# SOFTWARE ENGINEERING PROCESS REPORT

## AIM 1: TO CONSIDER THE WAYS IN WHICH THE SOFTWARE ENGINEERING PROCESS CAN BE MEASURED AND ASSESSED IN TERMS OF MEASURABLE DATA.

### INTRODUCTION

Measuring the software engineering process is a vital part of software development as it allows the use of information that has been collected and assessed to develop theories and models. From a qualitative and quantitative viewpoint, it is also used to support production planning, monitoring and control, and to determine a product's state in relation to a budget and a schedule.

In general, techniques such as Personal Software Progress is used to improve project estimation, which is done by gathering size, time, and defect data on an initial set of software projects and performing in-depth analyses on it. Furthermore, PSP incorporates two primary assumptions; the collected data and the analysis made by an individual can provide significant benefits to that individual and that these benefits are large enough that the developer will continue to use this technique. Back in the earlier generations, PSP used to create and print out forms where they were manually logged in terms of effort, size and defect information. Additional forms would use this data to support project approximation and quality assurance. As a result, substantial overhead were required while form filling. Nowadays, the common types of metrics used in measuring software engineering process are Formal Code Metrics, Developer Productivity Metrics, Agile Process Metrics, Operational Metrics, and Test Metrics.

### FORMAL CODE METRICS

Also known as Static Program Analysis coupled with Formal methods, it is primarily used to verify code and to identify and diagnose run-time errors. Formal methods consists of the result of a program being understood and reviewed through the use of rigorous mathematical methods, which includes denotational semantics, axiomatic semantics, operational semantics and abstract interpretation. It comes with two approaches; property based specification and model based specification. Property based specification characterises the operations performed on the system and the relationship that exists among these operations. It is comprised of signatures; which determines the syntax of the operations, and axioms; which defines the semantics of the operations through a set of equations. Model based specification uses tools of set theory, function theory and logic to develop an abstract model of the system and the operations performed. It is consist of the set of states of the system and it defines the legal operations that is executable on the system and how it influences the current state. Advantages associated with this method is that it is capable of detecting ambiguity, incompleteness and inconsistency in the code, it incrementally grows towards an effective solution after each iteration, and the formal specification language semantics is able to verify self-consistency. Disadvantages includes the method being time consuming and expensive, and extensive training is required since only few developers have the essential knowledge to implement this model.

Since formal methods alone tends to be overly tedious and human-intensive therefore, basic static code analysis techniques such as data flow analysis, control flow graph, taint analysis and lexical analysis are used alongside the formal methods. Static code analysis is comprised of debugging a software by examining its source code before running it. It is examined against a set of coding rules and performed by an automated tool.

Data Flow Analysis is a technique that utilises the gathering of run-time information about data in the software while it is in a static state. It is composed of three main components; a basic block, control flow analysis and control flow path. A basic block is a sequence of consecutive instructions where the control enters at the start of the block and exits at the end of it. The block cannot halt or branch out except at the end. A control flow graph is a visual representation of the code using nodes to illustrate multiple basic blocks. The directed edges are used to illustrate any jumps or branches from block to another. If a node has only an exit edge, this is known as an entry block, if it has an entry edge, this is known as an exit block.

Taint Analysis is used to identify variables that has been infected with user controllable input and traces them to possible vulnerable functions. This process is called 'sink'. If the infected variable proceeds to a sink without being resolved then it is flagged as a vulnerability. This approach may be viewed as a conservative estimation of the full verification of non-interference or the more general concept of secure information flow. Other applications of this analysis includes exploit detection, where user data can be tracked, SQL injection, buffer overflows can be detected, used in PERL tainted mode and can be used for web applications.

Lexical Analysis is used to convert source code into token syntax where the source code is abstracted and manipulated. It represents the initial phase of a compiler, where the lexical analyser works in close collaboration with the syntax analyser to detect invalid tokens from character streams and to identify lexemes which are defined by grammar rules.

