they receive from Better Code Hub, the extent to which the feedback is actionable, and the general experience the interviewee has had with Better Code Hub.
- **Experience with MP metric / CQI Report** - Scaled and non-scaled questions to determine the extent to which the interviewee agrees with the feedback they receive in the CQI report, the degree to which the feedback matches their expectations and is clear and understandable, and the understanding of the ”gold-plating” aspect as it relates to the CQI rank and the fairness therein.

Table 5.1: Response Scale for Scaled Questions

Scale Encoding	Response
1	Completely Disagree / Completely Unimportant
2	Moderately Disagree / Moderately Unimportant
3	Neither Agree nor Disagree / Somewhat Important
4	Moderately Agree / Moderately Important
5	Completely Agree / Completely Important

Interviewee candidates were selected based on their participation in active development on internal projects at SIG. As Better Code Hub currently only offers analysis on systems hosted on GitHub, this limited us to 3 internal projects at SIG to select candidates from. Due to SIGs relatively small development team, and the restriction to 3 internal projects, we were able to schedule 6 candidates involved in active software development to interview. As preparation for the interview, interviewees were asked to review a sampling of their own commits to the internal SIG projects utilizing the Maintainable Production feature in Better Code Hub. Each interviewee reviewed personalized feedback regarding their own contribution to a system, allowing for feedback from a varied selection of CQI reports with differing feedback personalized for the interviewee who generated the report. Interviewees were expected to review each commit’s CQI report in advance to formulate opinions on the feedback presented from the Maintainable Production model. Interviews lasted between 30-45 minutes and were conducted one on one with the interviewee and researcher.

Table 5.2: Interview Question List

Question	Scaled	Category
1. What is your current role in software development?	No	Demographics
2. How many years of experience do you have developing software?	No	Demographics
3. What is the typical type of project you have worked on?	No	Demographics
4. To what extent do you agree with the ISO 25010 definition of software maintainability?	Yes	ISO 25010
5. To what extent is it important to you to produce maintainable, as defined by this standard, when contributing to a software system?	Yes	ISO 25010
6. To what extent is it important that you feel you are continuously improving your own development efforts as far as maintainability is concerned as defined by ISO 25010?	Yes	ISO 25010
7. Do you feel Better Code Hub is accurate in measuring maintainability as defined in ISO 25010?	No	BCH
8. How often do you visit Better Code Hub to gain feedback on a software project's maintainability status?	No	BCH
9. Do you agree the feedback is useful in determining how maintainable the project is and how your own changes have affected the maintainability?	No	BCH
10. Is the feedback actionable, in that you know what improvements you yourself can make in the software to increase maintainability?	No	BCH
11. What, if any, aspects of Better Code Hub do you find most beneficial to your development habits?	No	BCH
12. Are you satisfied with the feedback Better Code Hub currently gives?	No	BCH
13. To what extent do you agree with the statement "If I did not use Better Code Hub, the quality of my software would be the same"?	Yes	BCH
14. To what extent do you agree "the calculation of the Commit Quality Impact Rank is clear and comprehensible?"	Yes	MP
15. How does the CQI Rank you receive affect your motivation going forward with development?	No	MP
16. Does the rank you receive match your expectations of the maintainability you produced with your contribution?	No	MP
17. To what extent do you agree that the metric is fair in penalizing Gold-Plating when analyzing your contributions?	Yes	MP
18. The Gold-plating feature attempts to highlight the interplay between productivity and software quality. Do you feel this metric accurately captures this aspect of development?	No	MP
19. Is the feedback provided has more, less, or the same amount of actionability going forward with development	No	MP
20. What, if anything, do you find most beneficial about the CQI [Maintainable Production] metric being included in Better Code Hub?	No	MP
21. Are you satisfied with the feedback provided in the CQI Report?	No	MP

5.2 Findings

As a means of processing the interview results, we first examine the measured feedback provided through scaled response and then summarize the results of the non-scaled per category. Non-scaled responses are evaluated based on the interpretation of the researcher. Full transcripts of the interviews can be found in **Appendix D** of this work.

5.2.1 Interviewee Demographics

The development role breakdown of the 6 candidates interviewed was as follows: 1 junior full stack engineer with client-side specialization, 1 full stack engineer with a server-side specialization, 1 technical lead, 1 research prototyper, and 2 client-side developers. Average development experience over the 6 interviewees was 5.6 years of professional software development, with a maximum of 10 years and a minimum of 1 year. The average educational level was a university Master's degree, with majors including: Computer Science, Mathematics, Multimedia Design, and Data Science.

Types of software projects interviewees had worked included: video game development, enterprise systems, multimedia systems, data analytics and machine learning applications. All interviewees reported as having experience working on web-based applications. Average development team size reported by interviewees as having typically work in ranged from 4-8 developers, with an average team size of 5. This data therefore indicates the average interviewee had development experience greater than 5 years, typically worked as a web application developer, and is familiar with working in collaboration with other developers on single software system.

5.2.2 Views on ISO 25010 Standard for Maintainability

In order to determine interviewee opinion on ISO 25010's definition of software maintainability, we first provided each interviewee the definition to review before asking them a series of scaled questions, namely questions 4, 5, and 6 from the list of interview questions as observed in **Table 5.2**. These questions were selected to determine the extent to which the interviewee agreed with / found the definition provided by the standard matched their own views on software maintainability, as the standard provides the underlying principles used within BCH and this iteration of the Maintainable Production model.

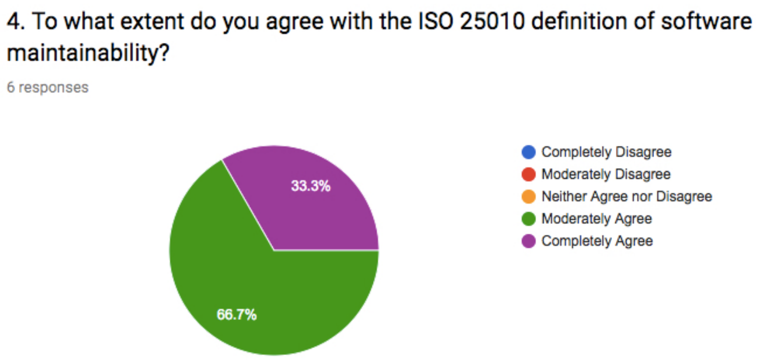


Figure 5.1: Responses Question 4: ISO 25010

Observing the measured responses to question 4 of the interview indicates all interviewees agreed with the ISO 25010 definition of maintainability, with 66.7% indicating "moderate" agreement (4 interviewees in total) with the remaining 33.3%, or 2 interviewees, in "complete" agreement. Follow-up questions were asked to those interviewees indicating a "moderately agree" response in order to better clarify the reason for the response.

Follow-up responses to this question included statements such as "Sometimes trying to make things really maintainable according to this can actually make the implementation more complex", "It is a model... there are many things that come into everyday practice that models cannot catch" and "it says nothing about what you can reuse or not or how we can make it better or worse." This indicates that while they agree with the overall definition supplied in ISO 25010, it was unclear / impractical in applying these principles directly from the standard. There was no mention of disagreement with the fundamental principles in the standard, but rather the interviewees which moderately agreed felt the standard does not give a concrete methodology in which maintainable code can be produced.

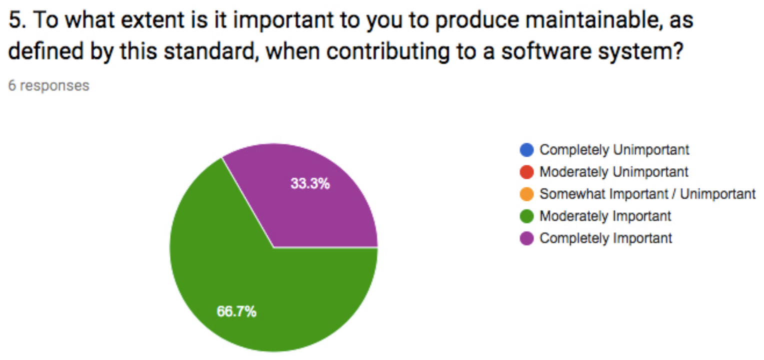


Figure 5.2: Responses Question 5: ISO 25010

With respect to question 5, we again observe all interviewees indicating that maintainable code production, as defined through ISO 25010, was important to their development efforts, with 2 respondents indicating a very high / "complete" importance. From those interviewees which indicated only "moderate importance", follow-up responses included statements such as "it is important to me but only to a certain extent", "the problem is the granularity. As I've told you with re-usability, for example, it is really hard to define", and "that really depends on the project... there are some projects you must deliver fast, so the aim is delivering it really quickly as opposed to having it really maintainable because in a couple of months that code-base is going to be really redundant."

From these responses again we observe that the ISO 25010 definition, while the underlying of principles of software maintainability are agreed on, the application of these principles is not always feasible or relevant in every case for all projects. The majority of interviewees made it clear that it was up to the developer's themselves to be decisive in when and to what extent they abide by the ISO 25010 standard when developing on a project.

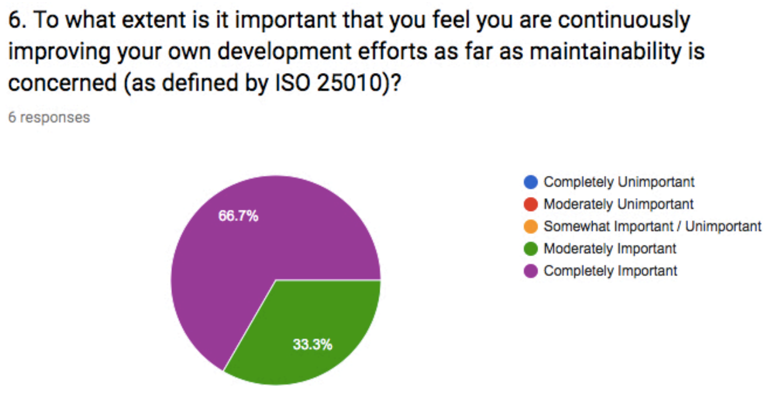


Figure 5.3: Responses Question 6: ISO 25010

Question 6 was asked as an attempt to gauge the extent to which the interviewee strives to make their code more maintainable. The assumption here is that should there be a positive response, they would find the Maintainable Production model more actionable than if they provided a negative response.

All respondents all indicated that it was important to them to improve their own maintainable code production habits (as defined by ISO 25010), with a majority indicating they found it "completely important." Interviewees indicated why it was important to them with statements such as it being "nice to track how far along are you with your [maintainability] goals" and "whenever there is new code to be written, maintainability is one of the most important things" and that "this would be pretty interesting to see, to get really good data over my development" when asked about tracking developer maintainability over time. Response that indicated a less positive opinion stated that "at some point there is an upper bound... at some point [software] is done from a maintainability perspective" and that you shouldn't "do maintainability for maintainability's sake." This reinforces the idea of the "Definition of Done" as it is defined in Better Code Hub and its use in the Maintainable Production Model.

In summary, we observe that all respondents indicate agreement with the principles defined by the ISO 25010 standard of Maintainability and that all interviewees agree that it is important to both produce and track maintainable code at a developer granularity. These findings are important to this research as it identifies the extent to which, at least from a domain-expert level of observation, maintainability is important to the development habits of developers and there is interest in the measuring and tracking of maintainability at the contribution level.

5.2.3 Views on Better Code Hub

After determining the interviewee's views on maintainability as defined by ISO 25010, we investigated how these views related to Better Code Hub itself. BCH was originally designed around this standard and took great strides in making the individual principles measurable through utilization of the 10 guidelines presented in the analysis tool. The goal of this segment of the interview was to determine the extent to which interviewees agreed that BCH's analysis measured the software maintainability principles defined in ISO 25010 as well as the extent to which they found BCH feedback actionable and satisfactory.

Question 7 ("Do you feel Better Code Hub is accurate...?") was asked as an attempt to gauge the interviewee's opinion on the accuracy to which BCH is accurate in measuring maintainability as defined by ISO 25010. The majority of respondents expressed positive responses, with statements such as "I think its sufficiently accurate", "it represents it... it captures it" and "from what Ive used it on I do believe it is accurate in the feedback it gave." One respondent expressed that while BCH is accurate to "the standard itself", but that the tool itself is not accurate as it is language agnostic and that do to this "it is not accurate [as] it is not tweaked to specific programming languages." By design this is intentional as BCH presents itself as a general-purpose software analysis tool which can measure a system without being having to be adjusted to a single language.

Question 8 ("How often do you visit Better Code Hub?") to determine the frequency in which the interviewees viewed BCH's feedback. Responses ranged from "only the merge / pull request" to "after every commit", however the general sentiment observed was that interviewees visit BCH with relative frequency. Every respondent agreed that most attention was paid at the "merge request" point of development as this was the point feedback was most useful to see the change from the previous build of the system.

As a means of understanding how usable is the feedback BCH currently provides to the interviewees, we asked questions regarding if the feedback highlighted the individuals impact on the system (9), how actionable was the feedback provided to the interviewee (10), and how satisfied were the interviewees with the feedback provided (12). With regards to individual impact, responses were less positive, with statements regarding the tracking of individual impact having "some lag there... it could be a bit stronger" and that the analysis itself is relevant "more at the system level than at the single change." Actionability was seen more favorably, with interviewee responses recording they know how to proceed with their development "most of the time" and that they "agree that it helps", however it "takes some time to really get what that information is" and that "particularly for some guidelines some [more] insight would be more helpful."

Overall, interviewees gave positive responses regarding satisfaction with the current BCH feedback, with only one respondent saying no due to "granularity issues" of the measurements employed by BCH. When asked which aspects of BCH they found most beneficial (11), examples given are the ease of BCH's integration with development flow and how "very time you add new code it will check it." Multiple interviewees also indicated they particularly liked that BCH provides a point of being satisfactorily done with maintainability, with one respondent stating "making something more maintainable by itself isn't particularly useful" if other aspects of the project are neglected.

From the subset of scaled questions in the interview question list, question 13 was asked to determine the value interviewees placed in BCH has a means of improving the maintainability of their software.

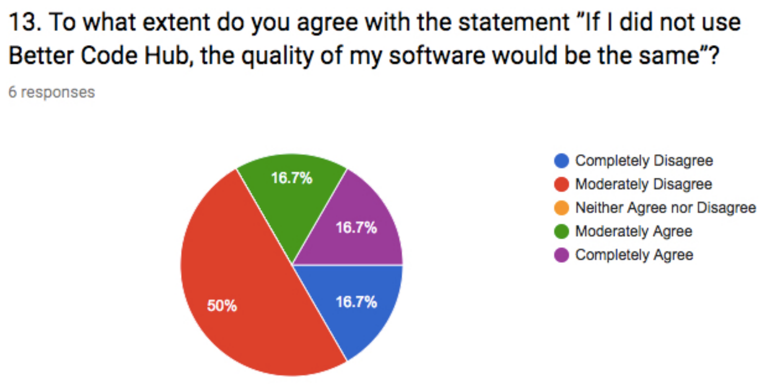


Figure 5.4: Response Question 13: Better Code Hub

Of the responses given, 4 of interviewees disagreed that BCH had no affect on the quality of their software, while 2 agreed with the statement. The respondent which "completely agreed" stated that they "dont really look at it" and that the measures better code hub provides fits their "personality" and that they already "do this without the feedback from Better Code Hub". However, they still stated they "believe there are people for whom this feedback is useful though, especially students just starting." This response would seem to indicate BCH is personally not useful in improving this respondents software quality due to level of experience, however they agreed it is useful in a general since for new developers. This finding was corroborated with the follow-up response from the second interviewee who "moderately" agreed BCH had no affect on their software quality, who stated BCH "sometimes catches some things but I think it comes from the experience as well... I would really use this to make sure that a team is on the same page." This would further suggest that in general BCH is beneficial for tracking the maintainability of projects and is capable of providing feedback which improves the quality of a given project, but the level to which it helps to improve quality is dependent on the experience of the developer.

Summarizing the feedback given, we observed that interviewees mostly agree that BCH is accurate in measuring software maintainability as defined by ISO 25010. The frequency at which they view BCH is typically at the merge / pull request of a worked branch, as this was seen as a more complete contribution where feedback is most necessary. With regards to the usability of the feedback, interviewees found the feedback to be actionable to a moderate extent as it provided enough information to move forward, however this feedback is not always clear for each guideline. Most interviewees were satisfied with feedback given, with most claiming that BCH has a real impact on the quality of their software, however the extent of this benefit depends on the experience level of the developer. These findings are relevant to this research as Better Code Hub is sole measure of software maintainability used in the Maintainable Production model.

5.2.4 Views on CQI Report Feedback

The final segment of the interview investigated interviewee opinion on the Maintainable Production feedback the CQI Report feature provided over the technical contributions they had made to a project. Questions 14-21 from the interview question list were asked as a means of determining the overall comprehensibility of the Maintainable Production model's calculation for the CQI rank along with the impact of the gold-plating penalty used therein, as well as the actionability of the feedback provided. The ultimate goal of this interview segment was to answer the research question, "Can we provide feedback over maintainable code production at the commit level which is a) coherent and b) actionable for developers?" as a means of taking the first steps toward validation of the model's use as a feedback tool for developers to improve their development habits.

Comprehensibility

Questions 14, 16, 17, and 18 from the interview question list observed in **Table 5.2** were utilized to observe the interviewees views on the comprehensibility of the Maintainable Production model. For the purpose of this research, we define comprehensibility as the extent to which the model is clearly understood by the interviewee and the extent to which it matches their own intuition regarding software maintainability.

The questions "to what extent do you agree that the calculation of the Commit Quality Impact Rank is clear and comprehensible?" (question 14) and "does the rank you receive match your expectations of the maintainability you produced with your contribution?" (question 16) were asked as an attempt to gauge the interviewees understanding of the underlying calculation in the model, and if that calculation matched their own intuitions. The assumption here being that if the interviewee in fact finds the CQI Rank measure clear, then they likely will agree with the feedback provided with the reverse being observed if the calculation is unclear. If most interviewee report the measure as being unclear, then we can conclude that the feedback provided by the CQI report in not sufficient in conveying the measures used in the Maintainable Production model.

Question 14, "to what extent do you agree that the calculation of the Commit Quality Impact Rank is clear and comprehensible?", was a scaled question with the following breakdown of responses.

