A-max(Static Code Analysis)

FUTURE SYSTEMS

제품소개서

인프라소프트(주)

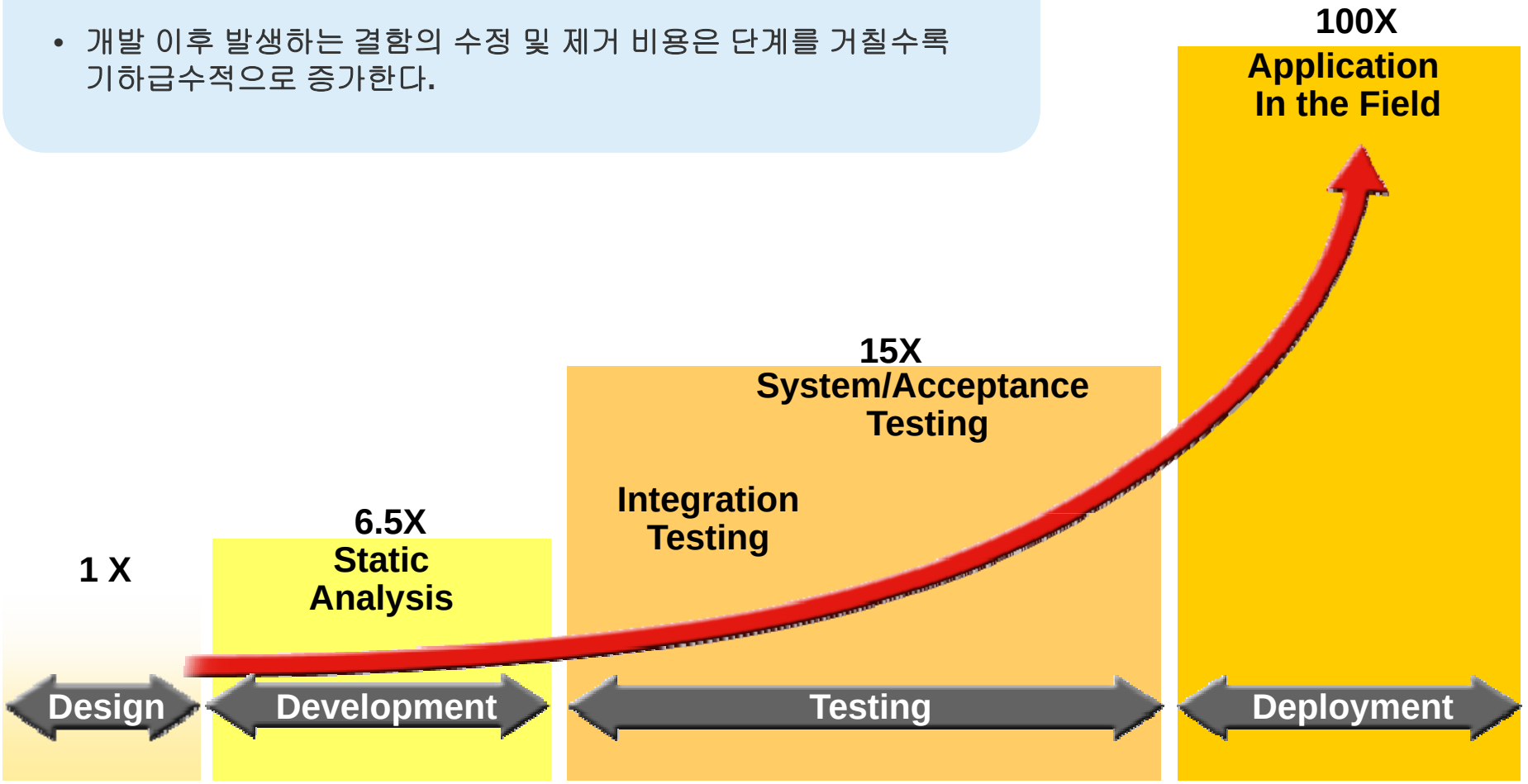
목 차

1. 배경

1. ASQ(Automated Software Quality)

Application Maximizer A-max

- 모든 소프트웨어는 최소한의 결함 요소를 항상 내포하고 있다.
- 개발 이후 발생하는 결함의 수정 및 제거 비용은 단계를 거칠수록 기하급수적으로 증가한다.



