ChangeMiner

CHANGEMINER WHITE PAPER

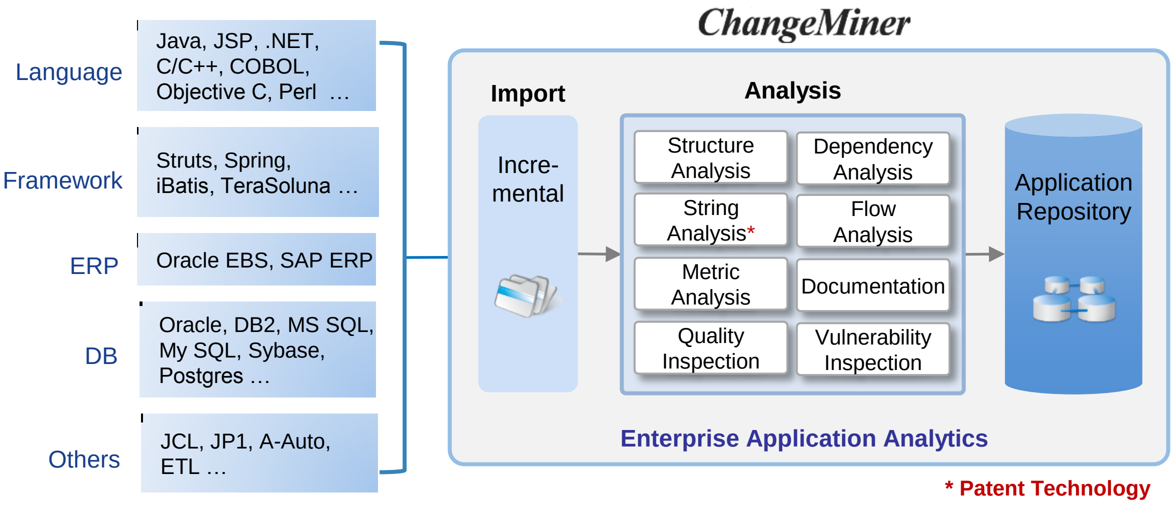
Robust string analysis engine enables the most accurate application change impact analysis!

GTONE
Governance Technology

What is ChangeMiner?

ChangeMiner is an application analytics tool that provides comprehensive insight into enterprise applications and databases. It enables organizations to improve development and maintenance productivity by delivering automated knowledgebase for complex applications using multiple languages and technologies.

ChangeMiner automatically imports and analyzes both source files and database schema based on patent technology to find the overall detail object level dependencies among them. Because ChangeMiner conducts everything based on fundamental user input configurations, user doesn't have to manually manage the object dependencies information.

