klocwork

Klocwork Architecture

Document Number: KW2023_4_006

Certified Klocwork version: Klocwork 2023.4

Author	Revision	Comments	Date
AWeekes	0.1	Initial draft created	May-09-13
AWeekes	0.2	Expanded content with reference to relevant Klocwork product names	May-27-13
AWeekes	0.3	Updated for 2015 re-certification	May-29-15
SBommaganti	1.0	Updated for Klocwork 2016	March-3-16
SBommaganti	1.1	Updated installation architecture to include CI	April-14-16
SBommaganti	1.2	Updated Klocwork version to 2016.1	June-25-16
SBommaganti	1.3	Updated Klocwork version to 2016.3	Nov-2-16
SBommaganti	2.0	Updated Klocwork version to 2017	March-2-17
SBommaganti	2.1	Updated Klocwork version to 2017.1 and added EN 50128 standard	July-5-17
SBommaganti	2.2	Updated version to 2017.2	Oct-16-17
SBommaganti	2.3	Updated version to 2017.3	Nov-2-17
MTofinetti	3.0	Updated version to Klocwork 2018	2018-04-16
MTofinetti	3.1	Updated version to Klocwork 2018.1	2018-06-13
MTofinetti	3.2	Updated version to Klocwork 2018.2	2018-09-27
MTofinetti	3.3	Updated version to Klocwork 2018.3	2018-11-28
MTofinetti	4.0	Updated version to Klocwork 2019	2019-03-22
MTofinetti	4.1	Updated version to Klocwork 2019.1	2019-05-14
MTofinetti	4.2	Updated version to Klocwork 2019.2	2019-07-25

MTofinetti	4.3	Updated version to Klocwork 2019.3; IEC 62304 added	2019-12-12
LRobertson	5.0	Updated version to Klocwork 2020.1	2020-03-12
ABedford	5.1	Updated version to Klocwork 2020.2	2020-06-29
ABedford	5.2	Updated version to Klocwork 2020.3	2020-09-14
ABedford	5.3	Updated version to Klocwork 2020.4	2021-02-24
ADunster	6.0	Updated version to Klocwork 2021.1	2021-04-26
ADunster	6.1	Updated version to Klocwork 2021.2	2021-08-17
ADunster	6.2	Updated version to Klocwork 2021.3	2021-11-30
ADunster	6.3	Updated version to Klocwork 2021.4	2022-01-18
JBritton	6.4	Updated version to Klocwork 2022.2 and rebrand	2022-05-26
JBritton	6.5	Updated version to Klocwork 2022.4	2022-12-17
JBritton	6.6	Updated version to Klocwork 2023.2	2023-07-26
JBritton	6.7	Updated version to Klocwork 2023.4	2023-12-29

Contents

Referenced Standards

Standards referenced in this document refer to the following versions:

Standard	Version
ISO 26262	ISO 26262:2018
IEC 61508	IEC 61508:2010
IEC 62304	IEC 62304:2006/AMD1:2015
EN 50128	EN 50128:2011/A2:2020