Source IDC and IBM Systems Sciences Institute

Application Maximizer A-max

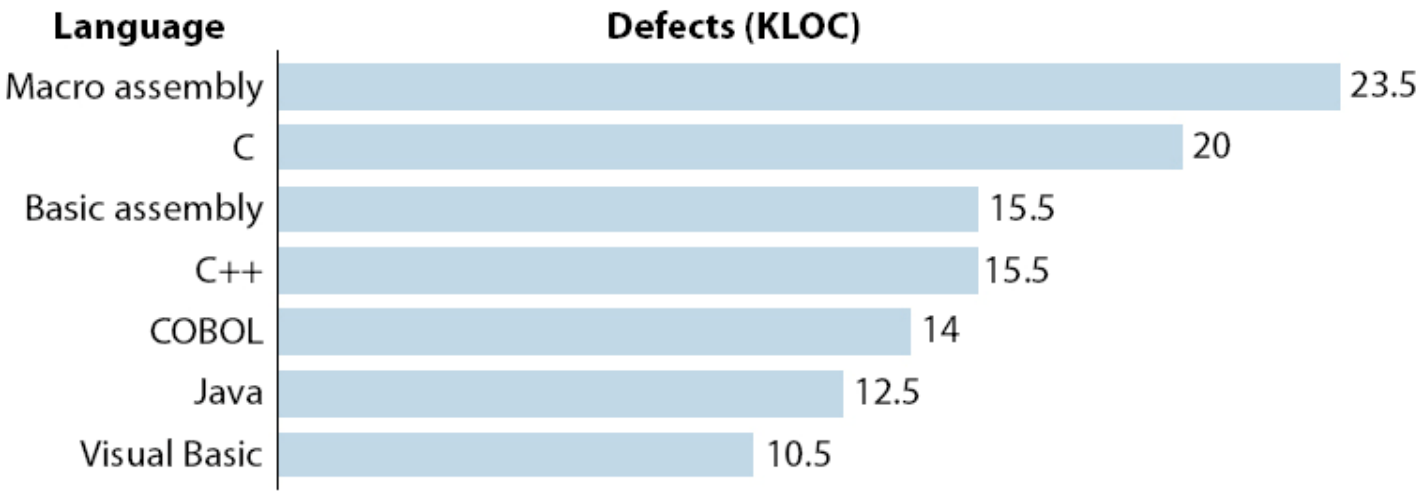
The Cost of QA

Role	Time to Correct	Cost per Defect	Average Costs	Difference
QA @ $23.43/hr	1.5 hrs	$35.15	4,000 * 35.15 = $140,600.00	Dev. Cycle contained inspections and testing; defects fixed before production
Dev @ $31.25/hr	9 hrs	$281.25	3,000 * 281.25 = $843,750.00	Dev. Cycle contained no inspections; detection in beta testing and post production support

Source: Giga Information Group

- **Average Cost**
  - ✓ 일반적인 **Large Application**에서는 약 **5,000**개의 결함요소 존재
  - ✓ **QA**를 통해서는 약 **4,000**개**(80%)**의 결함 요소 발견
  - ✓ **QA**를 배제한 개발에서는 약 **3,000**개**(60%)**의 결함 요소 발견

Figure 1 Defects Per KLOC Per Language



Source: T. Capers Jones, "Estimating Software Costs."

Source: Forrester Research, Inc.

