kiuwan

SOFTWARE ANALYTICS REPORT

Application: Carloan
Analysis label: no label
Analysis date: 2016/02/25 14:53
Report date: 2016/02/25 14:54

TABLE OF CONTENTS

kiuwan

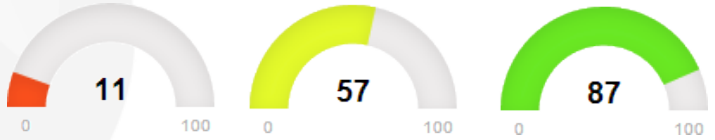
SOFTWARE EVALUATION METHODOLOGY

checKing Quality Model for Software (CQM) is Kiuwan methodology for evaluating the internal quality of a software product. **CQM is ISO-9126 based.** It defines us the internal quality scope and characteristics. CQM simplifies ISO 9126 focusing on internal quality. CQM proposes indicators for the following software characteristics:

- **Security.** The capability of the software product to protect information and data so that unauthorised persons or systems cannot read or modify them and authorised persons or systems are not denied access to them.
- **Reliability.** The capability of the software product to maintain a specified level of performance.
- **Efficiency.** The capability of the software product to provide appropriate performance relative to the amount of resources used under stated conditions.
- **Maintainability.** The capability of the software product to be modified. Modifications may include corrections, improvements or adaptability of the software to changes in environment and in requirements and functional specifications.
- **Portability.** The capability of the software product to be transferred from one environment to another.

CQM indicators are normalised to represent these regions:

- **0-30 region.** The characteristic pointed to by the indicator is in the RED zone. Improvements are needed.
- **30-70 region.** Represented by YELLOW and means that you have to keep your mind on this indicator. Your next moves will depend on your requirements.
- **70-100 region.** The GREEN zone. This is the zone where all indicators must be. No critical defects founded.



This normalisation allows the comparison of different characteristics between them; this means that you can say if the software is more maintainable than it is efficient or reliable. You are going to compare different version of the same application over time because the meaning of the indicator does not change. You can even compare two different technology applications.

APPLICATION & ANALYSIS INFORMATION

Kiuwan've analyzed for you a set of source files that is called application. In order to put in context this analysis and your results, kiuwan is going to give you some statistics:

Analysis Info

label	no label
date	2016/02/25 02:53
ordered by	spronghi spronghi
encoding	UTF-8
analyzed path	src.zip
languages found	java
analyzed files	109
unparsed files	0
unrecognized files	-

Application Info

name	Carloan
Bussiness value	critical
Times analyzed	7

Model Info

model name	CQM
model version	v1.1.15
engine version	-
active rules	2,430
active metrics	115

Note:

- unparsed files are the source code files of your application that kiuwan engine couldn't read during the analysis process.
- unrecognized files are the ones that were in your analyzed path but they aren't source code or they contains code in languages that Kiuwan doesn't support yet.

RISK INDEX

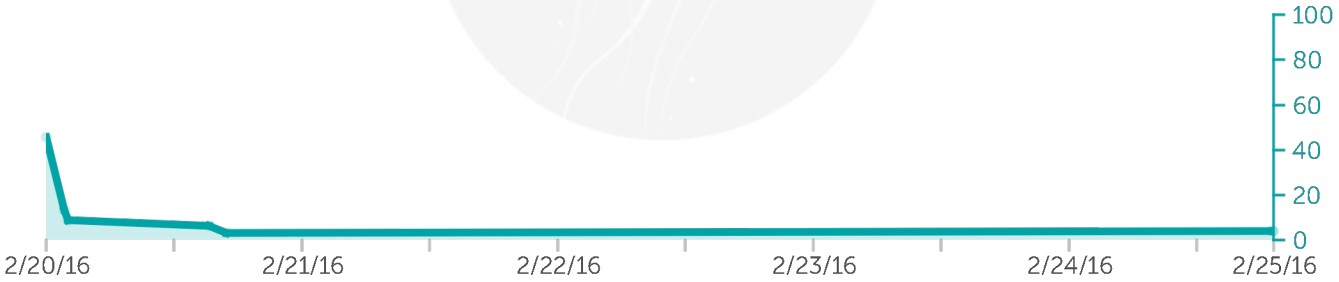
Risk index represents the potential problems that you are assuming for not paying attention to the quality of your source code. So far as you are (measured in effort) to get an acceptable quality level.

It is a number that concentrates all the evidence found in the source code of your application. It has been used your quality indicator and the effort that you need to spend to reach the quality level set as goal for you. So if you have poor quality, but if the effort needed to get better is low you are not assuming a high risk in this application because you are going to repair your problems easily. But if your effort needed to get better is very high your risk index will be high too.

Pay attention to risk index evolution in time.