Trademarks

Related Documents

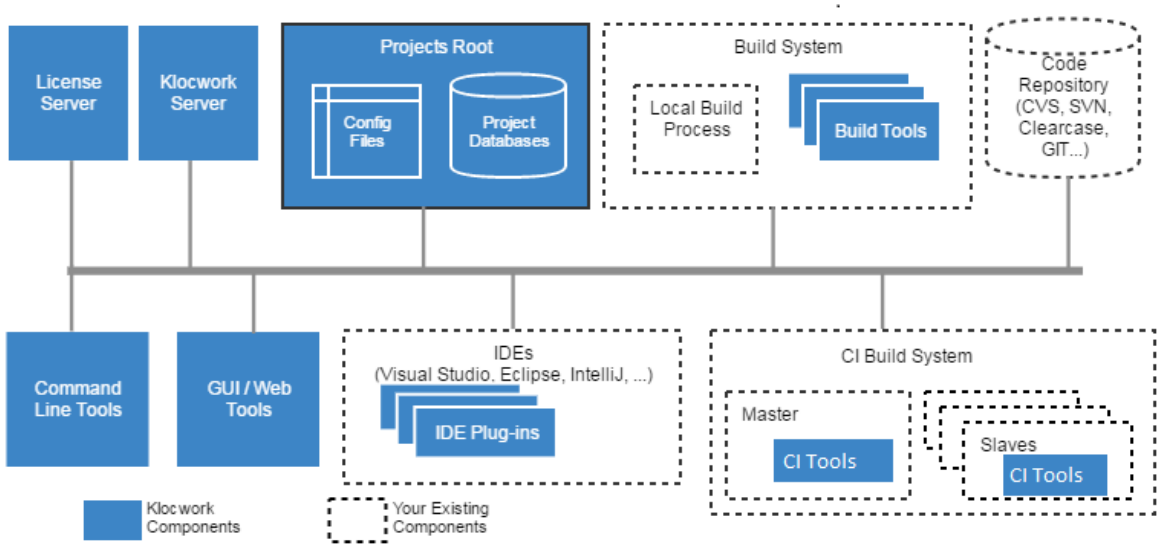
Document ID	Title
KW2023_4_001	Functional Safety Manual for Klocwork
KW2023_4_003	Klocwork ISO 26262 / IEC 61508 / IEC 62304 / EN 50128 Certified Checkers
KW2023_4_005	Klocwork Checker Qualification Pack
KW2023_4_009	Klocwork 2023.3 and 2023.4 Release Notes

Purpose

This document describes the installation and operational architecture of Klocwork.

Installation Architecture

A Klocwork installation includes both server-side components and client-side components. For the sake of understanding the installation architecture, we will focus on the server-side components.



Component	Details
License Server	A RLM license manager. You can also use your organization's RLM server. Default host: localhost Default port: 27000 plus 33133
Klocwork Server	An Apache Tomcat Web server. Provides access to Klocwork Review, Klocwork Inspect and Klocwork documentation. Klocwork clients connect to the Klocwork Server for project information. Note: Klocwork does not support using your own Apache Tomcat server. You must use the Web server packaged with Klocwork. Default host: localhost Default port: 8080 plus 8081 MariaDB database server. Stores defect data. Default host: localhost Default port: 3306

Component	Details
Klocwork projects_root directory	The data location for the Klocwork Servers and applications, including the project database tables, which are stored in MariaDB and Lucene databases. Note: The projects_root directory is not one of the components you see listed in the installation wizard, but the wizard lets you choose a location for it.
Klocwork build tools	Integration build analysis tools: • tools for managing Klocwork projects, analysis and access control (kwadmin and kwbuildproject) • tools to produce a build specification (kwinject, kwwrap, kwant, kwcsprojparser) • tools for running Klocwork integration build analysis (the analysis engines). It accurately identifies critical security and reliability issues through a sophisticated whole program analysis of C/C++, Java and C# code • tools for managing Klocwork projects and access control • sample projects
Klocwork Static Code Analysis	Drag and drop build reporting capabilities help development leads answer complex questions about the security, reliability and maintainability of the entire code base in minutes.
Client-side tools	A suite of client interfaces and IDE plug-ins is available to match most developer requirements. Client-side tools are installed separately from the server-side resources and are configured based on the individual developer's requirements. Client-side tools include: • Command-line tools • Klocwork Desktop • IDE plug-ins for Visual studio, Visual Studio Code, Eclipse, and IntelliJ IDEA • Klocwork Desktop Analysis - like spell-check for developers. Instant, accurate, and continuous feedback on the critical defects and security vulnerabilities introduced into your code, as you're writing it. Klocwork Refactoring simplifies the time-consuming task of code maintenance for C/C++. Automatically clean up your code and make it easier to understand within Visual Studio or Eclipse.
Continuous Integration tools	Klocwork's Continuous Integration (CI) capability enables your organization to identify and communicate errors faster, without waiting for nightly builds. Klocwork's CI tools integrate with third-party plug-ins and custom scripts for CI environments so that developers can verify the quality of source code as part of their pre-merge or post-commit processes.

