AR: Assessment Report

Apr 28 2016

JAVAWEBAPP

Version: R 5.4.S_BB

PURPOSE OF DOCUMENT:

Purpose of this report is to provide an objective assessment of the quality of the application. The primary audience for this report is IT executives who are responsible for JAVAWEBAPP application.

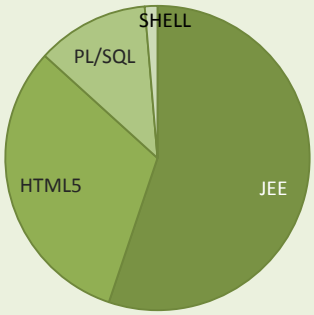
Table of Contents

1. Executive Summary

The Application Assessment evaluates the overall quality of the JAVAWEBAPP application.

JAVAWEBAPP is an extra large application and has a ***Total Quality Indicator (TQI) of 2.43 on a scale of 4.*** Each of the additional health metrics and their scores are identified below.

1.1. Application Characteristics



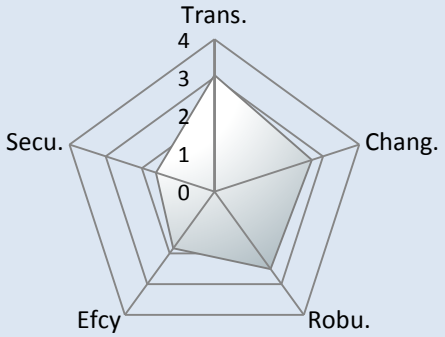
TOP 5 TECHNOLOGIES

Name	LoC
JEE	777,422
HTML5	444,448
PL/SQL	168,102
SHELL	18,928

TECHNICAL SIZE

Name	Value
kLoC	1,409
Files	8,662
Classes	8,622
SQL Art.	7,307
Tables	599

1.2. Summary of Quality Indicators



	TQI	Robu.	Efcy	Secu.	Trans.	Chang.
R 5.4.S_BB	2.43	2.51	1.84	1.61	3.06	2.69

1.3. Assessment Highlights

STATISTICS ON VIOLATIONS

Name	Value
Critical Violations	8,623
per File	0.99
per kLoC	6.12
Complex Objects	2,066
With Violations	2,066

Rule Name	# Violations
Avoid using Fields (non static final) from other Classes	5,338
Suspicious similar method names or signatures in an inheritance tree	1,780
The exception Exception should never been thrown. Always Subclass Exception and throw the subclassed Classes.	553
Avoid direct or indirect remote calls inside a loop	424
Avoid instantiations inside loops	93
Close database resources ASAP	68
Close the outermost stream ASAP	68
Avoid empty catch blocks	47

1.4. Analysis

JAVAWEBAPP contains an average amount of risk. The key risk areas pertain to the application's Security (specifically, architectural security), and Efficiency (how the application uses its resources). These risks may expose end users to data theft and sluggish responses to user input. While there are numerous instances of where these risks as injected into the application, the types of "violations" concentrate on a select few.

JAVAWEBAPP's strengths lie in its logical architecture making it easy to apprehend and to change. Therefore, the application has very low risk when it comes to Changeability and Transferability. The source code is well documented, written in a way that reduced complexity. Outside of some architectural non conformity issues, this application should be very easy to maintain and enhance.

Please see the remainder of the report regarding areas of focus to further improve the application.

1.5. Architecture Diagrams

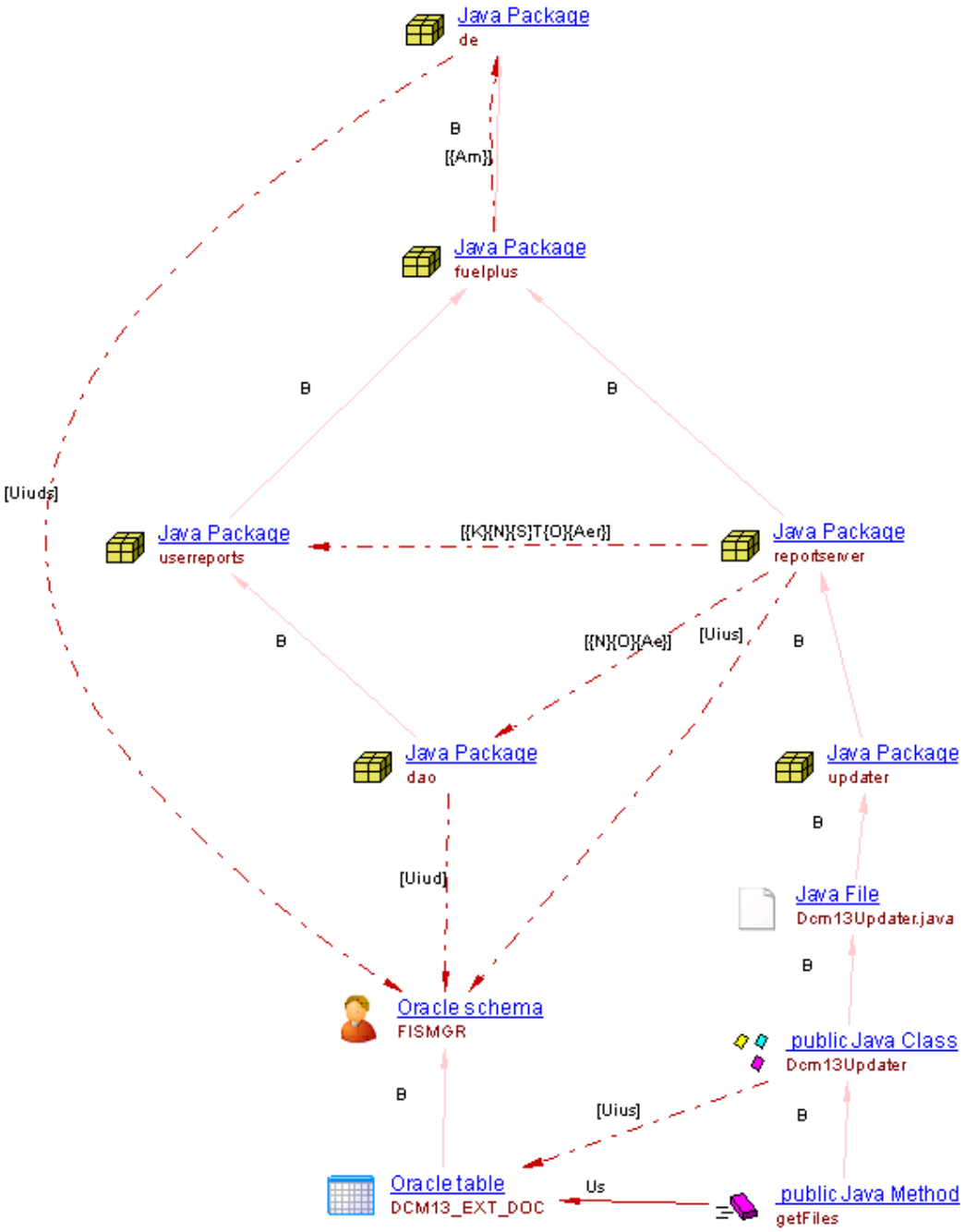


Figure 1 : Architecture diagram - "de" package (showing some files bypassing the DAO)