Static Code Analysis on its own, scales well but has many limitations attached. Most security vulnerability are very difficult to find automatically such as authentication and access control errors. It produces high number of false positives and cannot recognise configuration issues which are not represented in code. Tools that are based on this have difficulty in analysing code that cannot be compiled.

## BUGS PER LINE OF CODE

This metric is one of the easiest methods to judge a program's readiness to be released; by measuring its defect density. It's natural that a project cannot achieve zero defects in their code as it would cost thousands of dollars per line of code. Stated in Steve McConnell's Code Complete book, Microsoft Applications contain about 20 defects per 1000 lines of code during in-house testing and 0.5 defect per KLOC in production and the industrial average is about 15-50 errors per lines of delivered code.

However, the bugs can be predicted through a technique called Defect Pooling, where the defect reports are divided into two pools, where the distinction between the two pools are arbitrary. The number of unique defects reported at any given time is: $Defects_{Unique} = Defects_A + Defects_B – Defects_{A\&B.}$ and the total number of defects can be approximated by this formula: $Defects_{Total} = ( Defects_A * Defects_B ) / Defects_{A\&B.}$ Defect Seeding is another approach where defects are purposefully injected into the program by one group for detection by another group. The ratio of the number of seeded defects detected to the total number of defects seeded provides a raw approximation of the total number of

unseeded defects that has been detected. The formula given for this approximation: $IndigenousDefects_{Total} = ( SeededDefects_{Planted} / SeededDefects_{Found} ) * IndigenousDefects_{Found.}$

## TEST METRICS

Test metrics are used to monitor and control process and product which drives the project its ultimate goal without any deviation. The general categories of this metric includes code coverage, test tracking and efficiency, test effort, defect distribution, test execution and regression. Different applications of these examples are dependent on the organisational levels. At a software level, development teams may record requirement coverage, defect distribution, defect open and close rate and test execution trends whereas at the department level, the manager of this unit may monitor the testing and defect trends, and the organisation's mean time to detect a defect and the mean time to recover from the defects.

## CODE COVERAGE

Code coverage is a measure that depicts the degree in which the source code of a software is executed through a series of automated tests. It was one of the first techniques invented for systematic software testing and is part of a feedback loop in the development process. High test coverage is a term used to refer to a program that has a large portion of its source code executed during testing and has a lower chance of containing undetected software bugs in comparison to a program that has low test coverage. The common types of metrics used are function coverage; statement coverage, branches coverage, condition coverage and line coverage.

Function coverage is the number of functions defined that has been called. It analyses the values within a single object called a point or an item coverage. It also examines the relationships between different objects in a process is known as cross coverage.

Statement coverage is considered a white box testing approach as it involves the execution of all the statements within the source code at least once. The type of information collected can be used to check the behaviour and quality of the source code and the different path flows in the program. The primary issue of this method is that we cannot test the false condition in it.

Branches coverage is the number of branches of a control structure that has been executed. It tracks the destination that has been visited and flags statements that hasn't visit all of their possible branch destinations. It aims to ensure that each one of the possible branch from each decision point is executed at least once, therefore, ensuring that all reachable code is executed and that it does not lead to any abnormal behaviour of the code.

Condition coverage tests the possible outcomes of true or false for each condition at least once. This practice ensures that the tests explore all possible code paths.

Line coverage measures the amount of executable lines of code that has been completed however, it does not include loops or conditional statements.

Source code instrumentation is an approach of code coverage that involves adding specific code to the source files under analysis, and compiling them to produce dump data for run time analysis or component testing. It is designed to reduce both performance and memory overhead to the bare minimum and this is primarily done through limiting code coverage types, instrumented calls and optimising the information mode that is defined when using code coverage. This type of technique is

limited by execution coverage, as if the program does not reach a particular point of execution then instrumentation at that point gathers no data. As a result, there is possibility of an exponential increase in execution time which leads to its limitation to debugging contexts. Examples of applications that incorporates this technique, is code tracing, debugging and exception handling, profiling, performance counters and computer data logging.