II. **A-max**

Application Maximizer A-max

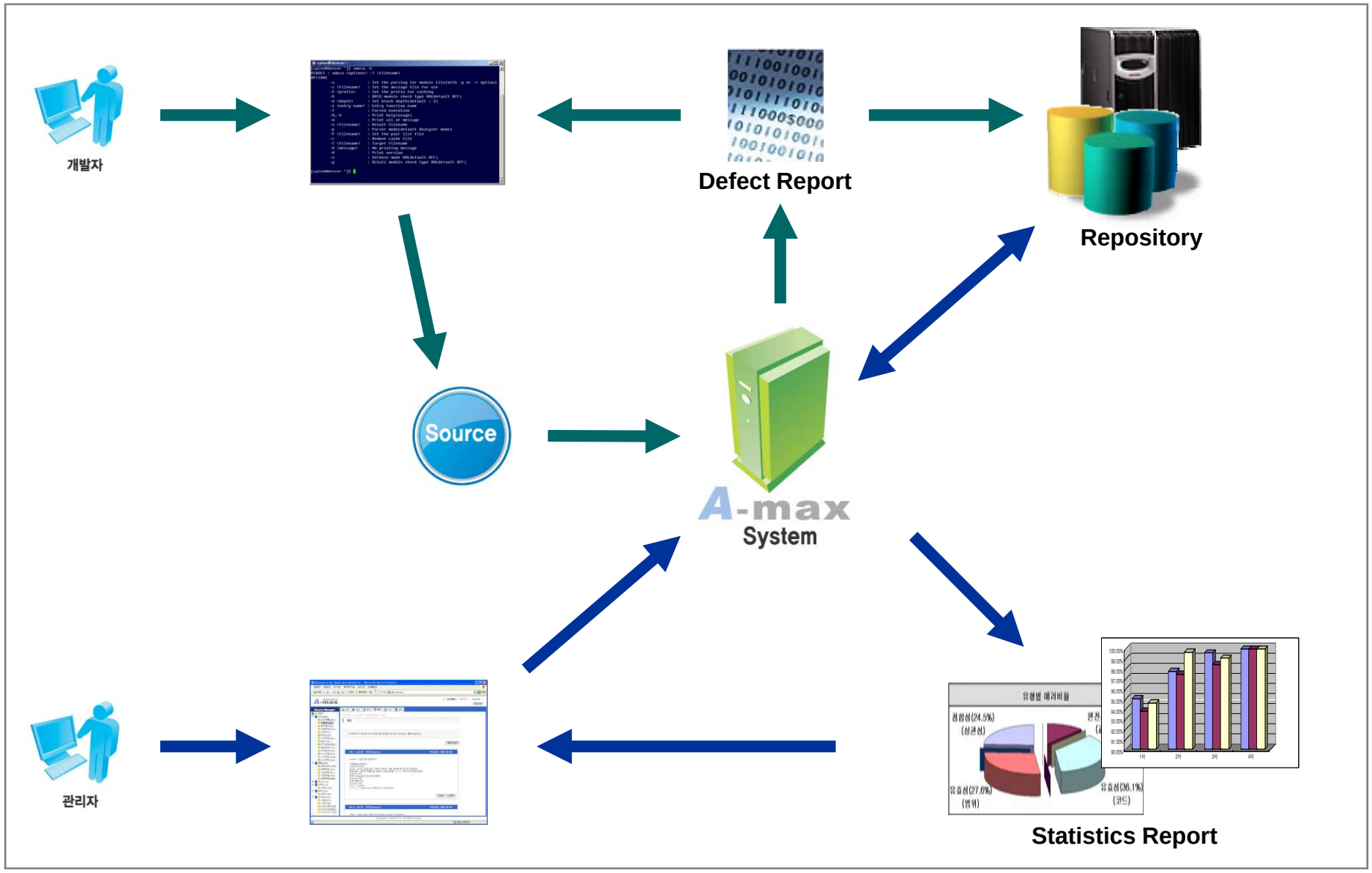
A-max

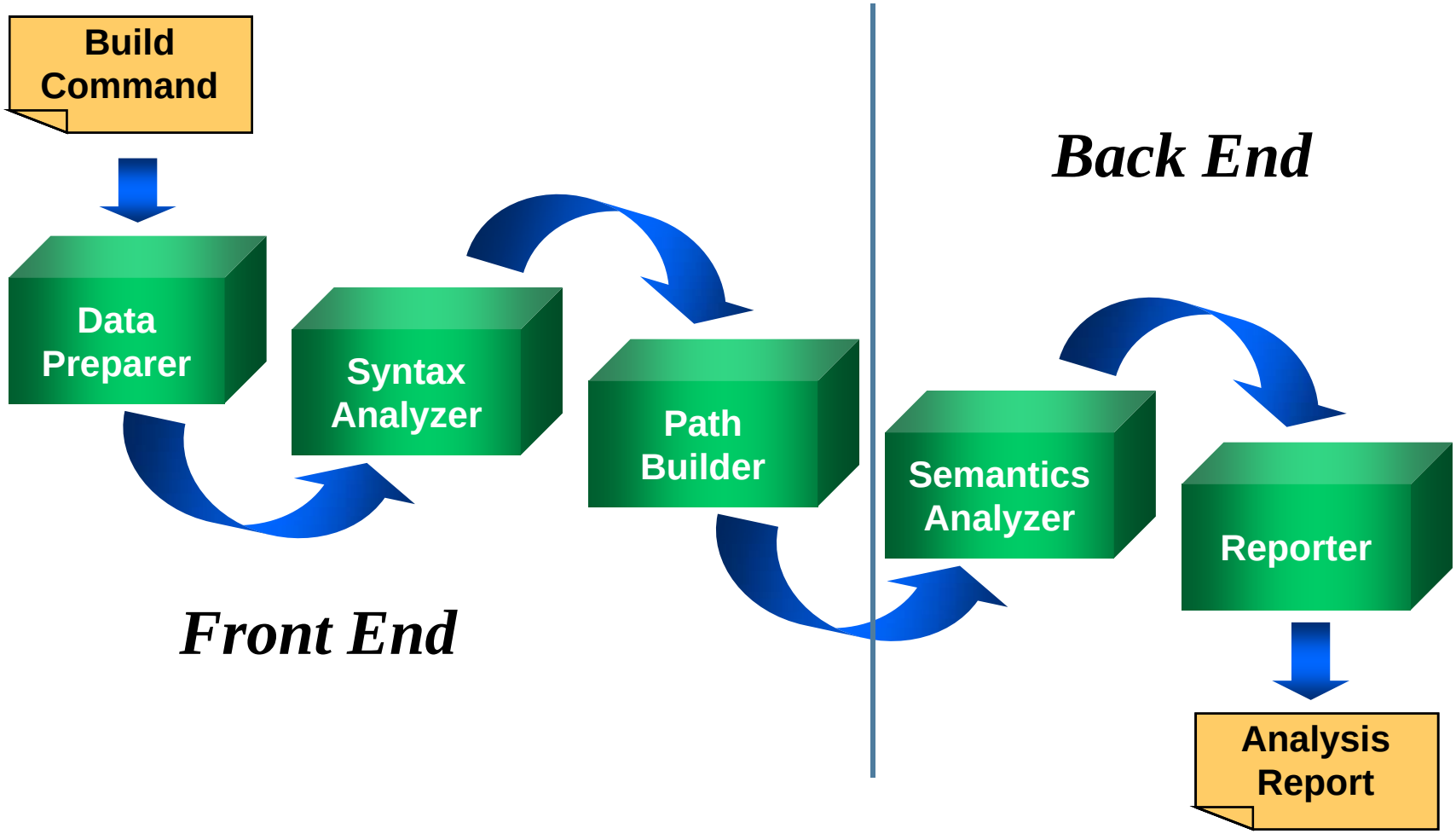
A-max S.A.
(Source Analyzer)