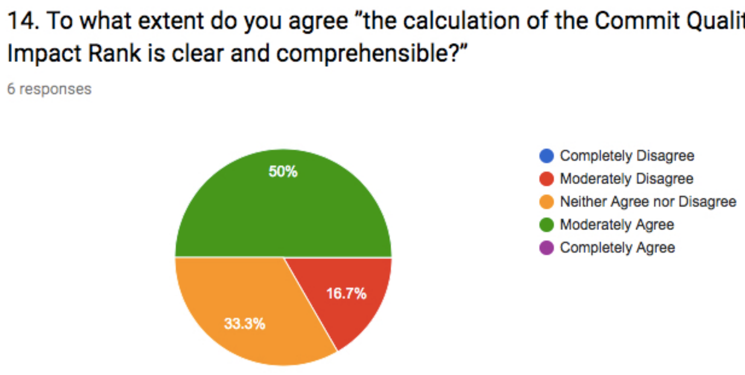


Figure 5.5: Response Question 14: CQI Report

Initial observation of the response breakdown indicates split opinion on the clarity of the model, with 50% of interviewees responding positively in agreement and 50% indicating a neutral of negative opinion regarding the clarity of the calculation. While only 1 interviewee (16.7% of total respondents) outright disagreed that the calculation of the CQI rank was clear and comprehensive, 2 interviewees (33.3%) indicated neither agreement nor disagreement with the understandibility of how the CQI rank was ultimately measured. Responses from these interviewees indicated opinion that the measure was too complex in that it "wants to capture a lot of different dimensions and put them into one value" as an undesirable trait and that how the "change in quality came to be is a bit unclear." Many references were made by interviewees, including those who were positive, that for some commits the results were very clear, however due to the "gold-plating" mechanic of the calculation some commits produced feedback which was more unclear than others.

The question "does the rank you receive match your expectations of the maintainability you produced with your contribution?" (question 16) highlighted this concern more concretely, showing uncertainty of the measure came mostly from the "gold-plating" penalty. Multiple interviewees brought up the issue of receiving feedback that they gold-plated in a commit, with one respondent finding the rank they received too "harsh" as they had added code "in [a] normal maintainable way" and was penalized, but that this shouldn't be an issue as "it improved the quality" of the system. However, when removing this aspect and only looking at those commits which did not have gold-plated feedback, the majority of interviewees agreed with the assessment given by the CQI report, with statements such as ”I would say I moderately agree, and actually I would say I would completely agree if I knew just a bit more about the tool” and ”There are some cardinal cases [where they disagree] but for the most part yes.” This would indicate that the main aspect of unclarity in the model is the manner in which a ”gold-plated” contribution is penalized.

As a means of investigating interviewee opinion on the ”gold-plating” aspect further, we first provided the definition of gold-plating as we define it in this research, namely the over-engineering of code past the point of being satisfactorily maintainable, and provided a summary of the nature in which gold-plating is measured in the MP model. We then asked the question ”to what extent do you agree the metric is fair in penalizing Gold-Plating when analyzing your contributions?” (question 17). This allowed us to observe the degree to which they understood the objectivity of measuring gold-plating and to what extent agree with the inclusion of the penalty in the Maintainable Production model.

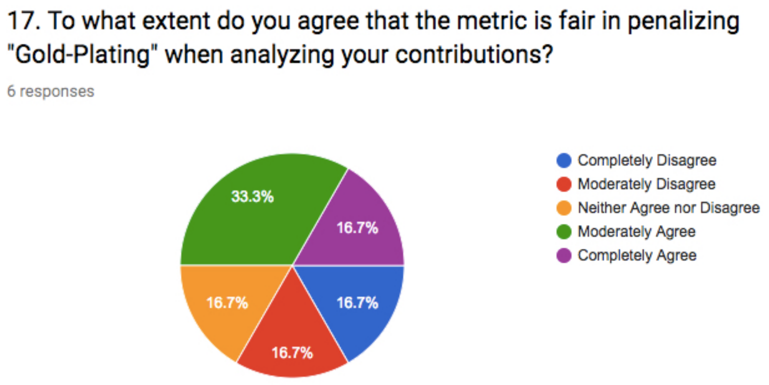


Figure 5.6: Caption

Again, we see a split opinion in the respondents with 50% in agreeance and 50% with a negative or neutral view on the fairness of the ”gold-plating” penalty. Removing the neutral responses we see 33.3% purely disagreeing that this aspect is fair, which indicates the ”gold-plating” measure is viewed slightly more positively than negatively, however with such a small sample group we can not come to a definitive conclusion on the fairness. Positive response indicated that the metric ”is showing what it is supposed to show... it is fair” while expressing concern ”that in specific situations, the impact that it gives may not be the ideal one. It lacks the nuance but it is definitely fair.” This was corroborated almost universally among respondents, with multiple statements indicating that while they believe the tracking of gold-plating ”is a good idea”, the manner in which it penalizes the overall CQI rank is ”way too harsh” for new feature development but is acceptable if the measure is analysing existing code that was simply refactored.

As a means of understanding this concern further, we can observe the responses from question 18 which asked ”the gold-plating feature attempts to highlight the interplay between productivity and software quality. Do you feel this metric accurately captures this aspect of software development?” Interviewee feedback indicated an almost universal positive response to this question, with responses ranging from ”yes, but should this be seen negatively? I think no” to ”yes, absolutely [it captures this relationship].” This indicates that the primary issue interviewees took concern with in the gold-plating measure is the degree to which this penalty affects the overall score. As all respondents agreed with the simple definition of gold-plating, the majority of feedback suggested that the model should separate refactored code which is gold-plated from new feature code, and that new code which is more maintainable than the original system should not be penalized.

In summary, we can conclude that the feedback provided over the Maintainable Production Model via the CQI report was only somewhat comprehensible, however this is primarily due to the gold-plating penalization attribute of the model. Interviewees pressed to only consider those commits which were not gold-plated found the model to be intuitive and matching their own expectations of the maintainability they delivered through their contributions. Regarding the gold-plating penalization, while they agreed with the definition of gold-plating provided and that the model accurately captures the interplay between developer productivity and software maintainability, interviewees found penalty too harsh in this iteration of the model and code contributions should be measured differently for "gold-plating" behavior depending if the contribution is solely a refactoring as opposed to new code development, or a mix of the two. From these findings, we can only conclude that the comprehensibility of the model cannot be definitively validated, and future work in adjusting the gold-plating penalization factor would be needed to fully validate the model under this aspect.

Actionability

As a means of assessing the actionability of the feedback provided by the CQI Report, interviewees were asked the questions 15 and 19 from the interview question list in **Table 5.2**. Question 15, "how does the CQI rank you receive affect your motivation going forward with development" was considered for actionability to gather a sense of how the feedback motivated the interviewee going forward with development, with the assumption being a less motivated developer would find it harder to proceed given the feedback over their Maintainable Production. In a similar vein question 18, "does the feedback provided have more, less, or the same amount of actionability going forward with development", to directly gauge the interviewees opinion on the perceived actionability of the feedback. Actionability in this sense is defined as the extent to which a developer knows how to proceed going forward with future development with respect to the maintainability of the system.

Responses to the question regarding the impact on motivation of the CQI Rank were slightly mixed. Most interviewees stated they would be "more motivated by a worse score" as this highlights areas in which they can improve, however one respondent made it clear that a negative feedback would be "a negative for my motivation and I would stop looking at it", however this was said mostly in the context of the gold-plating penalty returning a negative feedback. When asked if the gold-plating were not to be taken into account with the score, only mentioned as a side note, this interviewee agreed this would motivate them differently. Positive scores were seen as less motivating but still useful for most interviewees as a means of getting an "at glance view of where they are currently", but this feedback would not necessarily change their development habits. This indicates that indeed the metric has the ability to motivate developers to some degree, however the extent of which is dependent on the individual interviewee.

With regards to the question "Does the feedback have more, less, or the same amount of actionability as BCH currently does?", the general consensus among respondents was that the feedback provides roughly the same actionability as BCH if not more, however gold-plating feedback itself was seen as non-actionable. Interviewees felt that by receiving the feedback that a guideline was "gold-plated", they were lost as to the manner in which they correct this with statements asking "should I then make the code worse?" or if they should "ignore it and make the code less maintainable on the next commit." This is an unintended response to the gold-plating penalization, as its true purpose is inform the developer when they should stop increasing the maintainability of their code. This again indicates the lack of expressiveness found in the current iteration of the gold-plating aspect to the Maintainable Production Model, suggesting again that future work should look to refining the manner in which this penalization affects to overall CQI rank of a commit. Feedback that was void of the gold-plating penalty was found to be intuitive and clearly show how to proceed for the interviewees, indicating that this again is a contained issue regarding solely the "gold-plating" factor.

These findings suggest that the metric is at least as actionable as Better Code Hub itself, with those respondents stating it was more actionable particularly found the feedback to be "more far-reaching" and encourages more discussion at the individual developer level over the trade-off between productivity and maintainability for a given project. However, due to the perceived in-actionability found in the "gold-plating" feedback, the model as it currently exists cannot be definitively validated as a means of improving the actionability over BCH.

5.3 Discussion

As a first step in the validation process, this initial lab testing produced promising findings and the first steps in revising the Maintainable Production model. While the general principles of the model were found to be quite clear and understandable as well as actionable for the most part, the gold-plating penalization was observed to be a point of criticism for all interviewees involved. What is most interesting is that while all interviewees agreed with the basic definition of "gold-plating your code", and that the model was accurate in highlighting the interplay between productivity and maintainability, they felt that they should not be penalized as often, and in the case of one of the interviewees not at all, for gold-plating within their contribution. With respect to **Research Question 3**, "Can we provide feedback over maintainable code production at the commit level which is 1) coherent and 2) actionable?", these findings indicate an inconclusive result as we could only provide a partial confirmation for both the extent to which the feedback is coherent and the extent to which it is actionable due to the negative impact of the "Gold-plating" penalty on both of these aspects.

The first major point of interest for revising the MP model for future work is the necessity to categorize the contributed code as either modified existing code or new code, and measure gold-plating differently between the two. Interviewees expressed preference for only having the code which existed previously in the system and was modified under a commit to be measured for gold-plating, while new code should only be measured for the change in maintainability it introduced. The second point of interest for revising a the model in future work is the removal of the penalization factor in the overall CQI score, and simply state if the contribution gold-plated any of the guidelines or not. This would potentially alleviate both the comprehensibility and actionability concerns reported by interviewees while still tracking this aspect of productivity, as interviewees found this information useful but did not agree with degree to which it impacted their score.

We must reiterate that these findings are not intended to suggest the Maintainable Production Model is in any way fully validated through this interview process. These interviews were conducted as a means of testing the level to which the Maintainable Production Model makes sense with respect to the knowledge these interviews have on software maintainability in a laboratory setting. The TAR research method would suggest, and it is our recommendation as well, a move to external testing in a "real-world" environment to cross-validate the findings in this research before any conclusive validation over the model can be made. However, we feel the observed findings are a sufficient first step in the validation process and provided important recommendations to take into account for the next iteration of the Maintainable Production Model presented in this research.

5.4 Threats to Validity

- Interviewees were selected internally at SIG and in many cases have worked on Better Code Hub itself. This shows an inherent bias when questioning over specific aspects of Better Code Hub and the manner in which it measures software maintainability. We attempted to mitigate this by formulating the questions to introduce as minimal bias as possible and pressing interviewees firmly over the opinions held. We also provide keen discussion as a means of alleviating this bias in our analysis.
- The sampling of interviewees was very small in this research. This is due in part to the smaller development team at SIG and the projects in which they are working on. We decided to only use those developers / consultants who worked on systems which could be analyzed through Better Code Hub.
- The criteria for which the validation aspects for comprehensibility and actionability is based on the interpretation of the sole researcher, who may have attempted to show the Maintainable Production Model in a favorable manner. We attempted to mitigate this bias through keen discussion over each question asked relevant to these validation aspects and utilized only statements gathered from the interviewees.
- The majority of interviewee's were contributing to internal projects at SIG which already had a very high compliance rating through BCH, with the lowest receiving an 8 out of 10. As these projects are of such high compliance, the change in guidelines as a result of a developer's contribution will often be only rated neutrally or negatively, as "gold-plating" is penalized in the event the developer's improve compliant guidelines. This is a limitation brought on by the internal testing conducted. We attempted to mitigate this threat through scenario based questioning when discussing the ranking system with interviewees.

Chapter 6

Conclusion

In this research we set out to construct a model for measuring the maintainability change incurred by a system per the amount of code change introduced to the system by a single contribution by a developer. Software maintainability is seen as one of the most important factors regarding project success in the long term together with productivity, but a formal way of measuring this interplay has eluded the software engineering community. Our goal with this research was to measure maintainability change of a system over successive contributions from a developer in order to provide clear and actionable feedback to the developer over their maintainable code production.

We utilized Better Code Hub as the tool in which we measure the maintainability of a system through its measure of 10 individual code metrics, referred to as the 10 guidelines of future-proof code. We constructed a methodology for selecting the relevant guidelines based on robustness and granularity of the measure per guideline. This resulting selection of guidelines were utilized to measure the change in maintainability resulting from an individual developer's contribution to a project. Developer's contributions were measured as the volume of churned code in proportion to the overall system size. Once acquired, we mapped these two measures as a function of productivity, with the developer's contribution churn percentage as the production input while the change in maintainability as the output of the production allowing for the effective measure of maintainable code production within a developer's contribution. This production was then ranked against a benchmark analysis of Maintainable Production over 40 different software projects.

As a means of moving toward validation of this model, we also constructed a specialized feedback report within an experimental branch of Better Code Hub in which developers could review characteristics of Maintainable Production for a selected commit, which we call the Commit Quality Impact Report. This CQI report contained a breakdown of the measures used in calculating the overall rank of Maintainable Production, referred to in the report as the Commit Quality Impact Rank. Careful attention was given to the gold-plating penalization factor, which attempted to highlight the trade-off between continuously improving maintainability and developer productivity. Our goal in this phase of the research was to determine the overall comprehensibility and actionability of the feedback produced by the Maintainable Production Model, which we tested through a series of interviews with domain-experts in a lab environment.

The findings from this research indicate that while the Maintainable Production Model produces feedback which is mostly understandable and provides at least the same level of actionability as Better Code Hub itself, the gold-plating penalization factor introduces a great deal of uncertainty when evaluated the final rank of Maintainable Production. While all interviewees agreed that "gold-plating" is an issue to track and inform the developer over, the manner in which it penalizes the overall rank of Maintainable Production is too harsh and not robust enough. Overall, the model was found to be a beneficial addition to Better Code Hub, with promise for future iterations to mitigate the issues found in the "gold-plating" aspect of the model.

6.1 Research Question Answers

Regarding the research questions posed in **Section 1.2.1** of this work, we were able to provide the following answers:

6.1.1 Research Question 1

By what measures can we evaluate a developer's code contribution to a project as maintainable code production?

Answer: We measure maintainable code production as the rate of maintainability change or the $\Delta Maintainability$, measured over a subset of BCH's guidelines, incurred on the system as the result of the percentage of the system that was changed, measured as the $LOC_{churned} \div LOC_{system}$. This rate is referred to as the Maintainable Production of the developer.

6.1.2 Research Question 2

By what means can we evaluate the impact of a developer's maintainable code production to the maintainability of a given project?

Answer: We evaluate the impact of a developer's maintainable code production as the observed CQI Rank of a developer's contribution as it compares to the benchmark analysis of Maintainable Production. This rank is obtained through classification of the Maintainable Production through a quantile split of the benchmark distribution following a 5-30-30-30-5 scheme.

6.1.3 Research Question 3

Can we provide feedback over maintainable code production at the commit level which is comprehensive?

1) *Is the feedback coherent for developers?*

2) *Is the feedback actionable for developers?*

Answer: The experiment to test extent to which the feedback was found to be 1) coherent and 2) actionable was found to be inconclusive on both accounts as we can only provide a partial confirmation through the interviews with domain-experts. This was primarily due to the extent at which "Gold-Plated" code influenced the CQI Rank negatively was found to be too severe. Further iterations on the model are needed to fully validate the feedback contained in the CQI report over these aspects.

6.2 Future Work

The "gold-plating" penalization factor of the model caused the most concern during the interview process, indicating a strong need to iterate on this aspect for future work on the Maintainable Production Model. Some suggestions include a revision to the manner in which developer contributions are measured. Interviewees felt that only contributions which include refactored code should be subject to the "gold-plating" penalization, while new code should only be measured for the change in maintainability that is being introduced through the contribution. This change could be seen as removing the objectivity of the model, but makes sense in the context that a developer should not be motivated to make their new code worse as a means of receiving a better rank. An additional change suggested during the interview process was to remove the gold-plating penalization in the calculation of the rank all together and simply report if a guideline was gold-plated or not. This would require a new bench-marking of data, however this change could make strides to aligning developer expectations with the Maintainable Production rank they actually receive.

Apart for the gold-plating penalization, future work should move toward formally validating the model through empirical analysis. The findings in this research were qualitative based and did not explore the statistical significance of the Maintainable Production score in great detail. Further research should also look in exploring the Maintainable Production Model in an external environment for testing as the next step from the internal environment test conducted in this research.

Lastly, one could consider using a different analysis tool for measuring maintainability of a system. This research relied purely on Better Code Hub as it contained aspects which were found to be most optimal, namely the guideline approach to the tool and the defined points of satisfactory maintainability per guideline. It is not the case that this model relies on Better Code Hub, in fact any process for which a software maintainability delta can be acquired can be used in the model as long as relevant thresholds of compliance can be defined. This would allow this iteration to be used as a baseline for which any future iteration of the Maintainable Production model can be compared.

Acknowledgements

First of all, I would like to thank my family and friends for supporting me throughout all my years of study. I want to give special thanks to my partner, **Marit K.**, for sticking with me through the good times and the bad, for her unconditional support, and for never giving up on me through this whole experience.

I want to give a massive thanks to my university supervisor, **dr. Ana Oprescu**, for giving me clear guidance and pushing me to give my best during this research. Thank you for the support and the confidence to make it through this thesis, for pushing me to go to *CompSys 2018* even though I was damn reluctant, and for all the fun and laughs we shared along the way.

I want to thank the whole of SIG for welcoming me into the company as an intern. A very special thanks to **dr. Magiel Bruntink** for acting as my secondary supervisor within SIG, your support was instrumental in the completion of this research and I appreciate the time you took out of your busy schedule to ensure I had the guidance and the proper amount of challenge to achieve the goals we defined at the beginning of this project.

To my organization host and product owner of Better Code Hub, **Michiel Cuijpers**, I want to also extend my wholehearted appreciation for your support during my time with SIG. I thoroughly enjoyed our discussions over the impact and future of Better Code Hub, what it means to produce maintainable code, and the degree of confidence you showed in me and my work. I also want to thank you for allowing me to attend the Beyond-Banking hackathon as a representative of SIG, it was a great deal of fun.