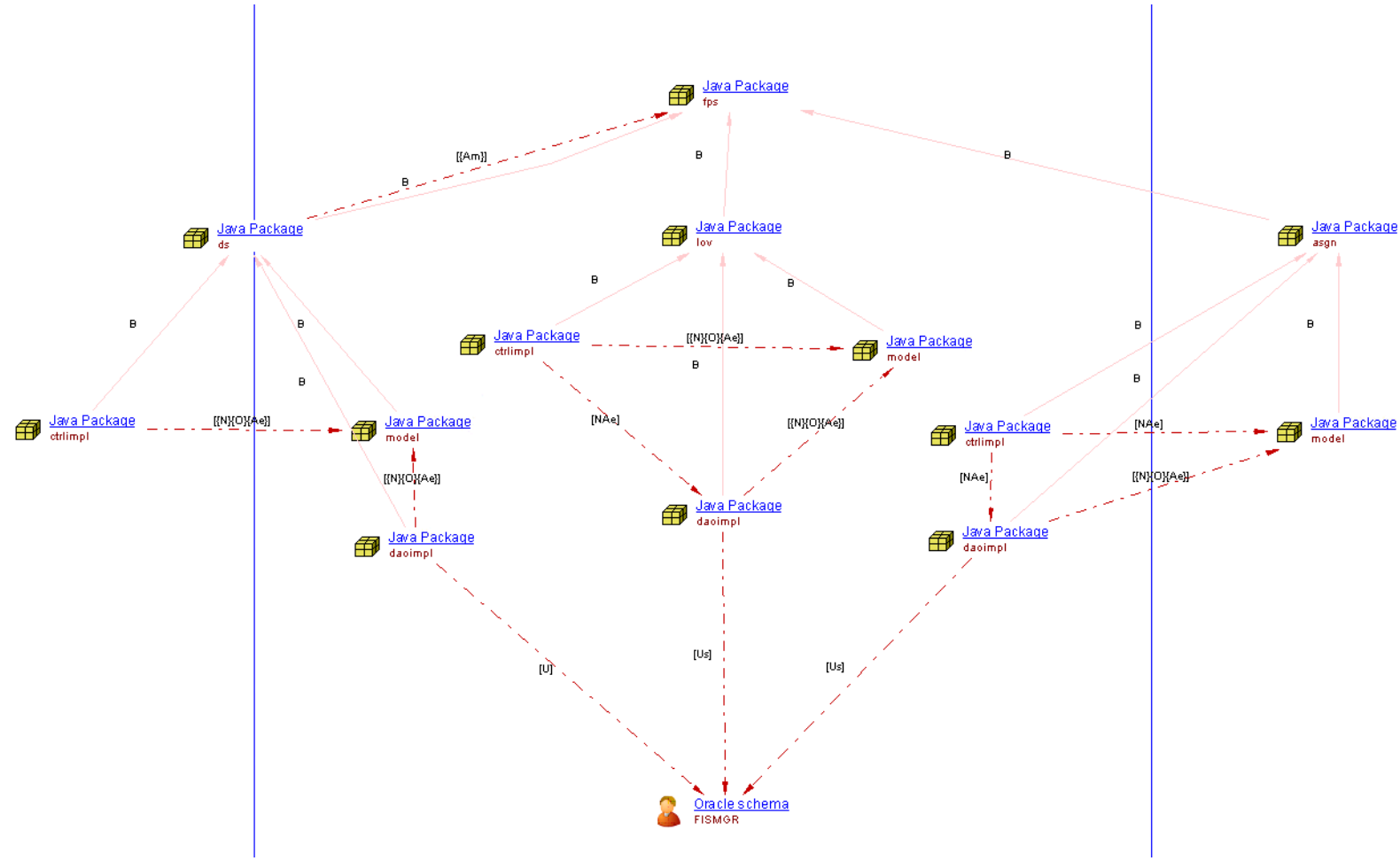
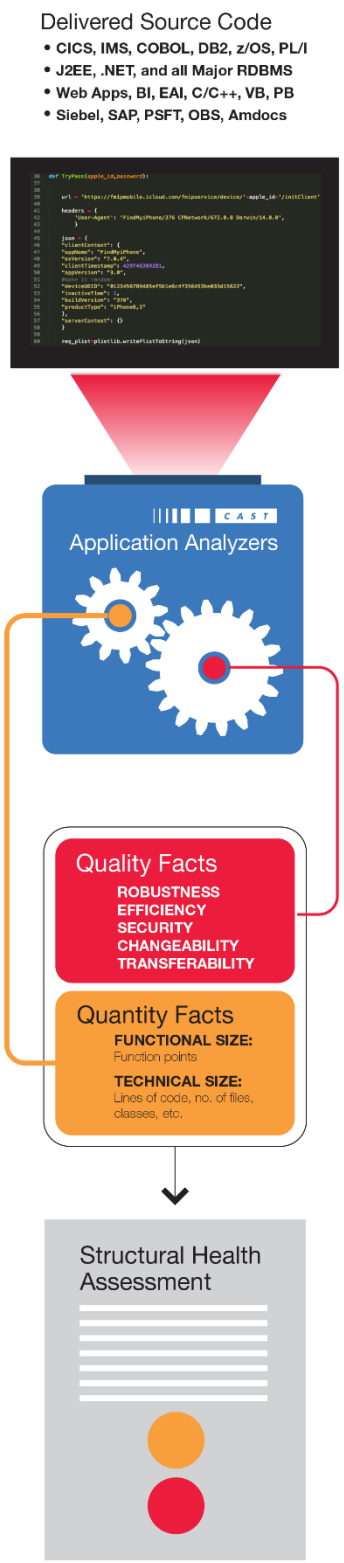


Figure 2: Architecture diagram - "bbs" package

2. Assessment Approach Overview



This assessment is an effort to determine the overall quality of the application and identify any risks that may be inherent in the application. The assessment looks at the implementation of the application to determine whether the application is constructed according to industry best practices, follows best practices for software engineering, and is maintainable.

This assessment is focused solely on the Source code and Database structure with no view to functionality provided by backend services.

CAST Application Intelligence Platform (AIP) is the industry leading automated code analysis platform, with coverage of all major development tools and languages. CAST AIP automatically scans and analyzes all of the source code and database elements that are part of an Enterprise system. CAST AIP applies over 1000+ metrics based on standards and measurements developed by the Software Engineering Institute (SEI), International Standards Organization (ISO), Consortium for IT Software Quality (CISQ), and Institute of Electrical and Electronics Engineers (IEEE). These metrics objectively measure software for the quality and quantity of work.

CAST AIP provides Application Analysts the ability to examine and drill down on critical application characteristics and attributes. The primary Application Health Factors that are addressed follow on the next page.

Health Factor	Description	Example business benefits
Robustness	Attributes that affect the stability of the application and the likelihood of introducing defects when modifying it	• Improves availability of the business function or service • Reduces risk of loss due to operational malfunction • Reduces cost of application ownership by reducing rework
Efficiency	Attributes that affect the performance of an application	• Reduces risk of losing customers from poor service or response • Improves productivity of those who use the application • Increases speed of making decisions and providing information • Improves ability to scale application to support business growth
Security	Attributes that affect an application's ability to prevent unauthorized intrusions	• Improves protection of competitive information-based assets • Reduces risk of loss in customer confidence or financial damages • Improves compliance with security-related standards and mandates
Transferability	Attributes that allow new teams or members to quickly understand and work with an application	• Reduces inefficiency in transferring application work between teams • Reduces learning curves • Reduces lock-in to suppliers
Changeability	Attributes that make an application easier and quicker to modify	• Improves business agility in responding to markets or customers • Reduces cost of ownership by reducing modification effort

3. How Can Technology Address Application Quality Challenges?

The quality attributes of an application can be characterized by the quality attributes of its component parts no more than the attributes of a molecule can be characterized by the attributes of its constituent atoms. Since high quality components do not equate to a high quality system in any field of engineering, code quality, although necessary, is not sufficient to ensure high quality applications. Organizations need the help of application quality diagnostic tools which can discover inter-component issues and measure the internal quality of the application across its tiers.

There are numerous commercial, freeware, and open source tools available that measure code quality specific to a programming language and are often integrated into Integrated Development Environments (IDEs). These tools are becoming standard components of every developer's toolset since they provide quick feedback during the coding and unit test process. However, these tools are not sufficient to address application quality since they cannot evaluate interactions across the various languages, technologies, and tiers of an application.

Technology that measures application quality analyzes the integrated software produced by a build once the code is checked into a central repository by all the developers. In addition to analyzing each component, application quality technology analyzes their interactions for the types of problems described in earlier sections. Moreover, application quality trends can be compared across builds or releases to monitor the progress against application quality objectives and evaluate the risks posed by the application.

Application quality measurement tools provide several benefits for both the development team and management:

- ***Visibility across application(s):*** Consistent and continuous analysis of all core business applications provides executives with the metrics and information needed to better manage their portfolio of applications and projects.
- ***Analysis of the internal quality of an application***: Reviewing the integrated software system for quality in order to detect architectural and structural problems that hide in interactions between tiers, provides application or project managers with continual status about application quality and risk.
- ***Team performance:*** Since a detailed knowledge of the whole system is usually beyond any individual developer's capabilities, analyzing application quality helps improves developer skills, the team's breadth of application knowledge, and the efficiency of team performance.

A dynamic business environment, new technology, and multiple sourcing options, amplify the complexity of business application software. Since even the most talented developers can no longer know all the nuances of all the different languages, technologies, and tiers in an application, their capability needs to be augmented by automated tools to evaluate the entire application. Without such assistance, defects hidden in the interactions between application tiers will place the business at risk for the outages, degraded service, security breaches, and corrupted data that are caused by poor quality applications.