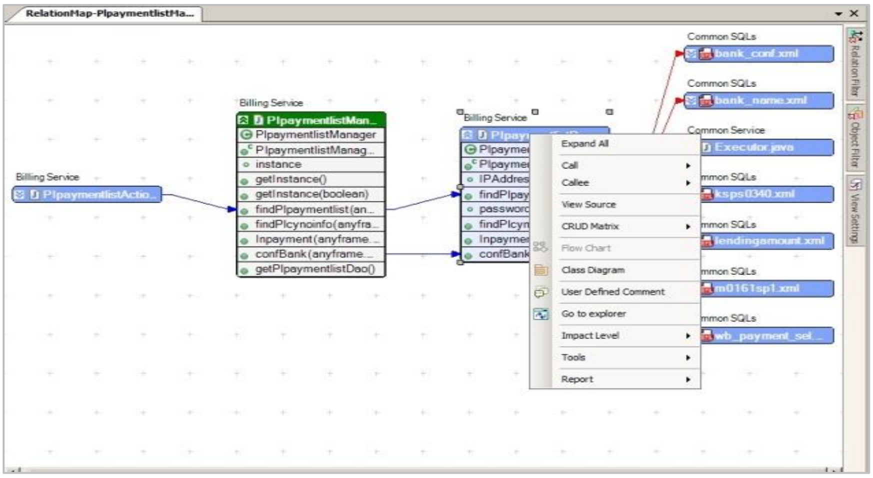


[Figure 1] Basic concept

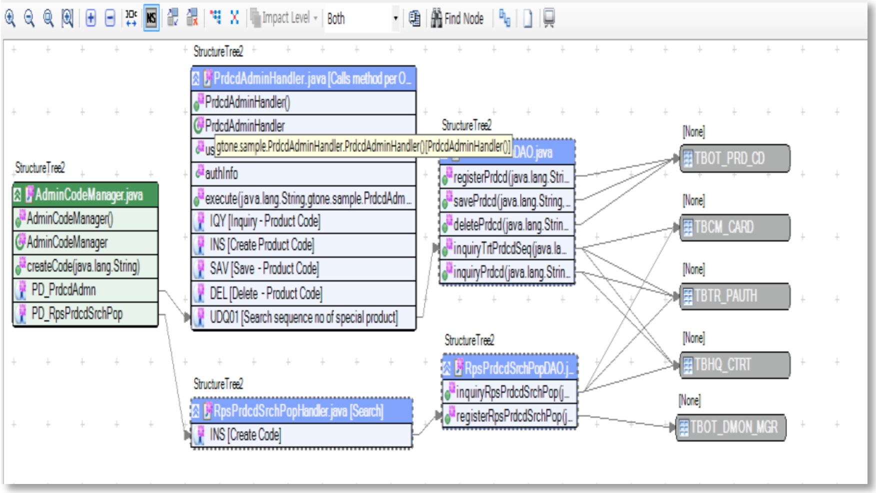
Information gained from ChangeMiner is shared through its application repository equipped with robust documentation feature. Application team can access the application knowledgebase anywhere via the Internet and get rapid understanding of how their applications are structured and dependent.

One of ChangeMiner's main usages is change impact analysis between application and database. For example, when a change is made in database schema, the potential consequences of the change need to be identified so that the application can be properly modified to maintain its integrity. With valuable analytics information such as end-to-end method call chain and CRUD[1] matrix, ChangeMiner provides highly productive change impact analysis or variable level root cause tracking of application failures to development team.

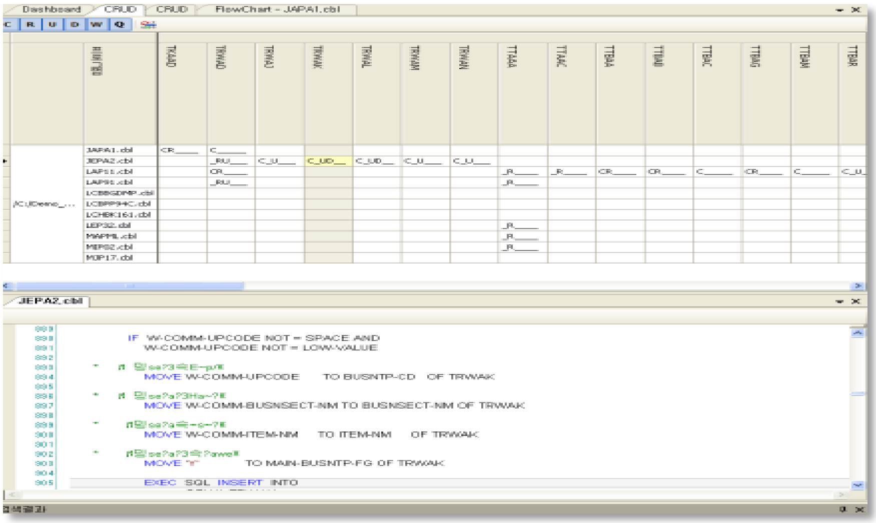
[1] CRUD (Create, Read, Update, Delete) matrix shows which database object is used by which program through SQL queries.