- Source code의 논리적 오류에 대한 검사
- 코드 내에서 사용된 리소스, 트랜잭션 및 SQL 정보에 대한 검사 및 보고
- 미들웨어 API 사용에 대한 검사 및 보고
- Reliability와 performance 향상을 위한 정보 및 통계 자료 제공
- Framework과 연동하기 위한 분석 방법 지원

A-max R.C.
(Rule Checker)

- 기 정의된 코딩 표준 및 규칙에 위배되는 코드에 대해서 검사
- 기본적인 코딩 표준 사항 제공 및 사용자 정의 코딩 표준 정책 수립 및 적용
- 코드 내에서 선언 및 사용된 모든 유형에 대하여 검사 및 보고
- 결과에 대해서 다양한 방법으로 보고(Display, File - .xml, .txt, .xls, etc.)


개발자
Defect Report
Repository
Source
A-max
System
관리자
Statistics Report


Build Command
Data Preparer
Syntax Analyzer
Path Builder
Semantics Analyzer
Reporter
Analysis Report
Front End
Back End

Application Maximizer A-max

정적 코드 분석

- 자원의 누수 검사**(Memory, File, Database, Cursor, etc.)**
- 소스 코드상의 정확한 결점 위치 보고
- **Unreachable Code** 사용 검사
- 소스 코드의 변경 없이 분석
- **Embedded SQL** 사용시의 트랜잭션의 명시적 사용 검사
- 코드에서 사용된 **SQL** 문장에 대한 메타 데이터 보고
- 미들웨어 사용시의 **TP/XA** 사용 모드 검사
- 모듈별 **In-Depth Analysis** 지원

표준 스타일 분석

- 금지 함수의 사용 여부 검사
- 파일명 및 함수명의 명명 규칙 준수 여부 검사
- 매크로 및 구조체 선언시의 명명 규칙 준수 여부 검사
- 사용자가 정의한 코딩 표준에 대한 검사 지원
- 경고 레벨 및 수준에 대한 사용자 정의 지원

특 징

1. 개발자 관리자 모두를 위한 솔루션

- 개발자는 컴파일 과정에서 자신의 소스 코드에 대한 검증
- 관리자는 해당 소스 코드를 이용한 애플리케이션 적용시 발생 가능 오류에 대한 점검

2. 소스 코드만으로 오류 분석 가능

- 별도의 추가적인 작업 없이 개발자가 작성한 소스 코드만으로 표준 검사 및 오류 검사 작업 진행
- 분석 결과에 대해서는 다양한 방법(Display, File - .xml, .txt, .xls, etc.)을 통하여 보고