Risk index

3.89



QUALITY

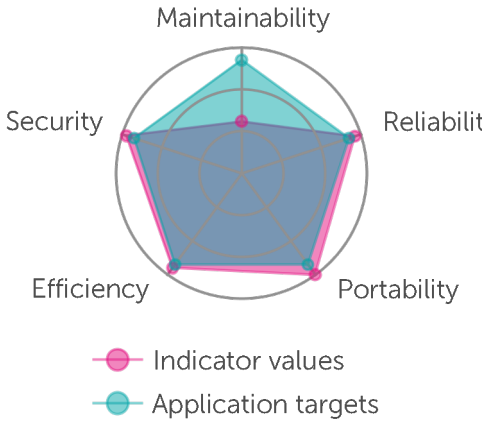
You get your global indicator at the right and a breakdown in software characteristics.

Pay attention to the **target values** that have been set when you configure your application. It is important that you know if your quality is better or not that these values and you modify them according to your requirements.

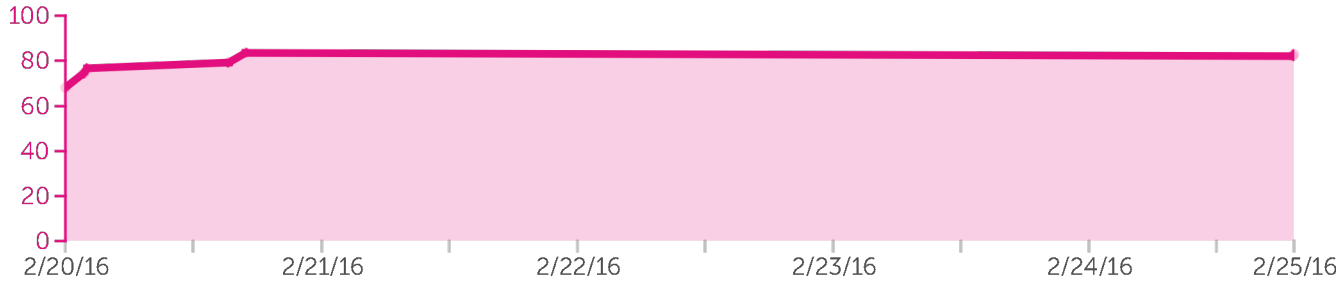
The quality evolution is another important point. **Are you improving?**

Global indicator

83.02



Characteristic	Quality	App. target
Maintainability	41	90
Reliability	95	90
Portability	100	90
Efficiency	93	90
Security	96	90



REPARATION EFFORTS

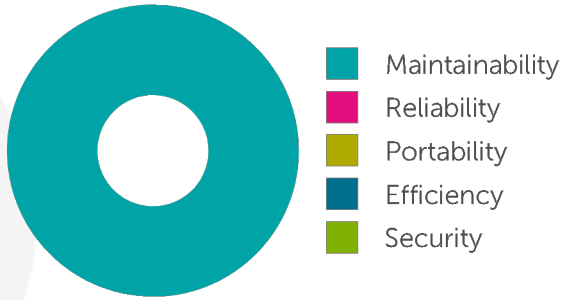
Now you know your quality level, you'll want to know how much it will cost to reach your goal.

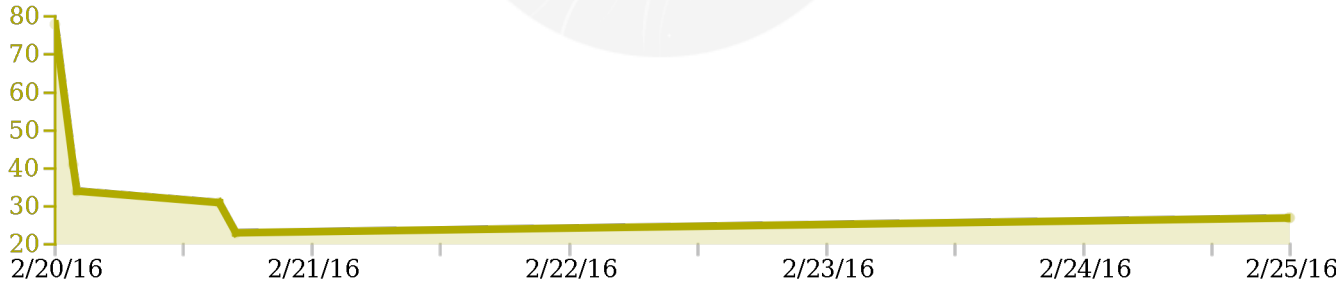
We've calculated for you the minimum set of defects will be corrected to achieve it. Here's what you need to invest. You can configure (for accuracy) the effort needed to correct each defect type.