Limitations of Code Coverage includes that it cannot detect conditions that are not in the code as it only monitors the lines of code or expressions in the code that has been executed.

## COHESION

Cohesion refers to the level of strength and unity between the different components of a software and how they are inter related to each other. It determines the strength of the source code by monitoring how close linked each module of the code is, and how rapidly it can perform the tasks given. In qualitative measurement process, the textual code of the program is analysed by sampling and evaluating them against coding standards of the respective programming language. Quantitative evaluation on the other hand, identifies the number of comprised modules and examines their actions according to predefined scales. It is identified as an ordinal type of measurement where modules with high cohesion are preferable as they include traits of robustness, reliability, reusability and understandability and correlates with loose coupling, whereas low cohesion modules include traits of being difficult to maintain, test, reuse and understand.

High cohesion modules are achieved if the functionalities embedded in a class have much in common through its method accesses, and that the methods perform small number of related activities by avoiding unrelated sets of data. This in return, reduces module complexity, increases system maintainability, as logical modifications in the domain would affect fewer modules. Module reusability is also increased as users would easily find the desired component among the set of operations provided by the module.

The various types of cohesions are as follows, coincidental cohesion, logical cohesion, temporal cohesion, procedural cohesion, communicational cohesion, sequential cohesion and functional cohesion. Coincidental cohesion is referred to as one of the worse type of cohesion, as parts of the module is grouped arbitrarily i.e. there is no meaningful relationship between the modules. Logical cohesion is when parts of the modules are categorised logically to perform the same tasks even though they are different in nature. Temporal cohesion is when parts of a module are grouped when they are processed at the same time during program execution. Procedural cohesion refers to the grouping of the modules that follow a certain sequence of execution. It can support different and possibly unrelated activities where the control is passed from one activity to the next. Communicational cohesion is the grouping of the modules when they operate on the same input or output data structure, by accessing or modifying it. Sequential cohesion is when parts of a module are classified as a group as the activities involved produces output data that serves as input data to the next. Functional cohesion is the grouping of modules when the execution of the activities is for one and only one problem related task. It is considered the best and most desirable type of cohesion but it is not necessarily achievable. Under certain circumstances, types such as communicational cohesion may be the highest level of cohesion that the software may be able to attain.

## CYCLOMATIC COMPLEXITY

Cyclomatic complexity is a quantitative metric used to indicate the complexity of a program through the analysis of linearly independent paths through a program's source code. Developed by Thomas J. McCabe, Sr in 1976, it uses the control flow graph of the program and basis path testing where the test cases would result in the cyclomatic complexity of the program. The nodes of the graph would represent groups of commands in the program and the directed edges connecting them, would signify if the second command may be executed immediately after the first command.

A source code containing a complexity of 1 would have no control flow statements. Therefore the complexity M is defined as: $M = E - N + 2P$, where E is the number of edges, N is the number of nodes and P is the number of connected components in the graph. An alternative to this formula is to view the graph in a way where each exit point is connected back to the entry point. In this case, the graph would be considered strongly connected and the complexity would be equivalent to the first Betti number of the graph. This is defined as: $M = E - N + P$.

For a single subroutine, the number of connected components would always equate to 1 and the complexity would be defined as $M = E - N + P$. Cyclomatic complexity can be applied to several programs or subprograms simultaneously and in this case, P would be equated to the number of programs in question, where each subprogram would be represented as a disconnected subset of the graph.

This type of measurement is used to limit the complexity of routines during development, where larger modules were split into smaller ones when the complexity exceeded 10. This approach was adopted by the NIST structured testing methodology. It can also be utilised to measure the "structuredness" of a program by iterating through the control flow graph, reducing it into its subgraphs that consist of a single entry and a single exit point, which are then replaced by a single node. If the program is structured then through this reduction process; it can be reduced to a single control flow graph node. On the other hand, if the program is unstructured, the iterative process would identify the irreducible component of the program.