[Figure 2] Example of end-to-end method call graph


Impact Level
Both
Find Node
StructureTree2
AdminCodeManager.java
AdminCodeManager()
AdminCodeManager
createCode(java.lang.String)
PD_PrdcdAdmn
PD_RpsPrdcdSrchPop
StructureTree2
PrdcdAdminHandler.java [Calls method per O...
PrdcdAdminHandler()
PrdcdAdminHandler
gtone.sample.PrdcdAdminHandler.PrdcdAdminHandler()[PrdcdAdminHandler()]
authInfo
execute(java.lang.String,gtone.sample.PrdcdAdm...
IQY [Inquiry - Product Code]
INS [Create Product Code]
SAV [Save - Product Code]
DEL [Delete - Product Code]
UDQ01 [Search sequence no of special product]
StructureTree2
...DAO.java
registerPrdcd(java.lang.Stri...
savePrdcd(java.lang.String...
deletePrdcd(java.lang.Strin...
inquiryTrtPrdcdSeq(java.la...
inquiryPrdcd(java.lang.Strin...
StructureTree2
RpsPrdcdSrchPopDAO.j...
inquiryRpsPrdcdSrchPop(j...
registerRpsPrdcdSrchPop(j...
StructureTree2
RpsPrdcdSrchPopHandler.java [Search]
INS [Create Code]
[None]
TBOT_PRD_CD
[None]
TBCM_CARD
[None]
TBTR_PAUTH
[None]
TBHQ_CTRT
[None]
TBOT_DMON_MGR

[Figure 3] Example of dependencies by condition values



	[illegible]	TKAAD	TRWAD	TRWAJ	TRWAK	TRWAL	TRWAM	TRWAN	TTAAA	TTAAC	TTBAA	TTBAB	TTBAC	TTBAG	TTBAM	TTBAR
/C:/Demo_...	JAPA1.cbl	CR___	C____													
	JEPA2.cbl		_RU__	C_U__	C_UD_	C_UD_	C_U__	C_U__								
	LAP11.cbl		CR___						_R___	_R___	CR___	CR___	C____	CR___	C____	C_U_
	LAP91.cbl		_RU__						_R___							
	LCBBGDMP.cbl															
	LCBRP94C.cbl															
	LCHBK161.cbl															
	LEP32.cbl								_R___							
	MAPPIL.cbl								_R___							
	MEP82.cbl								_R___							
	MOP17.cbl															

JEPA2.cbl

```
           IF  W-COMM-UPCODE NOT = SPACE AND
               W-COMM-UPCODE NOT = LOW-VALUE

      *    ꞃ 덩se?3늑E~p/E
               MOVE W-COMM-UPCODE      TO BUSNTP-CD    OF TRWAK

      *    ꞃ 덩se?a?3Ha~?E
               MOVE W-COMM-BUSNSECT-NM TO BUSNSECT-NM OF TRWAK

      *     ꞃ덩se?a늑~s~?E
               MOVE W-COMM-ITEM-NM     TO ITEM-NM     OF TRWAK

      *     ꞃ덩se?a?3늑?awe
               MOVE 'Y'           TO MAIN-BUSNTP-FG OF TRWAK

               EXEC  SQL  INSERT  INTO
```

검색결과

[Figure 4] Example of CRUD Matrix

What makes the tool special?

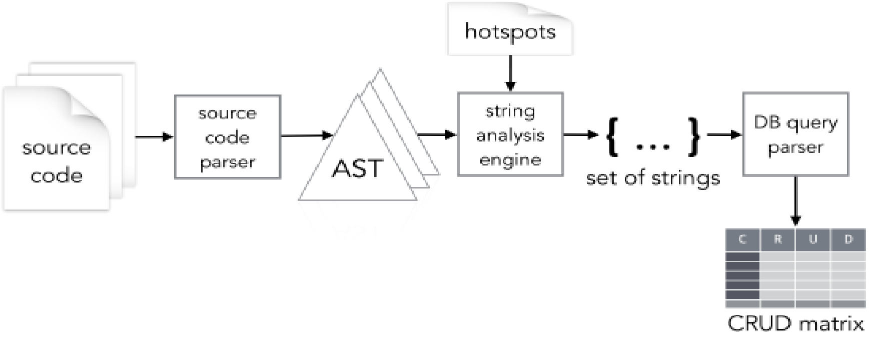
The Universal String Analyzer (USA) is the key technology that sets ChangeMiner apart from its competitors.