Effort to target

27 hours

Characteristic	Effort to target
Maintainability	27 h
Reliability	0 h
Portability	0 h
Efficiency	0 h
Security	0 h





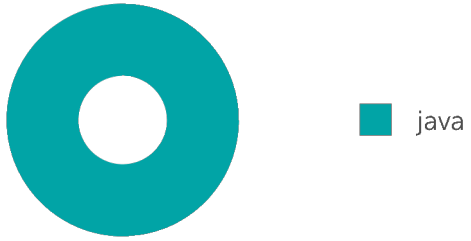
MAIN METRIC VALUES

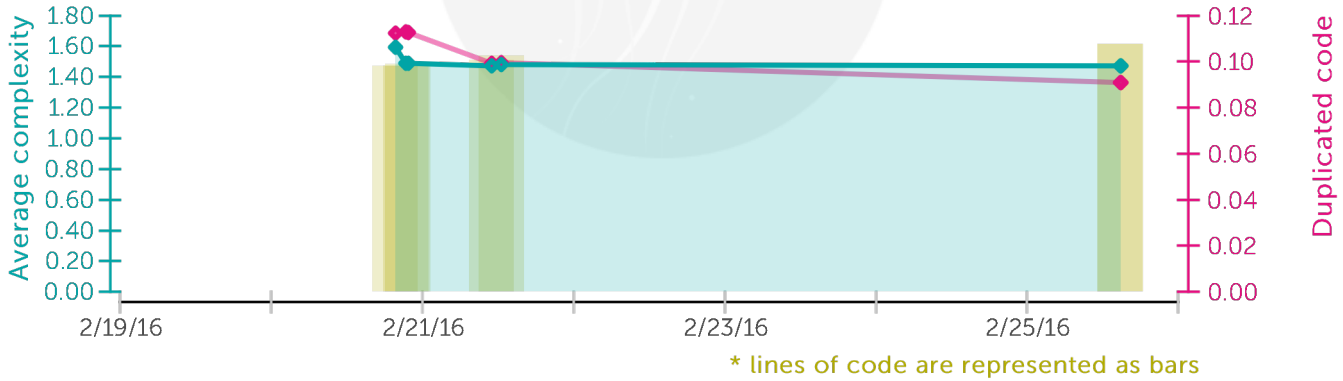
We've computed some metrics of your source code. Below are shown the most important ones.

- **Lines of code.** Excluding commented lines and blank lines.
- **Function points.** Functional size calculated by backfiring strategy.
- **Avg complexity.** Average of each function (or method).
- **Dup code.** The ratio of the duplicated code.

Lines of code	5,737
Function points	108
Average complexity	1.47
Duplicated code	9.09%

Lines of code by technology

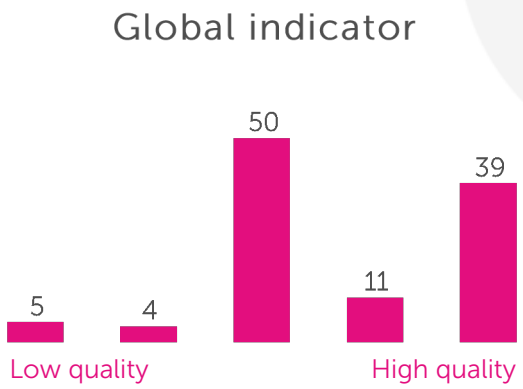




QUALITY DISTRIBUTION IN FILES

It's important to know if your quality is distributed homogeneously through files. In the chart we've used **quintiles** dividing the file quality value in five equal sized subsets, and then we tell you how many files fall in each subset.

In the table, you have a list of the files (the worst ones) that have reduce their quality indicator from previous analysis. If this is your first analysis, you will get the files with the lower quality values.



File name	Indicator	Previous	Delta
.../Customer.java	44.79	-	-
.../OperatorChecker.java	52.64	-	-
.../PasswordChecker.java	53.11	-	-
.../ContractType.java	53.86	-	-
.../PaymentType.java	53.86	-	-
.../SqlConnector.java	57.46	-	-
.../Location.java	58.23	-	-
.../Operator.java	63.71	-	-
.../PriceCategory.java	64.47	-	-
.../CustomerChecker.java	67.51	-	-
.../Agency.java	68.25	-	-
.../LoginService.java	69.1	-	-
.../PaymentChecker.java	71.08	-	-
.../UtilServiceFactory.java	71.9	-	-
.../PermissionControl.java	72.26	-	-
.../TableControl.java	72.26	-	-
.../ContractChecker.java	73.7	-	-
.../CarChecker.java	73.99	-	-
.../ServiceControl.java	74.1	-	-
.../Payment.java	74.65	-	-
.../Status.java	75.5	-	-
.../Contract.java	76.01	-	-
.../Car.java	76.24	-	-
.../ModelServiceFactory.java	76.46	-	-
.../FXLoader.java	76.49	-	-
.../Currency.java	76.57	-	-
.../ModelMenuButton.java	76.67	-	-