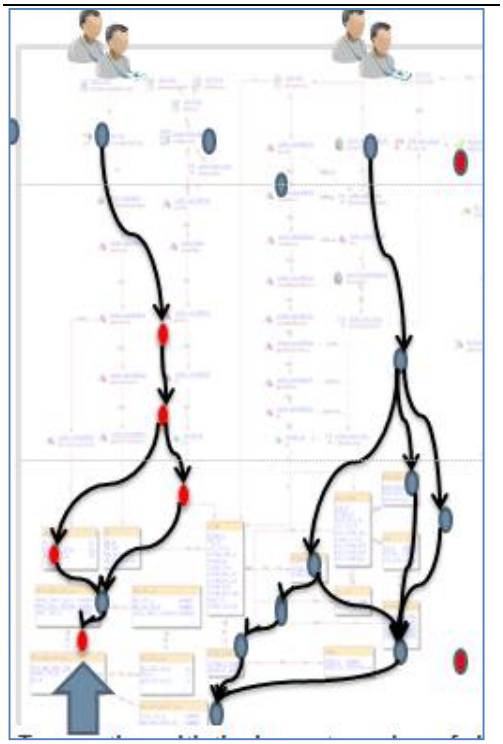
3.1. Potential Points of Failures: Critical rules

The CAST AIP quality model assess automatically the application and raise the main issue of the application through a weighted aggregation of more than +1000 rules across the different technology. The below list represent the different rules which contain some violation on some component which can create some abnormal behavior during the execusion of the application.

Rule Name	# Violations
Avoid using Fields (non static final) from other Classes	5,338
Suspicious similar method names or signatures in an inheritance tree	1,780
The exception Exception should never been thrown. Always Subclass Exception and throw the subclassed Classes.	553
Avoid direct or indirect remote calls inside a loop	424
Avoid instantiations inside loops	93
Avoid declaring Public Instance Variables	89
Close database resources ASAP	68
Close the outermost stream ASAP	68
Avoid empty catch blocks	47
Pages should use error handling page	34

3.2. Potential Points of Failures: Transaction wide Risk Index

Transaction Risk Index (TRI) enables easy identification of the riskiset transactions within the application

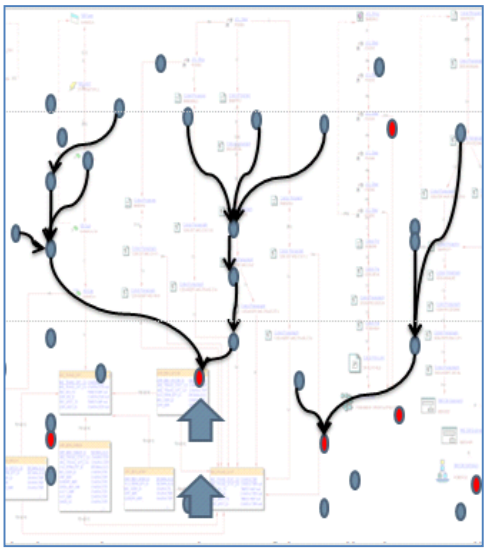


Transaction Wide Risk Index (TwRI) is an indicator of the riskiest transactions of the application. The TwRI number reflects the cumulative risk of the transaction based on the risk in the individual objects contributing to the transaction; in the below list the focus is on the efficiency of the application. The TwRI is calculated as a function of the rules violated, their weight/criticality, and the frequency of the violation across all objects in the path of the transaction. TwRI is a powerful metric to identify, prioritize and ultimately remediate riskiest transactions and their objects.

Transaction Entry Point	TRI
bbs.base.ctrl.FileUploadController.handleFormUpload	35,790
de.bigbusiness.reportserver.ReportServerSevlet	28,000
bbs.base.ctrl.FileUploadController.handleFormUpload	18,170
bbs.base.ctrl.FileUploadController.handleFormUpload2DB	14,450
de.bigbusiness.jobenv.batch.tasklet.BbBaseTasklet	14,400
bbs.base.ctrl.FileUploadController.migrateDocuments2DB	14,080
bbs.base.ctrl.FileDownloadController.getFileFromDb	13,820
bbs.exp.tdr.iata.controller.TenderIataExportController.getFileFromDb	10,420
de.bigbusiness.jobenv.javainst.ecb.EcbTransferPGAFile	7,150
de.bigbusiness.jobenv.controller.rest.BbJobController.upload	7,130
de.bigbusiness.jobenv.controller.rest.BbJobController.uploadFromUrl	7,130
de.bigbusiness.jobenv.javainst.ecb.ECBCleaner	6,350
de.bigbusiness.jobenv.javainst.ecb.EcbBatchServerDate	5,380
de.bigbusiness.jobenv.controller.rest.BbJobController.remove	5,170
de.bigbusiness.jobenv.controller.rest.BbJobController.download	5,170
bbs.base.ctrl.FileUploadController.handleFormUpload	35,790

3.3. Potential Point of Failures: Propagated Risk Index

Propagated Risk Index (PRI) enables easy identification of the riskiset objects/artifacts within the application



Propagated Risk Index (PRI) is a measurement of the riskiest artifacts or objects of the application along the Health Factors of Robustness, Performance and Security.

PRI takes into account the intrinsic risk of the component coupled with the level of use of the given object in the transaction. It systematically helps aggregate risk of the application in a relative manner allowing for identification, prioritization, and ultimately remediation of the riskiest objects.

The PRI number reflects the cummulative risk of the object based on its relationships and interdependencies. The PRI is calculated as a function of the rules violated, their weight/criticality, and the frequency of the violation.

The Top 15 objects with the highest PRI are:

Risk Factor	Object	Propagated Risk Index (PRI)
Robustness	bbs.base.dao.AbstractDao.buildCustomFilterWhere	6,882,940
Security	bbs.base.dao.AbstractDao.buildCustomFilterWhere	4,259,800
Robustness	bbs.base.dao.AbstractDao._buildQueryStatement	3,947,100
Security	bbs.base.dao.AbstractDao._buildQueryStatement	2,118,500
Robustness	bbs.base.dao.AbstractDao.postProcessSummaryResult	1,548,450
Security	bbs.base.dao.AbstractDao.postProcessSummaryResult	1,415,250
Robustness	bbs.base.dao.ResultSetWriter.ResultSetWriter	718,530
Security	bbs.base.ctrl.AbstractController.sendResultSave	509,040
Robustness	bbs.base.ctrl.AbstractController.sendResultSave	475,440
Efficiency	de.bigbusiness.loader.utility.ScriptRunner.runScript	422,400
Security	de.bigbusiness.loader.utility.ScriptRunner.runScript	409,200
Efficiency	de.bigbusiness.loader.utility.ScriptRunner.runScript	377,600
Security	de.bigbusiness.loader.utility.ScriptRunner.runScript	365,800
Security	de.bigbusiness.loader.utility.ScriptRunner.runScript	365,800
Efficiency	de.bigbusiness.loader.savers.OracleSqlSaver.getConnection	305,900

3.4. Strengths and Weaknesses

Main Weaknesses

Highest risks for this application pertains to Security and Efficiency, specifically for the following technical criteria:

- Error exception handling
- Secure coding - input validation
- Secure coding – time and state
- Architecture - multi-layers and data access

Main Strengths

Lowest risks for this application pertains to Changeability and Maintainability, this is because:

- CWE rules all completely respected except 2 log forging violations
- Good usage of a data access layer
- Low complexity on the different elements, making the code transferable and changeable
- Low dependence with OS and platform

4. Measures of Robustness

The root causes of poor reliability are found in a combination of non- compliance with good architectural and coding practices. This non-compliance can be detected by measuring the static quality attributes of an application. Assessing the static attributes underlying an application's reliability provides an estimate of the level of business risk and the likelihood of potential application failures and defects the application will experience when placed in operation.

Technical criterion name	Grade
Architecture - Multi-Layers and Data Access	1.00
Architecture - Object-level Dependencies	3.31
Architecture - OS and Platform Independence	4.00
Architecture - Reuse	2.02
Complexity - Algorithmic and Control Structure Complexity	3.50

Measures: Assessing reliability requires checks of at least the following software engineering best practices and technical attributes:

- Application Architecture Practices
  - Multi-layer design compliance
  - Coupling ratio
  - Component or pattern re-use ratio
- Coding Practices
  - Error & Exception handling (for all layers GUI, Logic & Data)
  - Compliance with Object-Oriented and Structured Programming best practices (when applicable)
- Complexity
  - Transaction complexity level
  - Complexity of algorithms
  - Complexity of programming practices
  - Dirty programming

Depending on the application architecture and the third-party components used (such as external libraries or frameworks), custom checks should be defined along the lines drawn by the above list of best practices to ensure a better assessment of the reliability of the delivered software.