Thank you to all in the research department of SIG for welcoming me into the group as an equal and making it a fun and educational experience. I want to give a special thanks to **Marco di Biase**, **Cuiting Chen**, and **Enrique Larios Vargas** for assisting me throughout my research and bringing new perspectives to my work which I may have otherwise missed.

Equally so, I want to thank the entire SIG development team for assistance in implementing my experiments into Better Code Hub and for their participation in my work. Special thanks to **Mircea Cadariu** for walking me through the Better Code Hub pipeline and helping me sort out my data-binding issues I kept running into! Thank you **Thomas Kraus** for the keen feedback on the code I produced as part of my experiment and for the pleasant discussions we shared over my research.

Thank you also to my second reader, **dr. Ana Varbanescu**, for being so flexible and accommodating in reviewing my work and for the helpful feedback you provided at *CompSys 2018*!

I want to also thank the rest of the guys that interned with me at SIG: Simon Schneider, Edwin Ouwehand, Mees Kalf, Cornelius Ries and Wojciech Czabanski! Thank you all for the support and good times we shared together.

And last but not least, I want to thank all of the students and professors who participated in the MSc. Software Engineering class of 2017-2018! It was a hell of an experience and I wish you all the very best going forward!

Bibliography

[1] Better code hub. URL: https://bettercodehub.com/.

[2] Github api for java - kohsuke kawaguchi. URL: http://github-api.kohsuke.org/.

[3] Angularjs developer guide, 2018. URL: https://docs.angularjs.org/guide/concepts/.

[4] Github graphql api v4, 2018. URL: https://developer.github.com/v4/.

[5] How gpa is calculated, 2018. URL: https://nces.ed.gov/nationsreportcard/hsts/howgpa.aspx.

[6] Java 8, 2018. URL: http://www.oracle.com/technetwork/java/javase/.

[7] maintainability, n., 2018. URL: https://www.thefreedictionary.com/maintainability.

[8] Mongodb for giant ideas, 2018. URL: https://www.mongodb.com/.

[9] productivity, n., 2018. URL: http://www.oed.com.

[10] Typescript - javascript that scales., 2018. URL: https://www.typescriptlang.org/.

[11] A. J. Albrecht and J. E. Gaffney. Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Transactions on Software Engineering*, SE-9(6):639–648, Nov 1983.

[12] Donald Anselmo and Henry Ledgard. Measuring productivity in the software industry. *Communications of the ACM*, 46(11):121–125, 2003.

[13] Alex Antonov. *Spring Boot Cookbook*. Packt Publishing, 2015.

[14] Gerald M Berns. Assessing software maintainability. *Communications of the ACM*, 27(1):14–23, 1984.

[15] Barry W Boehm. Improving software productivity. In *Computer*. Citeseer, 1987.

[16] Barry W Boehm, Ray Madachy, Bert Steece, et al. *Software cost estimation with Cocomo II with Cdrom*. Prentice Hall PTR, 2000.

[17] Bordoloi, Bijoy, and Luchetski. Software metrics: Quantifying and analyzing software for total quality management. In *Systems Development Handbook. 4 ed.* CRC Press LLC, 2000.

[18] Alan Bryman and Emma Bell. *Business research methods*. Oxford University Press, USA, 2015.

[19] David Card. The challenge of productivity measurement. In *Proceedings: Pacific Northwest Software Quality Conference*, 01 2006.

[20] I Robert Chiang and Vijay S Mookerjee. Improving software team productivity. *Communications of the ACM*, 47(5):89–93, 2004.

[21] NINTH EDITION, Howard Anton, Chris Rorres, Christine Black, Blaise DeSesa, Molly K Gregas, Elizabeth M. Grobe, and Charles A. Grobe. Elementary linear algebra with applications. 2008.

[22] Lile P Hattori and Michele Lanza. On the nature of commits. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages III–63. IEEE Press, 2008.

[23] Ilja Heitlager, Tobias Kuipers, and Joost Visser. A practical model for measuring maintainability. In *null*, pages 30–39. IEEE, 2007.

[24] Israel Herraiz, Gregorio Robles, Jesús M González-Barahona, Andrea Capiluppi, and Juan F Ramil. Comparison between slocs and number of files as size metrics for software evolution analysis. In *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*, pages 8–pp. IEEE, 2006.

[25] ISO/IEC. Iso/iec 25010 system and software quality models. Technical report, 2010.

[26] Leonard J Kazmier. *Schaum's Easy Outline of Business Statistics.* McGraw-Hill Trade, 2003.

[27] Barbara Kitchenham and Shari Lawrence Pfleeger. Software quality: the elusive target [special issues section]. *IEEE software*, 13(1):12–21, 1996.

[28] Thomas J McCabe. A complexity measure. *IEEE Transactions on software Engineering*, (4):308–320, 1976.

[29] Andrew Meneely, Ben Smith, and Laurie Williams. Validating software metrics: A spectrum of philosophies. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 21(4):24, 2012.

[30] André N Meyer, Laura E Barton, Gail C Murphy, Thomas Zimmermann, and Thomas Fritz. The work life of developers: Activities, switches and perceived productivity. *IEEE Transactions on Software Engineering*, 43(12):1178–1193, 2017.

[31] Vu Nguyen, Sophia Deeds-Rubin, Thomas Tan, and Barry W. Boehm. A sloc counting standard. 2007.

[32] Ranjith Purushothaman and Dewayne E Perry. Toward understanding the rhetoric of small source code changes. *IEEE Transactions on Software Engineering*, 31(6):511–526, 2005.

[33] John M Roche. Software metrics and measurement principles. *ACM SIGSOFT Software Engineering Notes*, 19(1):77–85, 1994.

[34] J. Rosenberg. Some misconceptions about lines of code. In *Proceedings Fourth International Software Metrics Symposium*, pages 137–142, Nov 1997.

[35] KP Srinivasan and T Devi. A novel software metrics and software coding measurement in software engineering. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(1):303–308, 2014.

[36] KP Srinivasan and T Devi. Software metrics validation methodologies in software engineering. *International Journal of Software Engineering & Applications*, 5(6):87, 2014.

[37] C. R. Symons. Function point analysis: difficulties and improvements. *IEEE Transactions on Software Engineering*, 14(1):2–11, Jan 1988.

[38] Joost Visser, Sylvan Rigal, Rob van der Leek, Pascal van Eck, and Gijs Wijnholds. *Building Maintainable Software, Java Edition: Ten Guidelines for Future-Proof Code.* O'Reilly Media, Inc., 1st edition, 2016.

[39] Roel Wieringa and Ayşe Moralı. Technical action research as a validation method in information systems design science. In *International Conference on Design Science Research in Information Systems*, pages 220–238. Springer, 2012.

Acronyms

BCH Better Code Hub.

CI Continuous Integration.

CQI Commit Quality Impact.

LOC Lines of Code.

MP Maintainable Production.

SIG Software Improvement Group.

TAR Technical Action Research.

UI User Interface.

Appendices

Appendix A

Benchmark Projects

1. WeblateOrg/weblate
2. storybooks/storybook
3. bokeh/bokeh
4. QualitySoftwareDeveloper/test-repo
5. powerunit/powerunit-extensions-matchers
6. RocketChat/Rocket.Chat.ReactNative
7. CommandPost/CommandPost
8. ICTU/quality-report
9. contentmonkey/contentmonkey
10. NLog/NLog
11. wulkanowy/wulkanowy
12. erxes/erxes
13. Zwinne2018Recruiter/Zwinne_Recruiter.io
14. rualark/MGen
15. lodygens/xtremweb-hep
16. nnadeau/pybotics
17. ILSCeV/Lara
18. TheDemocracyFoundation/epitome
19. tasks-delivery/front-editor
20. artyomsv/board-web
21. ImpressCMS/impresscms
22. cirept/QA_Toolbox
23. wongjiahau/ttap-web
24. Blockchaingers/consaint

25. joskuijpers/bep_codefeedr
26. fszlin/certes
27. OpenMTR/OpenMTR
28. DimaStoyanov/Monopoly
29. raynode/nx-logger
30. seanst13/AC40001-Honours-Project
31. spectre-team/native-algorithms
32. WLM1ke/PortfolioOptimizer
33. QoboLtd/project-template-cakephp
34. jsim2010/papyr
35. JoeKarlsson/bechdel-test
36. Techprimate/TPPDF
37. salesforce/CustomerSuccess
38. kamejosh/waffle
39. Pr0methean/BetterRandom
40. Software-Improvement-Group/BetterCodeHub-Edge

Appendix B

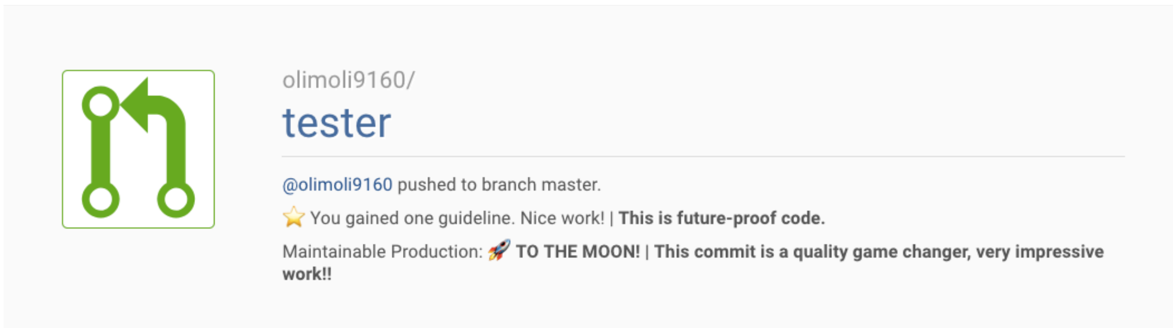
ISO 25010 Maintainability Definition for Interviews

Maintainability - According to ISO 25010, software maintainability represents the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements. This characteristic is composed of the following sub-characteristics:

- **Modularity.** Degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.
- **Reusability.** Degree to which an asset can be used in more than one system, or in building other assets.
- **Analysability.** Degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified.
- **Modifiability.** Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality.
- **Testability.** Degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met.

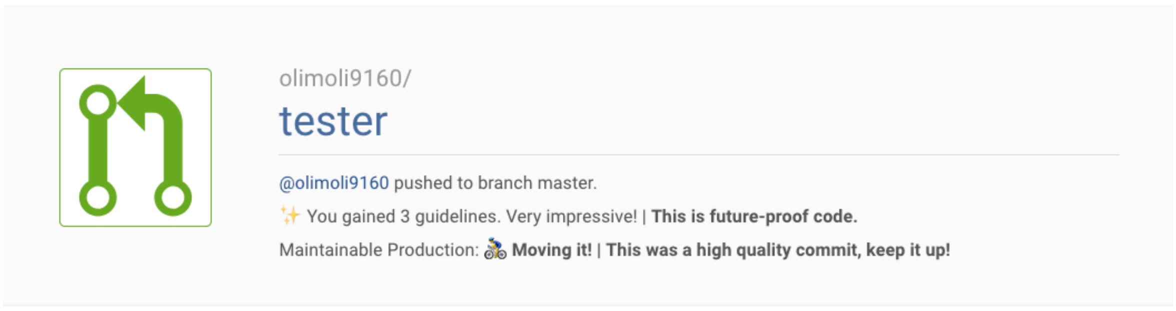
Appendix C

CQI Report Tagline Examples


olimoli9160/
tester

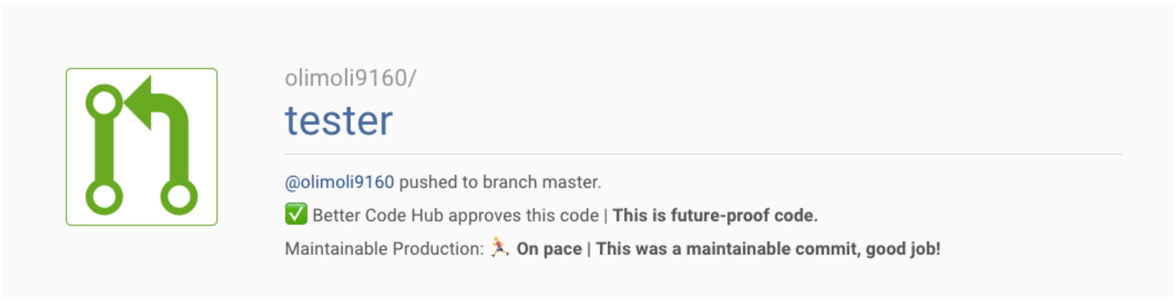
@olimoli9160 pushed to branch master.
⭐ You gained one guideline. Nice work! | **This is future-proof code.**
Maintainable Production: 🚀 **TO THE MOON! | This commit is a quality game changer, very impressive work!!**

Figure C.1: Tagline with CQI Rank A


olimoli9160/
tester

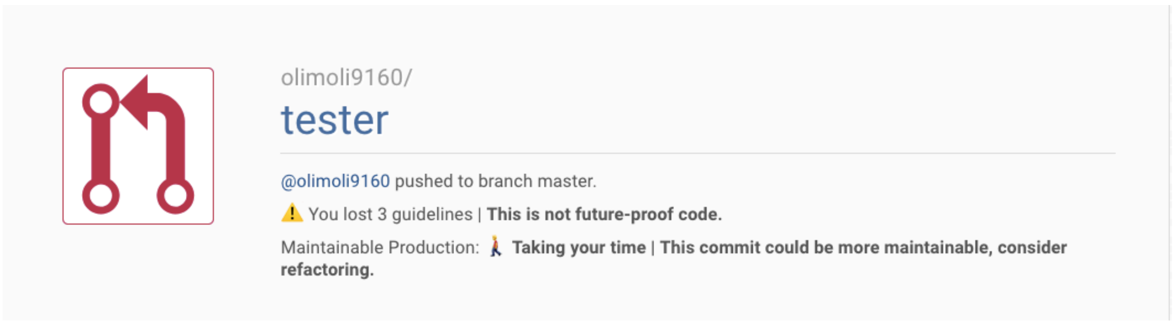
@olimoli9160 pushed to branch master.
✨ You gained 3 guidelines. Very impressive! | **This is future-proof code.**
Maintainable Production: 🚴 **Moving it! | This was a high quality commit, keep it up!**

Figure C.2: Tagline with CQI Rank B


olimoli9160/
tester

@olimoli9160 pushed to branch master.
✅ Better Code Hub approves this code | **This is future-proof code.**
Maintainable Production: 🏃 **On pace | This was a maintainable commit, good job!**

Figure C.3: Tagline with CQI Rank C


olimoli9160/

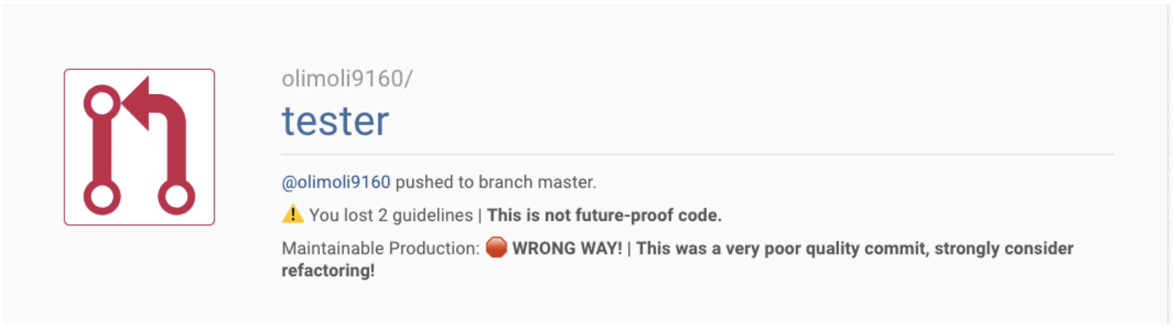
tester

@olimoli9160 pushed to branch master.

⚠️ You lost 3 guidelines | **This is not future-proof code.**

Maintainable Production: 🚶 **Taking your time | This commit could be more maintainable, consider refactoring.**

Figure C.4: Tagline with CQI Rank D


olimoli9160/

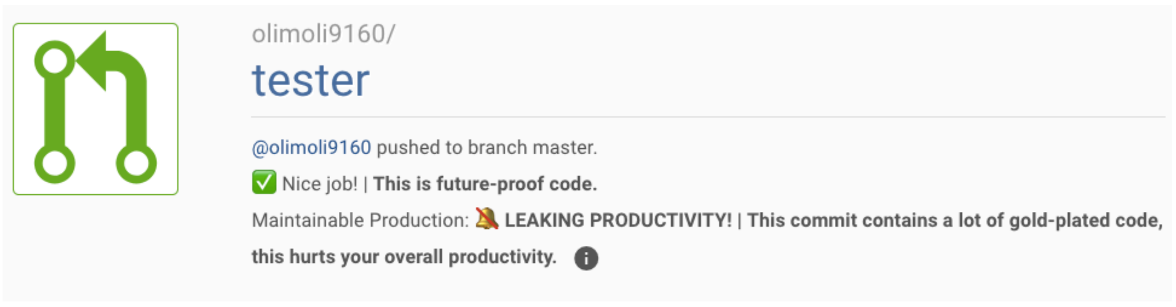
tester

@olimoli9160 pushed to branch master.

⚠️ You lost 2 guidelines | **This is not future-proof code.**

Maintainable Production: 🛑 **WRONG WAY! | This was a very poor quality commit, strongly consider refactoring!**

Figure C.5: Tagline with CQI Rank F


olimoli9160/

tester

@olimoli9160 pushed to branch master.

✅ Nice job! | **This is future-proof code.**

Maintainable Production: 🔕 **LEAKING PRODUCTIVITY! | This commit contains a lot of gold-plated code, this hurts your overall productivity.** ℹ

Figure C.6: Tagline indicating gold-plating with CQI Rank C-F

Appendix D

Interview Transcriptions

D.1 Interview - Full Stack Developer / Front End Specialist

Author:	What is your current role in software development?
Full Stack Dev 1:	I'm a Front End Developer. My official title here is Full Stack Developer but I only work on the front end with AngularJS and Typescript.
Author:	How many years of experience do you have doing software development professionally?
Full Stack Dev 1:	About 1.5 years now I think.
Author:	What is the typical type of project you have developed on? Enterprise, mobile, web, SaaS?
Full Stack Dev 1:	Just web applications that use Angular at SIG.
Author:	What is your educational background?
Full Stack Dev 1:	I have a Bachelor in Mathematics, a Master Degree in Computer Science, and I don't quite have a PHD because you know you run out of money and time and you have to do a real job haha.
Author:	Yeah, haha it's a big commitment right? You're still working on it.
Full Stack Dev 1:	Yes, I hope to finish it sometime. I'm not working on it currently but it is there, looming over me constantly.
Author:	So I'm going to ask you some questions over maintainability and this ISO 25010 standard. Do you need time to look over that?
Full Stack Dev 1:	I should know it because I used to be a consultant also but its a bit fuzzy on the details.
Author:	Yeah no problem, I'll give you a bit of time to look over it now.
Full Stack Dev 1:	Ok, yeah.
Author:	Ready? Alright so my first question is, to what extent do you agree with the ISO 25010 definition of maintainability with regards to software quality?
Full Stack Dev 1:	The overall definition? Yeah it's hard to give an answer with this scale because its somewhere between a 4 (Moderately Agree) or 5 (Completely Agree) as there are some things I don't really agree with, but I mostly agree with it.