3. 통합적인 분석 가능

- 모듈 및 여타 관련 정보를 포함하는 통합적인 분석 가능
- Framework과 연동된 기능에 대한 추가적인 정보 제공을 통하여 분석

4. 기존의 관리 솔루션과의 연계 가능

- 기존의 형상 관리, 영향 분석 및 여타 관리 솔루션과의 연계 방안 지원

Application Maximizer A-max

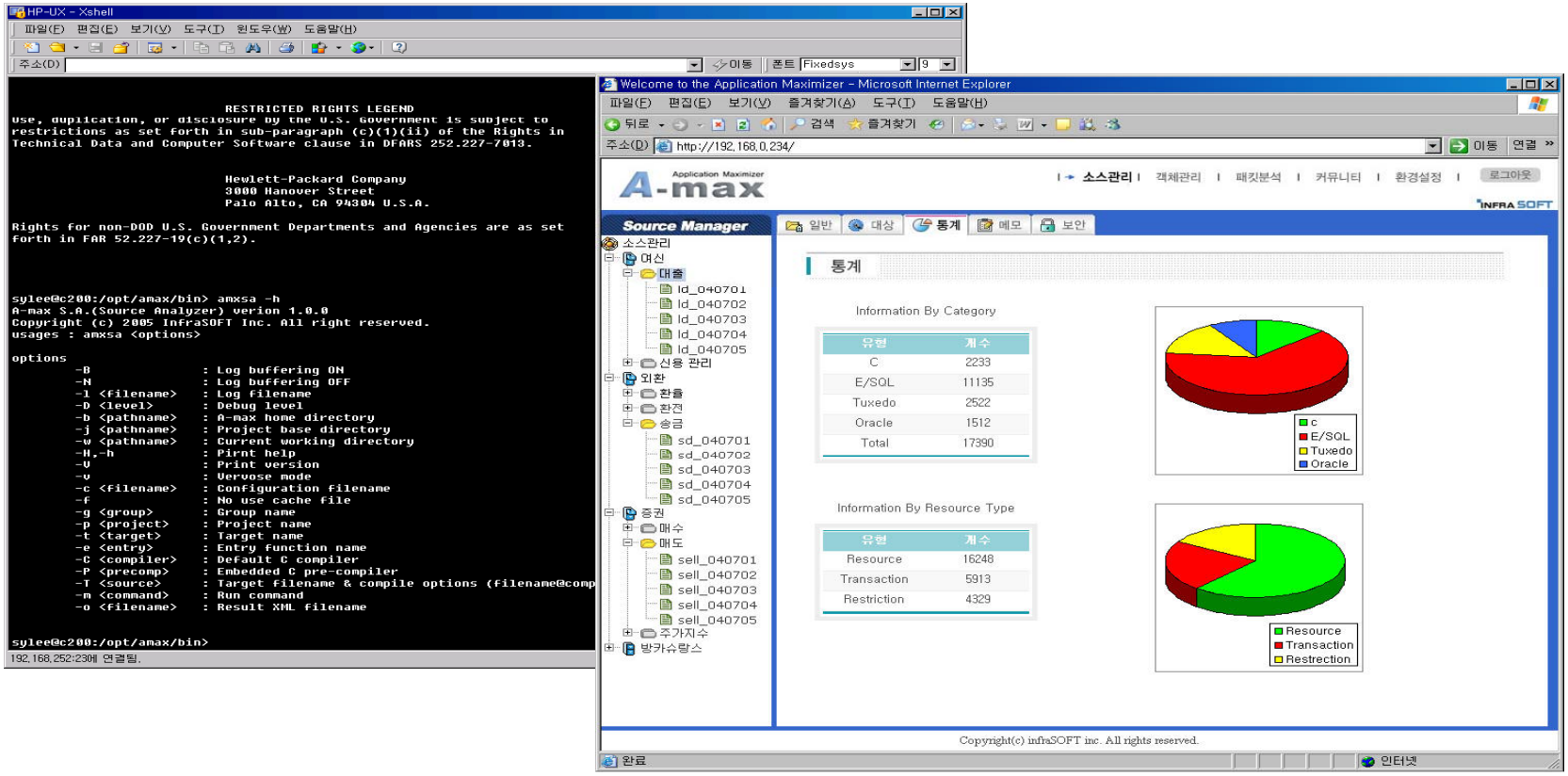
기 대 효 과

소스 코드의 표준화	1. 소스 코드 작성의 표준화 정책의 수립을 통하여 일관된 소스 코드의 작성 2. 가독성을 높이고, 향후 수정 및 보완 작업시 효율적인 작업이 가능
시스템 안정성	1. 소스 코드 레벨에서 위험요소 제거를 통한 애플리케이션의 안정성 확보 2. 안정된 애플리케이션의 운영을 통한 시스템 안정성 도모
개발 속도 향상	1. 오류 발견에 필요한 시간 및 인력에 대한 절감을 통하여 개발 속도 향상 2. 가독성 높은 소스 코드를 통한 수정 및 보완 작업 용이
중앙집중적 관리	1. 소스 코드에 대한 일관화된 관리 및 현황 관리 용이
품질향상 및 유지관리	1. 소스 레벨에서의 오류 가능성 대처를 통한 애플리케이션의 품질 향상 기여 2. 향후 애플리케이션의 업그레이드시 소스 코드에 대한 수정 및 변경 용이

5. 사용자 환경

Application Maximizer A-max

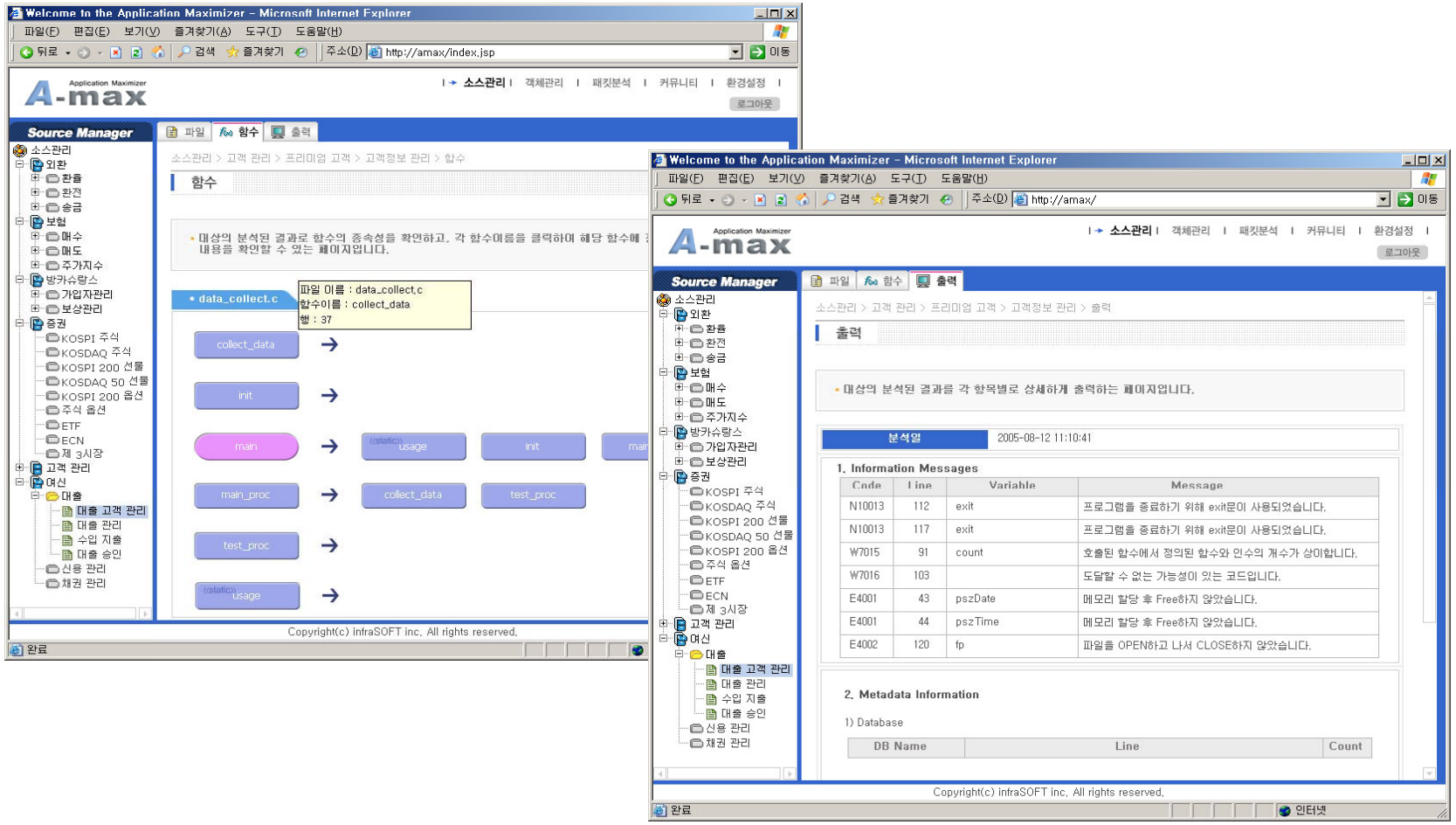
- **GUI** 및 **Command Line Interface**를 이용한 개발자 및 관리자를 위한 다양한 사용 방법 제공
- **Compiling-Time** 정도의 분석 시간으로써 분석 및 결과 제공 가능
- 분석된 소스에 대한 통합적인 관리 및 분석 통계 자료 제공을 통하여 일관된 관리 방법 제공