Rule Name	# Violations
Methods must have JavaDoc comments	49,793
Methods must have appropriate JavaDoc @param tags	33,693
Methods must have appropriate JavaDoc @return tags	25,624
Avoid Methods with a very low comment/code ratio	24,481
Private fields must have JavaDoc Comments	20,307

Rules Descriptions for Top Critical Violation Rules For Business Criterion Robustness	
Rule Name	Suspicious similar method names or signatures in an inheritance tree
Rationale	When programming it is very easy to make a mistake when naming a method to override or in its signature. This may occur when writing the inherited class, but also when changing the signature of the basic class to add a new parameter or when changing the type of a parameter. One of the more typical examples is the overriding of equals methods with an argument type that is different from the Object class.
Description	Find all methods in an inheritance tree that have the same signature but whose name differs only by capitalization and methods that have the same name but are overridden with a different signature (this signature doesn't exist in the super class) with the following restrictions: - where the class implements an interface, if one method with the same signature exists in the class then no violation will be reported even if other methods exist with different signatures - where the class extends another class, a violation will be reported for a method only when the number of parameters are the same and one of the parameters of the signature of the child class method inherits from a parameter in the same place as the parent class method with the same name.
Remediation	Fix the name of the method or the signature and if you use JSE 5.0 or later add the @Override annotation to inform the compiler that the method is meant to override a method declared in a superclass. If the method marked with @Override fails to correctly override a method in one of its superclasses, the compiler generates an error.
# Violations	1780

Top Riskiest Artifacts
bbs.ds.daoimpl.acc.InvItemsManuallyClearedDaoImpl.PreQuerySP.execute
bbs.base.LogErrorMessage.execute
bbs.ds.daoimpl.acc.InvDaoImpl.PreQuerySP.execute
bbs.ds.daoimpl.acc.InvoiceItemsIplEditEventsDaoImpl.PreQuerySP.execute
bbs.ds.daoimpl.acc.InvoiceItemsIplEventsByInvDaoImpl.PreQuerySP.execute

Rules Descriptions for Top Critical Violation Rules For Business Criterion Robustness	
Rule Name	The exception Exception should never been thrown. Always Subclass Exception and throw the subclassed Classes.
Rationale	Whenever a method throws an exception of type Exception, it prevents its callers from carrying out the specific recovery process that is needed and as a consequence this will threaten both application robustness and security. For example, each exception related to resource allocation whose catch does not explicitly release the resource might create a "resource leak". When a leak occurs on a limited set of available resources, such as a database connection, the application can then become unusable because resources cannot be allocated any more. The application also becomes difficult to support and run in production as root-cause analysis is made more difficult. The support teams might not even be aware that something went wrong (by catching Exception, RuntimeException might not be visible any more).
Description	This Quality Rule reports all methods throwing an exception of type Exception. The exception Exception should never been thrown.
Remediation	The method must throw a Subclass of the generic Exception that provides valuable information about the exception that occurred in order to help programmers calling this method to write the appropriate recovery or error management code.
# Violations	553

Top Riskiest Artifacts
bbs.base.ctrl.AbstractController.sendResultSave
de.bigbusiness.loader.GenericXMLLoader.checkFile
de.bigbusiness.loader.GenericXMLLoader.checkFile
bbs.base.ctrl.AbstractController.sendResultServiceListByIds
de.bigbusiness.reportserver.birt.BirtEngine.getUTCTimestampDiff

Rules Descriptions for Top Critical Violation Rules For Business Criterion Robustness	
Rule Name	Avoid empty catch blocks
Rationale	An empty catch block defeats the purpose of exceptions. When an exception occurs, nothing happens and the program fails for an unknown reason. The application can be in an unknown state that will affect subsequent processing. Since the reason for the issue (the exception type and potential embedded message) are ignored, it will require more time to fix the issue.
Description	This metric reports all methods with at least one empty catch block (empty or only containing comments). In a Try and Catch statement, Catch blocks should have code to handle the thrown exception. If they are empty or only contain comments, the Exception will not be handled.
Remediation	The exception must be handled correctly according to its type.
# Violations	47

Top Riskiest Artifacts
bbs.base.dao.AbstractDao.postProcessSummaryResult
bbs.base.ctrl.AbstractController.sendResultSave
bbs.base.ctrl.A ... ntroller.sendResultServiceListByIds
de.bigbusiness.jobenv.jobs.remote.BbSftpExecutor.execute
bbetl.jobs.instances.Xls2csvJob.execute

Rules Descriptions for Top Critical Violation Rules For Business Criterion Robustness	
Rule Name	Avoid non-thread safe singleton
Rationale	If singleton is invoked in a multi-threaded program, you could end up creating multiple instances of the class which will make the application instable.
Description	All singleton that initialize the static field that refer to the single instance in a non synchronized method will be reported. A singleton is defined as: - a class with a static member with the same type or parent type (extended or implemented) as the class - a static method that refers the instance and return an object of same type or a parent type (extended or implemented) - a class that has only private constructors
Remediation	To remediate to this issue (in case of multi-threaded environment), there is two solutions: 1/ declare the field that hold the unique instance as static final and initialize it in the declaration 2/ synchronize the method that initialize the field
# Violations	4

Top Riskiest Artifacts
de.bigbusiness.csv2xml.logging.Csv2XmlLogger
de.bigbusiness.loader.logging.LoaderLogger
de.bigbusiness.loader.logging.LoaderLogger
de.bigbusiness.csv2xml.csv.CsvParserImpl

```
public class Csv2XmlLogger {

	private static final String TERMINATED_WITH_ERRORS = "Abnormal loader end";

	public static String LOADER_NAME = null;

	private static Csv2XmlLogger _instance = null;

	private Logger logger = Logger.getLogger("csv2xml");
	private Handler consoleHandler;

	private Logger debugLogger = Logger.getLogger("csv2xml.debug");
	private Handler debugHandler;

	private boolean debugMode = false;

	private Csv2XmlLogger() {
		logger.setUseParentHandlers(false);
		debugLogger.setUseParentHandlers(false);

		// create logger to log message on console
		consoleHandler = new ConsoleHandler();
		consoleHandler.setFormatter(new LogFormatter());
		consoleHandler.setLevel(Level.INFO);
		logger.addHandler(consoleHandler);

		// create logger for debug mode
		debugLogger.setLevel(Level.OFF);
//		createDebugHandler();
//		debugLogger.addHandler(debugHandler);

	}

	public static Csv2XmlLogger getInstance() {
		if (_instance == null) {
			_instance = new Csv2XmlLogger();
		}

		return _instance;
	}
```

Figure 3 : Avoid non thread safe singleton

Rules Descriptions for Top Critical Violation Rules For Business Criterion Robustness	
Rule Name	Avoid cyclical calls and inheritances between packages
Rationale	When two packages refer to each other through a call, the result is a circular dependency. Neither packages can function without the other, and so neither is reusable without the other. In some cases redesign may eliminate these dependencies. When circular references are necessary, redesign it to ensure reusability. The same problem happen when some classes from a package A inherit from classes of a package B and other classes from package B inherit from other classes from package A. This rule can be extended to circular dependencies for more than 2 packages (for example a package A call a package B that call a package C, that call package A).
Description	This metric reports all packages that have one-on-one and more static circular dependencies. Dependencies mean: - references through static methods call - references through class fields - references through inheritance Note that all these links are static link and not runtime. The threshold parameter permit to define the maximal number of packages to cross for a cycle. Note that cycle notion means here a directed path in a graph that is directed by dependencies relations.
Remediation	If there are circular relationships among packages, the partitioning is not clear and should be redesigned. Use CAST Enlighten to see all dependencies to fix.
# Violations	27

Top Riskiest Artifacts
de.bigbusiness.loader
bbs.base
bbs.base.context
de.bigbusiness.csv2xml
de.bigbusiness.csv2xml.converters