It is hard to accurately analyze applications communicating with databases through SQL queries that are mostly composed at run-time by concatenating literal and user-supplied strings. To overcome the technical obstacle, ChangeMiner uses not only lexical parsing technology which is widely used, but also special string analysis technology called Universal String Analyzer (patent).

The Universal String Analyzer embedded inside ChangeMiner is a static string-analysis engine intended to work for applications written in any programming language. It has an ability to accurately extract dynamic strings in source code without running program based on the following advanced technologies:

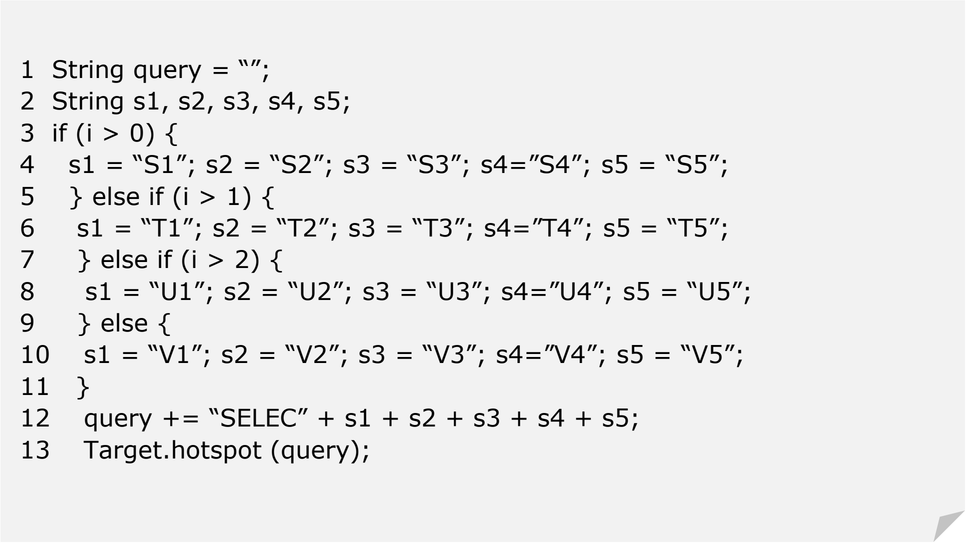
- Inter-procedural path-sensitive analysis for reducing false positive
- Deep data structure analysis for high accuracy

The structure of USA is shown in Figure 5. The analysis engine takes an AST representation of a source code and generates a graph containing control-flow and data-flow. The graph is then simplified into a string graph that can be traversed to produce a set of query strings.



[Figure 5] The architecture of string-analysis engine

Because the engine is designed to support path-sensitive string analysis, each string expression contains path information about whether it is in true branch or false branch. For example, consider the code segment in Figure 6, it is made-up but simple enough to show why path-sensitive analysis is desirable. In lines 3 through 11, variables s1, s2, s3, s4, and s5 are assigned different string values according to given conditions. They are then later concatenated and assigned to a variable query. The number of different string queries line 12 can have is obviously four. However, since path-insensitive analysis does not use path information, it typically gives all combinations 1,024 (= 4^5) possible strings, resulting in that many false positives, not to mention the waste of time and space.