Cyclomatic complexity can also be used to determine the number of test cases that are necessary to attain accurate test coverage of a particular module. The complexity would act as an upper bound for the number of test cases necessary to achieve a complete branch coverage, as well as a lower bound for the number of paths that exists in the control flow graph, where the assumption is made that each test case takes one path and the number of cases needed to attain path coverage is equal to the number of paths that can actually be taken. In relation to Cohesion, there is a possible correlation between a higher complexity program that has a lower level of cohesion as there is more decision points as a result of implementing more than a single well-defined function. A higher cyclomatic complexity rate also correlates to a higher frequency of defects occurring in a function.

## COUPLING

Coupling is the measure of the interdependence between software modules. It is generally contrasted with cohesion, where low coupling is associated with high cohesion and vice versa. Low coupling is an indication of a well-structured system, and combined with high cohesion, it contributes to the general goals of high readability and maintainability. There are two types of coupling; Efferent coupling and Afferent coupling.

Efferent coupling measures the amount of classes in which a given class is dependent on. This includes inheritance, interface implementation, parameter types, variable types and exceptions. It is also known as outgoing dependencies and is often used to calculate the instability of a component in software architecture, where 0 indicates that the component is maximally stable and 1 for maximally unstable.

Afferent coupling measures the number of external classes coupled to classes of a package due to internal coupling. If the result returned is 0, then the package does not consist of any classes or no external classes are using the package's classes. This type of measurement is mainly applicable to object oriented systems.

The instability index is a measure of efferent coupling against the total coupling. If there is high afferent coupling but no efferent coupling, it is still acceptable to assume that the particular class is stable. Furthermore, if a class has high efferent and high afferent values then it tends to be a medium that is highly susceptible to bugs.

### PROGRAM EXECUTION TIME

There are many approaches that exist to measure execution time, but no single technique. It is primarily dependent on a composition of multiple attributes such as resolution, accuracy, granularity and difficulty. Resolution depicts the limitations of the timing hardware, while accuracy describes the closeness of the measured value to the actual time if the perfect measurement was obtained. Granularity describes the portion of the code that can be measured. Coarse granularity define methods that would measure execution time per process, per procedure or per function basis, while fine granularity define methods that can be used to measure execution time of a small segment of code such as a loop or a single instruction.

The data that is collected is typically used to refine estimations, optimise code, analyse real-time performance and to debug timing errors. The optimisation of code can be implemented through coarse grain methods or fine grain methods, depending on the portion of the code that needs to be optimised. If it is on a global scale then coarse grain techniques are generally used, whereas for localised optimisation, fine grain techniques would be sufficient to trace the execution time of a single line of code. Analysing real time performance can incorporate a coarse grain technique but would not provide sufficient accuracy, therefore fine grain techniques are usually used. Debugging timing errors requires a fine grain method combined with maximum resolution as it is often necessary to not only measure the user code but also the real time operating system code. It will allow for the detection of any anomalies that may occur such as missed deadlines or tasks that are not performing at the desired rate.

## AIM 2: TO DELIVER AN OVERVIEW OF THE COMPUTATIONAL PLATFORMS AVAILABLE TO PERFORM THIS WORK AND THE ALGORITHMIC APPROACHES AVAILABLE.

### HACKYSTAT

Hackystat is an open source framework for gathering, analysing, visualising, and interpreting data as well as the annotation and dissemination of the software development process. The procedure is done in an unobtrusive manner where software sensors are attached to their development tools and returns raw data on the development process to be stored in a web service called the Hackystat Sensorbase. This is repository that can be queried by other web services to produce high level abstractions of the collected data such as visualisations or annotations or to be integrated with other internet based communications.