Author: So you mostly agree (4)?

Full Stack Dev 1: Yeah.

Author: Ok, what do you disagree with as that is more interesting.

Full Stack Dev 1: It's hard to put into words. Sometimes trying to make things really maintainable according to this can actually make the implementation more complex, even though according to this scale the complexity isn't very high. You end up trying to break things down into blocks and you have to take into consideration edge cases you have to deal with and all that stuff. It can make things so much more complicated for the myself but also a future maintainer.

Author: You feel like this forces you to micromanage too much?

Full Stack Dev 1: Yes, yes.

Author: Ok but barring that you still mostly agree with the definition?

Full Stack Dev 1: Yeah

Author: Ok great uh keeping that in mine, so to what extent is it important to you to produce maintainable code when contributing to a system?

Full Stack Dev 1: It's important in the sense that I myself enjoy it a lot more to work in code that has small units and units that are not complex where making a little change here doesn't impact surprising parts of the application because you didn't even know they were connected. So in that sense it is important. I think people can only keep a certain amount of stuff in their head and so if I'm reading a unit and I'm working on it, and it does A B C D, and I need to change B I don't need to know about the others. I just want to go into the function that only deals with B, and not the other stuff. I tend to try to keep things small so you can understand and control the impact of your changes.

Author: To what extent is it important that you feel you are continuously improving your own development efforts as far as quality is concerned?

Full Stack Dev 1: Again 4 (Moderately Agree) or 5 (Completely Agree). It is important to me but only to a certain extent, it should not.. you shouldn't do maintainability for maintainability's sake. I'm not really consciously thinking about this definition (ISO 25010) but I always review my own code before I give the merge request to someone else to see if it creates too much confusions or too many layers of abstraction. Its more for me, I don't like long units or complex units so I have a tendency to follow these rules but I'm not consciously thinking of them.

Author: I'd like to talk to you about Better Code Hub now. You've worked on Better Code Hub, you're a bit "intimate" with it you could say. Better Code Hub presents itself as a tool that can measure how maintainable a system is according to a list of 10 code aspect guidelines. Do you feel that the tool is accurate in this measurement? Does the analysis fit the ISO 25010 standard of maintainability?

Full Stack Dev 1: I think it's sufficiently accurate, in that I think the part you could go overboard (with maintainability), it tries to protect you from that by saying "doesn't have to be perfect, it just has to be good enough." I don't look at it very often.

Author: Ah that was my follow up questions, how often do you visit Better Code Hub itself to view the feedback it provides over the project you are working on?

Full Stack Dev 1: No, I just look at the merge request and use my own judgment to determine if it is maintainable, are these long units, are they complex, do I see circular dependency has been created somewhere. Is some knowledge leaking from another thing. In that sense I don't use Better Code Hub actively to kind of look at my commits.

Author: Let's say you have a commit which is of questionable quality, would you actually check Better Code Hub for this? Or would you just do your own code review?

Full Stack Dev 1: Umm that hasn't happened yet haha, yeah.

Author: Ok, that's a valid answer as well haha. So we're trying to see a bit I guess of how Better Code Hub's feedback sits with you, so is.. is the feedback provided useful in determining how maintainable the project being developed is and how your changes have affected the maintainability of the project?

Full Stack Dev 1: The thing with (Internal SIG project) is that it has had a 10 out of 10 for a very long time. I think if someone told me it dropped from a 10 out of 10 to a 9 (in Better Code Hub), I would start looking at it.

Author: So the score itself is very important to you, as when it drops thats when you want to go look at Better Code Hub to see what happened? That's what is most important to you?

Full Stack Dev 1: Yes! But that's also if it was not 10 out of 10 I would look at it more to... if a guideline was not compliant to see what can I do here in context of what I'm working on because outside of that you're not... you're not doing anything useful. Making something more maintainable by itself isn't particularly useful.

Author: Going back to one of your earlier responses you gave about Better Code Hub giving you this nice point of being "done" with maintainability, you don't need to keep working further.. this is something you value most from the feedback Better Code Hub gives you.

Full Stack Dev 1: Yes.

Author: Ok great, so... if a project were to drop a guideline, do you think the feedback is actionable, in that you know what improvements you yourself can make in the software to increase its maintainability?

Full Stack Dev 1: So the problem is that we work with (internal monitor) and act more on what is reported from that and we don't act on what Better Code Hub tells us.

Author: So if we move away from SIG projects, if you have a personal project would you use Better Code Hub with it?

Full Stack Dev 1: I might, I might if I want to put my code up on GitHub in public.

Author: So in that sense if you were to use it, do you think the feedback is actionable from Better Code Hub? Do you know what to do with it?

Full Stack Dev 1: Yes but I have inside information. I know what I can do if I have a bad guideline to improve it, without Better Code Hub telling me. If I try to remove that bias I think... well particularly for some guidelines some insight would be more helpful, particularly the ones that have to do with separation of concerns and stuff.. those are a bit magical.

Author: Yeah so you mean the architectural guidelines?

Full Stack Dev 1: Yeah those are more difficult to pin down.

Author: Ok so let me ask you are you satisfied with the feedback Better Code Hub currently gives?

Full Stack Dev 1: Well yeah, I don't really look at it so I'm fine with it.

Author: To what extent do you agree with the statement If I did not use Better Code Hub, the quality of my software would be the same?

Full Stack Dev 1: I completely agree, if I didn't use Better Code Hub my quality would be the same.

Author: Completely agree, would you like to expand on that?

Full Stack Dev 1: Yeah because when I look at it, I myself like to have the small units and the nice structure and I don't like cyclic dependencies so its.. maybe because I became a developer at SIG and it fits my personality I do this without the feedback from Better Code Hub. I do believe there are people for whom this feedback is useful though, especially students just starting.

Author: To what extent do you agree that the calculation of the Commit Quality Impact Rank is clear and comprehensible?

Full Stack Dev 1: Ahh I go with 2 (moderately disagree)

Author: Ok you say moderately disagree, what is not easy to understand?

Full Stack Dev 1: So there is this stuff about percentiles and stuff, as a mathematician I get it.. but you know, when I first opened this as a user my thought was "whoa that's a lot of information, what does it all mean" and I thought, oh this is still in the realm of research. I didn't immediately feel I can identify with this with what I did in my commits.

Author: Right so if you strip away some of this UI stuff and the reason for all this text being there is to give a summary of what this metric means and everything.. is it understandable the underlying formula that gives you this rank?

Full Stack Dev 1: Not so much. I mean I saw your proposal some time ago explaining everything but I didn't read it, so that's my fault.

Author: Ah its ok, no worries!

Full Stack Dev 1: Yeah so that.. it's a bit difficult. My sense is that you want to capture a lot of different dimensions and put them into one value.

Author: Yeah it's quite complex. So basically we're looking at two builds in relation and how close you move to this kind of quality point of being done. So when you're dealing with Non-compliant code it makes sense right, because if you improve code you move closer so the distance is less, and the reverse is true if you make code worse you make the distance greater. If you're on the compliant side of the spectrum here its a bit different. You actually brought this up earlier when you talked about the point of being done, in that you should not focus on maintainability for maintainability's sake. So if you're moving from build A to B, you've improve the code but we penalize you for that, as you moved away from the definition of done and so you actually get a negative score here.

Full Stack Dev 1: Yeah I think that's where I kind of... so I made a change that was your basic functionality kind of stuff. I had made it in my normal maintainable way and it tells me I was gold-plating. And I was like "NO, I was adding a new feature!" and the feature I added was maintainable so it improved the quality.

Author: Right so if you remove the idea of functionality there and only look at maintainability, the metric is objective and harsh as it only tells you what you've done, and in this case you have improved the quality past their point of being done so you have indeed gold-plated. It's harsh because it ranks gold-plated just as bad as non-compliant code. But its trying to give a sense that you need to focus your development efforts elsewhere as you're leaking productivity, but it might be the case that you created a new feature and you weren't concerned about maintainability, and the quality of the code was so good that it is going to give you this gold-plated penalty.

Full Stack Dev 1: Right so the code you've added is qualitatively better than what you had, or even the same but because there is more of it, the rating is improved and then you get "oh you're gold-plating and you're leaking productivity." and I think "leaking productivity? I've been productive as hell!" So I think the kind of assumption in Better Code Hub, and maybe in this, is that any change you're making is for the sake of maintainability! But you're not always doing this!

Author: Yeah indeed

Full Stack Dev 1: So if its already compliant, and you're making it more compliant, that is not useful. But if that happens, it happens because you happen to add new code of higher quality.. and then that shouldn't be considered gold-plating as you're not making it prettier than it already is. I get it to a certain extent but not every positive change to a compliant guideline is gold-plating.

Author: So let me ask you then how does this rank you receive affect your motivation? So if you get a D and it says gold-plating what is your response to that?

Full Stack Dev 1: Well I think I wasn't deliberatively gold-plating, I was fixing a bug and adding a test so that wouldn't happen again, like what did I do wrong? I didn't do anything wrong!

Author: So would you say it negatively affects your motivation or would you just not want to look at this?

Full Stack Dev 1: It would negatively, it would be negative for my motivation and I would stop looking at it.

Author: Right. So you're currently working on systems which are highly rated, 9 out of 10 and 10 out of 10, how can you not gold-plate right? But can you imagine a developer not working on such a good system, 3 out of 10 or something, do you think the metric would work better in that situation?

Full Stack Dev 1: Yeah I think it would help as it tells you how your moving closer to this definition of done and yeah... I think if you're working on something that is not there yet it definitely helps. It's just from the otherside (very compliant) that it's frustrating.

Author: Right, so its kind of a combined feedback from Better Code Hub. Maybe you got a D here but Better Code Hub tells you this is future-proof code and they approve of it. So it's not really supposed to be a replacement but an addition. Do you see value in this sort of feedback?

Full Stack Dev 1: Yeah I think it should be focused on things that are not there yet (as far as compliance). Or things that have gone from compliant to non-compliant, but things that are compliant to just not worry about this.

Author: So If you took away the gold-plating and only said "you've improved this guideline" and its a positive change, would that motivate you differently?

Full Stack Dev 1: Yes, yes. The thing is when people say "oh, you're gold-plating" that is a negative thing, and its red so its also saying its not nice and that you're doing things you should not be doing.

Author: So if you weren't informed about gold-plating do you think you would come to a natural stopping point yourself when it comes to maintainability without getting all this feedback?

Full Stack Dev 1: So I wouldn't work on maintainability for maintainability's sake, I would probably as long as a guideline is not compliant try to just take it with my improvements, as long as the unit is within my context. So even if it is already compliant and I find a unit in a file that I'm working on that is overly long, I may try to break it down because I think this is not nice and I have to read too many lines, so in that sense I would continue to work on the maintainability.

Author: But you're saying its not a priority, you're not worried about reaching this point where you are constantly gold-plating and wasting productivity as it isn't an issue for you?

Full Stack Dev 1: No, its not.

Author: So let me ask you this, to what extent do you agree that the metric is fair in penalizing Gold-Plating when analyzing your contributions?

Full Stack Dev 1: I don't think you're surprised if I say I completely disagree here right?

Author: No, no I'm not haha.

Author: So the Gold-plating feature attempts to highlight the interplay between productivity and software quality. Do you feel this metric accurately captures this aspect of development?

Full Stack Dev 1: It captures if you are writing code that is higher quality than the system already is, if that's gold-plating then yeah it does. But is that a negative thing? I'd say no.

Author: Negative for the developer or for the system you mean, to clarify?

Full Stack Dev 1: For the system is it a negative thing? Because you're adding better code to it that the system was before?

Author: Yeah so the system is more maintainable right, we agree on that. But if your focus is on constantly adding higher quality code to the system through your contributions, do you think the system could suffer from this as there may be a trade off and you need to be making the system more secure or more functional instead of worrying about quality? This is a quality feedback system, we don't say we know about functionality or anything else.

Full Stack Dev 1: So that's the difficulty.. so if you are deliberately spending hours and hours focusing on the maintainability then yeah, you are wasting productivity and this shows this. But if its the natural process of developing this code and it's.. yeah I don't think there's a deliberate focus on maintainability, I'm just writing it that way because I prefer it that way.

Author: Ok. So the feedback this metric provides do you feel it is more, less, or kind of the same as far as actionability going forward with development, so do you know what to do next?

Full Stack Dev 1: No, because it often for me tells me I'm gold-plating. I'm not deliberately doing this.

Author: Ok ok, so let me ask you what do you find most beneficial about this metric being included in Better Code Hub? Can you see a place where it can be useful to other developers or systems?

Full Stack Dev 1: Yes, because if.. just looking at what Better Code Hub gives us as feedback, I would not know that there was negative changes in these guidelines so finely. Better Code Hub doesn't tell you how much of a change has happened with a commit. And if you can keep track of this over time, and if you have a person that you review and they all the time have a negative change in a guideline, you can say "dude, you need to pay better attention to your complexity of your stuff because it keeps going down." And at some point you'll be so close to reach this definition of done that you're going to (make this guideline) go under it.

Author: So all in all, are you satisfied with the feedback provided in the Commit Quality Impact Report?

Full Stack Dev 1: No. So the fine grained change reporting I like it but the gold-plating is too too harsh. I mean certainly there I wasn't trying to improve that guideline, but then "Oh, I'm gold-plating."

Author: So maybe you'd like to see it split up a bit more, like that you made a positive change here but then we report it as maybe gold-plating down here, but here we just tell you if it's positive or negative and only this has an affect on your rating? Not the gold-plating?

Full Stack Dev 1: Yeah you could go.. maybe more pose it as a question, "Are you maybe focusing too much on the maintainability?" Because then I can just say "No I wasn't", and ignore whatever is underneath (reporting of gold-plating).

Author: Ok Ok, well that's all the questions I have so...

Full Stack Dev 1: OK, was I helpful?

Author: Yeah you were great, thank you! It was a nice interview, longer than I expected but useful.

D.2 Interview - Technical Lead / Senior Full Stack Developer

Author: So, I'd first like to ask what is your current role in software development?

Technical Lead: Yeah I'm a software engineer, senior full stack. Um currently I'm the Technical Lead of Better Code Hub so I'm not always actually developing (on the product).

Author: Ok and how many years of experience do you have doing software development professionally?

Technical Lead: Well I've been working here at SIG for 4 years, but I was also a Technical Consultant in the beginning. We still did development on (internal projects) but I also went to speak with clients about their software. After some time I kind of transitioned into full time development.

Author: And did you have any professional experience before joining SIG?

Technical Lead: Yeah so I had a part-time job for about a year while I was in university.

Author: So we can say about 5 years of professional development experience then?

Technical Lead: Yeah, yeah sure.

Author: Great, ok. So What is the typical type of project you have developed on? Enterprise, mobile, web, SaaS?

Technical Lead: Yeah so... You could say enterprise, mostly web applications... Better Code Hub is indeed Software as a Service.

Author: Ok and what is the average team size you've worked in?

Technical Lead: 4-6 people.. the development group here all cooperate together but we work tend to work on other projects, moving around from (internal project) to (internal project)...

Author: So you kind of have a rotation with no fixed team you mean?

Technical Lead: Yeah indeed, a rotation.

Author: Ok cool, and what about your educational background if you don't mind me asking?

Technical Lead: No its fine. So I have my Masters from TU Delft, and my bachelor from my hometown university in Romania...

Author: Computer Science?

Technical Lead: Yeah, computer science. I didn't study software engineering itself, more of the theoretical computer science.

Author: Right, right. So the next few questions are scaled questions and I want to ask you to review this ISO 25010 standard of software quality, namely this Maintainability definition. I'm sure you know it well from working here at SIG, but maybe you need to refresh the details?

Technical Lead: Yeah sure, of course I know the model, particularly we used it when I was a consultant but it has been some time since I actually saw the kind of formal text of it.

Author: Yeah sure you can review it, just let me know when you're ready.

Technical Lead: Ok, yeah.

Author: Great, ok. So to what extent do you agree with the standard that is given there (ISO 25010), that definition of maintainability? This is a scaled question but you're free to give an open response as well.

Technical Lead: 4, moderately agree

Author: M oderately agree, ok. What specifically would you say you disagree with then.

Technical Lead: Well for example, it doesn't look at anything about runtime aspects. It's more a static view. It is a model, there's this saying that "all models are wrong but useful" and I think... so its, there are many things that come into everyday practice that models cannot catch. Like for example, you know a roadmap and features need to evolve that's tricky, that's kind of like a feeling than trying to capture it in a model.

Author: Indeed. Barring that you say you agree that its good..

Technical Lead: Yeah its important for me in that it gives a baseline. You know our discussion of "definition of done" and acceptable level and stuff, but that's why I didn't give it the full extent (completely agree) because there are things you need to overrule at times.

Author: Ok, so with regards to that... this maintainability model.. to what extent is it important to you to produce maintainable code according to this model when you are doing your development? So if you actually are creating new code, do you keep this model in mind constantly?

Technical Lead: Yeah, yeah

Author: Is it important to you... so out of all the aspects there are to code developement.. functionality, requirement delivery, maintainability, security.. where does this maintainability fall in this heirarchy, if there is one?

Technical Lead: It falls very high because it impacts many other things. If you have a maintainable code base then its easy to trouble shoot it, easy to optimize performance as you can see what happens where so with the profiler you can quickly navigate to the problem area of the code base to refactor it. Bugs are typically found in complicated code but if its maintainable then it makes it easier. Very high, as I said there is still this notion of experience and decision-making on the spot but its... I place it high yeah.

Author: Right, ok. So this is another scale question.. to what extent is it important that you feel you are continuously improving your own development efforts with respect to maintainability? For example if you made some code last week that was of some level of maintainability do you try this week to improve on it?

Technical Lead: Yes, completely agree.

Author: So its something you really would want to track? You really want to see how your code is becoming more or less maintainable over time?

Technical Lead: Yes its a bit like Strava, right? Strava has this idea of a quantified self and its nice to track how far along are you with your goals and see improvement in an objective way.