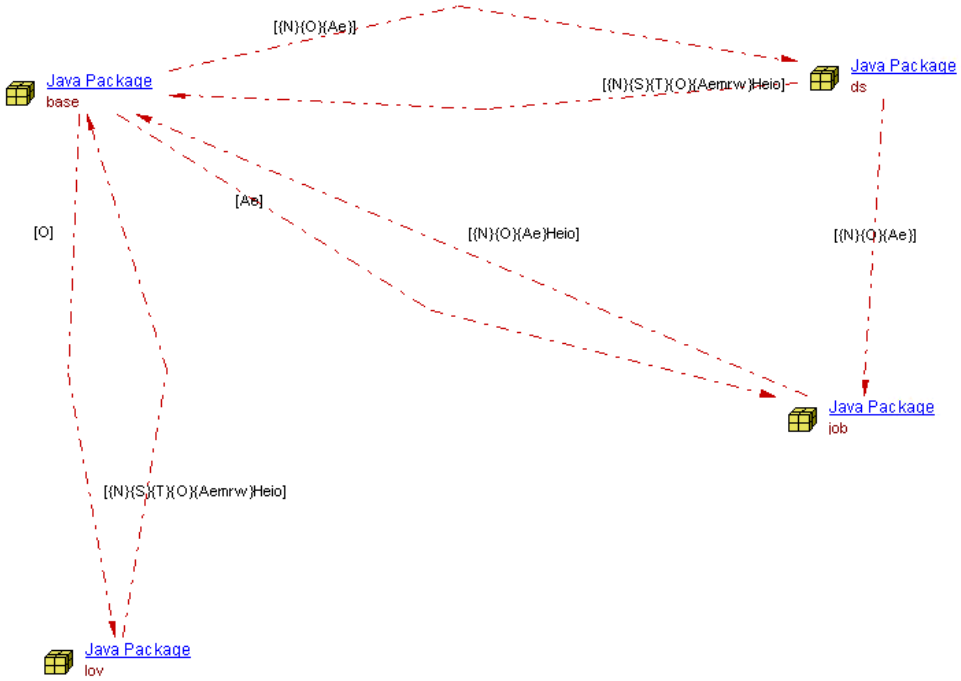


Figure 4 : Architectural diagram - Cyclical calls and inheritance between packages

5. Measures of Efficiency

As with Reliability, the causes of performance inefficiency are often found in violations of good architectural and coding practice which can be detected by measuring the static quality attributes of an application. These static attributes predict potential operational performance bottlenecks and future scalability problems, especially for applications requiring high execution speed for handling complex algorithms or huge volumes of data.

Technical criterion name	Grade
Complexity - Dynamic Instantiation	4.00
Complexity - SQL Queries	3.71
Efficiency - Expensive Calls in Loops	1.61
Efficiency - Memory, Network and Disk Space Management	1.42
Efficiency - SQL and Data Handling Performance	1.38

Measures: Assessing performance efficiency requires checking at least the following software engineering best practices and technical attributes:

- Application Architecture Practices
  - Appropriate interactions with expensive and/or remote resources
  - Data access performance and data management
  - Memory, network and disk space management
- Coding Practices
  - Compliance with Object-Oriented and Structured Programming best practices
  - Compliance with SQL best practices

Rules Descriptions for Top Critical Violation Rules For Business Criterion Efficiency	
Rule Name	Avoid direct or indirect remote calls inside a loop
Rationale	All remote calls made inside a loop will badly impact the performances of your application.
Description	Reports the following remote call inside a loop at a depth level less than <depth level>: * SQL Statement through loop (example: SQL cursor on SQL Server, nested cursors on Oracle) * Stored procedure called many times from the client in a loop. * EJB3 Session remote method * ABAP RFC <depth level> is a parameter that can be changed at will.
Remediation	This loop execution can be delegated to the server side of the application so that not network latency will occur.
# Violations	424

Top Riskiest Artifacts
bbs.job.FuelTicketCsvImport.write_quantities_info_to_db
de.bigbusiness.loader.logging.DbmsOutput.show
bbs.ds.daoimpl.acc.AdjustmentDaoImpl.insert
bbs.ds.daoimpl.acc.AdjustmentDaoImpl.update
bbs.ds.daoimpl.acc.AdjustmentPercetangeDaoImpl.update

Several issues are found on the riskiest component:

- Close() and preparecall() could be outside the loop
- No catch error
- No checking that debug is enabled

```
private void write_quantities_info_to_db(Connection conn,
                Integer fuel_trn_id, List<FFQ09_Fuel_Trn_Qty> model_list)
                throws Exception {

        String sql = "begin po_fop_ftt.insert_into_ffq09( :p_fuel_trn_id, :p_ln_no, :p_upl_qty, :p_upl_unit, :p_qty_type); end; ";

        Iterator<FFQ09_Fuel_Trn_Qty> it = model_list.iterator();

        while (it.hasNext()) {
                FFQ09_Fuel_Trn_Qty model = (FFQ09_Fuel_Trn_Qty) it.next();

                CallableStatement insert_ffq09 = conn.prepareCall(sql);

                if (model.getUpl_qty() != null)
                {
                        insert_ffq09.setInt("p_fuel_trn_id", fuel_trn_id);
                        insert_ffq09.setInt("p_ln_no", model.getLn_no());
                        insert_ffq09.setDouble("p_upl_qty", model.getUpl_qty());
                        insert_ffq09.setString("p_upl_unit", model.getUpl_unit());
                        insert_ffq09.setString("p_qty_type", model.getQty_type());

                        insert_ffq09.execute();

                        insert_ffq09.close();

                        log.debug("Record successfully inserted in insert_ffq09 table");
                }
        }

}
```

Rules Descriptions for Top Critical Violation Rules For Business Criterion Efficiency	
Rule Name	Avoid instantiations inside loops
Rationale	One of the fundamental OO performance management principles is this: Avoid excessive object creation. This doesn't mean that you should give up the benefits of object-oriented programming by not creating any objects, but you should be wary of object creation inside of tight loops when executing performance-critical code. Object creation is expensive enough that you should avoid unnecessarily creating temporary or intermediate objects in situations where performance is an issue.
Description	Reports all artifacts with loops (for, while, do while) that contain object instantiations (object creation).. Java artifacts include all methods and constructors with the following exclusions: - the cases where the instantiation appear at the end of a return or throw statement are excluded. - the case where the instantiation occurs in a call to one of the following methods: . java.util.Collection.add . java.util.Map.put . java.lang.StringBuilder.append . java.lang.StringBuilder.insert . java.lang.StringBuilder.replace . java.lang.StringBuffer.append . java.lang.StringBuffer.insert . java.lang.StringBuffer.replace . and any method with the same name in their respective derived classes (e.g. java.util.ArrayList. add(int index, E element)), as ArrayList implements java.util.Collection and has the same name as the authorized method add).
Remediation	Redesign the loop.
# Violations	93

Top Riskiest Artifacts
de.bigbusiness.loader.utility.ScriptRunner.runScript
de.bigbusiness.loader.savers.OracleSqlSaver.readJobParams
bbs.base.ctrl.AbstractController.prepareQueryContext
de.bigbusiness.loader.savers.OracleSqlSaver.readJobParams
de.bigbusiness.reportserver.service.model.base.ReportParam.getSqlValueConv

Rules Descriptions for Top Critical Violation Rules For Business Criterion Efficiency	
Rule Name	Close database resources ASAP
Rationale	A frequent issue when dealing with database resource is resource leak. This mainly comes from an incorrect code that miss to close the connection in any cases. Incorrect resource management is a common source of failures in production applications, with the usual pitfalls being database connections and file descriptors remaining opened after an exception has occurred somewhere else in the code. This leads to application servers being frequently restarted when resource exhaustion occurs, because operating systems and server applications generally have an upper-bound limit for resources.
Description	The following methods are taken into account: - JDBC: . open: java.sql.DriverManager.getConnection(String) . close: java.sql.Connection.close() - JDBC: . open: java.sql.Connection.createStatement() . close: java.sql.Statement.close() - JDBC: . open: java.sql.Connection.prepareStatement(...) . close: java.sql.PreparedStatement.close() - JDBC: . open: java.sql.Connection.prepareCall(...) . close: java.sql.CallableStatement.close() - JDBC: . open: java.sql.PreparedStatement.executeQuery() . close: java.sql.ResultSet.close() - JPA: . open: javax.persistence.Persistence.createEntityManagerFactory(String) . close: javax.persistence.EntityManagerFactory.close() - JPA: . open: javax.persistence.EntityManagerFactory.createEntityManager() . close: javax.persistence.EntityManager.close() - Hibernate: . open: org.hibernate.SessionFactory.openSession() . close: org.hibernate.Session.close() - Hibernate: . open: org.hibernate.cfg.Configuration.buildSessionFactory() . close: org.hibernate.SessionFactory.close() - Spring: . open: org.springframework.orm.hibernate3.SessionFactoryUtils.getSession(...) . close: org.springframework.orm.hibernate3.SessionFactoryUtils.closeSession(...)
Remediation	You can: - close the resource in a finally block (only explicit closing is considered valid) - or annotate this resource with @Cleanup annotation (lombok.Cleanup) - or use the try with resource to declare the resource that must be closed (available in java 7) - or use Spring JDBC Template that open and close the connection for you (http://static.springsource.org/spring/docs/3.2.x/spring-framework-reference/html/jdbc.html) - or use CDI with @Dispose annotation
# Violations	68

