Codeprism

CODEPRISM WHITE PAPER

Enhancing Application Quality and Saving Development Cost with Static Analysis Technology

GTONE
Governance Technology

Overview

There is a strong link between business risk and software risk because the software defects can directly impact customer satisfaction, brand image, revenue, and time-to-market. Discoveries of code defects often happen late in the development life cycle or in production, which can cause significant post-development work.[1]

Finding software defects at the initial development stage will reduce huge amount of defect-fixing cost. However, checking source codes of software with manual way relies on developers' experiences and needs large amount of times and costs.

Therefore, organizations need an efficient method to find defects and ensure software quality. Automated code review or inspection is one of options to effectively discover software defects in the early stage of development life cycle, which checks software's source code for compliance with a predefined set of rules or best practices.

The static analysis tool can be used to assist with automated code inspection.[2] It compares favorably to manual reviews, but they can be done faster and more efficiently. The tool also encapsulates deep knowledge of underlying rules and semantics required to perform this type of analysis such that it does not require the human code reviewer to have the same level of expertise as an expert human auditor.

Its aim is automatically detecting and locating defects in source code. Those defects can be broadly divided into two categories; security vulnerability and quality. The vulnerability is a weakness which allows an attacker to attack a system, decreasing system's security assurance. The defects associated with software quality may vary ranging from potential errors and bad performance factors to non-compliance with development standards.

In this paper, we will explorer the static analysis technology and its benefits.

---

[1] Forrester Research, Software Integrity Report

[2] Code inspection means a formal testing technique where the programmer reviews source code with a group who ask questions analyzing the program logic, analyzing the code with respect to a checklist of historically common programming errors, and analyzing its compliance with coding standards.

Static Analysis Technology

Static analysis is the analysis of computer software that is performed without actually executing programs. It normally requires just source code as an input. The results from the static analysis are generated with a complete view of every possible execution path, rather than some aspect of a necessarily limited observed runtime behavior.

On the other hand, dynamic analysis is the analysis that is performed by executing programs on a real or virtual processor. For dynamic analysis to be effective, the target program must be executed with sufficient test inputs to produce interesting behavior. It usually takes a lot of time and is difficult to know whether the analysis covers all possible cases.

There are two major families of static analysis technology; pattern matching (also known as syntactic analysis) and path flow analysis (also known as semantic analysis). The first one is most frequently used to validation of the basic syntax and structure of code, whereas the second one is applied for more complete types of analysis depending on understanding a code execution path.

Pattern matching analysis

The pattern matching technology is based on an Abstract Syntax Tree (AST) or parse tree which is a tree-structured representation of the source code as might be generated by parsing of a compiler. This tree contains a rich breakdown of the structure of the source code, allowing search some straightforward patterns including:

- Code layout
- Usage of comments
- Conformance to naming conventions
- Function call restrictions
- Identification of dangerous or bad coding practices
- Many other simple patterns

Anything that can be inferred from the code without requiring knowledge of that code's runtime behavior is typically a target for AST checking. This type of checking is relatively simple to do, and useful for enforcing development standards or industry recommended best practices.

Path flow analysis

The path flow is simply source code's execution path flow. Some weakness in source code can not be expressed as simple syntactic patterns. For instance, in C language, a defect such as "an allocated memory cell must be disposed before the program termination" can not be described using a simple syntactic pattern, but can be specified using a semantic flow pattern.

To detect the above example, for instance, code execution paths should be examined with control-flow and data-flow graph.[3] This type of technology enables to locate sophisticated defects including:

- Memory leaks
- Invalid pointer dereferences
- Buffer overflow
- Many other possible error conditions

You've probably had an X-ray examination of some part of your body. Health care professionals use it to look for broken bones with relatively low-resolution picture. The pattern matching technology is analogous to X-ray examination of source code.

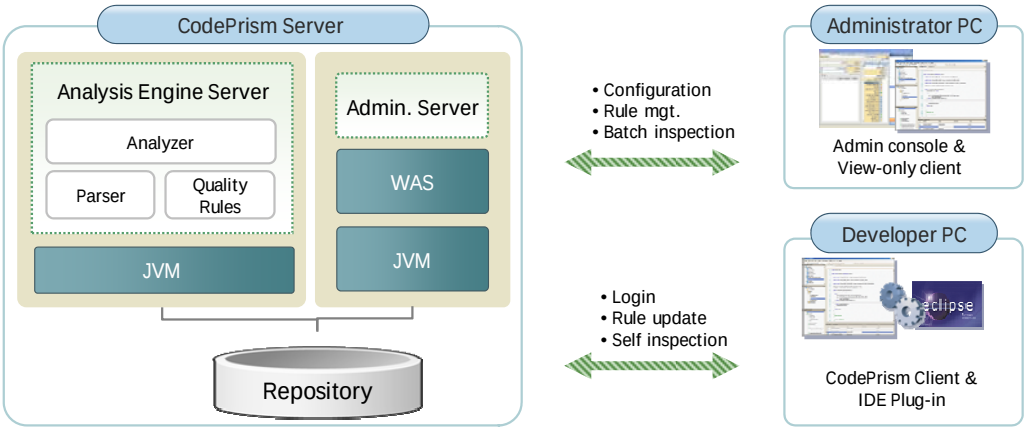
On the other hand, an MRI (Magnetic Resonance Imaging) scan allows high-resolution pictures of many organs and tissues to be taken that are invisible to standard x-rays. The path flow analysis technology can be thought as MRI for source code.