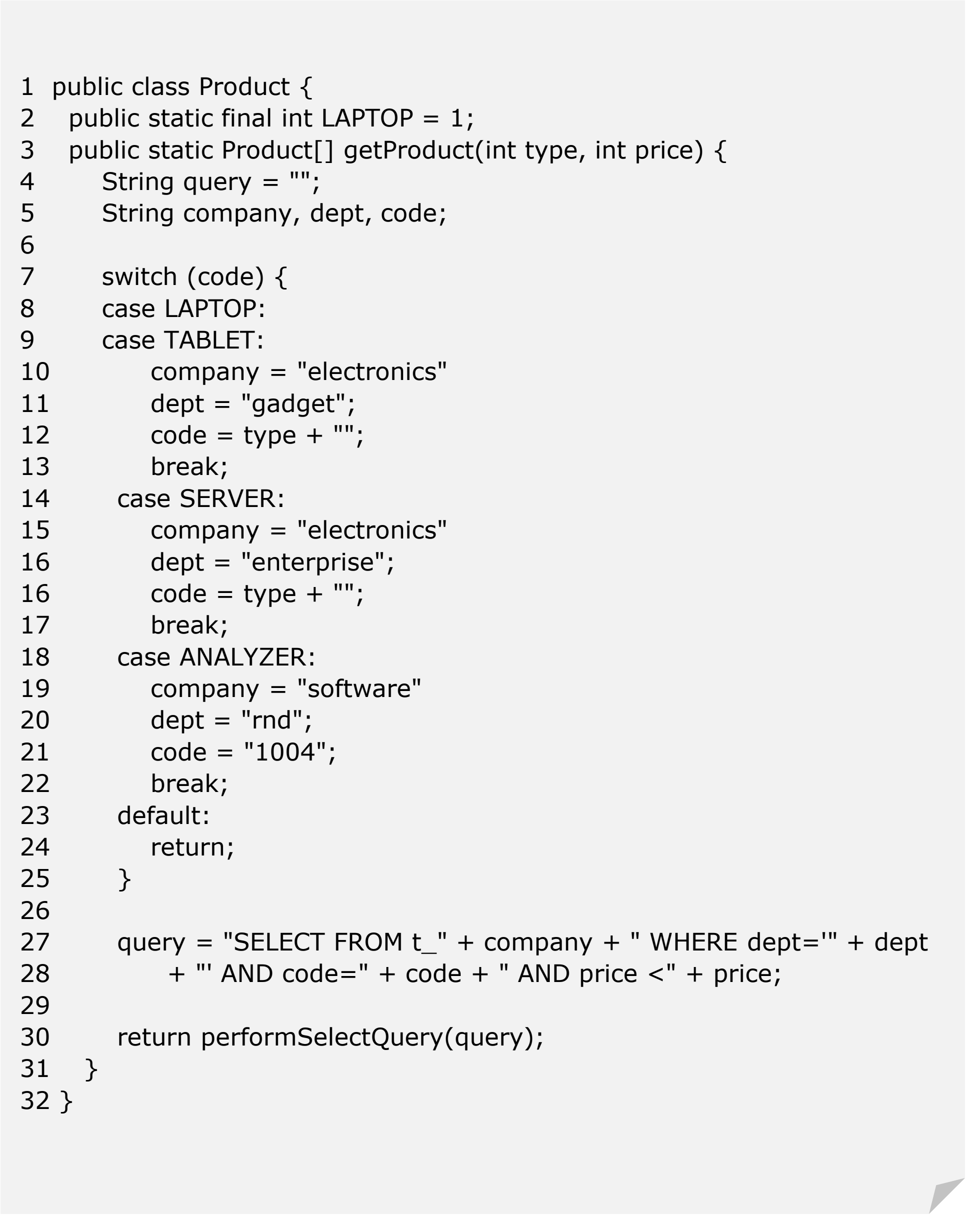


```
String query = "";
String s1, s2, s3, s4, s5;
if (i > 0) {
  s1 = "S1"; s2 = "S2"; s3 = "S3"; s4="S4"; s5 = "S5";
  } else if (i > 1) {
   s1 = "T1"; s2 = "T2"; s3 = "T3"; s4="T4"; s5 = "T5";
   } else if (i > 2) {
    s1 = "U1"; s2 = "U2"; s3 = "U3"; s4="U4"; s5 = "U5";
   } else {
    s1 = "V1"; s2 = "V2"; s3 = "V3"; s4="V4"; s5 = "V5";
   }
    query += "SELEC" + s1 + s2 + s3 + s4 + s5;
    Target.hotspot (query);
```

[Figure 6] An example showing the need of path-sensitive analysis

Figure 7 shows a real world code example. In this example, 4 variables - company, dept, code, price – are used to create a query. Out of the 4 variables, three variables – company, dept, code – through three execution paths will get three possible strings. When general path-insensitive analysis is used, estimated query's possible strings are - execution paths ^ variables = 3^3 = 27.