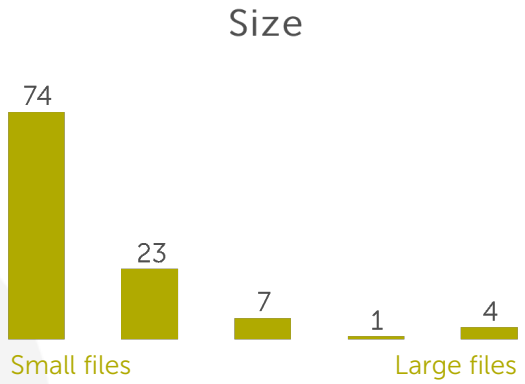
kiuwan

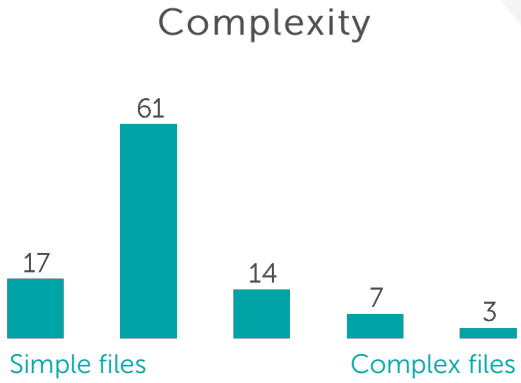
METRIC DISTRIBUTION IN FILES

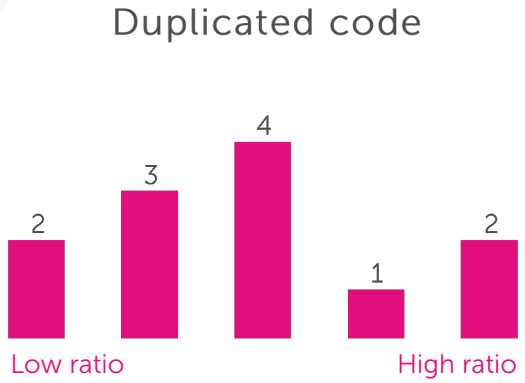
It's important to know if your metrics are distributed homogeneously through files. We've used **quintiles** dividing the file level metric values in five equal sized subsets, and then we tell you how many files fall in each subset.

We give you the most three popular metrics:

- **size** as lines of code of each file.
- **complexity** as average cyclomatic complexity of each function per file.
- **dup code** as the ratio of the duplicated code of each file.







TOP 10 REPAIR FIRST DEFECTS

Here you have a list of top 10 defect types that you have in your application. These defects are the ones that once eliminated give more benefit by unit time of effort. If you want to start fixing quality issues, you must start with this to maximize your time.

Rank	Defects	Files	Rule name	Lang.	Characteristic	Priority	Effort
1	1	1	Avoid parameter method names that provoke conflicts with class members names.	java	Maintainability	2	3m
2	7	2	Avoid unused private methods and constructors.	java	Maintainability	2	21m
3	2	1	Avoid queries in the database except from the specific classes.	java	Security	1	1h
4	1	1	Use instanceof within an equals() method implementation.	java	Efficiency	2	3m
5	5	5	Provide Javadoc comments for public fields.	java	Maintainability	2	30m
6	1	1	Avoid comparing floating point types.	java	Reliability	2	6m
7	8	8	Use getClass() in the equals() method implementation.	java	Efficiency	2	48m
8	11	11	Avoid usage of * in import statements.	java	Maintainability	3	33m
9	3	1	Avoid if/else-if chains performing type testing.	java	Maintainability	2	1h 30
10	3	3	Avoid nested IF sentences with too many levels.	java	Maintainability	2	1h 30

If you want a complete list of defects to repair in order to reach a quality target or to spend a bag of budgeted hours you can take an **action plan** report from kiuwan.com in **what if** function.

kiuwan

ABOUT OPTIMYTH

Optimyth is an independent software company specialized in solutions to support large companies with their rationalization, quality and productivity management initiatives whilst improving the overall maintainability of the software estate. Our solutions raise the quality and increase the performance of business systems. They improve the efficiency, cost control and productivity within an organization, thus achieving maximum IT value.

These solutions are based on our state of the art checKing products: an Integrated Application Quality Management Portal which aggregates information from all the stages of a development life cycle into one "single pane of glass". And, an Application Inventory Management system that allows companies to untangle the complexity of their application through automatic discovery of software components, and the relationships between them at different levels, enabling rationalization initiatives and impact analysis capabilities to reduce maintenance costs.

www.kiuwan.com

@KiuwanJelly

kiuwan fan page