The basic Hackystat architecture and information flow consists of an XML database, a web server, developer tools with sensors attached, a browser and a mailer. The sensors gathers activity data, size data and defect data at 30 second intervals. The user is registered with a Hackystat server, which sets up an account with a password that prevents unauthorised access to their metric data or any uploading of anonymous data. On the server side, analysis programs are ran regularly on all the metrics that are

provided by the user. One of the fundamental analysis that is performed, abstracts the raw data stream into a presentation of the user's activity at a 5 minute interval over the course of a day. The results of the analysis are available to each user from their account home page of the web server and can be extracted manually to support project development activities. The framework can also define alerts, where there are given threshold values for the analyses that are being performed. If the threshold value has been exceeded then an email is sent to the user to indicate that the newly discovered data may be of interest to them along with an URL to display more details on the data in question at the server. This is known as Complexity Threshold Alert and it enables the system to track the complexity of the classes that the user has worked on.

The project was initiated in early 2001, with the operational release of the server and a small set of sensors in July. The server is coded in Java and is comprised of 200 classes, 1000 functions, and 15,000 non-comment lines of code. The JBuilder Code of the client side is much smaller as it only contains 200 lines of Java and 400 lines of Lisp.

Effort data is collected when the user actively modifies a file and has a fixed grain size of 5 minute increments. If the user is not actively changing a file, then they are considered to be idle. Size data is collected in terms of the amount of new methods added during a given day, while pre-release defect data is automatically gathered by attaching a sensor to a unit testing mechanism and post-release defect data is gathered by attaching a sensor to a bug reporting system.

## TIMEULAR

Timeular is device that supports the user in visualising their time spent on certain activities which leads to encouraged productivity. Companies such as Google, Facebook, Cisco, Intuit, Audi and Airbus all incorporates this platform. The tracker monitors time-sinks and takes action based on real time data. This is done by identifying what the user would like to track, and linking the physical device to the software through Bluetooth. The sides of the tracking device is customisable, but it is generally 8 sided, where each side represents a task, and the size that is currently facing up, is the activity the user is currently performing. Statistics are produced to give the user insights on how they are spending their time. They are able to edit and monitor the time entries freely through the app and export the data in CSV format, or interact with the public API.

It is one of the easiest ways to track time, and allows the users to become aware of their work efforts. It encourages effort spent on activities that would drive the user towards their goals. It makes time tracking tactile, where the user would resist performing less productive tasks and it immediately offers data on the amount of time spent on the previous activity.

## GITHUB

Github is a web-based version control service using Git. It provides various functionality, such as source code management, access control, and several collaboration features including bug tracking, feature requests, and task management. It supports social networking functions such as feeds, followers, and a social network graph to represent how the developers work on their version, also known as forks, of a repository and what fork is newest. Users are able to review changes to their code or someone else's code, and the commit history. Pull requests are also available with code review and comments.

Github's code review tools are built into every pull requests. It supports the review process of code and improves the overall quality of it, integrating it neatly into a user's workflow. Pull requests

allows for a team to preview changes in context with a single user's code against what is being proposed. This is known as Diff, and it makes comparison on any modifications made in a topic branch against the base branch and allows for possible merging. History of commits, comments, and references related to a pull request are available in a timeline-style interface. It will also highlight the recent modifications made.

The project management tools provided by Github supports user's productivity, allowing them to coordinate and align their ideas and stay on schedule. They are able to create what's known as an "issue", which is used to track ideas, enhancements, tasks or bugs. Milestones can be added to track progress on groups of issues or pull requests in a repository, and can be assigned to members of the group.

Github largely integrates Git, which is used to support software development work and to maintain and track the history of their work. It supports non-linear development due to rapid branching and merging, and provides specific tools for visualising and navigating. It gives the user a local copy of the full development history and the modifications that are made can also be imported and merged. Git is also very fast and scalable, with performance tests carried out by Mozilla, is showed. That it was an order of magnitude quicker than some version control systems where fetching version history from a locally stored repository was much more faster than fetching it from a remote server.