When path-sensitive analysis is used, however, three variables – company, dept, code- through three execution paths will be assigned one possible string. Estimated number of possible query strings is number of execution paths, in this case, three; therefore, possible string's proportion is 9:1. Which means path-sensitive analysis will have 9 times the accuracy than path-insensitive analysis.

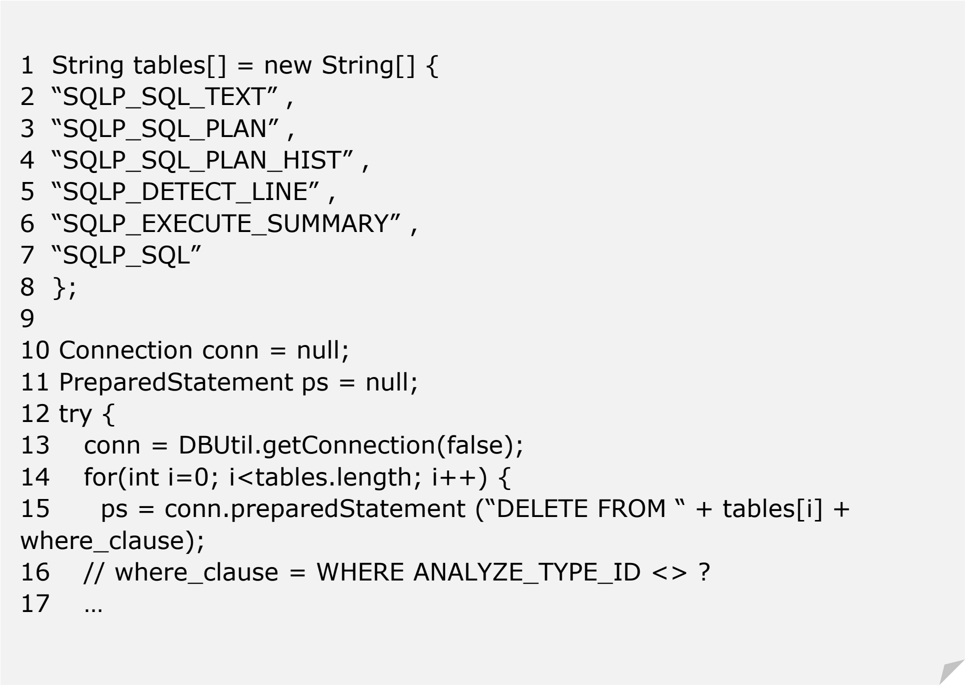


```
public class Product {
  public static final int LAPTOP = 1;
  public static Product[] getProduct(int type, int price) {
      String query = "";
      String company, dept, code;

      switch (code) {
      case LAPTOP:
      case TABLET:
            company = "electronics"
            dept = "gadget";
            code = type + "";
            break;
        case SERVER:
            company = "electronics"
            dept = "enterprise";
            code = type + "";
            break;
        case ANALYZER:
            company = "software"
            dept = "rnd";
            code = "1004";
            break;
        default:
            return;
        }

        query = "SELECT FROM t_" + company + " WHERE dept='" + dept
              + "' AND code=" + code + " AND price <" + price;

        return performSelectQuery(query);
    }
}
```