Author: Ok, great. So now I'm going to talk to you a little about Better Code Hub. Better Code Hub presents itself as a tool that can measure how maintainable a system is according to a list of 10 guidelines, and these 10 guidelines are supposed to reflect some aspects of this ISO 25010 standard. Do you feel that the tool is accurate in this measurement? Does it fit this model of maintainability?

Technical Lead: Yeah, yeah

Author: How accurate would you say it is then?

Technical Lead: It was based on this model, so I mean the way the metrics were aggregated it.. it's derived from it. Still there is the human touch to it so I can't really say in terms of academic rigor.. I can not say that it precisely or whatever.. but ah it comes pretty close. I mean, it represents it. Let's say, it captures it.

Author: Ok ok, so you could say it is a replacement. Say I don't know, if you see this (ISO 25010) is too cumbersome to understand, you would say Better Code Hub is a suitable replacement? In that if you're satisfying Better Code Hub here, you are satisfying ISO 25010? There's a one to one relationship?

Technical Lead: Yeah, yeah, yeah. It makes it (ISO 25010) practical. It makes it less abstract and gives you direct feedback on your code.

Author: Ok. How often do you visit Better Code Hub itself to view the feedback it provides over the project you are working on? I mean I know you work on Better Code Hub but if you work on (internal projects) do you actually check Better Code Hub to see how well you're doing?

Technical Lead: Yeah

Author: And how often would you say you do that?

Technical Lead: Well ideally after every commit but definitely at the merge request. That happens for Better Code Hub, we check it all the time. I'm actually preparing a blog post about how Better Code Hub uses itself to help build itself.

Author: Haha a bit of a recursive thing?

Technical Lead: Haha exactly. So yeah we.. frequency it depends on the amount of work. It depends on the team, right now its smaller.

Author: Do you use Better Code Hub in a private project?

Technical Lead: I don't have a private project right now. I actually started one and I want to use it, but I only have the boilerplate (of the project) there so...

Author: Ah Ok

Technical Lead: Yeah so there is no logic there but I would like to as you know, you should dogfood your own work. So not only that you get better code bases but so you actually learn how to make the product more useful.

Author: Ok. So maybe this is not directly for you, do you feel the feedback provided is useful in determining how maintainable the project being developed is and how your changes have affected the maintainability of the project?

Technical Lead: Yes

Author: So you believe any new developers will sit down and see this feedback and make sense of it and, you know, say "this is useful to me, I can see how maintainable my system is and I know how to go forward with my development now?" Do you think it is actionable in that way?

Technical Lead: I think if you understand the model (ISO 25010) beforehand it is very actionable. There may be a gap in that a person just out of nowhere, a developer that is just using this thing for the first time, there may be need to spend some time sampling the FAQ section and uh, maybe reading the book. For example, right now I can see that when we show them the bundle chart, it doesn't stick that well. But, if you have this background and spent some time with it, I think it could work out fine. I think it could be a little more fleshed out and carry you through a bit faster to understanding it. It has a bit of a learning curve.

Author: Do you think it currently highlights well the changed you yourself have made to a project? Do you think Better Code Hub is really clear in saying ”this is your individual impact to the project” as opposed to what is happening overall?

Technical Lead: Yeah so that can be a bit.. there's some lag there. That could be a bit stronger.

Author: What aspects of Better Code Hub do you find most beneficial to your development efforts?

Technical Lead: Well it covers the entire code base so it has the end to end perspective. So you talk about units, you talk about modules, architecture components... this gives a nice overall look at the maintainability of the system versus other tools which are a little bit more fine-grained. They have too many unit metrics and nothing about components and that stuff. This full picture is really important but also... there are some interesting metrics in there that are about the calls? That is really nice and if we get the bundle charts a bit more explainable and a bit more industrialized, then developers get a bit more quickly conscious of understanding coupling and how things depend on each other and where calls are coming from. That will allow for making an opinion on the next step, where can we split this up. Just randomly? No, you look at the calls to see where this part impacts this other. What I'm trying to say is there is some interesting feedback in there that I haven't seen in other tools.

Author: Yeah, Ok, great. Are you satisfied with the feedback Better Code Hub currently gives?

Technical Lead: I am satisfied, but the accuracy could be improved a bit. The fact we don't have a compiler under the hood sometimes can cause things to go a little bit wrong. There are some little things there that can put you off. Other than this, the aggregation and type of feedback you get, that's nice. Its a bit of a caveat, these things are a bit of a thorn... but its a work in progress we are continuously improving upon.

Author: Well, every software is kind of a work in progress, there's always something you can improve. But you're saying for the most part you are happy with the feedback?

Technical Lead: Yeah

Author: Ok. So this is also a scaled question so to what extent do you agree with the statement If I did not use Better Code Hub, the quality of my software would be the same?

Technical Lead: So yeah I disagree, I moderately disagree.

Author: Ok so why not completely then? Or why not neutral?

Technical Lead: Its because it has a nice way of working, it has nice coverage but as I said, its like the discussion of the maintainability.. ISO standard. Its a comprehensive view but it is not everything, it has a lot of overarching benefits but its not everything.

Author: Ok, fair enough. So I want to talk to you now about the Maintainable Production metric, or this Commit Quality Rank. You've seen it?

Technical Lead: Yeah I've looked at a few commits and opened the delta report.

Author: Ok so you've seen it, you've seen the full report. To what extent do you agree that the calculation of the Commit Quality Impact Rank is clear and comprehensible? Do you understand it well?

Technical Lead: Yeah. So the text, and the information icon which shows the popup to tell you what the letters (rank) mean and how they translate to the percentiles are used.

Author: So its clear how you got this letter grade? If you get a D or B it's clear to you what has occured to make you get this grade?

Technical Lead: Yeah, what went in.. so the volume is also factored in and what is good and bad is always clear.

Author: Ok so what would you give it as far as the scale goes then?

Technical Lead: I would give it a 4 (moderately agree). So I agree with the feedback but... do we have time to expand or is now the time to get into...

Author: Yeah we're going to get into it, so yeah you can expand. Go ahead.

Technical Lead: Yeah so it is important feedback that you get, but sometimes there are other things that need to... you know it's a tradeoff. I had an instance where I urged someone to remove duplication so then I got "hey, this is gold-plating." I fully agree with that, it's a removal that may not be needed. But then this, I had to counter-balance with the fact that "you know what, this is in the front-end. If we have duplication yeah, it will matter for the run-time performance in the browser because the code base will be a bit bigger if you leave that in." So then I decided to overrule it (the gold-plating feedback) but then the feedback is nice because it created a discussion and that's the most important thing. Even though it sometimes doesn't match intuition, its a bit like the ISO standard, it doesn't offer this 100 percent accuracy but because it exists as a concept it leads to nicer outcomes. So that's what makes me moderately agree, I do agree with this caveat that sometimes its.. you have to kind of..

Author: Right I get it. So this model is built on Better Code Hub, and Better Code Hub can't capture this kind of contextual issue, you know you have this tradeoff of you have this run-time and this maintainability, maybe run-time is more important in this case but at least it gives you that feedback. And that feedback is understandable from what I hear from you?

Technical Lead: Yeah, it is something that adds to the discussion. I would not think in this way otherwise, I would say let's just do it. But if we always just focus on.. basically it prevents gold-plating and that is a useful discussion to have, even though as I said (the issue of the front-end duplication) I think this issue captures this trade-off and how you have to look at it?

Author: So you're saying without that feedback you wouldn't have this thought into that trade-off?

Technical Lead: No, no I wouldn't. The default would be to say "hey duplication, please. That's it." but no, you have to think about the fact that we're in a business and you cannot always foresee how the things evolve. Sometimes you just really write the code the first time and then wait for new features to be ordered, and then not gold-plate but apply a bit more thinking and thought. The goal is not to write code, we solve problems and we need to have feedback like this metric provides to nuance the whole discussion in ways that ultimately benefit the business. We can stay here and optimize code all day, but where will we be?

Author: Ok, indeed. So this Commit Quality Impact Rank you receive, how does it affect your motivation going forward with development? So does it change your development habits I guess is what I'm asking?

Technical Lead: Yeah, yeah. So it's a bit like in student life, you want to have straight A's but for me it's more a tool to adjust the behavior.. I can't say that I feel some kind of intrinsic feeling about itself (the grade), I just want to see where I can apply it in practice.

Author: Does the rank you receive generally match your expectations of the quality you're delivering to the system? So you get this B, does it match what you think you were going to get?

Technical Lead: Some cases yes and some cases no. I have some examples in which I removed code and I got the feedback that I'm "leaking productivity". Maybe the calibration needs to be looked into for these specific examples. In general, you get that you expect what you see but there are some examples which we can discuss to see what happened there.

Author: Ok. So you talked a lot about the gold-plating thing and that the metric is trying to prevent that. That is not the main goal of the metric per se, but I understand in the case that the projects you are working on.. these are all really high quality according to Better Code Hub. They are all 9s or 10s. So any positive changes you make are going to result in gold-plating. Its hard to see how you could do that in a really well maintained system, but try to imagine a 3 out of 10 system. Then gold-plating likely is not going to be seen as so much of an issue, maybe on those 3 guidelines which are compliant but for the most part you are not going to be seeing gold-plating that often. You will more likely be seeing how close you are getting to becoming compliant. But let's focus on the gold-plating, it is heavily penalized. It's penalized as heavily as bad code, or non-compliant code let's say. The idea behind it is that it is beneficial to make your code maintainable, but there is a point of diminishing returns right? Where you should stop focusing on maintainability and put your development efforts elsewhere so that you actually deliver a functional product that is secure and all these other things. The metric is "harsh" in that even if you introduce new code that is a higher quality than the current system, you will be penalized for gold-plating if that system is already compliant. This is kind of not intuitive. You would think, like "I'm introducing new code which is of high quality so that's good right?" But the idea behind that is that, with these Agile studies that if you had created the code to the same standard of what the system already was, you likely would have been able to employ it faster and ship faster, with excess quality being classified as waste to be eliminated from your production. Do you agree with these ideas?

Technical Lead: Yeah so I think these are good ideas, and I can see how the metric is a bit "harsh", but I would be interested to see if models can be created to be robust against that. So, I see what you mentioned that you could "put your effort elsewhere", but it would be nice if you could also handle this case about what can you do when it's already good (the system is already compliant).

Author: So to what extent do you agree that the metric is fair in penalizing Gold-Plating when analyzing your contributions?

Technical Lead: For this instance (a compliant system) it is not fair. Yeah you... its a bit harsh. It would be nice if it was robust against this. But for the other case in which the it is not compliant, I see it being very fair. I see it working pretty well. But if you're already compliant, I see it being a bit too harsh.

Author: So what kind of value do you see in it then as far as how much you agree in that the metric in fair?

Technical Lead: That the metric is fair? That it's not harsh?

Author: Well more that it's fair in the sense that it is objective, its reporting exactly what is being done. Do you believe that it is fair in its reporting?

Technical Lead: Oh in that sense it is fair. It is built like that and is showing what it is supposed to show. So I completely agree that it is fair. It's just that in specific situations, the impact that it gives may not be the ideal one. It lacks the nuance but it is definitely fair, it may be interpreted in the way that you hope is not the wrong way. Something like that. So thanks for clarifying, I agree with it that it is fair.

Author: So the Gold-plating feature attempts to highlight the interplay between productivity and software quality. Do you feel this metric accurately captures this aspect of development?

Technical Lead: So its, it would not be an issue if you introduce code of that is of a high quality and gold-plated. It is input that you have to factor in and draw a conclusion about it.

Author: Ok so do you see the relationship to productivity then, productivity as a whole? So if I'm working on maintainability and I'm getting this feedback back from the metric, and it tells me "Oh I've gold-plated on this guideline" or "Oh I'm moving in the right direction", do you see the relationship to productivity there?

Technical Lead: Yeah, yeah because you have to focus on getting the most amount done with the highest quality without gold-plating.

Author: So this is the point of having the gold-plating in (the calculation)?

Technical Lead: Yeah you're trying to de-incentivize the person from trying to work too hard on maintainability. Yeah, as a software engineer you have to be able to take this input, this is valuable input... yeah that's the thing. It can be somehow "harsh", I can see that, but it's just another input to develop a solution. So yeah getting feedback that is a little bit not what you want... yeah you can't say you want it this way, you need to take it as another input to change your behavior.

Author: Do you think the feedback provided with this new metric more, less, or the same as far as actionability going forward with development, so do you know what to do next?

Technical Lead: Yeah, so in terms of like.. we didn't have this before right? We kind of just gave like a state of quality. The new behavior that this unlocks is that it gives you a bit, ok "if I have done this in this way, if I really follow it, maybe I could have done more in the same amount of time."

Author: Ok so you think it is more actionable?

Technical Lead: It is more actionable because it is more far reaching. It makes you think, like "ok, what did I do here that caused this feedback and what can I do next to kind of deliver faster at an acceptable level of quality so we can do more per unit of time?"

Author: Ok, great. So what do you find most beneficial about this metric being included in Better Code Hub?

Technical Lead: The most beneficial is that as a team, we can... Better Code Hub gives a "definition of done" for quality and what we can now potentially have is a definition of done for Maintainable Production. The romantic story is that we can do more at an acceptable quality, and get more value out of our time.

Author: Are you satisfied with the feedback provided in the Commit Quality Impact Report?

Technical Lead: Yeah. So there are things that can be improved about the UI, for example reducing text and doing some drop downs or whatever. But yeah it's nicely linked to GitHub, it's not too much information off the bat and you can access it and gain more information with it. I am a bit biased as I've known what you've been working on so I've had to put myself in the perspective of a first time user..

Author: Indeed so.. This is why I chose to interview you, you are the technical lead and you have this intimate relationship with Better Code Hub, but also being the technical lead you more often have to put yourself in the shoes of the user. It's your product, you don't want it to be seen in a bad light but you also want to see it improved. This is something that is easier for you to conceptualize what this feedback means to the user. So yeah that's all the questions I had for you, I appreciate you taking the time to sit with me and talk about it.

Technical Lead: Sure thing! It was nice.

D.3 Interview - Research Developer / Prototyper

Author: So, what is your current role in software development?

Prototyper: Yeah I'm only doing prototyping nowadays, I only write software for research which is mainly prototyping.

Author: Ok, have you done professional development?

Prototyper: Yeah I did it for about 3 years.

Author: 3 years, great. What type of development did you typically work on? Web applications, was it enterprise software..?

Prototyper: Yeah mainly web applications. We did web development but later I did.. the last year I only did data science stuff.

Author: And were you working with a team when you did this?

Prototyper: Yeah, yeah.

Author: So how big were your teams on average?

Prototyper: Hmm 25 people.

Author: 25 people, ok.

Prototyper: Yeah at max.

Author: Ah ok and were you working in subgroups or...

Prototyper: Yeah we were working remotely most of us. So the teams were about... yeah it depends, about 8 to 20 people.

Author: Ok so you're used to working on highly collaborative projects?

Prototyper: Yeah, yeah.

Author: Ok, ok. Your education background.. what was your main study.

Prototyper: Computer Science. Yeah I studied a Bachelor of Mathematics and a Master in Computer Science with Data Science. I didn't study software engineering but more data science.

Author: Ok and you're working on your PHD now?

Prototyper: Yes.

Author: So when do you think you'll finish?

Prototyper: 3 years, haha. 3 years from now.

Author: 3 years, great. Ok so I'm going to ask you a few questions about maintainability, the next few questions are scaled questions, the next 3. They have to do with the ISO 25010 standard for (software) Maintainability. You've gotten a chance to review this?

Prototyper: Uh huh, yeah I have.

Author: So yeah, to what extent do you agree with the ISO 25010 definition of maintainability matches software quality, let's say? So do you agree with this maintainability model?

Prototyper: Yeah I moderately agree with it, I don't completely agree with it.

Author: Ok, you moderately agree with it. Why do you not completely agree with it?

Prototyper: So first of all, what I don't like about standards, and particularly what I don't like about this, is that it is really high-level and not easy to understand for people who use software. For example, if you have reusability, you have "the degree to which an asset can be used more than once", but the asset size is not specified.

Author: It's not specified enough?

Prototyper: Exactly, it says nothing about what you can reuse or not or how we can make it better or worse. It just... yeah ok, you can reuse something from a project or not.

Author: So it's like the practicality bit that is missing for you, you say?

Prototyper: Yeah, I would say it is easy to misinterpret it.

Author: Ok, and do you agree that this is an accurate way of measuring it (software quality) as far as how good a software project is?

Prototyper: I wouldn't agree nor disagree with this. These are just guidelines to me.

Author: Ok. So with this standard in mind, to what extent is it important to you that you are producing maintainable code?

Prototyper: With this standard in mind?

Author: Yeah, this standard.

Prototyper: Well I have a focus on maintainability, with this standard all of them (the aspects defined in ISO 25010). I think the problem is the granularity. As I've told you with reusability, for example, it is really hard to define.

Author: Ok, ok. So overall these are nice ideas is what I'm getting from you?

Prototyper: Yeah.

Author: ... but they are not actionable in your eyes?

Prototyper: Yes.

Author: Ok, going off of that, let me ask you to what extent is it important to you that you are continuously improving your development for producing maintainable code?

Prototyper: For me it is very important.

Author: So just to kind of flesh it out more, if you're working on code one week and you make some contributions to the system and you think "this is maintainable code", is it that you will keep developing in the same way or are you going to try and push yourself to make it even better over the next few contributions?

Prototyper: I think it depends, at some point there is an upper bound. So at some point it is done from a maintainability perspective, but if I add new functionality to it then it should adhere to the last level (of the system with respect to maintainability).

Author: Ok great, so I'm going to ask you a few questions now about Better Code Hub. You've used it right?

Prototyper: Yes.

Author: So Better Code Hub presents itself as a tool that can measure how maintainable a system is with these 10 guidelines it has, and these 10 guidelines are supposed to map to the aspects of ISO 25010. Do you feel the tool is accurate in this measurement? Do you agree that the tool meets this standard?

Prototyper: The standard itself I think yes, but I don't think it is always accurate.

Author: Why do you think it is not accurate?

Prototyper: Haha yeah I don't know, so this is just coming from experience that it is not accurate but beyond that I think it is not tweaked to specific programming languages.

Author: Right, so it presents itself as a general-purpose language... so you could put anything into it and it will give you a rating that is acceptable regardless of language.

Prototyper: Right.

Author: So you feel it still doesn't nail it then?

Prototyper: Yeah, for example when I use it with Python, it had plenty of errors that are dependent on the programming language. But if you use it with Java it works a bit better.

Author: So then let me ask you, with Java this model (ISO 25010) you would say fits. Would you say it also fits with Python?

