57 Quality Checks in COBOL-Programs with ITP-PANORAMA

Master your software with ITP-PANORAMA!

Why is testing of code quality so important?

Testing the quality of COBOL programs is critical for meeting requirements of BASEL Commission regarding the prevention of "Operational Risk." At the same time, it is an important step to improving the software and ensuring accurate support of business processes in the day-to-day IT operations.

All details of the scanned software are available for management, architects and developers in the repository of ITP-PANORAMA. If boundaries are violated it will be shown in the error list. A mouse-click on the error message will show the relevant code line in the program listing. The metrics themselves can later be used for statistical analysis. A history of all metrics is stored and allows trend discovery.

Tests and Rules to Judge Code Quality

The goal is to make a program understandable and easier to maintain. Only then is a program transparent and clear. All tests/rules are based on the knowledge that not all possibilities that a program system offers, should be used. The user can define the set of rules he/she wants to be checked and documented. However, ITP-PANORAMA will always produce and store all rules. The user will have them available whenever wanted.

Tests for COBOL (Language Features)

- **PROGRAM-ID – Prname[ALP1]?**
  Prname should correspond with File name of Source File.
- **FILE Section – FILE-STATUS?**
  A FILE STATUS Variable should exist for each FD statement. It can be accessed later for error checks.
- **Linkage Section – 01 Stage?**
  Symbols in the Linkage Section have no memory of their own. They are set usually by the developer or set during run-time execution. If such a field is not initialized but accessed during run-time a system crash will happen. Panorama will check whether level 01 initializing has been programmed.
- **Program Entry USING List?**
  A long list of handover-parameters makes a program difficult to understand. Has a defined number (n) of parameters exceeded?
- **Code outside of Section/Paragraph?**
  Code outside a Section / Paragraph is unsightly and makes a program difficult to understand.

- **First Section/Paragraph too long?**
  First Section/Paragraph contains Code for the main control of a program. It should not exceed a defined (n) number of lines of code.
- **Section/Paragraph too long?**
  Is a defined number of lines of code (n) exceeded?
- **Conditional Expressions too complex?**
  Conditions in IF, PERFORM and WHEN statements that contain too many parameters are difficult to read. This has to be avoided. Is a defined number (n) exceeded?
- **Conditional Expressions with different Data Types?**
  Comparing data fields with different data types has negative impact on the performance, loss of precision and unwanted conversions. This has to be avoided. Are these rules broken?
- **Arithmetic Operations with different Data Types?**
  If different data types are used in a compute it can have negative impact on the performance, loss of precision and unwanted conversions. This has to be avoided. Are these rules broken?
- **MOVE with different Data Types?**
  If different data types are used in a MOVE it can have negative impact on the performance, loss of precision and unwanted conversions. This has to be avoided. Are these rules broken?
- **MOVE fields of different length?**
  Moving a field into a field that is shorter in length causes errors. This has to be avoided.
- **Too deep nesting in a Program?**
  Too many branches in a code-block make a program difficult to understand. Is a defined number (n) of nestings exceeded?
- **Code Blocks too long?**
  Control-flow statement that build code blocks (IF-THEN-ELSE, EVALUATE-WHEN, PERFORM-END-PERFORM) should not exceed a certain length. Do they exceed the defined (n) number?
- **Missing End-characters ”.“ in the Code?**
  Some Statements (CALL, COPY) need a **”.“** at the end. Are there missing ones?
- **ALTER Statement?**
  Is ALTER being used and did they produce self-modified code?