[Figure 7] Sample code od dynamic query

When creating a query using real enterprise application, there are a lot of possible variables and execution paths which can be used. If there are two execution paths and ten variables in use, with path-insensitive analysis it will result in 2^10=1024 possible strings and with path-sensitive analysis it will result in 2 possible strings. With the path proportion 512:1, path-insensitive analysis needs to analyze over 500 unnecessary strings to get the necessary query number. This makes path-sensitive analysis extremely valuable. ChangeMiner’s embedded string analysis engine supports path-sensitive analysis

Let's take another simple example of string analysis. It shows engine's deep structure analysis capability.



```
String tables[] = new String[] {
"SQLP_SQL_TEXT" ,
"SQLP_SQL_PLAN" ,
"SQLP_SQL_PLAN_HIST" ,
"SQLP_DETECT_LINE" ,
"SQLP_EXECUTE_SUMMARY" ,
"SQLP_SQL"
};

Connection conn = null;
PreparedStatement ps = null;
try {
  conn = DBUtil.getConnection(false);
  for(int i=0; i<tables.length; i++) {
    ps = conn.preparedStatement ("DELETE FROM " + tables[i] +
where_clause);
  // where_clause = WHERE ANALYZE_TYPE_ID <> ?
  …
```

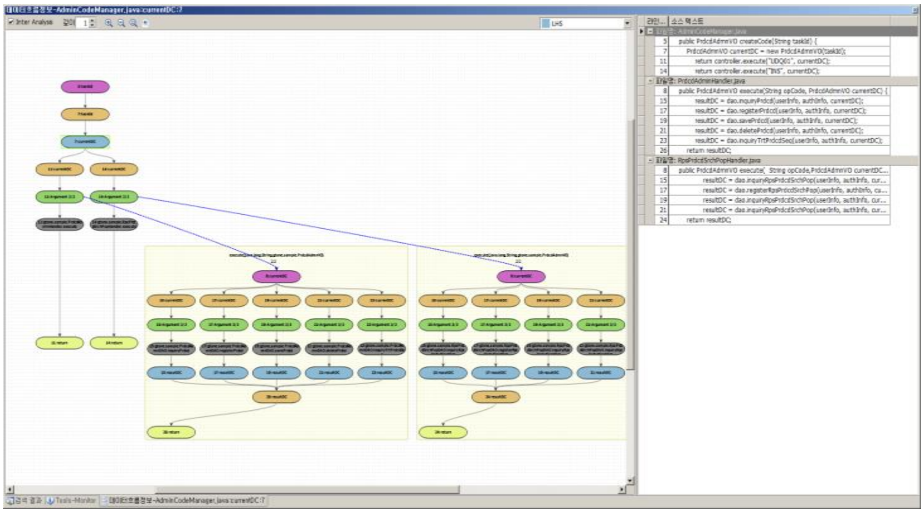
[Figure 8] Sample code of structured data

If an analysis engine doesn't understand the static array in the example above, it will just give user the following insufficient string:
"DELETE FROM null WHERE ANALYZE_TYPE_ID <> ?"

ChangeMiner's engine, however, understands how to analyze static string array, which will give user the following strings (dynamic queries) exactly:
"DELETE FROM SQLP_SQL_TEXT WHERE ANALYZE_TYPE_ID <> ?"
"DELETE FROM SQLP_SQL_PLAN WHERE ANALYZE_TYPE_ID <> ?"
"DELETE FROM SQLP_SQL_PLAN_HIST WHERE ANALYZE_TYPE_ID <> ?"
"DELETE FROM SQLP_DETECT_LINE WHERE ANALYZE_TYPE_ID <> ?"
"DELETE FROM SQLP_EXECUTE_SUMMARY WHERE ANALYZE_TYPE_ID <> ?"
"DELETE FROM SQLP_SQL WHERE ANALYZE_TYPE_ID <> ?"