5. 사용자 환경 - 계속

Application Maximizer A-max

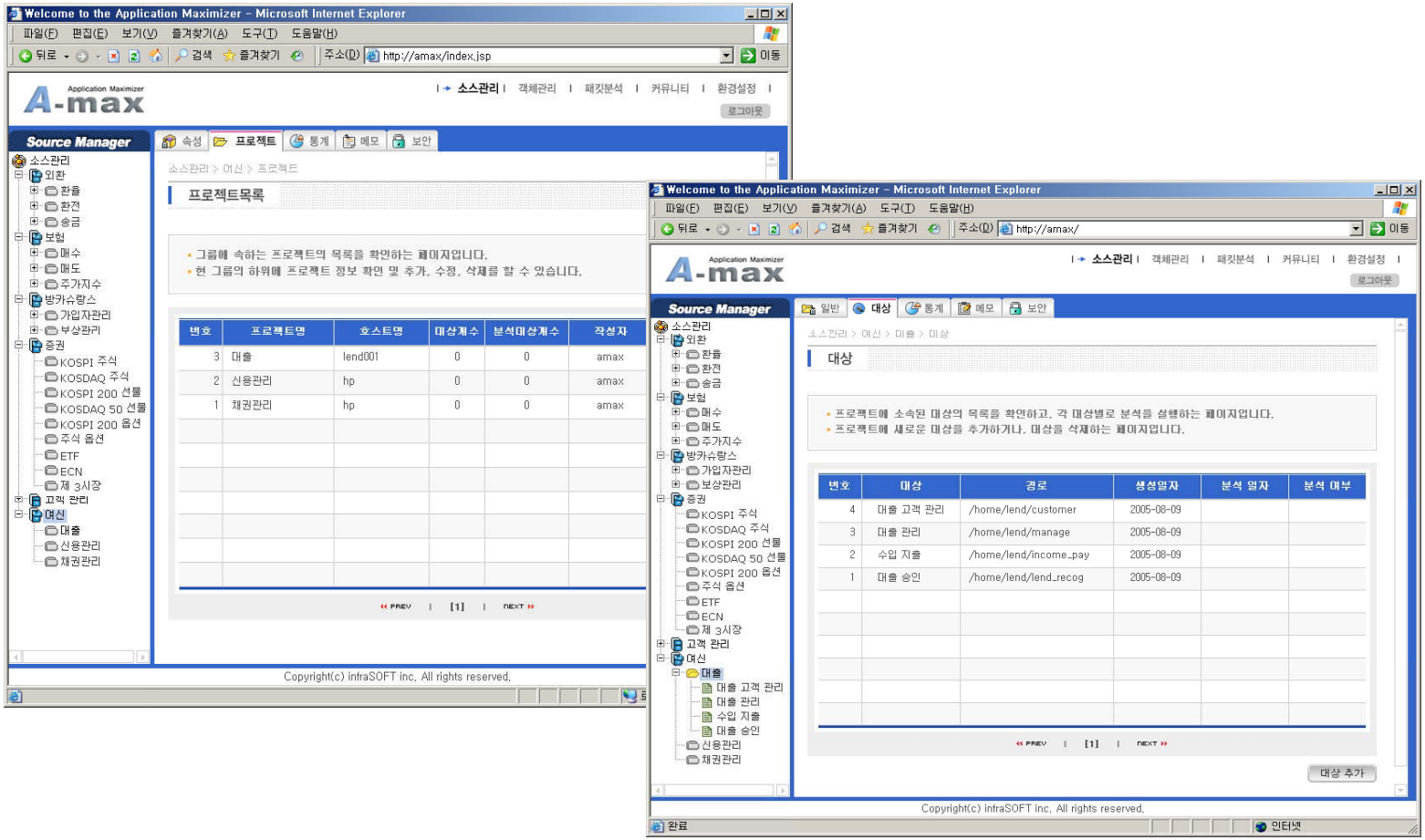
- 대상에 대한 분석 결과와 **DB** 및 **Queries**사용 정보와 같은 부가적인 정보에 대해서 구분하여 리포팅
- 분석결과의 **GUI** 결과로써 함수의 종속성 관계에 대해서 도식화하여 리포팅