- **Pointer Datatype and SET ADDRESS OF?**
  Is Pointer Datatype / SET ADDRESS OF used to change variable names and content?
- **GOTO?**
  A long list of targets in a GOTO DEPENDING ON statement can make a program difficult to read. It is also possible that the maximal number of target has been exceeded. Has a defined number of targets been exceeded?
- **INDEX Variable?**
  If an INDEX is used to access a data field without and end check, it is possible that it is reaching over the end. Did the developer program an end check?
- **PERFORM-THRU?**
  Has a PERFORM-THRU Statement been used to influence a program-flow?
- **PERFORM Z THRU A?**
  Was a PERFORM-THRU as end of a Code Position used that is positioned before the beginning?
- **CALL with Variables?**
  Has a variable been used in a CALL Statement that is set during the run of a program? Dynamic Calls are making it difficult to follow the program flow, unless one is using the Dataflow Analysis of ITP-PANORAMA.
- **Addressing with Constant Values e.g. B.: A (17:3)?**
  Access to data in fields with a fix index should be avoided. It is better to define a specific field/group.
- **Addressing with Constant Values, outside a certain area?**
  Was the length of a field exceeded as it was accessed by a fix index?
- **EXEC SQL ”FROM“**
  Was a defined number (n) of FROM Tables exceeded?
- **EXEC SQL ”WHERE“**
  Was a defined amount (n) of conditions in a SQL WHERE Statement exceeded? Was the WHERE Statement (n) too deeply nested?
- **COPY REPLACING?**
  Was REPLACING used? Was a defined number of REPLACINGs (n) exceeded? REPLACINGs are difficult to understand and should be avoided.
- **COPYBOOK in a COPYBOOK?**
  Was a COPY statement used in copies? Was a defined number (n) exceeded? This kind of programming makes a program difficult to understand.

- **Procedural Statements in COPYBOOK?**
  Was program code found in copies? This kind of programming makes a program difficult to understand.
- **DECL with unusual levels?**
  01 levels should not be defined in a COPYBOOK. Usage of uncommon levels (3, 17, 22…) are an indication of improper programming.
- **TAB (ASCII 9) in source code?**
  Usage of TAB should be avoided, because it influences the display of the code in the Text Editor.
- **Have opened files be closed?**
  A file opened with an OPEN statement has to be closed by a CLOSE statement.
- **Are Files that have been defined used?**
  If a FD has been made, an OPEN/READ/WRITE/CLOSE statement should be executed.
- **SQL LIKE?**
  In a SQL LIKE instruction a search should not start with a wildcard. A Wildcard at the beginning makes it necessary to ignore all Index Information of a column and always search through the entire table. This will have a negative influence on performance.
- **SQL ORDER BY?**
  In a SQL ORDER BY instruction a column name and not its number should be used to improve the readability of a program.
- **SQL WHERE Arithmetic?**
  In a SQL WHERE instruction no calculation should be executed. This will have a negative influence on the performance.
- **SQL WHERE Length?**
  A SQL WHERE instruction should not exceed a defined number of statements (n).
- **SQL JOIN Quantities?**
  A SQL JOIN instruction should not link more than a defined number (n) of tables.
- **WITH DEBUGGING MODE?**
  ”D“ Statements should not be executed in code that is used in production. It could lead to unwanted behavior if an instruction WITH DEBUGGING MODE is used.
- **OCCURES 1?**
  If a variable is not an Array, it should not be defined with OCCURES 1.

- **COPYBOOKS should not control a Structure?**
  The listed Statements should not be used in a Copybook:

  - IDENTIFICATION DIVISION.
  - PROGRAM-ID.
  - AUTHOR.
  - INSTALLATION.
  - DATE-WRITTEN.
  - DATE-COMPILED.
  - ENVIRONMENT DIVISION.
  - CONFIGURATION SECTION.
  - SOURCE-COMPUTER.
  - OBJECT-COMPUTER.
  - SPECIAL-NAMES. DECIMAL-POINT IS COMMA.
  - I-O CONTROL.
  - FILE-CONTROL.
  - SELECT.
  - DATA DIVISION.
  - FILE SECTION.
  - WORKING-STORAGE SECTION.
  - SCREEN.
  - REPORT.
  - INPUT-OUTPUT SECTION.
  - LINKAGE SECTION.
  - PROCEDURE DIVISION.