Prototyper: Yeah for sure, the model itself fits but the way its applied (in Better Code Hub) does not. Actually I think the model is good at a general level, I just think it is applied... it needs specifics.

Author: It's an implementation issue?

Prototyper: Yeah... it's more an interpretation issue.

Author: Interpretation, ok. How often do you visit Better Code Hub when you develop your code?

Prototyper: All the time.

Author: So you actually visit the UI and get the feedback each time and get the feedback directly from the UI?

Prototyper: Yeah, yeah.

Author: Ok, great. Frequency? Would you say after every commit?

Prototyper: Yeah, after every commit!... Well, not really no but after every merge yes.

Author: After every merge, ok. Is the feedback provided useful in determining how maintainable your system is?

Prototyper: Yeah the problem is you can't avoid some stuff that is in Better Code Hub. For example, I had code duplication in imports and I could not avoid it or scope it out without it becoming very messy. But beyond that I think the feedback is very useful.

Author: Do you think it really shows how your individual change has affected the system? Is that single change really highlighted for what has happened in that one change?

Prototyper: No, not at all. It's more at the system level than at the single change.

Author: Ok, so is the feedback actionable for you? Do you know what to do in your development to make it better?

Prototyper: Sometimes yes, sometimes no, but most of the time yes. Often it says to read some chapters from the book but you're supposed to pay for Better Code Hub so yeah, to just say read this chapter that's kind of shitty.

Author: Yeah, yeah. Ok but from the feedback that is there, that is presented to you.. so if you see that duplication has gone down, do you know what to do to rectify this issue to make it better?

Prototyper: Yeah, yes.

Author: Ok so what, if any, aspects of Better Code Hub do you think are most beneficial to your development efforts?

Prototyper: So I like that every time you add new code it will check it, but what I don't see as a clear feedback is... if I develop a new feature and push it or just analyze it, I don't see the metrics.. they are increased or decreased, but I don't see the balance between "Oh I added new functionality which is good for the system, even though it is a bit less maintainable." So there is no tradeoff between what you actually do in terms of code, and they are less maintainable...

Author: So this goes back to what you said earlier about the granularity?

Prototyper: Yes.

Author: Are you satisfied with the feedback Better Code Hub currently gives you?

Prototyper: No.

Author: So you don't. What would you like to see improved?

Prototyper: First of all, it should solve these issues (the granularity issue discussed before). Beyond that, I think they have this bar after they give you feedback which is really not clear what it is. Also, there is a lot of terminology in the feedback that is not coming from developers but is coming from researchers. The other one is the one with the book, I have the book (10 Guidelines to Building Maintainable Software) but other people do not have the book, and I'm paying for the system so give me the book or PDF.

Author: Earlier you said that it is important to you to continuously improving you coding habits for maintainability. Do you think Better Code Hub allows you to do this?

Prototyper: I think Better Code Hub is more like a referee, it's not really a facilitator. It's more like a referee. Without it you would not know anything, with it you have a referee that is not always right.

Author: So it has "bad calls" now and then?

Prototyper: Yeah, yeah. You have sort of... if you just develop on GitHub and you have no static analysis tools, you'll never know what happens. With Better Code Hub you have an indication of what happens.

Author: To what extent do you agree with the statement "If I did not use Better Code Hub, the quality of my software would be the same?"

Prototyper: I would say I moderately disagree.

Author: So you do see it as something that is really enhancing your development efforts?

Prototyper: It is, but not because... it's because it gives you something rather than nothing. Otherwise you would not have any references. With Better Code Hub you have references.

Author: Ok, so the references is the clear benefit here?

Prototyper: Yes.

Author: Ok, so I want to talk to you now about the Maintainable Production Metric...

Prototyper: Your metric?? :D

Author: Yes, my metric haha. You've seen the feedback and analyzed a few of your commits. So yeah, this is another scaled question, to what extent do you agree that the calculation of the Commit Quality Impact Rank... or that letter grade, rank you get... is clear and comprehensible? Do you know how that value came to be?

Prototyper: Not all the time. So at some point it was really really clear, but for some commits... I'm not sure if it was wrong or if I got it wrong. I would say I moderately agree then.

Author: Moderately agree?

Prototyper: Yeah for some commits it was nice, but other times I really had some issues understanding how it got there.

Author: Would you call those like edge cases then.

Prototyper: Yes. I think it is more a problem with the implementation.

Author: Ok, so how does that rank you receive affect your motivation? If you get a D or an A or B, are you feeling more or less or at all motivated?

Prototyper: I think I would be more motivated with a lower score so I can find something to improve. I cannot answer this question confidently as I need more time with the metric.

Author: Does it affect your views of what you think of software quality?

Prototyper: The metric itself?

Author: The feedback you receive. For example, you are reported to have a positive impact or a negative impact to the system... but then you also have this idea of gold-plating or no impact..

Prototyper: So you mean the definition of software quality or my own opinion of software quality?

Author: Yeah so I guess I'm asking... you contributed something you expected to be of a certain quality and then you get this feedback, does it match that expectation?

Prototyper: The feedback itself? Yes. Yes.

Author: So you think it's accurate in this measurement?

Prototyper: There are some cardinal cases but for the most part yes.

Author: Ok, ok.

Prototyper: At some points I think it overlaps with Better Code Hub.

Author: So you're saying there are issues in Better Code Hub itself affecting it?

Prototyper: Yeah but not only necessarily the feedback but also the way you display it, and the way the you display it which makes you think the feedback is the same.. but then after some time you see that this is really something different.

Author: Ok, fair enough. Yeah so let's talk a little bit about gold-plating. It's heavily penalized under this metric, its actually seen as being just as bad as having bad code (non-compliant code) right? And the idea behind that is that while making your code more maintainable is clearly beneficial, there is this point of diminishing returns where the code is future-proof, you don't need to be focusing on quality. You need to put your development efforts elsewhere. The metric could be said to be "harsh" or rather objective in that it is only going to tell you what you impacted (as far as maintainability) within that commit. So even if you're introducing new code, and it is of a higher quality than what the system already was, you're going to be penalized for gold-plating. Likely that doesn't match intuition, but if you think about it... you've introduced code that could have been of the same quality as the system itself and likely that would have been faster to add and easier to implement than over-engineering this code. It might not be the case also, it might be you just naturally produce code in this way, but you can see how...

Prototyper: Yeah, I understand that. I understand that. But developers also evolve, they are humans that don't do the same thing, the same way twice necessarily.

Author: Yeah I can understand that. So let me ask you, this was a personal project you were working on when you analyzed this code?

Prototyper: It's a library, it's an open-source project I've been working on for a while. That's why maintainability matters.

Author: So let me ask you, to what extent do you agree that the metric is fair in analyzing gold-plating when analyzing your contributions?

Prototyper: I moderately agree.

Author: Ok, but you feel there are some times where you feel it is not accurate?

Prototyper: So I think gold-plating (the tracking of) is a good idea, but I think in some commits it was way too harsh. I mean you get penalized a lot for just doing some gold-plating. Which in the end, it's really on the edge of being not good.

Author: Right, ok. The gold-plating feature attempts to highlight this interplay between productivity and software quality. Do you feel this metric accurately captures this aspect of development?

Prototyper: Yeah, I completely agree with that.

Author: Completely agree, ok. So the feedback provided to you do you feel it is more, less or the same as far as actionability? Has it made it more clear going forward?

Prototyper: No, I moderately disagree.

Author: Moderately disagree, so you feel it is less actionable?

Prototyper: Yeah, the gold-plating.

Author: The gold-plating specifically?

Prototyper: Yes, because you can not undo that.

Author: No you can't undo that, so in some ways it encourages you to make worse code?

Prototyper: Yeah but.. you can't undo it in your refactoring. Getting rid of good code...

Author: Yeah indeed but the idea is to give you a trend of your development to see "oh I'm gold-plating, and gold-plating and gold-plating." Not from a single feedback will you get this feeling, but from an aggregated view would you..?

Prototyper: Yeah so the thing with Better Code Hub is that your code is getting better and better and better. Then with gold-plating you would have to do worse. So if you get a D from gold-plating, because you get penalized a lot just from gold-plating, your next commit you get an F... or whatever... then you ask "why is it decreasing? I'm adding code, I'm doing it good. Maybe it's a bit too engineered but..."

Author: Yeah so the idea behind the feedback is the feedback you get from this metric and then what you get from Better Code Hub, they work together. One is not supposed to override the other. So maybe you get this feedback "this code (system) is future-proof" and this metric attempts to highlight what happened within that single commit, did you gold-plate in that commit.

Prototyper: Yeah but the feedback is not actionable, I can not doing anything about that single commit. I can not go back and refactor or write other code to make it worse to improve the metric.

Author: Are you satisfied with the feedback given to you in this Commit Quality Report?

Prototyper: Yes, I moderately agree. There is some wording there you can change... I tried to use it as a normal developer. There are some things in there with the wording some seemed cluttered and be more simple.

Author: Right so I mean, this feedback is for research purposes. It is not a production ready sort of feedback. But you would say you are quite happy with the feedback you are getting?

Prototyper: Yeah.

Author: So is there anything you'd like to see improved?

Prototyper: Yeah so this scale of code churn.. the amount of code you've added? That's not really clear how that relates to the overall scale. So if you can relate them in a visual element that would be nice.

Author: So you're talking about the percentage of code churn in relation to the system? You understand how that works in the calculation of the metric right?

Prototyper: Yeah, yeah. I just think making a visual element would make it better.

Author: Alright, great. That's all the questions I have, thanks.

Prototyper: Alright, cool.

D.4 Interview - Front End Developer 1 / Contracted

Author: What is your current role in software development?

Front-end Dev 1: I'm currently hired as a front-end programmer on contract for SIG. I'm helping them revamp the Better Code Hub website.

Author: And how many years have you been doing software development professionally?

Front-end Dev 1: I would think it would be around 10 years.

Author: Do you also work on personal projects on the side?

Front-end Dev 1: Yeah, a few.

Author: Ok, cool. What is your educational background if you don't mind me asking?

Front-end Dev 1: I studied multimedia design and I was more into animation and graphics.. visual representations of stuff. Along the way I discovered Flash and from there I got into programming, making my first few games and micro-sites. I feel in love with programming, and have been doing it ever since.

Author: Cool, cool. So what sort of projects have you typically worked on? Were they more web applications or mobile.. maybe enterprise software?

Front-end Dev 1: Typically web applications. I've built usually a lot of heavy front-end websites, but on the other hand I've done some experimental stuff like Unity3D or Unreal Engine or mobile apps. Yeah but mainly front end work on websites.

Author: Ok, and on these projects you would typically work in a team?

Front-end Dev 1: Yep.

Author: So what would be the average team size you're used to working in?

Front-end Dev 1: Around 4-5 people, but just including the devs. Not the sales people or anything.

Author: I want to talk to you now about maintainability a little bit, actually in the sense of this ISO 25010 standard. Have you seen this standard before?

Front-end Dev 1: Umm no.

Author: Yeah you probably have not seen the formal definitions of it. So what I would like to ask you is to review it for a moment just so you can formulate some opinions and then I'll have some follow-up questions.

Front-end Dev 1: Ok. (after 1 minute) Ok, yeah.

Author: Yeah? Ok so you've taken this in and have some opinion on it. To what extent do you agree with this definition given here in this ISO standard?

Front-end Dev 1: I would say a 5, completely agree.

Author: Completely agree, ok. So there are no aspects of maintainability you feel this standard does not encapsulate or anything like that? It really does encapsulate the full idea (of software maintainability)?

Front-end Dev 1: I would say it fully encapsulates it.

Author: Ok, so going with this standard of maintainability, to what extent is it important to you that you are producing maintainable code?

Front-end Dev 1: That really depends on the project, because there are some... I'm coming from an advertising background, and there are some projects you must deliver fast, so the aim is delivering it really quickly as opposed to having it really maintainable because in a couple of months that codebase is going to be really redundant. However, if you're working on software projects or even like a company website intended to last for years, then it is really important that these steps are adhered to.

Author: Do you strive to make it your focus then when you're doing your development on a software project? So like with all the aspects of software development... functionality, security, run-time, maintainability.. where would you say maintainability falls in this hierarchy let's say?

Front-end Dev 1: Um I would say it would be almost at the top, around 2 or 3, because sometimes you have to sacrifice maintainability in order to deliver something or in order to fit within a particular framework. That happens a lot when you're working with a lot of different frameworks which have predefined ways of doing things instead of just working with vanilla code. However, maintainability is a very, very, very high focus.

Author: Ok, great. So to what extent is it important to you that you are continuously improving in your maintainability habits when developing code?

Front-end Dev 1: Completely agree (very important).

Author: Completely agree. Ok. Let's try to flesh it out a little bit then. So, if you make some code last week and you think "this is maintainable code, I'm proud of this" do you then strive for the next week to do even better the next week?

Front-end Dev 1: Old code or new code?

Author: Let's say in both instances.

Front-end Dev 1: I used to do a lot of refactoring of old code but then I began to realize it is a bit redundant, it was good practice but as you get more experienced you start building code in a more maintainable way so you refactor less often. But yeah, whenever there is new code to be written, maintainability is one of the most important things. That's actually one of the most time consuming aspects of writing code because you spend a lot of time thinking on how to structure something as opposed to writing it.

Author: Ok so is this something you personally would want to keep track of? To see how maintainable you are over a particular project or in general?

Front-end Dev 1: Yeah I think this would be pretty interesting to see, to get really good data over my development.

Author: So lets talk a little bit about Better Code Hub. You're getting more familiar with it, you've been working on it a bit. Better Code Hub presents itself as this tool which can measure how maintainable a system is according to this standard (ISO 25010) using these 10 code aspect guidelines. Do you feel the tool is accurate in measuring maintainability? Does it fit this standard?

Front-end Dev 1: From what I have seen so far I do believe it does, yes. I've only used it on a few projects but so far I would say yes. I also know Better Code Hub can support many different languages and I haven't seen how it works across all of these projects but from what I've used it on I do believe it is accurate in the feedback it gave me.

Author: Ok how would you rate it with this scale then (Likert)?

Front-end Dev 1: I would say moderately agree. Just because I haven't seen it across all projects it can support.

Author: How often do you visit Better Code Hub's UI to get a sense of how maintainability your project is?

Front-end Dev 1: Lately, every day but it's also because I'm working on it haha.

Author: Right, for example when you make a commit are you checking Better Code Hub each time? Or is it only in the pull request?

Front-end Dev 1: Oh so yeah it is mainly in a pull request.

Author: Do you think the feedback is useful in determining how maintainable is the system at the time?

Front-end Dev 1: Yes, and I think it can be very useful for teams. Especially if you're hiring junior developers, then they may spend more time on each commit but if you know what you're doing then mainly checking the pull request is important.

Author: Do you agree that it reflects the changes that you yourself have made to the project? Does it really highlight what you have changed as far as maintainability of the project is concerned?

Front-end Dev 1: I would moderately agree.

Author: Ok so is that feedback actionable, in that you know what to do going forward?

Front-end Dev 1: Yes, again I'd say moderately agree. I agree that it helps so that's why I agree, but I think sometimes going into the content (of Better Code Hub) takes some time to really get what that information is. I think I would use Better Code Hub more when I myself find doubts in my code and I want to see what it thinks, but I wouldn't use it for each commit.

Author: Are you satisfied with the current feedback Better Code Hub gives?

Front-end Dev 1: Yes.

Author: So nothing you would want to see improved?

Front-end Dev 1: At the moment no, not anything I could think of.

Author: Ok so let me ask you this, to what extent would you say you agree with the statement "if I did not use Better Code Hub, the quality of my code would be the same?"

Front-end Dev 1: I would say I moderately agree.

Author: Moderately agree, ok. So you would say Better Code Hub is not having a significant affect on your software quality?

Front-end Dev 1: I don't think so at this point. I think sometimes it catches some things but I think it comes from the experience as well. The projects I've ran through it are all high quality, like a 9 or a 10, and I only missed one because I had no tests. However, this was a personal project so I didn't care to have tests at the time. I would really use this to make sure that a team is on the same page.

Author: Ok, so lets imagine a scenario where you're working on a work project, production code, that is not so maintainable, let's say a 2 or 3. Would you be more invested in using Better Code Hub in that sense?

Front-end Dev 1: Definitely, definitely.

Author: Ok so now I'm going to talk to you about this maintainable production metric. You've seen the feedback it's given you, you've analyzed a few commits and seen this Commit Quality Impact report there which gives you this letter grade based on your commit's affect on the maintainability of the system. To what extent do you agree that the calculation of that rank is clear and comprehensible for you?

Front-end Dev 1: That's a tough one, that's a tough one. I would say I neither agree nor disagree here, mainly because there are some things that help quite a lot to see but some things that are also redundant. Yeah it's difficult. I think there is a lot of stuff happening behind the scenes, which makes sense to not show to the user as long as the result makes sense. It's mainly how this change in quality came to be is a bit unclear.

Author: Ok, we'll get a little bit more into that in a moment but first I'd like to ask you how the rank you receive affects your motivation.

Front-end Dev 1: So far it has been interesting to get this idea of which direction the project is going (with respect to maintainability). At the moment, everything seems to be "ok, ok, ok" and if there was a dip I would be more interested to see where did it dip, and where can I improve that. I think once you reach a certain level, the motivation is already there. It's like you've reached this good pace. I think I'm only more motivated to do things differently if I see it going down.

Author: Ok, just to clarify, getting a good grade is not as motivating for you as getting a worse grade? You're more interested in this negative response as it shows you there is something to improve?

Front-end Dev 1: Yes, yes. It's like avoiding the bad grade and staying on the podium. I think really I'm most interested in seeing what happened with the guidelines than seeing an actual letter grade, or rather getting that grade and seeing a really detailed breakdown.

Author: Does the rank you receive generally match your expectations with regards to the quality you've delivered to the system?

Front-end Dev 1: Yes, I would say yes. I would say I moderately agree, and actually I would say I would completely agree if I knew just a bit more about the tool... but I think that's more from the developer in me in that I want to know every little bit of how it works.

Author: So let's talk a little bit about gold-plating, because I think this is something everyone has an opinion on. Particularly if you have a project that is already very maintainable, such as what you may be working on here at SIG. So let's just talk about the definition of gold-plating. Would you agree that "gold-plating occurs when you attempt to increase the quality of a system past the point of when it is already good enough for production purposes?"

Front-end Dev 1: For me gold-plating sounds like something that you are making better, as opposed to doing something that is not that useful. So the term for me sounds like "I've gold-plated something, I've polished it and made it better."

Author: Made it better, indeed.

Front-end Dev 1: But what I think it means in this context is that you've done something that wasn't needed, you've wasted time.