5. 사용자 환경 - 계속

Application Maximizer A-max

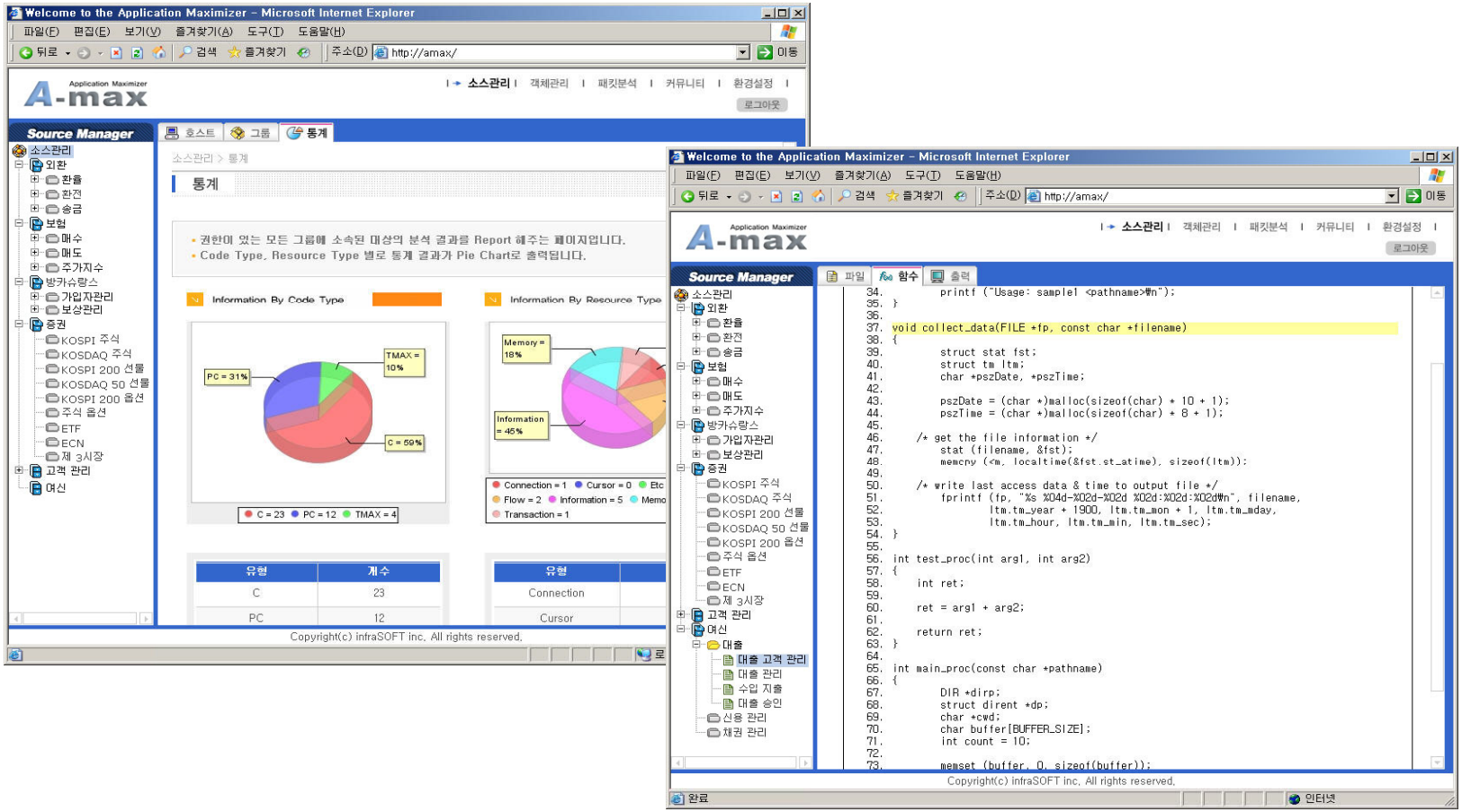
- 대상 소스의 성격, 개발팀 및 부서 등에 따라 **Grouping**하여 구분하여 관리 가능
- 대상 소스 및 서비스 등과 관련된 라이브러리 소스에 대해서도 일관되게 관리 가능



5. 사용자 환경 - 계속

Application Maximizer A-max

- **Group** 및 **Project**와 같이 **Grouping**된 대상에 대해서 **Code Type**별 **Resource Type**별 통계 결과를 리포팅
- **Client**환경에서 직접 리포팅된 대상에 대한 소스 파일을 **Viewing** 가능