---

[3] A control-flow graph (CFG) in computer science is a representation, using graph notation, of all paths that might be traversed through a program during its execution. A data-flow graph (DFG) is a graph which represents data dependencies between a numbers of operations.

What is CodePrism?

GTONE's CodePrism is a source code's quality inspection tool based on static analysis technology. It detects software defects in the quality point of view without running programs. The following figure shows its architecture and working mechanism.



The analysis engine server is based on J2EE technology for platform independency and stability. There are 2 kinds of servers; analysis server and admin server. The analysis server is responsible for followings:

- Collects source files from remote machine (using FTP) or local machine.
- Parses and analyzes collected source files with predefined quality rules.
- Stores analysis results to central repository.

The admin server, J2EE web application, takes care of followings:

- Processes requests from client to show analysis results.
- Processes requests from admin console to conduct administration works.

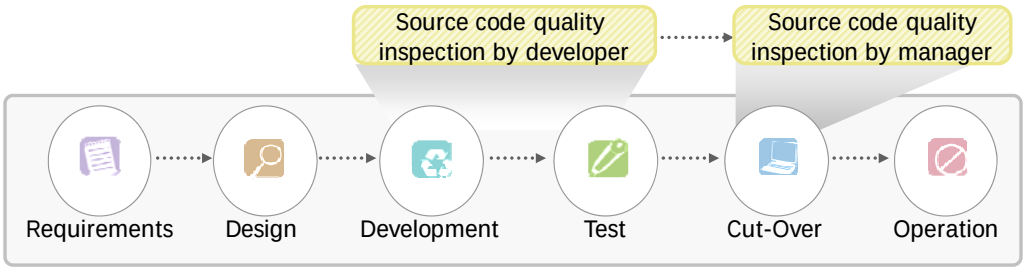
The repository is based on relational database. It contains all analyzing result data for application managers and developers. There is a view-only client program for managers or general users, which just supports view of the analysis results. Note that the view-only client is only for server side Inspection results. The client program is based on Microsoft .NET technology for rich user interface. It supports zero configuration installation and automatic version upgrade for user convenience. Users just need internet connection to use it.

For developers, Eclipse plug-in (for Java language) and standalone inspection client (for other languages) are available. They contains analysis engine inside, enabling developers to inspect source code on their own development environment.

Administrator may configure server environment and manage quality inspection rules with admin console. Because the CodePrism server supports scheduling and manual analysis, administrator can setup scheduled batch inspection or on-demand inspection in the server side.

Developers should login to the admin server from inspection client or Eclipse plug-in to verify user account and download updated rules before analyzing their own source code.

The following figure explains general usage of the tool.



- At the development stage, developers can check their source codes by themselves
- At the test stage, developers or QA (Quality Assurance) team can apply checking rules to facilitate more accurate and flexible testing.
- At the cut-over stage, managers try to check the quality of all source codes at once.

The tool supports various languages such as COBOL, Java, JSP, C, SQL, C#, and VB.NET with hundreds of pre-defined inspection rules. It also supports user defined custom rules based on regular expression and APIs, allowing users to flexibly extend the inspection rules.

What type of defects can be detected?

In this section, we will go through some examples of software defects which can be identified using the static analysis tool. Although many more types of defects can be detected by CodePrism, the examples in this section are just for the reader's better understanding.

Legacy application's quality problems

When applications become legacy, SDLC (Software Development Life Cycle) tasks such as analysis, change and testing take longer time. Furthermore more errors and performance issues are occurred in the legacy applications.