Author: Right, so the way it's defined at SIG is that you've taken a system which is already future-proof as far as maintainability is concerned and still tried to improve it. In this way, because you're so focused on maintainability you keep trying to constantly improve it, so you've kind of spent time you didn't need in doing that so you've wasted productivity in that sense. Does that make sense to you?

Front-end Dev 1: Ok yeah, that totally makes sense to me. I just never heard this term before. Is this known in the industry?

Author: Yeah it's kind of a common idea or term but perhaps called differently. At SIG this is how they define it so far as I know.

Front-end Dev 1: Ah ok, I think I would have called it something like over-polishing or...

Author: You could say it is most similar to saying something is "over-engineered".

Front-end Dev 1: Over-engineering. Yeah, that makes more sense.

Author: Ok, we're on the same page then. So with this addition to Better Code Hub, gold-plating is heavily penalized. It is seen almost as bad as having non-compliant code. The idea is that while it is beneficial to making a system more maintainable, there is the point of diminishing returns where you should be focusing your development efforts elsewhere. You're actually not being as efficient with your productivity as you could be. Simply having a maintainable system does not necessarily mean it is automatically successful if other aspects are lacking such as functionality, security or poor run-time. The emphasis is really on reaching this "definition of done" and stopping. However, the metric is objective in that it only tells you what you've done, have you made the quality go up or down with your contribution. So if you add new code to the system, and it is of a higher quality than the system was before but the system was already compliant, you would be penalized for gold-plating. That's because, yes while this is new code, you've increased the system past this done point. Now I'd like to ask if this is clear for you?

Front-end Dev 1: Even if it's a new feature?

Author: Even if it's a new feature. The idea is from the Agile methodology which states if you had created the feature with same quality the compliant system already had, you would likely have deployed and implemented it faster and this excess quality would be seen as "waste" which you should eliminate from your production. It's really if the quality is improved past this done point.

Front-end Dev 1: But then wouldn't you wrtie the feature in a bad way as to not be penalized?

Author: You would say that you would write it in a maintainable way, so that it neither increases nor decreases the overall system quality. You've already reached this point of future-proof code. Better Code Hub doesn't care about what functionality you may add or features being created, it only looks at how maintainable your system is. So with this idea, we're trying to say what happened with the system's maintainability in response to your contribution. Did you over-engineer it, gold-plate it? Or did you actually maintain the system?

Front-end Dev 1: So how is gold-plating actually calculated then?

Author: It's based on changes to each of the guidelines we are analyzing. If you cause a guideline to increase past a certain point, then it is compliant. If you continue to increase the quality after passing this point, it is gold-plating.

Front-end Dev 1: And this point is stationary?

Author: Yes, as a means of saying in a general sense where you should stop. This point was determined through a historical bench-marking of many many projects SIG has worked on. So if you're really moving far from this point after being compliant, then you are gold-plating.

Front-end Dev 1: Ok. I agree it is gold-plating then if it is existing code, for sure. But I think if it is new code, this point should move along with it. I think if you're adding new code to the system that is good, then the system should move along with it.

Author: So you think this "definition of done" should be flexible and not stationary?

Front-end Dev 1: Yeah I think so. Because if you're adding new code that is far past this point, would you still be penalized if the system is already good?

Author: Yeah so this point is defined through a risk profile based on percentages of the system which are good or bad. So if you keep moving further past this minimal point in the positive side you will be penalized for gold-plating. It doesn't look to see if this feature satisfied a requirement, it doesn't look to see if it is improving the security of... it's only looking at maintainability.

Front-end Dev 1: Ok.

Author: So to what extent do you think this metric is fair in penalizing gold-plating?

Front-end Dev 1: I think it is fair for existing code, I still have reservations for new code being added. If you add a new feature and it is written in a better way than what was there existing code, I think this should be seen as an improvement. It doesn't affect the other stuff.

Author: Right, so this is only when it is compliant. What about when the system is non-compliant over many guidelines?

Front-end Dev 1: When you're moving toward compliance? This I agree with completely.

Author: So it is really in this compliance area it is not fair when adding new code?

Front-end Dev 1: Yeah so there is something there that does not seem fair. There's also this idea of a new team member adding new code and a project member adding new code... if you're then penalized for adding better code than theirs that could be demotivating. So what, that means do you have to reduce the way you write your new features to their level?

Author: With this metric yeah you would be encouraged to write things on the same level, only as good it needs to be.

Front-end Dev 1: Ok so if you're a new developer, and you write code a certain way to an existing system which is better, maybe not even over-engineered just that you write better, how likely is it you would be penalized?

Author: Well I would say unlikely but yeah, if the system is already better... it looks at snapshots of the system in Better Code Hub to determine how compliant it is. If it is non-compliant then you would not be penalized though if you are then it can happen.

Front-end Dev 1: Ah ok.

Author: So how would you rank the metric based on fairness then?

Front-end Dev 1: I think I would go with a 3, somewhat agree / disagree. I'm still not sure how it helps. I think it's very interesting as a concept, I like the concept, but I'd have to see how it could be used.. I'd like to use it more.

Author: Ok, fair enough. So this gold-plating feature attempts to highlight this interplay between productivity and software quality. Do you feel that the metric accurately captures this aspect of development? Do you think it actually shows a trade-off between the two?

Front-end Dev 1: Yes, absolutely. Is it possible to measure time in gold-plating? To say yeah ok, you used so many hours to actually gold-plate this?

Author: Yeah you could, though we don't look at time in this measurement. We really wanted to focus on what happens with a technical contribution to the system. But we could potentially add this in future work where we see, how much quality do you produce per contribution and how long did the contribution take. This could give a more nuanced view, but for now we don't look at this.

Front-end Dev 1: Ok.

Author: So you say yes, this does capture this aspect of how productivity is moving in relation to software quality?

Front-end Dev 1: Yes, yes I do.

Author: Ok so is the feedback given to you by this metric more, less or the same as far as actionability compared to the feedback Better Code Hub originally provides? Do you know what to do given this feedback going forward?

Front-end Dev 1: I would say yes, it would be clearer if... it's clear in the situations where a file is being over-worked on but in situations where the gold-plating is due to new code being added it is not so clear how to move forward. I wouldn't know what to do there, am I supposed to make my code worse?

Author: So is it more or less actionable then?

Front-end Dev 1: It depends on the situation, its actionable if I worked too much on this one file but new additions that are reported gold-plating, do I leave it as it is or do I go back and rewrite it in a worse way? It's good to have the data, but in the end the person will have to decide what to do.

Author: So that still pertains to when you are kind of in this compliant side of the spectrum, but what about with non-compliant systems? From what I understand that is intuitive to you. Would you say that is clear for actionability, just the compliant side of things is unclear?

Front-end Dev 1: Yes, exactly.

Author: What do you find most beneficial about this metric being included in Better Code Hub?

Front-end Dev 1: I think it is really nice to see what is being over-worked on. I like that is can show you how many times to refactor something and where you are spending a lot of time.

Author: Ok. Earlier you gave me a response that you find it important to continuously improve your development habits with regards to maintainability. Do you think that this metric helps with that?

Front-end Dev 1: I think it maybe doesn't help with improving the maintainability for your own development habits. I think it won't help you improve due to how heavily penalized the gold-plating is with respect to new development. It makes me feel like I would have to downgrade my code in certain situations, meaning I'd be making it less maintainable or less clean.

Author: Ok, are you satisfied with the feedback this is giving you.. the commit analysis report?

Front-end Dev 1: Yeah, yeah.

Author: Anything to be improved?

Front-end Dev 1: This is more like visual stuff I guess but I think the text is a bit long and maybe that could be shortened into small chunks. I think if you're going to check what the tool does each time you don't need to know the definition of each thing. It would be nice to have a smaller UI to tell you what happens with each thing to make it easier for first time users. Stuff like that. What would also be cool is to get that single value right in your CI integration in GitHub. You know that letter grade.

Author: Ah you mean the CI check feedback?

Front-end Dev 1: Exactly! So you get like that right there. Because right now it says whatever Better Code Hub says, but if that value (the maintainable production rank) could be added right there it can be like "ah ok, why has that changed?"

Author: Ah that's nice feedback, thanks. Yeah so I think that's all the questions I have for you, I really appreciate you sitting with me.

Front-end Dev 1: Yeah man cool, thanks for doing this. I really like how you structured everything, it was very professional.

Author: Ah thanks, I tried.

D.5 Interview - Front End Developer 2 / Contracted

Author: The first question I'd like to ask you is, what is your current role in software development?

Front-end Dev 2: I'm a front-end developer and I'm hired by SIG as a contractor. I'm in the middle of redesign, so not only new design but also adding functionality that fits with the concept of this redesign, so its really focused on front-end development in an Angular project at the moment.

Author: Ok so is that your typical development, you're a front-end guy.

Front-end Dev 2: Yeah, so I've been specializing for the past 3 or 4 years in that field of work. Before that I did a lot of other things. In the first 2 or 3 years of my career I worked as a Java developer, C# developer, I did a lot of very old XML stuff, ancient Java stuff, I'm familiar with all the build tooling in both of those worlds kind of. But along the way I wanted to know more about front-end development as I like the visual aspect of building software, so that's why I wanted to specialize.

Author: How many years of experience do you have doing software development?

Front-end Dev 2: That would be 7, professionally.

Author: Any little side projects you work on also?

Front-end Dev 2: Not really, no. Its a bit up and down with the side projects, for me it is the work that is nice and that's enough at some point.

Author: You need to get away from the code at some point right?

Front-end Dev 2: Sometimes yes.

Author: Ok so these projects that you've worked on, they've typically been web applications or mobile applications or...

Front-end Dev 2: Yeah they are mostly web applications. I have worked on projects dealing with server-side processing, but the past 4 years has been solely web applications.

Author: Ok, and you typically are working in a team right?

Front-end Dev 2: Yeah, yeah.

Author: And what would you say is the average size of the team?

Front-end Dev 2: I think its around 5, that includes developers and testers typically. This depends on where you go and which company your working, there may be an analyst on board. I always say the product owner is not part of the team. So I would say 5 or 6.

Author: So it would be fair to say you're used to working on highly collaborative projects?

Front-end Dev 2: Yes.

Author: Ok great. So now I would like to ask you some questions on software maintainability, specifically this ISO 25010 standard. You've had a chance to look at it?

Front-end Dev 2: Yeah.

Author: So let me ask you first of all, to what extent do you agree with this standard's definition of software maintainability as it relates to quality? Does it capture the total essence of maintainability?

Front-end Dev 2: I would say a 4 (moderately agree).

Author: Ok so what would you say you disagree with then?

Front-end Dev 2: I would not say I disagree with any of these aspects. What colors my answer a little bit is that I don't like the extremes in this. What I always miss here, or maybe its in between the lines, but I miss the reality aspect. The practicality. You always face situations in which you say "I know I should follow this rule or specification as a means of fixing this, but right now this is a better solution because reason A, B, C... whatever. I still think this is a good guideline, and everyone should know about this, but you shouldn't blindly follow this standard. The reality that comes with facing a certain problem or encounter a certain question, that always is coloring your approach.

Author: It's contextual you'd say?

Front-end Dev 2: Exactly.

Author: Ok, so with this standard in mind how important is it to you that you are actually producing maintainable code?

Front-end Dev 2: For me it is very important (5). Particularly now in my current job, I'm typically working on a project for around a year's time and then move on to a new project. When you move on to a new project and you see new code, its very important that you make that code familiar as quickly as possible. So writing code that follows maintainability guideline makes it easier to grab yourself. Of course you have situations where you work on the same code base for many years, and its all "in the head" and that's fine, but I don't think that is how modern software development should be. Most important is readability and understandability for other people. It always gives you a nice feeling when you leave code better than the way you found it, and I think it's important to be part of a team that has this same mentality.

Author: Ok great. To what extent is it important to you that you feel you are continuously improving in your, let's say, maintainable code production? In that if you make code that you feel is of a certain maintainable level, do you strive to make it better the next time you contribute to a system?

Front-end Dev 2: Umm no, because I'm a realistic type of guy and I know there is a reason you build certain things, like you need a certain functionality or you need your software to perform in a certain way so there's a business model behind it. I know that everything is driven by the business. In a side project this can be different, as you may be deep diving into a technology to learn everything and make it better. But real life projects like at SIG, at some point it is enough.

Author: Ok so let me ask you how to what extent would you be interested in having this idea tracked, the manner in which you make maintainable code? Would you like to see a trend over time of how you're doing as far as maintainability goes?

Front-end Dev 2: Yeah I would. Especially if you're part of a development team and really owning a product, then certain ways to see how to make your code better or work faster is really important. It gives you the right kick to come to work everyday and feel well as a development team. So again I would say a 4 (moderately agree).

Author: So let's talk a bit about Better Code Hub. Better Code Hub presents itself as this tool that can measure how maintainable a system is according to these 10 guidelines. These guidelines themselves are derived from the principles found in the ISO 25010 standard. Do you feel Better Code Hub is accurate in measuring maintainability according to ISO 25010? Do you feel it fits it?

Front-end Dev 2: I have not worked that much with Better Code Hub. I'd like to work with it more, and actually include it more in my own personal work flow. With the limited knowledge I have I think it gives good insights for the developer to put the right focus on thigns. It gives the right information to see "ok this, this is good. You could have been better on this part." Overall as a complete measurement system for your software product, I don't think it is covering everything. It's a workflow tool, but when I think of measuring systems it is really about dashboards.

Author: So yeah it's not a holistic view of the system. It's focusing on the maintainability as it relates to these 10 guidelines. Would you say in that sense it matches the maintainability standard? Or to what degree do you think it matches the standard?

Front-end Dev 2: I think it matches the standard around a 4 (moderately agree). I trust the tool, I understand the concepts behind it, but I would not say it is always leading in my decisions going forward with development.

Author: How often do you actually visit Better Code Hub to see how well a commit or merge request is doing according to the tool?

Front-end Dev 2: Currently, almost never. Way to little actually. When I worked on Better Code Hub itself I did not check my individual commits but rather the merge request or the total score.

Author: Ok. Do you think the feedback is useful in determining how maintainable the system is?

Front-end Dev 2: I think it is indeed useful, but on the other hand I think it is a lazy tool in the sense that.. I can imagine you end up in a situation where you have an 8 or 9 out of 10 and that could be good enough for the business. While there are things that are red and alarming, and you need to fix stuff, but if it's good enough to your own standard then the tool could be helpful in showing you "you're on a very good track now, but if you fix this it could be even better". It would be nice to receive really focused areas, like if you're in a specific role in a development team you can focus on different aspects whereas in another team you may not do that. It might not be beneficial to put your effort there. That might not have been completely the question but it triggered me. I think BCH could be more useful as a helping hand in a more active way.

Author: Ok that kind of leads into my next question, do you think the feedback is actionable? So you see how everything is graded, but I get the feeling from you that the actionability is not there?

Front-end Dev 2: Yeah sometimes not enough. If you have a system that needs attention, then it is clear and actionable because if you see 5 out of 10 then you clearly need to do some work here. However if it's an 8 or 9 out of 10 then it depends on your own mentality and your team's mentality, mabye you have a good reason to leave it without going for the 10. At the same time, I also know that those could be considered excuses, but it's not always as black and white as following the underlying model. It strongly depends on the people working on the software itself.

Author: What aspects, if any, do you find most beneficial in Better Code Hub?

Front-end Dev 2: I really like that it's a tool which integrates nicely into a common work-flow of software developers. Currently it only integrates into GitHub, that could be a limitation, but the concept is right. It's very visible in your daily work-flow, so that's the one thing I like the most.

Author: Are you satisfied with the feedback?

Front-end Dev 2: In general, yes. Right now it's really that you have to click a couple of times before you see the feedback, where it could be more helpful to see the urgent things first you know. To really get it in your face, which by itself makes it more actionable already. I think its a bit old-fashioned here and there in the application flow of BCH itself, you really have to go look for it.

Author: To what extent do you agree with the statement "If I did not use Better Code Hub, the quality of my software would be the same."

Front-end Dev 2: I'd say a 2 (moderately disagree). I disagree as I don't agree with those people that say they don't need a tool, I don't believe that. We're all human, we're missing things, we may all know the rules but we don't follow them always.

Author: So now I want to talk to you about this maintainable production metric, or this Commit Quality Report you get. This is an experimental branch that integrates this model which does some calculations to give you a grade of your commit as far as maintainability is concerned. You've had chance to look at it?

Front-end Dev 2: Yes, let me just open it again here.

Author: So yeah it gives you this letter grade and some calculation there, it is a bit hidden away exactly how these things are measured, but we try to give you enough information there to understand it. So I'd like to ask you, to what extent do you agree that this calculation of the rank, the Commit Quality Impact rank, you receive is clear and comprehensible?

Front-end Dev 2: Well it is clear as you give the definition within the Commit Quality report. The biggest question I had was, is this all? It's telling you something if you put the right focus on your code in this kind of way, but is it also making my code good or bad. So that wasn't the most clear. Is this the only thing I have to look at when I check my commit or is there more to it?

Author: Yeah so this is really intended to be an analysis of your commit, while the feedback Better Code Hub gives is analysis over the system. So this is an addendum to it, they are meant to work together. In this way the feedback you get can be a little more nuanced, as Better Code Hub may tell you "this is future-proof code" but then the commit may tell you "you were leaking productivity here or you affected the guideline negatively by this much."

Front-end Dev 2: Yeah I understand this, it makes sense and I understand the concept. The problem may be that I've inspected few commits, I can imagine if I inspected more then it may get more clear. For me, apparently I touched quite some code which was good already, which could be seen as a waste of effort and is report as "gold-plated." So I see some are red in this block, while they are green in the other block. So I'm unsure if it really is bad or not.

Author: Ok so lets say for this instance it is not clear. I will get to the "gold-plating" a bit later, but I will say that this is kind of objective in the sense that you've added code to the system and the system you're working on is quite good, a 9 out of 10 in Better Code Hub. So in this case it may be the case that you see a lot of instances where you gold-plated the code. I'll tell you more the specifics on that in a bit but I'd like to ask you to keep in mind also those systems that are not that good. I know it's hard to kind of remove this bias, but we'll get to that in a bit.

Front-end Dev 2: Ok, I see.

Author: Ok so as far as the letter grade you get for your commit, how does this rank you receive affect your motivation going forward? Is this something that impacts you negatively or positively?

Front-end Dev 2: It's not impacting me negatively. I would say positively because, well in this case it's a D and orange so it doesn't look so good, so I'm being triggered to look into what is going on and what is this telling me.

Author: I assume you have some expectation of the code you deliver, does this rank meet your expectation as far as maintainability of your contribution?