## JIRA

Jira is a closed source issue tracker developed by Atlassian that supports bug tracking and agile project management. Companies such as Fedora Commons, Hibernate, Twitter, Skype Technologies and NASA incorporates this user of this tracker. It is written primarily in Java and integrates the use of source control programs such as Clearcase, Concurrent Versions System and Git. It was initially used as a pure issue tracking software but is now adopted as a project management tool.

It is built for a software team to plan and track their software development progress. Members of the team can create stories and issues, plan sprints and distribute tasks across the software team. They are able to prioritise certain task work and create discussion on any activity with full context and complete visibility. Overall, this improves team performance, with the real time data provided that can be put into visualisation.

Jira in terms of structure, is consist of workflows, issues types, custom fields, screens, field configuration, notifications and permissions. The Jira issue is the main component that tracks bugs or issue that underlies with the project. It is classified into the following types; new feature, sub-task, technical task, bug, epic, improvement, story or task. Other components includes features, teams, modules and subprojects which can be used generate statistical reports and display it on dashboards. When an issue is created, it is divided into different fields which are known as screens. Screens can be edited and transitioned during the workflow.

Jira Agile is an approach used by development teams who follows a roadmap of planned features for upcoming versions of their product. There are two modes associated with this approach; Plan Mode and Work Mode. Plan mode displays all the user stories that are created for the project while in work mode, active sprint information is displayed.

To track progress a Burndown Chart is used to display the actual and estimated amount of work done in the sprint. A typical burndown chart would consist of two lines where one would indicate the actual task remaining and the other would indicate the ideal task remaining during the scrum cycle. Aside

from the chart, there are other representations available, such as a Sprint Report, Epic Report, Version Report, Velocity Chart, Control chart and a Cumulative Flow Diagram.

## BUGZILLA

Bugzilla is one of Jira's main competitors where it is also a web-based general bug tracker and testing tool. However, there is potential that could turn Bugzilla into a technical support ticket system, task management tool or a project management tool, but it is primarily used to track software defects.

The features of Bugzilla includes an optimised database structure for increased performance and scalability, security for protected confidentiality, an advanced querying tool, integrated email capabilities and comprehensive permissions system. These features would allow for improved communication between members of a software development team, increased product quality, and it ensures accountability and increased productivity.

It is comprised of a system that send the user results of a particular query on a schedule that the user specifies. The access to this system can be limited by the user's preferences, and they are able to send the reports to others as well as themselves. The reporting system allows for visualisation of the bug database by viewing the table as line graphs, bar graphs, or in a spreadsheet format. Users can also monitor the amount of time spent for them to fix a bug against a deadline where the fix must be completed by.

## CODE CHURN

Code Churn is a measure that indicates the rate in which a code evolves. It supports several uses; visualising the development process, reasons about delivery risks and tracks trends by task. Code churns provides diagrams for representing the code delivered. It can be used to predict post-release defects, as increased code churn would depict the program becoming more volatile as it gets closer to the deadline. The user is able to inspect the size and impact of the tasks being performed and using this information to decide whether the tasks are of an appropriate level.

Commit Activity charts can be used to define potential productivity issues like an increased in authors without a corresponding increase in commits and churn. This would indicate that there is more software developers working on the project than the actual software architecture can support.

The active contributors trends would depict the number of authors in the codebase over time. The information is calculated through observing the first and last recorded contribution times for each author. Correlating the number of active contributors to these churn metrics would allow the user to evaluate the effects of a project when it is scaled up or down.

CodeScene allows for the investigation of the impact made on the codebase by the project management tasks in terms of both code churn and collaboration. This aids in the partition of large tasks into smaller individual tasks as large tasks are hard to reason about and less predictable to plan. Smaller tasks are usually well-focused which is why they are preferable over larger ones. Individual commits can be grouped into a task, and a lead time can be calculated for each task. This is time that is passed between the first and last commit referencing a specific task and it can be used to track tasks that drift in time and to analyse the effects of the process changing in an organisation.