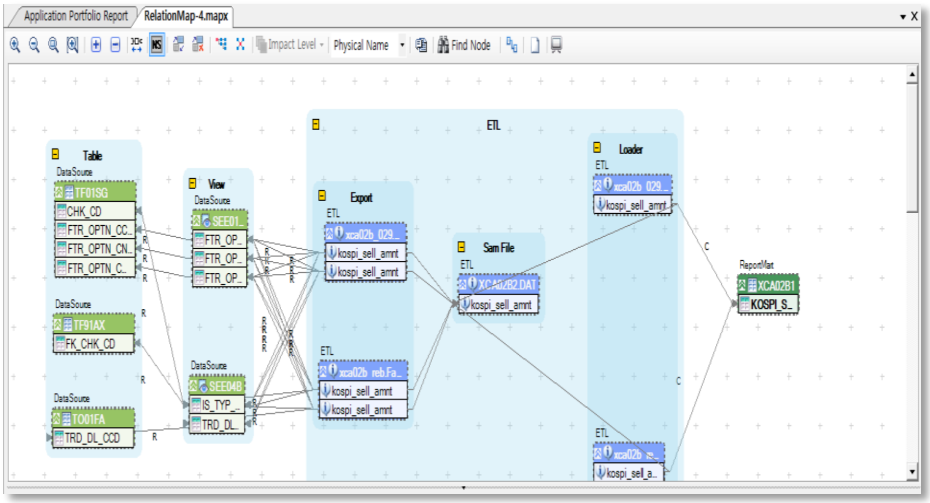
With this kind of deep structure data analysis accuracy, users will not miss dependencies between programs and database tables.

ChangeMiner also offers unique and valuable features such as inter-procedural data flow analysis to track application failure root cause, various Structure Data Tree APIs which can be used to extract business point-of-view information, data lineage visualization, semantic code inspections and more.



라인...	소스 텍스트
	파일명: AdminCodeManager.java
5	public PrdcdAdminVO createCode(String taskId) {
7	PrdcdAdminVO currentDC = new PrdcdAdminVO(taskId);
11	return controller.execute("UDQ01", currentDC);
14	return controller.execute("INS", currentDC);
	파일명: PrdcdAdminHandler.java
8	public PrdcdAdminVO execute(String opCode, PrdcdAdminVO currentDC) {
15	resultDC = dao.inquiryPrdcd(userInfo, authInfo, currentDC);
17	resultDC = dao.registerPrdcd(userInfo, authInfo, currentDC);
19	resultDC = dao.savePrdcd(userInfo, authInfo, currentDC);
21	resultDC = dao.deletePrdcd(userInfo, authInfo, currentDC);
23	resultDC = dao.inquiryTrtPrdcdSeq(userInfo, authInfo, currentDC);
26	return resultDC;
	파일명: RpsPrdcdSrchPopHandler.java
8	public PrdcdAdminVO execute(String opCode,PrdcdAdminVO currentDC...
15	resultDC = dao.inquiryRpsPrdcdSrchPop(userInfo, authInfo, cur...
17	resultDC = dao.registerRpsPrdcdSrchPop(userInfo, authInfo, cu...
19	resultDC = dao.inquiryRpsPrdcdSrchPop(userInfo, authInfo, cur...
21	resultDC = dao.inquiryRpsPrdcdSrchPop(userInfo, authInfo, cur...
24	return resultDC;

[Figure 9] Example of inter-procedural data flow analysis



[Figure 10] Example of data lineage

Conclusion

The Universal String Analysis engine is minimizing false positives and enabling rapid analysis by only using branch’s path information, avoiding complicated value analysis. Its advanced technology makes change impact analysis more accurate than ever before. Our analysis is especially effective when conditionals are nested, its branches assign multiple string variables, and the variables are concatenated afterwards. Even when false positives are inevitable, performance is generally improved in comparison to ones without path-sensitive analysis.

One of mega IT service companies in Japan saved time 75% by using ChangeMiner’s impact analysis. Before using ChangeMiner, developers spent 95 minutes finding and documenting change impact scope of a particular table column across the overall architecture. With ChangeMiner’s end-to-end call chain and CRUD information, developers can finish the same amount of work within just 24 minutes.

http://www.gtonesoft.com

http://www.changeminer.com

Excellence in application governance

Restricted Rights Legend