Top Riskiest Artifacts	Violation #
de.bigbusiness.loader.utility.ScriptRunner.runScript	
de.bigbusiness.loader.savers.OracleSqlSaver.getConnection	
de.bigbusiness.loader.savers.OracleSqlSaver.readJobParams	
de.bigbusiness.loader.savers.OracleSqlSaver.getJobCode	
de.bigbusiness.acc.loader.iata.SqlSaver.protokoll	

Rules Descriptions for Top Critical Violation Rules For Business Criterion Efficiency	
Rule Name	Close the outermost stream ASAP
Rationale	A frequent issue when dealing with stream is resource leak. This mainly comes from an incorrect code that miss to close the stream in any cases. Incorrect resource management is a common source of failures in production applications, with the usual pitfalls being database connections and file descriptors remaining opened after an exception has occurred somewhere else in the code. This leads to application servers being frequently restarted when resource exhaustion occurs, because operating systems and server applications generally have an upper-bound limit for resources.
Description	Reports methods that open a stream in the body and that: - doesn't close the outermost stream in a finally block. Note that the number of calls to open a stream and the methods in the finally must be the same. - or doesn't annotate this stream with @Cleanup annotation (lombok.Cleanup) - or doesn't use the try with resource to declare the stream that must be closed The following objects are taken into account: - output streams - input streams - readers - writers - channel
Remediation	You can: - use the try with resource to declare the resource that must be closed (available in java 7) - or close the resource in a finally block. - or annotate this resource with @Cleanup annotation (lombok.Cleanup)
# Violations	68

Top Riskiest Artifacts	Violation #
de.bigbusiness.loader.utility.ScriptRunner.runScript	
bbs.base.ctrl.AbstractController.sendResultSave	
de.bigbusiness.loader.savers.HsqldbSqlSaver.initDatabase	
de.bigbusiness.reportserver.service.model.dynamic.parser.DynamicReportXMLParser.process	
bbs.base.ctrl.AbstractController.sendResultServiceListByIds	

Rules Descriptions for Top Critical Violation Rules For Business Criterion Efficiency	
Rule Name	Avoid to call a function in a termination loop
Rationale	When calling a function in a end loop, the function will be computed for each loop iteration and will decrease dramatically performances.
Description	Reports all JavaScript code that call a function in a loop termination.
Remediation	Use a variable to store the result of a function and use it as the loop termination.
# Violations	32

Top Riskiest Artifacts
S:\Src_Bis\bbs.test.src\WebContent\ClientExtjs\bbs_base_debug.js/CAST_HTML5_JavaScript_SourceCode.doExportGrid
S:\Src_Bis\bbs.test.src\WebContent\ClientExtjs\app\base\GridDataExportWindow.js/CAST_HTML5_JavaScript_SourceCode.doExportGrid
S:\Src_Bis\bbs.test.src\WebContent\ClientExtjs\app\base\dc\AbstractDcvFilterPropGrid.js/CAST_HTML5_JavaScript_SourceCode._getFieldsPosition_
S:\Src_Bis\bbs.test.src\WebContent\ClientExtjs\app\dialog\acc\K6_17.js/CAST_HTML5_JavaScript_SourceCode._isValidFilter_
S:\Src_Bis\bbs.test.src\WebContent\ClientExtjs\bbs-core-all-debug.js/CAST_HTML5_JavaScript_SourceCode.valueChanged

6. Measures of Security

Most security vulnerabilities result from poor coding and architectural practices such as SQL injection or cross-site scripting. These are well documented in lists maintained by CWE http://cwe.mitre.org/, and CERT.

Technical criterion name	Grade
Architecture - Multi-Layers and Data Access	1.35
Architecture - OS and Platform Independence	4.00
Efficiency - Memory, Network and Disk Space Management	1.46
Programming Practices - Error and Exception Handling	1.24
Programming Practices - Unexpected Behavior	2.47

Rules Descriptions for Top Critical Violation Rules For Business Criterion Security	
Rule Name	Favor PreparedStatement or CallableStatement over Statement
Rationale	There are two benefits to use PreparedStatement or CallableStatement over Statement: - Performance: PreparedStatement gives better performance when compared to Statement because it is pre-parsed. CallableStatement is even more efficient but as it uses a stored procedure in the database, it is less portable, - Security: to prevent SQL Injection Attacks.
Description	All references to the method java.sql.Connection.createStatement() must be avoided.
Remediation	use java.sql.Connection.prepareStatement() or java.sql.Connection.prepareCall() instead.
# Violations	31

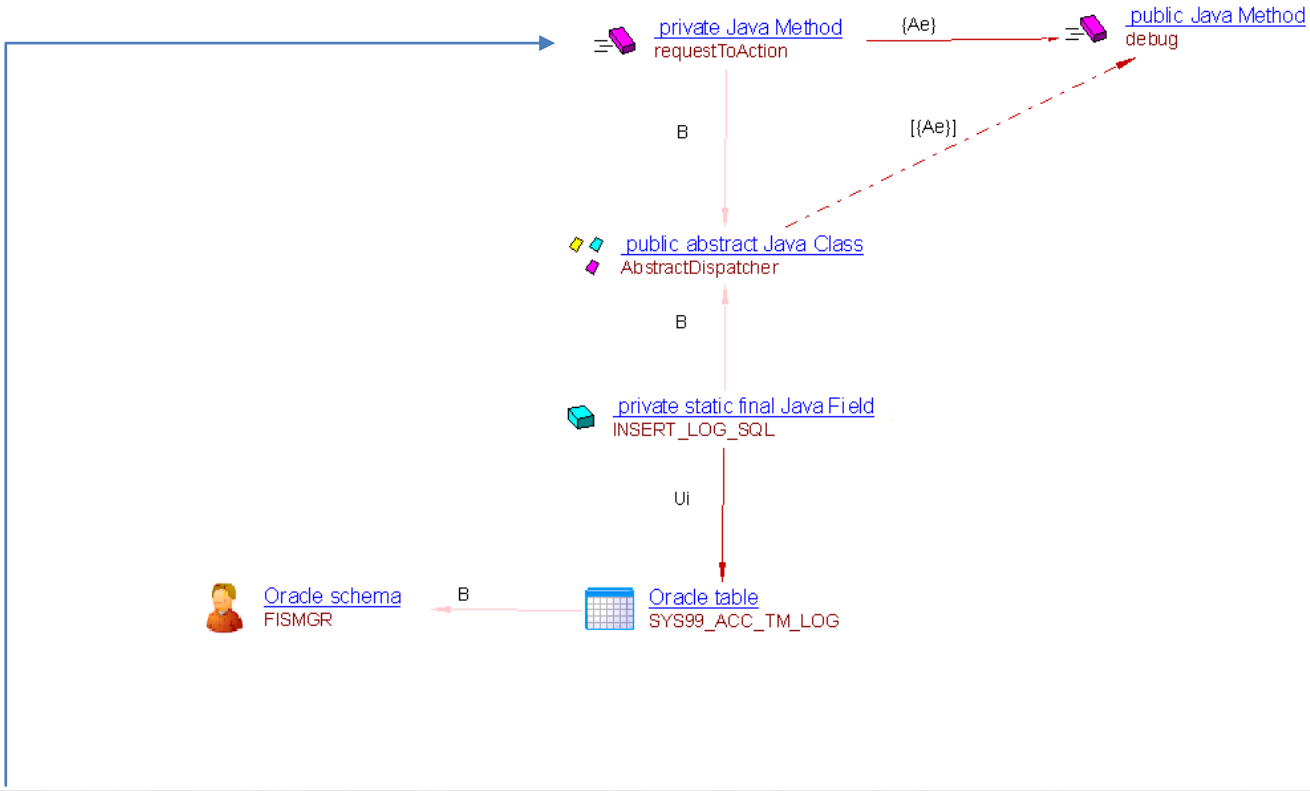
Top Riskiest Artifacts	Violation #
de.bigbusiness.csv2xml.converters	
de.bigbusiness.loader.savers.OracleSqlSaver.getJobCode	
de.bigbusiness.reportserver.service.model.special.SpecialReport.getTitle	
de.bigbusiness.loader.savers.HsqldbSqlSaver.getNextSeq	
de.bigbusiness.loader.savers.OracleSqlSaver.getNextSeq	