## AIM 3: TO CONSIDER THE ETHICAL CONCERNS SURROUNDING THIS KIND OF ANALYTICS.

There are many ethical concerns which must be considered when measuring software engineering process in relation to privacy, accuracy, property, accessibility and the effects on quality of life. Unauthorised access to information that has been gathered about a specific user's progress in software development would lead to privacy concerns as the data attained can be used for unknown purposes. The tools provided for this type of analytics are described to be focussed on the individuals, interactions, working software and customer collaboration. However, it is observed to be more focussed on software process and documentation. It is discovered that the data gathered does not encompass all the work accomplished every day, rather it was provided by the informants in a selective manner.

Software development analytics may also place more pressure onto developers to meet certain criteria within a time limit. This could increase the level of stress of the development team which results in the task being more focussed on meeting requirements rather than solving the problem at hand leading to reduced code quality. The selection of metrics used to perform software development analytics can also result in unethical effects. If poorly chosen, the contribution of each member of a development team may not be accurately assessed and efforts of the members could be misjudged, leading to unfairly assigned promotions or demotions.

Another recurring ethical question on software development analytics is who handles the data that is collected and who has access to the data? For apparent privacy reasons, there should be limited access to the statistical results of the data gathering, but that is not to say that this data would not be shared or transferred elsewhere once it has been accessed. Depending on how detailed the data collected is, it possesses the potential to define a general idea of any user's lifestyle, therefore, if this type of data leaks, it could lead to severe infringement of privacy.

Collected data by marketing companies can be utilised to support target advertisement. This is done through predicting the user's purchase preferences and it's possible for this resource to be sold to other companies to aid them with their sales.

## SOURCES


https://stevemcconnell.com/articles/gauging-software-readiness-with-defect-tracking/
https://www.researchgate.net/publication/4016775_Beyond_the_Personal_Software_Process_Metrics_collection_and_analysis_for_the_differently_disciplined
http://ecomputernotes.com/software-engineering/formal-methods-model
https://en.wikipedia.org/wiki/Static_program_analysis
https://www.tutorialspoint.com/compiler_design/compiler_design_lexical_analysis.htm
http://web.cs.iastate.edu/~weile/cs513x/2018spring/taintanalysis.pdf
https://en.wikipedia.org/wiki/Code_coverage
https://www.ibm.com/support/knowledgecenter/en/SSSHUF_8.0.0/com.ibm.rational.testrt.doc/topics/cinstruovw.html
https://confluence.atlassian.com/clover/about-code-coverage-71599496.html
https://www.ibm.com/support/knowledgecenter/en/SSSHUF_8.0.0/com.ibm.rational.testrt.doc/topics/cinstruovw.html
http://www.synthworks.com/blog/2013/04/20/why-you-need-functional-coverage/
https://www.zyxware.com/articles/4161/what-is-statement-coverage-in-testing
https://www.sealights.io/test-metrics/
https://www.techwalla.com/articles/what-is-cohesion-in-software-engineering
https://en.wikipedia.org/wiki/Cohesion_(computer_science)

http://pages.cpsc.ucalgary.ca/~eberly/Courses/CPSC333/Lectures/Design/cohesion.html
https://en.wikipedia.org/wiki/Cyclomatic_complexity
https://en.wikipedia.org/wiki/Coupling_(computer_programming)
http://www.informit.com/articles/article.aspx?p=1561879&seqNum=3
http://www.drdobbs.com/embedded-systems/measuring-execution-time-and-real-time-p/193502123
https://hackystat.github.io
https://timeular.com/?v=d2cb7bbc0d23
https://en.wikipedia.org/wiki/Jira_(software)
https://www.atlassian.com/software/jira
https://www.guru99.com/jira-tutorial-a-complete-guide-for-beginners.html
https://en.wikipedia.org/wiki/Bugzilla
https://www.bugzilla.org/features/
https://codescene.io/docs/guides/technical/code-churn.html