Component	Details
	Klocwork CI build analysis tools include the following: • tools to produce a build specification (kwinject, kwant, kwmaven, kwwrap, kwcsprojparser, kwvcprojparser, kwmaven) • command-line analysis tools (kwciagent) • tools for running Klocwork CI build analysis (the analysis engines)

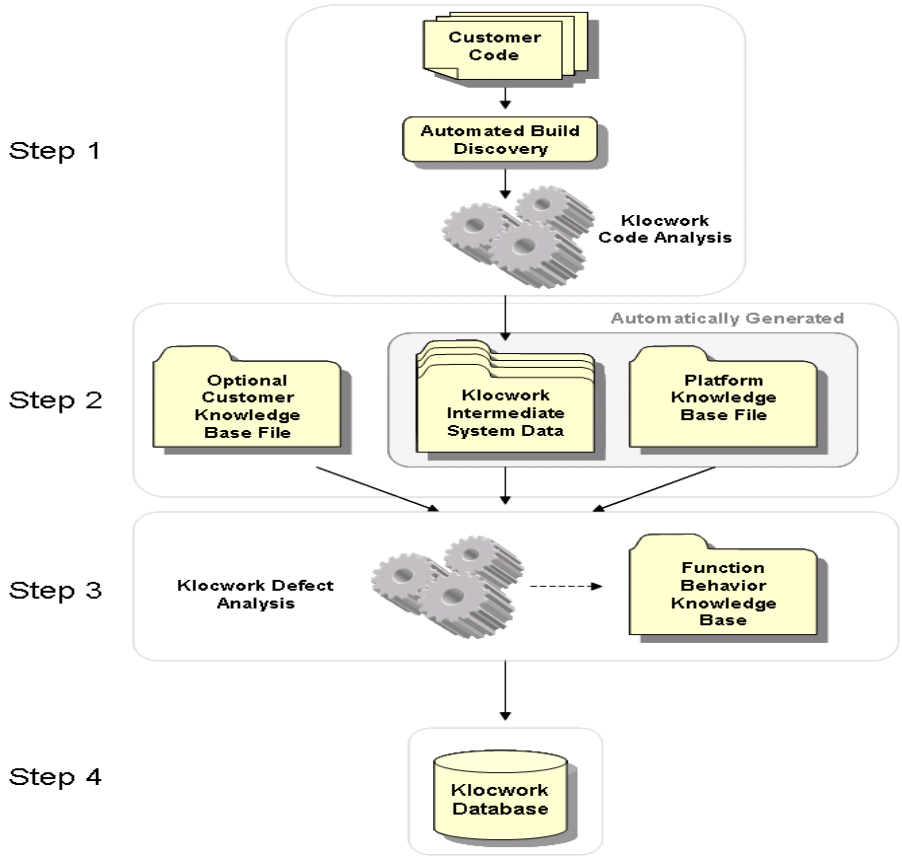
Operational Architecture

The operational architecture supports a series of processes that start with the customer's source code, and ultimately generate a collection of discovered code defects.

While we focus on checkers and their capabilities, what Klocwork provides is a piece of core technology based on a well-published research called symbolic execution. Klocwork includes an engine and a set of solvers that in abstract seek to predict what your code will do under every possible execution scenario, without ever actually executing your code. It is a simulation. Then, each checker uses that engine to find situations that that checker cares about.