Rules Descriptions for Top Critical Violation Rules For Business Criterion Security	
Rule Name	Avoid improper processing of the execution status of data handling operations
Rationale	Software without consistent and complete handling of errors and exceptions makes it impossible to accurately identify and adequately respond to unusual and unexpected situations.
Description	This rule identifies Java control elements that embed data resource access actions without checking return code or handling error. This quality rule checks methods that contain calls to SQL insert, update, select, create table enclosed in a try/catch block. Cases where SQL calls in functions that throw exceptions to be caught higher in the call graph are not addressed by this rule. Either empty catch blocks, catch blocks with comments only, or only write to a stream are violating the rule. Appropriate logging or other treatment of exception are said to be valid. Note: This quality rule implements the rule ASCSM-CWE-252 of the CISQ standard.
Remediation	Implement a consistent and complete handling of errors and exceptions to make it possible to accurately identify and adequately respond to unusual and unexpected situations. Using a logger library is a good practice.
# Violations	3

Top Riskiest Artifacts
bbs.job.MailerJob.executeInternal
bbetl.jobs.instances.Flx2flevntJob.cleanUp
bbetl.jobs.instances.Mfx2grfuelconsJob.cleanUp

Rules Descriptions for Top Critical Violation Rules For Business Criterion Security	
Rule Name	Avoid Log forging vulnerabilities (CWE-117)
Rationale	Writing unvalidated, unsanitized user input to log files can allow an attacker to forge log entries or inject malicious content into the logs. Applications typically use log files to store a history of events or transactions for later review, statistics gathering, or debugging. Depending on the nature of the application, the task of reviewing log files may be performed manually or sometimes automated with a tool that automatically gathers log data for important events or trending information. Interpretation of the log files may be hindered or misdirected if an attacker can supply data to the application that is subsequently logged verbatim.
Description	This metric uses CAST dataflow engine to detect a call path where input data from the user is written into the application logs without prior validation & sanitization. The list of user input methods, Log writing and as well as the validation method can be customized. In Java, the Set methods of the Form Beans are automatically taken into account as input methods.
Remediation	Code the appropriate input validation as close as possible to the user input call.
# Violations	2

Top Riskiest Artifacts
bbs.base.dispatch.AbstractDispatcher.handleRequestInternal
bbs.base.dispatch.AbstractDispatcher.requestToAction




```
private void requestToAction(HttpServletRequest request) throws BadRequestException {
        if (log.isDebugEnabled()) {
                log.debug("*** Converting request to action ...");
        }

        @SuppressWarnings("unchecked")
        Iterator<String> it = (Iterator<String>) request.getParameterMap().keySet().iterator();
        while (it.hasNext()) {
                String param = it.next();
                this.requestMap.put(param, request.getParameter(param));
                if (log.isDebugEnabled()) {
                        if (!param.contains("psw")) {
                                log.debug(" ->Request param " + param + "=" + (String) this.requestMap.get(param));
                        }
                }
        }
        String path = request.getPathInfo();
```

Figure 5: Achitectural diagram - log forging vulnerability

Rules Descriptions for Top Critical Violation Rules For Business Criterion Security	
Rule Name	Avoid Fields in Servlet Classes that are not final static
Rationale	Servlets must be programmed in a thread-safe manner, because the controller will share the same instance for multiple simultaneous requests. In addition to the servlet's threading model, if your intention is to store request-specific state and if your container provides clustering facilities, there's no guarantee that the same servlet instance will receive all the requests (from one user or all users) in a Web application. The use of non static final fields within a Servlet creates a security breach as this object is shared among multiple sessions and thus can lead to confidential data leaks.
Description	Find all non final static fields that belong to Servlets. All Classes that extend HttpServlet at any level are considered as Servlets.
Remediation	Review the Class design. Store global information in HttpSession, or use stateful session beans that are specifically targeted for this purpose. For temporary storage in a Servlet use local variables that are scoped within the doGet or doPost methods (or the service method).
# Violations	8

Top Riskiest Artifacts
bbs.base.servlet.BbDevCenterServlet.jsonParser
bbs.base.servlet.BbDevCenterServlet.projects
de.bigbusiness.reportserver.ReportServerSevlet.birtConfigPath
de.bigbusiness.reportserver.ReportServerSevlet.logger
bbs.base.servlet.BbDevCenterServlet.logger

Rules Descriptions for Top Critical Violation Rules For Business Criterion Security	
Rule Name	Avoid functions without returning exit code (Shell)
Rationale	Every function should have a returning exit code. This will provide a modularity to the code.
Description	This rule searches all the functions without an exit code.
Remediation	Try to reduce the number of function not having a returning exit code.
# Violations	633

Top Riskiest Artifacts
S:\Source\sha_application\SOURCEE\svn_sources\product\src\server\unix_specific\DLH\jobs\sap_dlh.sh.SHELLProgram.sap_dlh.SHELLFunction.fis_echo
S:\Source\sha_application\SOURCEE\svn_sources\product\src\data\others\idl.sh.SHELLProgram.idl.SHELLFunction.perform_add_check
S:\Source\sha_application\SOURCEE\svn_sources\product\src\data\others\idl.sh.SHELLProgram.idl.SHELLFunction.run_loader
S:\Source\sha_application\SOURCEE\svn_sources\product\src\db\scripts\install\configure.sh.SHELLProgram.configure.SHELLFunction.generate
S:\Source\sha_application\SOURCEE\svn_sources\product\src\db\scripts\install\create\cre_db_instance.sh.SHELLProgram.cre_db_instance.SHELLFunction.make_lower

7. Measures of Changeability and Transferability

Changeability and Transferability, also known as maintenability, includes concepts of modularity, clarity, testability, and reusability from one development team to another. These do not take the form of critical issues at the code level. Rather, poor maintainability is typically the result of thousands of minor violations with best practices around documentation, complexity avoidance strategy, and basic programming practices that make the difference between clean and easy to read code vs. ugly and difficult to read code.

Transferability

Technical criterion name	Grade
Architecture - Object-level Dependencies	3.31
Complexity - Algorithmic and Control Structure Complexity	3.52
Complexity - Dynamic Instantiation	4.00
Complexity - OO Inheritance and Polymorphism	3.77
Complexity - SQL Queries	3.71

Changeability

Technical criterion name	Grade
Architecture - Multi-Layers and Data Access	1.50
Architecture - Object-level Dependencies	3.31
Architecture - OS and Platform Independence	4.00
Architecture - Reuse	2.10
Complexity - Algorithmic and Control Structure Complexity	3.52

Measures: Assessing maintainability requires checking the following software engineering best practices and technical attributes:

- Application Architecture Practices
  - Multi-layer design compliance
  - Coupling ratio
  - Component or pattern re-use ratio
- Programming Practices (code level)
  - Compliance with Object-Oriented and Structured Programming best practices (when applicable)
- Complexity
  - Complexity level of transactions
  - Complexity of algorithms
  - Complexity of programming practices
  - Dirty programming
- Documentation
  - Code readability
  - Architecture, Programs and Code documentation embedded in source code
  - Source code file organization cleanliness
- Portability
  - Hardware, OS, middleware, software components and database independence

Rules Descriptions for Top Critical Violation Rules For Business Criterion Changeability	
Rule Name	Avoid using Fields (non static final) from other Classes
Description	To respect OO encapsulation concepts, Fields should not be accessed from outside the Class without going through their accessors.
# Violations	5,388

Top Riskiest Artifacts
bbs.base.dao.ResultSetWriter.getNumberMask
de.bigbusiness.loader.savers.OracleSqlSaver.getJobCode
bbs.base.model.AbstractMarshaller.setFormatRelatedToLocale
de.bigbusiness.loader.savers.OracleSqlSaver.getConnection
de.bigbusiness.loader.savers.OracleSqlSaver.getJobCode