Let's take a simple COBOL example.

```
1001           05 TAB1 OCCURS 1000.
1002                10 SALES PIC 9(7) PACKED-DECIMAL.
1003                10 EXPENSES PIC 9(7) PACKED-DECIMAL.
1004                10 INVENTORY PIC 9(7) PACKED-DECIMAL.
…              …
```

There is a performance issue in the line number 1001 because index variable is not declared when an array variable is declared. The declaration of index variable for accessing to array elements can improve the performance when an array variable is declared. Therefore, the 1001 line should be fixed as follow:

```
05 TAB2 OCCURS 1000 INDEXED BY TABINDX.
```

Here is another sample for SQL statement in COBOL program.

```
2001          SQL-RR0Q0719.
2002                        EXEC SQL
2003                        SELECT  CIST9, CIST9_RGIL, MCMNE,
2004
2005                                  CPVCI, CPFXO_TLNC, FCPGO, NABTI,
2006                                  …  CCOTA_Y999, CCAB9
2007                        INTO
2008                                :DCLRR007T00.CIST9
2009                               , :DCLRR007T00.CIST9-RGIL
2010                                  ...
2011          , :DCLRR007T00.CCAB9
2012                        FROM GRUP.RR007T00
2013                        WHERE
2014                          FCPGO = 'C'
2015                        AND
2016                          CPVCI   =   :DCLRR007T00.CPVCI
2017                        END-EXEC.
2018                  END-SQL-OR0Q0719.
```

The line number 2014 may cause some DB2 database performance problem. The line uses a constant ‘C’ in WHERE clause instead of BIND variable. It could use the database resources heavily. Therefore, the code line would be modified as follow:

```
WHERE
   FCPGO = :COL_VALUE
```

CodePrism can rapidly detect those kinds of coding patterns in COBOL program and SQL statement, allowing developers to reduce search time significantly.

Best coding practices

Using best practices in programming greatly reduces the probability of introducing errors into your applications, regardless of which software development model is being used to create that application.

Best coding practices gives you a way to analyze your source code so that certain rules and patterns can be detected automatically and that the knowledge obtained through previous years of experience by industry experts is implemented in an appropriate way. It should be understood that these practices are not just a way to enforce naming conventions in your code.

Let's take a simple Java code example from open source world.

```
1001    import java.util.ArrayList;
1002    import java.util.Collection;
1003
1004    public class Test {
1005
1006        public static void main(String[] args) {
1007            Collection c=new ArrayList();
1008            Integer obj=new Integer(1);
1009            c.add(obj);
1010
1011            Integer[] a=(Integer [])c.toArray();
1012
1013            Integer[] b=(Integer [])c.toArray(new Integer[c.size()]);
1014        }
1015    }
```

According to popular Java best coding practices, when you need to get an array of a class from your Collection, you should pass an array of the desired class as the parameter of the toArray() method. Otherwise you will get a ClassCastException.

The line number 1011 would throw a ClassCastException if executed. The best practice for this case is the line number 1013. If an unskilled developer doesn't know this best practice, his or her source code would have problem like this.

Here is another Java example from open source world.

```
1001    public class SeniorClass {
1002      public SeniorClass(){
1003          toString(); //may throw NullPointerException if overridden
1004      }
1005      public String toString(){
1006        return "IAmSeniorClass";
1007      }
1008    }
1009    public class JuniorClass extends SeniorClass {
1010      private String name;
1011      public JuniorClass(){
1012        super(); //Automatic call leads to NullPointerException
1013        name = "JuniorClass";
1014      }
1015      public String toString(){
1016        return name.toUpperCase();
1017      }
1018    }
```

Calling overridable methods during construction poses a risk of invoking methods on an incompletely constructed object and can be difficult to discern. It may leave the sub-class unable to construct its super-class or forced to replicate the construction process completely within itself, losing the ability to call super() method. If the default constructor contains a call to an overridable method, the sub-class may be completely uninstantiable.

In a team environment, best coding practices ensure the use of standards and uniform coding, reducing oversight errors and the time spent in later maintenance works. Therefore, managers definitely want developers to follow best practices and CodePrism can give the knowledge that the code produced by the developers meets all the guidelines mandated in pre-defined best practices.

Runtime error-prone code

NULL pointer dereference

A pointer is a programming language data type whose value refers directly to (or "points to") another value stored elsewhere in the computer memory using its address.

A null pointer has a reserved value, often but not necessarily the value zero, indicating that it refers to no object. Since a null-valued pointer does not refer to a meaningful object, an attempt to dereference a null pointer usually causes a run-time error. If this error is left unhandled, the program terminates immediately.

In the case of C on a general computer, execution halts with a segmentation fault because the literal address of NULL is never allocated to a running program. In most C programming environments malloc() returns a null pointer if it is unable to allocate the memory region requested, which notifies the caller that there is insufficient memory available.

```
1001          int *foo(int i) {
1002          int *x;
1003           if (i > 0) {
1004             x = malloc(sizeof(int));
1005          *x = i;
1006           } else {
1007             x = NULL;
1008           }
1009           return x;
1010           }
1011           int main() {
1012           int *x = foo(0);
1013           int y = *x; /*** Null pointer dereference ***/
1014          ...
1015           }
```

In the given example code, because the function foo( ) call’s parameter value is not bigger than 0 at the line number 1013, it will return null pointer. Therefore, a pointer variable x has a null pointer in line number 1014. CodePrism can detect this kind of null pointer dereference case.