- **SQL CREATE TABLE?**
  A *Create Table* creates a permanent table in a database. This is usually not allowed and permitted. One works better with temporary tables.
- **SQL EXISTS?**
  A SQL EXISTS statement makes it necessary that a Sub Query for each record will be reevaluated. It affects negatively the performance.
- **The listed COBOL Statements should be avoided to keep the code portable:**
  - EXHIBIT
  - NOTE
  - TRANSFORM
  - EXAMINE
  - ON
- **Opening and Closing of Files in a Loop?**
  Multiple opening and closing of a file has a negative effect on the performance.
- **Opening and Closing of a CURSOR in a Loop?**
  Repeated opening and closing of a CURSOR affects the performance as if the cursor is not closed.
- **Zyclometric Complexity?**
  The McCabe measure of a Section/Paragraph should not exceed a defined number (n).
- **EVALUATE WHEN IF?**
  There should be no IF condition in a WHEN Block.

- **CICS LINK/XCTL?**
  The COMMAREA length should be passed.
- **SQL SELECT *?**
  The columns of a select should be listed explicitly in order to increase the readability.
- **SQL DELETE/UPDATE?**
  A SQL DELETE or UPDATE instruction should always have a WHERE.
- **DISPLAY/ACCEPT?**
  Input or output on a console should be avoided.
- **SORT/MERGE?**
  Usage of SORT or MERGE statements can influence the performance negatively and should be avoided.
- **WHEN OTHER?**
  In an EVALUATE block there should always be a WHEN OTHER instruction to be prepared for unexpected contents.
- **EXIT PROGRAM?**
  After an EXIT PROGRAM instruction should be no further code in this Paragraph/Section.
- **SQL SELECT in SELECT?**
  A SQL SELECT statement within another SQL SELECT statement makes the code confusing. Has a defined number (n) of SELECT in SELECT been exceeded?
- **SQL LOCK?**
  A SQL LOCK statement blocks a table for other programs. This can lead to errors in the program system and should be avoided.
- **Block End?**
  Each control block in a program should be completed with the matching END Statement (END-IF, END-WRITE, END-READ ...)

Rules for Symbol Names

An important condition for the readability of a program is fulfilled when symbols have meaningful names. A symbol should explain themselves and may follow the internal rules. The symbol length plays a role here as well as the observance of prefix or Sufix rules. ITP-PANORAMA will examine symbol names via rules. Rule violations will be reported. Measures to the symbol names are available for statistical evaluation. An individual rule can be stored for each class of symbol.

List of Symbol classes (incompletely)

- Program-Name
- Source File Name (.CBL)
- Copy File Name (.CPY)
- File-Name (FD)
- File Assigned Name (FD)
- FILE-STATUS Name
- DATA RECORD Name
- Linkage Section 01 Level Name (mostly Parameter)s
- 01 Level Name
- Group-Name
- Field Name
- 77 Level Field Name
- 88 Level Field Name
- Table Name (OCCURS)
- Section Name
- Paragraph Name
- SQL Table Name
- SQL Column Name
- INDEX Variables

Metrics

Some quality characters can be read from metrics. ITP-PANORAMA offers the following metrics per **Program, Section und Paragraph:**

- Zyclometric Complexity - McCabe Counts
- Halstead Counts n, n1, n2,N, N1, N2 and derived from Halstead-Length (HL) and Halstead-Volume (HV)

Per Program Line of Code:

- Function Points – For each statement ITP-PANORAMA calculates a value that is then stored in the statement. For analyses, searches or selections the Function Point value can be displayed with two mouse-clicks. This is an easy way to judge the complexity of the selected amount of code.

Scanning of Comments

ITP-PANORAMA will scan the comments in the source code and store them in the Hyper-Cube Repository. This makes it possible to quickly access knowledge from the comments. In addition, users can define a list of "interesting" words and word patterns. The scanner will search through the comments for these words/patterns and store them in the Hyper-Cube Repository. Developers can analyze these comments and learn whether they follow the company's internal rules.

Summary

The examination of the listed criteria can be performed with each run of the scanner. After the clean-up of errors and poor references, violations of the rules can be resolved promptly. As a result, the quality of the software is enhanced and maintainability improved. The downtime caused by errors that were overlooked in the tests is minimized.

The introduction of the quality tests is limited to setting the nominal sizes in the rules. These are pre-defined by ITP and will be good enough for the start. Adjustment can be made any time according to experiences made. Analyzing the history will show the progress in improving the quality of the software.

"It seems that you better check the quality of your software with ITP-PANORAMA quickly!" [ALP2]