Rules Descriptions for Top Critical Violation Rules For Business Criterion Transferability	
Rule Name	Avoid hiding static Methods
Rationale	Hiding is all about polymorphism. This means that the OO designer expects to override methods and use polymorphism so that code calling methods through a base class will end up executing different methods depending on the instance being used. This is not the case with static methods. When static methods are called, there is no polymorphism in play. It is always the static method of the type used to reference the object used that is called. Hiding static methods is a misuse of OO practices that results in misunderstanding of what is going to be executed at runtime and thus leads to unexpected behavior, jeopardizing the stability of the application.
Description	Hiding Static Methods is not allowed. This Quality Rule retrieves all static methods that are redefined in subclasses i.e. "implicitly hidden". A Static Method MyMethod of Class MySuperClass is "implicitly hidden" in Subclass MySubClass if MySubClass contains a similar declaration of MyMethod (i.e. same signature).
Remediation	Review the design of the Method
# Violations	4

Top Riskiest Artifacts
org.eclipse.wb.swt.SWTResourceManager.decorateImage
org.eclipse.wb.swt.SWTResourceManager.decorateImage
org.eclipse.wb.swt.SWTResourceManager.dispose
org.eclipse.wb.swt.SWTResourceManager.disposeImages

Rules Descriptions for Top Critical Violation Rules For Business Criterion Transferability	
Rule Name	Proper overriding of 'finalize()'
Rationale	A call to 'super.finalize()' ensures the finalize behavior will still work.
Description	When overriding the 'finalize()' Method, a call to 'super.finalize()' is necessary.
Remediation	Review the Method's definition.
# Violations	2
Top Riskiest Artifacts	
de.bigbusiness.acc.loader.iata.SqlSaver.finalize	
de.bigbusiness.mvd.loader.SqlSaver.finalize	

Rules Descriptions for Top Critical Violation Rules For Business Criterion Changeability	
Rule Name	Avoid Too Many Copy Pasted Artifacts
Rationale	A program with a lot of duplication is hard to change. It might be required to change every copy of a copy/pasted code while it is very difficult to locate these copy/pasted code elements. Copy-and-paste is not always bad for a quick urgent "hack", but it is always a very dangerous practice in the long run.
Description	This metric measures the ratio between the number of duplicated, copy/pasted artifacts and the total number of artifacts. Copy / Paste detection is based on statistical detection methods. The statistical methods used compute a similarity metric between all artifacts. Artifacts are reported as copy / pasted when the similarity is higher than 90% (see metric parameter SIMILARITY). Like any statistical method, the detection algorithms require a well sized sample in order to provide significant results: testing these algorithms with a couple of classes will not do the job, a real life application's source code is required to yield usable results. The minimal size required stands at around 5000 lines of code. Below such a size, the algorithms detect the full list of exact copies for the copy/paste code detection but slightly modified copy/paste code will not always be detected. Also, for optimal efficiency, the copy/pasted code detection is enabled only for artifacts larger than 10 lines of code (methods, functions, procedures, triggers, and programs). CISQ rule: ASCMM-MNT-19.
Remediation	Review the Method definition
# Violations	5,987

Top Riskiest Artifacts
de.bigbusiness.loader.AbstractFlxTableMapper.setParentMapper (flxloader)
de.bigbusiness.loader.AbstractFlxTableMapper.setParentMapper (serverflx)
bbs.base.dao.AbstractDao.injectFormattersIntoSp
bbs.base.model.AbstractMarshaller.formatDateToString
bbs.base.ctrl.AbstractController.injectFormattersIntoDao

8. Appendix: Understanding Quality Indicators, Quality Rules

CAST AIP has 1000+ quality rules and each rule produces a Grade. Depending on the impact the grades are aggregated into high level Indicators: **Quality indicators** and **Best practices indicators**.

Each aggregation is a weighted average of the contributing metrics grades where certain metric grades are flagged critical, i.e. it is nearly a defect. We talk about **Critical Violations**.

Quality Indicators

The structure, classification and terminology are from the ISO 9126-3 and the subsequent ISO 25000:2005 quality model. The main focus is on internal structural quality. Subcategories have been created to handle specific areas like business application architecture and technical characteristics such as data access and manipulation or the notion of transactions. The dependence tree between software quality characteristics and their measurable attributes is represented in the following diagram, where each of the 5 characteristics that matter for the user or owner of the business system depends on measurable attributes: Application Architecture Practices, Coding Practices, Application Complexity, Documentation, Portability, and Technical & Functional Volume.

Quality Indicator	Description
Performance / Efficiency	The source code and software architecture attributes are the elements that ensure high performance once the application is in run-time mode. Efficiency is especially important for applications in high execution speed environments such as algorithmic or transactional processing where performance and scalability are paramount. An analysis of source code efficiency and scalability provides a clear picture of the latent business risks and the harm they can cause to customer satisfaction due to response-time degradation.
Robustness / Reliability	An attribute of resiliency and structural solidity. Reliability measures the level of risk and the likelihood of potential application failures. It also measures the defects injected due to modifications made to the software (its "stability" as termed by ISO). The goal for checking and monitoring Reliability is to reduce and prevent application downtime, application outages and errors that directly affect users, and enhance the image of IT and its impact on a company's business performance.
Security	A measure of the likelihood of potential security breaches due to poor coding and architectural practices. This quantifies the risk of encountering critical vulnerabilities that damage the business and provides a list of prevention measures.
Transferability	The effort necessary to diagnose the cause of a failure or section of code to be modified. It establishes the level of dependency on specific developers
Changeability	The effort necessary to modify the source code. It establishes the level of responsiveness to business-driven change requests
TQI	Total Quality Index (TQI) is computed on all the measures made by the CAST AIP

Best practices Indicators

Health Factor	Description
Programming Practices	Measures the level of compliance of the application to coding best practices. Compliance to best practices reduces risks of failures in production and improves productivity through increased readability and reduced debugging.
Architectural Design	Measures the level of compliance of the application to software architecture and design rules. Compliance to architecture rules improves productivity through better use of existing frameworks and code and reduced debugging.
Documentation	Measures the level of compliance of the application to code documentation best practices. Compliance to documentation best practices improves productivity through increased readability and faster understanding of source code.

The risk level of a grade shall be assessed according to the below scale

Scale	Risk Level
4	Low Risk
3	Moderate Risk
2	High Risk
1	Very High Risk

9. Appendix: Importance of measuring all layers of an application

Measuring the technical quality of business software applications is evolving from an art to a science with the availability of software tools that automate the process of code analysis. However, it is critical to understand that there are two categories of software quality with very different implications for operational performance. The first category is Code Quality which measures individual or small collections of coded components written in a single language and occupying a single tier (e.g., user interface, logic, or data) in an application. The second category, Application Quality, analyzes the software across all of the application's languages, tiers, and technologies to measure how well all an application's components come together to create its operational performance and overall maintainability.

Although the code quality of individual components is important, by itself it will not ensure the overall quality of the application. Quality is not an intrinsic property of code: the exact same piece of code can be excellent in quality or highly dangerous depending on the context in which it operates. Ignoring the larger context in which the code operates – the multitude of connections with other code, databases, middleware, and APIs – will often generate a large number of false positives.

Today's business applications are complex, built in multiple languages on multiple technologies. Even more challenging, these applications usually interact with other applications built on different technologies. Analyzing the quality of modern applications is monstrously complex and can only be accomplished with automated software that analyzes the inner structure of all components and evaluates their interactions in the context of the entire business application.

Typical application quality problems are listed below to clarify the distinction between application and code quality. Performance testing alone is not sufficient to detect these application quality problems.

9.1. Bypassing the Architecture.

Components in one tier of a multi-tier application are typically designed to access components in another tier only through an intermediate "traffic management" component. Bypassing this traffic management component will usually result in a cascade of problems.

9.2. Failure to Control Processing Volumes.

Applications can behave erratically when they fail to control the amount of data or processing they allow. This problem is often caused by a failure to incorporate controls in each of several different architectural tiers.

9.3. Application Resource Imbalances.

When database resources in a connection pool are mismatched with the number of request threads from an application, resource contention will block the threads until a resource becomes available, tying up CPU resources with the waiting threads and slowing application response times to a crawl.

9.4. Security Weaknesses.

Applications are vulnerable to security attacks when they lack appropriate sanitization checks on user inputs in all relevant tiers of the application.

9.5. Lack of Defensive Mechanisms.

Since the developers implementing one tier cannot anticipate every situation, they must implement defensive code that sustains the application's performance in the face of stresses or failures affecting other tiers. Tiers that lack these defensive structures are fragile because they fail to protect themselves from problems in their

interaction with other tiers. Each of these application quality problems will result in unpredictable application performance, business disruption, data corruption, and make it difficult to alter the application in response to pressing business needs. Reliably detecting these problems requires an analysis of each application component in the context of the entire application as a whole – an evaluation of application rather than code quality.