- So, if a checker is looking for memory leaks, it is going to be looking for code paths that the simulation engine generates, where memory is allocated and never released.
- If it is looking for dangling pointers, it is looking for a situation where you assign a pointer to some heap location, you use it, you think you've gotten rid of it, but you haven't. It's still perfectly valid.
- If we're looking for NULL pointer dereferences, you have a pointer which could be assigned a NULL value, and you dereference it at some point causing a crash.
- If you are applying MISRA standards for coding style / coding standards, such as where you have assigned a 16-bit unsigned integer, and then try to use it unqualified in a context that would normally require a 32-bit signed integer.

This common technology underlies everything that Klocwork does. Each of the checkers exercises that common technology in a way that is specific to what those checkers want to discover.



Step 1 – Build and Code Analysis

Customer Code

Files, libraries and other components that are part of the full or partial system build and which are normally used to generate object code and executables. Supported languages include C, C++, C# and Java.

Automated Build Discovery

Information regarding how a customer builds their full software system is critical for accurate, comprehensive defect and vulnerability detection:

- Compile options
- include directories (-I)
- defines (-D)

Automated tools gather the required information by observing a customer's build in its native environment. Automated tools can also extract build information from Microsoft Visual Studio project files.

Klocwork Code Analysis

Code is parsed and compiled correctly using the Klocwork analysis engine and generates Klocwork Intermediate System Data.

Step 2 – Code and System Knowledge Aggregation

Upon completion of the front-end processing provided by the Klocwork compiler, various intermediate assets are persisted to, or read from, disk in order to make inter-procedural data flow analysis possible:

Object files

Much like a normal compiler, the Klocwork compiler emits a "binary" form of the input to an object file. In this way, the Klocwork compiler works most like a multi-translation unit optimizing tool chain, in that the content of the object file is an intermediate representation of the code, and not final executable code. We call this format Mid-level Intermediate Format, or MIR for short, and it consists of a first-order logic serialization of the Abstract Syntax Tree and associated semantics information that are produced within the compilation process.

Platform knowledge base(s) (KB)

Source code is always written with a platform, or set of platforms, in mind as the target host environment. This could be a typical Posix-style environment, or a more specific real time environment, or something consumer facing such as Windows. Each of these platforms exposes many different API calls that programmers typically make use of, and to correctly analyze the data flow of that program code, Klocwork provides a variety of platform knowledge bases that encode the functional behavior of such system APIs.

In addition to the provided platform knowledge base(s), the customer may choose to enhance or replace the analysis of functions within either third-party, platform, or their own code by providing a knowledge base of their own. Just like a platform KB, this is responsible for defining the functional behavior of one or more APIs in a way that the Klocwork data flow engine can then perform analysis of functions that call these APIs, without having to see their source code.

Step 3 – Inter-Procedural Defect Analysis

Klocwork Defect Analysis

Sophisticated logical analysis follows the possible values of variables and different execution paths through the system. Multiple techniques provide the right balance between scalability and a robust, accurate list of defects.

In general, Klocwork makes use of a fundamental technology called symbolic execution. This allows the engine to reason about the relationship between different symbols, or variables, within the data flow of any function in the program. By doing so, we can perform deep analysis that leverages the symbolic solvers to perform such things as:

- Path elimination; removing, or reducing, possible paths through the function based on data flows that can be reasoned to be impossible based on the logical combination of axioms gathered,
- Alias analysis; following inter-relationships of symbols through multiple levels of assignment and re-assignment, so that even though the original symbol set may

no longer be relevant, a replacement symbol set can be used to continue analysis.

Function Behavior Knowledge Base (FBKB)

Automatically generated file of all the function behavior within the software system that is used as part of the Defect Analysis to ensure accurate analysis. Also used at the developer desktop to ensure accurate analysis results when the developer is only analyzing a portion of the system

Step 4 – Database Generation

Defects, security vulnerabilities, architectural models, and metrics are stored into a scalable, searchable database. Information from each analysis is stored into the database, allowing build over build analysis and comparison. Defects and vulnerabilities are automatically tracked across builds, so that the same defects are automatically tracked, even if line numbers change.