Front-end Dev 2: Ummm let's see. Well actually what I'm missing, or what I expected before I looked into this report, was that I would get a report on how my commit impacts the guideline measurements and that is not what I got out of this report. I see some "gold-plating" things going on and at first I didn't know what gold-plating was so I was happy to see it explained in the report, but maybe that is just a lack of knowledge before I started using it. I expected just a reflection of my commit and that this change I did impacted this guideline because you did or did not follow this rule we cover in the guideline.

Author: Ok so if instead of it saying "gold-plated", but rather positive change here would that be more clear. Because that is in fact what you did, you made a positive change to this guideline.

Front-end Dev 2: Yeah but to what extent.... well ok you show that. Maybe my expectation was just off as I was expecting something really showing a code fragment and you have feedback saying "you helped here" or "go back to the drawing board here as this block of code is not following this guideline."

Author: Ok let me rephrase the question a bit then. You contributed this code to the project, did you have some notion as to the maintainability of the code you were delivering?

Front-end Dev 2: Well like I said we are not really following the Better Code Hub scoring, so I didn't have anything in mind regarding the individual guidelines. For me that is really a subconscious thing, if I observe something being too big or complex I change it without counting the lines or whatever. That's why I really like a system that is alerting a bit more. For me that works better.

Author: Hmm ok, what I get from you is that you did not have very high expectations... you created the code and you felt it was good, and maybe "good" did not mean maintainability but it satisfied some requirements you had. So you contributed and you got this CQI rank here. It's a D so you said "ah this doesn't look so good." It saying that you didn't do that good, did that match the expectation of what you had for that contribution?

Front-end Dev 2: Ah it's difficult, because for me when I consider how maintainable is my contribution is when I submit the pull request. So for me, a commit is a temporary snapshot and I'm not really focused on it being maintainable for the commit's sake. I tend to make a commit at the end of the workweek even if the code is broken or not finished as I don't want to lose my work in the event my laptop gets stolen over the weekend or something.

Author: Ok, that's fair enough. Now let's talk about gold-plating. So Better Code Hub has this fundamental principle of the "Definition of Done", this is seen as the point to which you've satisfied these guidelines so the system is deemed satisfactorily maintainable and is future-proof. The idea here is that if you continue improving the maintainability past this point, you are over-engineering your code in some ways so in fact this extra work is a loss in productivity, which we call "gold-plating". So you are improving the code quality but likely you could put your development efforts elsewhere, because there is a point of diminishing returns when you continuously increase the maintainability of your code and eventually you need to focus on other aspects which may be lacking. Just because a system is very maintainable does not necessarily mean it will be successful if other areas are lacking. So this is the definition we will use, to what extent do you agree with this definition of "gold-plating"?

Front-end Dev 2: Yeah I'd say a 4 (moderately agree).

Author: Ok a 4. So in this metric specifically gold-plating is heavily penalized. It's seen as bad as non-compliant code in Better Code Hub. It's also objective as it looks only at the degree to which maintainability is changed through the contribution, not the specifics of the contribution itself. So even if you are adding new code to the system, if that code is of a higher maintainability than the already compliant system is and by its addition will increase the overall maintainability of the system then you will be penalized for gold-plating. So to what extent would say the metric is fair in penalizing gold-plating?

Front-end Dev 2: Yeah, that's a good question. My feeling says 2 (moderately disagree).

Author: Would you like to expand on that a bit?

Front-end Dev 2: Well I think the addition that you just said [about new code being penalized for gold-plating] is triggering me to give it a 2, because I don't think by adding a feature which is of high quality and increases the maintainability of the system... that should not be seen as gold-plating. I don't agree on that part.

Author: Ok so what about existing code? Do you find it fair to measure on refactored....

Front-end Dev 2: Yeah that is more debatable. Ummm....

Author: Ok so let's try to put it into 2 different buckets, code that is being added to a "done" system vs code that is being refactored on a "done" system.

Front-end Dev 2: Yeah ok but it doesn't say that the part of the software that you touched in this commit is also done, right? It doesn't say that it was good enough?

Author: Right, so if it was good enough... Better Code Hub looks at 2 snapshots, it looks at what was there prior to the commit and what is the maintainability with the commit added. So if you did a refactoring, if you touched old code while the system was already at it's compliance goals.. for example you are working in a file and see a big unit, so you go ahead and break it up, but the unit size was already good... then would you say that is gold-plating?

Front-end Dev 2: No, because I think that part which you touched is better now. So the next developer that comes along and has to modify this feature or file is more happy than if the unit was left alone.

Author: Ok but if Better Code Hub tells you the system is future proof... that the code is maintainable enough, let's say 90% of units are good and the guidelines are compliant... if you still decide to go fix that unit Better Code Hub will tell you you're "gold-plating" because you spent time and development effort to make a change that was not really necessary in this sense. Indeed it is making it better, that is something that is agreed upon. But in this way, do you see how this could be a loss in productivity in that you didn't need to make that change and could have spent your time doing something like adding a new feature or something like that?

Front-end Dev 2: I don't think so, well for 2 reasons. One which I just explained, it helps the colleague who is working the same piece of code happier and the other is the fact that it makes the developer actually contributing the code happier. Not because it's nice to make things better, but it may annoy you and in turn decrease your productivity if you have to work with an ugly piece of code for a full day. So sometimes it is more beneficial to spend just a bit of effort to make it better and then the rest of the day maybe you go faster or more efficient because you have increased your happiness for that day a little bit.

Author: Ok, fair enough. So now you know how this metric grades gold-plating, is that clear to you?

Front-end Dev 2: Yes.

Author: And just to confirm, you still give it a 2, you moderately disagree that it is fair in the manner in which it penalizes gold-plating?

Front-end Dev 2: Yes, because I think it is very black-or-white and that is what I don't like about this approach with gold-plating.

Author: Ok great. Do you think there needs to be some nuance built into that measure? Do you think tracking the gold-plating has a place, it should be reported but not necessarily affect your score or that there is some threshold for which an improvement is not always called gold-plating if the change is relatively small, but large changes to an already maintainable system should be?

Front-end Dev 2: Yes, I think it is nice to have a place where you inform the developer that they are gold-plating code. I don't agree that it should be penalized as heavy as a real violation in your commit quality. You're not going to revert it. If you make things worse you have a good reason to revert it because you see it has negatively affected your code. If you improve it though and get penalized for gold-plating, it would be stupid to then revert it or make it worse. For me that is the kind of distinction that needs to be made in that sense.

Author: Ok, ok. This gold-plating feature is intended to highlight the interplay between improving software quality and productivity. Do you feel that the metric captures this aspect?

Front-end Dev 2: Yeah I definitely see where it is going, but for me its too black or white with the gold-plating. To me, developer happiness is a very important part to developer productivity and that comes from having a good chair, to not having to hunt for a desk everyday, to these kinds of things. If you really have the task to work on a certain class or feature and you don't agree with how it's built, you should be allowed to go ahead and make it better as a clean start for you to be more productive.

Author: Ok, fair enough. Is the feedback provided here more, less, or the same as far as actionability goes?

Front-end Dev 2: It increases actionability by knowing that gold-plating is penalized similarly as making code worse. If I just see this D here [CQI rank], I wouldn't know if it is fine to include this commit in a pull request and let someone review it or if I need to go back to the drawing board.

Author: What do you find most beneficial, if anything, about this sort of grading being included in Better Code Hub?

Front-end Dev 2: Yeah I think it's nice. I don't think that gold-plating should be categorized together with really violating the guidelines, but I definitely see a place for providing this sort of feedback. I think it is good to have that insight.

Author: Ok so if we take away the gold-plating and we gave you the message positive impact instead, and then we took away the penalization, then this commit would likely be a B or something higher but we left the gold-plating messaging. Would you feel that is a more accurate feedback of what happened within the commit? So you're getting a "good" score in that sense but you're still being notified of the gold-plating.

Front-end Dev 2: Yeah, I would say that is more accurate.

Author: Ok, great. Are you satisfied with the feedback given in the CQI report?

Front-end Dev 2: Yes and no, in short I do see a place for these indications. I just don't think gold-plating should be seen as negatively as making a negative change to the product.

Author: Ok, great. That's all the questions I have for you. Thank you very much for sitting with me this is useful information.

Front-end Dev 2: Cool, you're welcome.

D.6 Interview - Full Stack Developer 2

Author: So let me first ask what is your current role in software development?

Full Stack Dev 2: Well in my day to day job I'm mostly involved in the .NET stack. I consider myself a full stack developer, so I go all the way from back end to front end where I deal with TypeScript and some more web related technologies.

Author: Ok great and how many years have you been a developer professionally?

Full Stack Dev 2: It's been around 7 years I believe. I started in 2011 so its going to be 7 years this year.

Author: And what type of projects do you typically work on? Is it primarily web development, enterprise software, mobile...

Full Stack Dev 2: I think every job I've done was sort of its own. I started in game development, then moved on to telecommunications projects which is mostly embedded software. I think moved to software which created optimization software so I guess more enterprise related stuff. Now I work also in an enterprise project with over 100 developers with some web development.

Author: Wow, so a pretty nice spread. What would you say is your typical team size you've worked in?

Full Stack Dev 2: It's mostly been Scrum teams so it has been roughly 7 to 9 people, sometimes less like 6 - 8 people. It depends on a per team basis.

Author: Do you have any personal projects you work on also?

Full Stack Dev 2: I used to have more personal projects but I don't do that much anymore.

Author: Ok. So it's fiarly safe to say you are used to working in highly collaborative projects?

Full Stack Dev 2: Definitely, yes.

Author: Ok great. If you don't mind me asking what is your educational background?

Full Stack Dev 2: I have a bachelor's degree in computer science.

Author: Great. So now I want to ask you a few questions regarding software maintainability, specifically this definition provided by ISO 25010 standard you've reviewed. I'd like to ask you, to what extent do you agree with this definition?

Full Stack Dev 2: I'd say I moderately agree...

Author: Ok so what aspects would you say you disagree with then?

Full Stack Dev 2: I think in most projects I would completely agree really but for example in game development projects you don't really care much about the maintainability, as the project is done one at a time so when it is done you typically don't revisit that code. So in those situations it is not as applicable, if it was not for that then I would say I completely agree.

Author: So just to clarify, you completely agree with the definition in that it encapsulates what software maintainability is but there are some edge cases where it is not so applicable.

Full Stack Dev 2: Yes, definitely (5).

Author: Ok, great. So let me ask, to what extent is it important to you that you are producing maintainable code according to that standard?

Full Stack Dev 2: Yes, I think it is extremely important. I'd rank maintainability as like the 2nd or 3rd most important aspect of my code, following functionality.

Author: To what extent is it important to you that you feel you are continuously improving your own development habits with regards to maintainability? Would this be something you wish to track?

Full Stack Dev 2: I think it's very important (5), yes

Author: Do you think you would come to a natural stopping point in improving your maintainability habits? Is there a point where you feel the code is done?

Full Stack Dev 2: That's a hard question. Sometimes I try to find a balance between how much maintainability I want to deliver versus adding new features, but I don't really have any strict rules for that. It depends on the workload I think of the team, so if the workload is higher then maybe I don't divert time to maintainability.

Author: I'd like to ask you few questions about Better Code Hub now. Better Code Hub presents itself as this tool which can effectively measure software maintainability to the standard of ISO 25010 with its 10 guidelines. You've used the tool and analyzed some of your commits before. Do you feel the tool is accurate in this measure? Does it capture the standard well?

Full Stack Dev 2: As far as I see, it's good enough. It fits the standard. It gives enough overview of what you want to see as a developer without getting too much... too much details and statistics.

Author: Ok. How often do you visit the BCH UI to review a system when you're working? Do you check it after each commit or each merge request or some other frequency?

Full Stack Dev 2: I think with these kind of tools I like to check it once per day.

Author: is the feedback it provides useful for you to determine how maintainable is your system at that point in time?

Full Stack Dev 2: I think for me what is important is to figure out how the system is improving and where to go next, which it does well.

Author: So you think it is in fact actionable?

Full Stack Dev 2: Yes, I think it is actionable yes.

Author: What aspects, if any, do you find most beneficial to your development efforts?

Full Stack Dev 2: I think what I find most useful is the refactoring candidates as it allows me to pinpoint the areas which need most attention, and outlines the severity of the problems.

Author: So it's really about the actionability it gives you through the refactoring candidates?

Full Stack Dev 2: Yes, yes.

Author: Are you satisfied with the feedback it currently gives you?

Full Stack Dev 2: Well I think in some cases it is a bit minimalistic... sometimes it is just "yes" or "no" and doesn't really show you well how much you've improved from commit to commit. I think that could be a little bit changed.

Author: So you would like a better overview at the commit level?

Full Stack Dev 2: I think that would be nice, yes.

Author: So let me ask you this, to what extent do you agree with the statement ”If I did not use Better Code Hub, the quality of my software would be the same?”

Full Stack Dev 2: I know I would have problems measuring the maintainability just myself so I would say it would make it worse if I did not use the tool. It gives you an easy to access highlight of where you are which allows you to proceed.

Author: Ok so would you say you completely disagree or disagree?

Full Stack Dev 2: I would say I completely disagree.

Author: Ok so now I'd like to talk to you about this Maintainable Production metric that I've asked you to take a look at. You've analyzed some commits with it and seen the CQI report that it gives?

Full Stack Dev 2: Yes I have.

Author: Ok, great. To what extent do you agree that the calculation of the CQI rank you get is clear and comprehensible? You understand how you received this rank?

Full Stack Dev 2: Ok so I understand the categories of how you receive the rank that you do, it's described well in the report. What I don't really get is what is the goal in that? Should I strive to only have commits be in the category with least improvement [the A rank] or are commits which are ranked C good enough? What I found most useful is the breakdown of which guidelines were impacted and which were gold-plated. I'd moderately agree that it is clear.

Author: Ok so the grade was not so important to you but the breakdown itself was more useful?

Full Stack Dev 2: Yeah, indeed. The breakdown was more helpful.

Author: How does the rank you receive affect your motivation going forward?

Full Stack Dev 2: If I produce code which receives a worse rank I'm actually more interested to see why that is. It's a positive motivation then.

Author: So you're saying receiving a worse score motivates you more positively than receiving a good score? You want to see what can be improved?

Full Stack Dev 2: Actually yes.

Author: Does the rank you receive generally match your expectations of the maintainability you delivered with your contribution?

Full Stack Dev 2: So I think I haven't worked with it long enough to have a good overview, but the ones that I did analyze were more or less matching indeed.

Author: So with this metric there is this idea of gold-plating, which we define as the over-engineering of code past this point of satisfactory maintainability. Basically that you are continuously trying to improve the quality when it is not needed as the code is future proof already, and you've reached this point of diminishing returns as far as maintainability is concerned. Is that a clear definition that you can agree with?

Full Stack Dev 2: Yes, it's clear to me. It makes sense.

Author: Yeah so the idea is that by gold-plating you are losing productivity, because you are making unnecessary changes or improvements when you could have devoted your efforts and attention elsewhere to areas which may be lacking.

Full Stack Dev 2: Yes I completely agree with that definition. I've experienced it myself and seen other people gold-plating and know that it can be a problem.

Author: Ok great. So with this metric gold-plating is heavily penalized, it is in fact penalized as bad as having non-compliant code in the system. The idea is that we really incentivize you not to gold-plate in an objective sense. This means that even new code that is being added, if it improves the maintainability of a guideline that is already compliant, you will receive a penalty there for gold-plating. We want you to reach this point of satisfactory complaince or maintainability, but we don't want you to over-engineer it...

Full Stack Dev 2: So let me just double check... if you deliver new code which is more maintainable than what the current system is, then you will still be penalized for gold-plating?

Author: IF the system is compliant under those guidelines that are improve, otherwise it is seen as a normal positive change which you are rewarded for. If the system is already good, then you would be penalized. It also depends on how all these guidelines move together, right? So maybe you gold-plate a little on this one guideline, if the others are all positive changes then overall you would receive a better rank. Only because you gold-plated it is not as high as it could be. So I'd like to ask you to what extent do you agree that the manner in which gold-plating is penalized is fair?

Full Stack Dev 2: Hmm so I think there should exist a metric to track this idea but to what extent do I think it is fair... so I when I analyzed a few guidelines I was surprised to see this gold-plating pop-up and see that it was something negative here. I think it can be too surprising in this case as I think "well I'm improving the guideline" but then in fact I'm penalized.

Author: So let's walk back a bit to the definition you agreed with earlier. Let's say you have the guideline "write short units" and it is compliant, and then you make a contribution which increases this guideline further, would you say that is gold-plating?

Full Stack Dev 2: Yeah it's still gold-plating.

Author: So is it that you feel the feedback for gold-plating is not nuanced enough or is it more confusion over the kind of intertwining of the guidelines in the calculation? How the impacts are measured together?

Full Stack Dev 2: Yeah so it's mostly about how the impacts over the guidelines are measured together, not necessarily the feedback of gold-plating.

Author: Ok, I think that's fair enough. So this gold-plating feature tries to capture or highlight this interplay between increasing software maintainability and developer productivity. So if you're gold-plating you may be leaking productivity while if you're improving you are making good use of your effort in this sense. Do you feel that this metric accurately captures this aspect of development?

Full Stack Dev 2: I think that is what is useful the most with this feature is that tries to show what the overall maintainability of the commit is and how it contributes to the overall maintainability. So I think it does well in saying how maintainable the commit is.

Author: OK, ok. But do you see the relationship clearly to productivity then? Is that clear from the feedback?

Full Stack Dev 2: Well I see it when there is a lot of gold-plating involved.

Author: Ok, ok. So then do you agree that the gold-plating feature highlights this?

Full Stack Dev 2: Absolutely, yes. I think it is accurate but can be confusing, it's not so intuitive. But I do understand what the feedback is intending to provide.

Author: So would you say the feedback provided by this report is more, less, or the same as far as actionability is concerned as compared to the feedback BCH currently gives? Do you know what to do going forward with development?

Full Stack Dev 2: I think it is a bit better yes, I think it outlines which areas I specifically don't need to focus on better and which are more relevant.

Author: Ok great. What do you find most beneficial about this metric being included in Better Code Hub?

Full Stack Dev 2: I think it would be interesting to investigate the historical overview of commits measured with this report. I'd like to see how maintainable are my commits over a long period of time, to see a trend. I think it would be useful to improve as a developer in producing more maintainable code.

Author: Would you say you are satisfied with the feedback in this CQI report?

Full Stack Dev 2: I would say it is decent. I'd prefer seeing this trend line overtime, to see how this commit actually performs against your historical average. I think it would be more motivating.

Author: Ok, great. I think that is all the questions I have for you, I appreciate you taking the time to sit with me and participate in this little experiment.

Full Stack Dev 2: You're most welcome!