Memory leaks

Memory allocation and the correct releasing of that memory are very important particularly in C and C++ code. A memory leak is the gradual loss of available computer memory when a program repeatedly fails to return memory that it has obtained for temporary use. As a result, the available memory for that application or that part of the operating system becomes exhausted and the programs can no longer function.

```
1001          void foo() {
1002          int *x = malloc(sizeof(int));
1003          int *y = malloc(sizeof(int));
1004          *x = 1;
1005          *y = 2;
1006          y = x;  /*** memory leak ***/
1007          // ...
1008          }…
```

In the line number 1003, it creates a heap object and is pointed by a pointer Y. However, in the line number 1005, another pointer is assigned to the y, As a result, any pointer doesn't point the heap object so that there would no way to access the heap object. This kind of statement is detected as a memory leak in CodePrism.

Dangling pointer dereference

If a pointer is defined firstly or released, the pointer becomes a dangling pointer that points unknown location of memory. Accessing to the dangling pointer may result in unpredictable behavior.

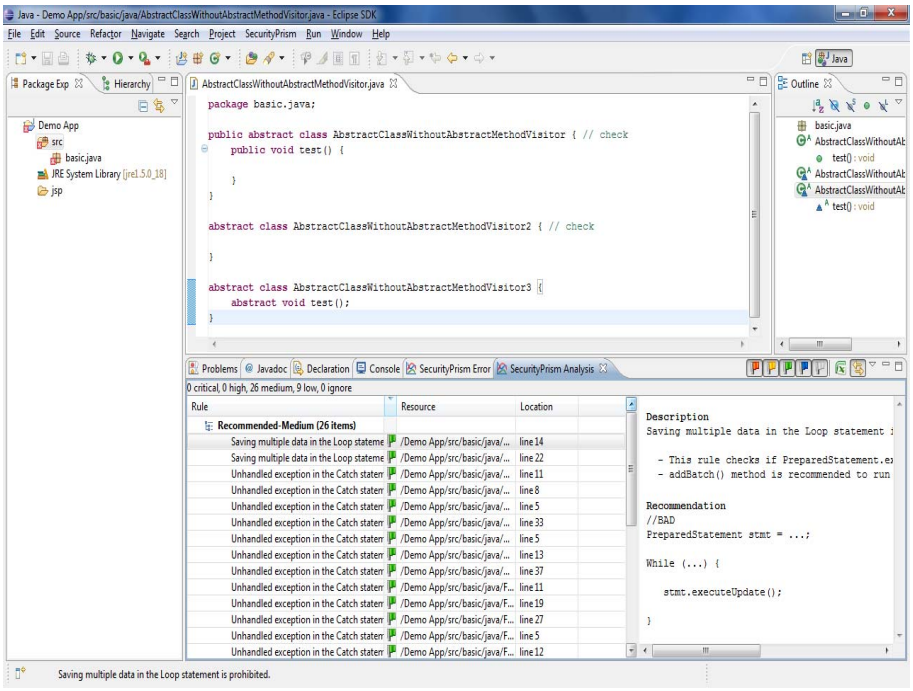
```
1001          int foo(int i) {
1002           int *x;
1003           int *y = malloc(sizeof(int));
1004           *y = i;
1005           *x = i;    /*** dangling pointer dereference ***/
1006           free(y);
1007           return *y;   /*** dangling pointer dereference ***/
1008           }
```

The line number 1005 is accessing to uninitialized pointer variable x. The line number 1007 is accessing to pointer variable y which is released by free(). CodePrism understands this kind of dangling pointer dereference.

Conclusion

Finding software defects at the initial development stage will reduce huge amount of cost. However, checking source codes of software with manual way relies on developers' experiences and needs large amount of times and costs.

CodePrism, advanced static analysis tool, enables you to find defects in your source code as early as possible. A key advantage of the tool is that it can be applied before coding is complete, saving quality assurance time and cost. IT organization can also improve maintenance efficiency by enforcing developer's coding standards compliance at the development stage.



GTONE's unique value is seamless integration between its own application governance solutions. Not likely other simple code inspection tools, CodePrism can be integrated with ChangeMiner, robust enterprise application mining tool, within GTONE's AppGovernance suite.

It means that application mining and code quality inspection can be conducted at once, which provides more comprehensive and multi-dimensional information on your application in single environment.

For more information about ChangeMiner, please visit to **http://www.gtonesoft.com.**

http://www.gtonesoft.com