Application Maximizer A-max

Check Items

1. Unreleased Memory
2. Unclosed File
3. Unclosed Cursor
4. Unclosed Connection
5. Uncommitted Transaction
6. Unreachable Code
7. Cursor/Prepare Declaration in a Loop
8. Unused Cursor and Prepare
9. DB Query not in a transaction
10. DB Connection with no queries
11. Functions with an incorrect number of parameters
12. XA Operations in Non-XA mode or Vice Versa

Case 1. Unreleased Memory

661, 662라인에서 할당한 변수에 대해 **702**라인에서 정상적으로 **free()**를 하고 함수를 **return**하지만, **732**라인에서는 해제하지 않고 **return**한다.

- ✓ 메모리 누수로 인하여 실행 속도가 저하된다.
- ✓ 서비스 다운 및 기능 정지 등의 요인이 될 수 있다

A-max

```
      ...
661: in  = (IN  *)malloc( sizeof ...);
662: out = (OUT *)malloc( sizeof ...);
      ...
690: if (ret != 0) {
693:     free((char *) in) ;
694:     free((char *) out) ;
          ...
701:     error(err_code, msg_hdr,...);
702:     return FAIL;
703: }
      ...
724: if ( sqlca.sqlcode != 0 ) {
726:     sprintf(errmsg->msg2,“Code: %d\n", …);
727:     error_mesg(errmsg, 1
728:         , “Open cursor error"
729:         , errmsg->msg2
730:         , "");
731:     error(err_code, msg_hdr,...);
732:     return FAIL;
733: }
      ...
3498: void error(ecode, m_hdr, emsg,…)
3504: {
          ...
3557:     return ecode;
3558: }
```

Case 2. Unclosed File

169, 184, 199라인에서 각각 해당 파일을 **open**하고, **259**라인에서 **SQL error** 발생할 경우와 **316**라인에서 **error** 발생할 경우 **close**하지 않고 **return**한다.

- ✓ 파일에 대해서 접근이 거부된다.
- ✓ 파일의 자료 일부가 상실되거나, 파일 자체가 훼손 될 수 있다.
- ✓ 운영체제에서 **Open** 파일의 개수를 제한함에 따라 최종적으로는 파일을 **Open**하는데 **error**가 발생할 수 있다.

A-max

```
54: int file_open_func()
55: {
        ...
85:     FILE *fp_0, *fp_1, *fp_2;
        ...
169:    if ((fp_0 = fopen(fname_0, "w")) != NULL)
        ...
184:    if ((fp_1 = fopen(fname_1, "w")) != NULL)
        ...
199:    if ((fp_2 = fopen(fname_2, "w")) != NULL)
        ...
251:    if(sqlca.sqlcode != ORA_SUCCESS)
252:    {
                ...
259:            return ret;
260:    }
        ...
308:    if (ret != SUCCESS )
309:    {
                ...
316:            return ret;
317:    }
        ...
381:    fclose(fp_0 );
382:    fclose(fp_1 );
383:    fclose(fp_2 );
        ...
421:    return ret;
422: }
```

Case 3. Unclosed Cursor

307라인에서 커서를 **open**하고 **407**라인에서 **close**하지만**, 337**라인에서 **Fetch error**가 발생할 경우와 **401**라인에서 **error**가 발생할 경우 **close** 하지 않고 **return**한다.

- ✓ 자원 누수로 인하여 실행 속도가 저하된다.
- ✓ 서비스 다운 및 기능 정지 등의 요인이 될 수 있다.

A-max

```
    ...
307: EXEC SQL OPEN C_INF_CURSOR;
308:
309: if (sqlca.sqlcode != ORA_SUCCESS) {
310:    fprintf(stderr, "Found error to open cursor!");
311:    return FAIL;
312: }
        ...
320:    EXEC SQL FETCH C_INF_CURSOR
321:                 INTO    :C_STR_NAME;
        ...
        else if (sqlca.sqlcode !=  ORA_SUCCESS) {
            ...
337:        return RtnStatus ;
338:    }
    ...
393:    if (rt = setInfoTail(C_STR_NAME->arr, ...)
394:        != SUCCESS
395:    ) {
            ...
401:        return ret_status;
402:    }
    ...
407: EXEC SQL CLOSE C_INF_CURSOR;
    ...
```

Case 4. Unclosed Connection

96라인의 **DB connection**에 대해서 해당 프로그램은 명시적으로 **disconnection**을 하지 않는다.

✓ **Database Connection**을 종료하지 않고 프로그램을 종료하게 되면, 진행 중이던 트랜잭션이 **Rollback**된다.

A-max

```
      ...
50: main(int argc, char *argv[])
51: {
      ...
96:  EXEC SQL connect to 'call_center';
97:    if (SQLCODE)
98:    {
99:       err = set_errmsg();
100:      printErr(Datbase connection failure");
101:      exit (err);
102:   }
      ...
305:  EXEC SQL commit work;
306:   if (SQLCODE)
307:   {
308:      err = set_errmsg();
309:      printErr("Could not update user info.!");
310:      exit (err);
311:   }
      ...
313:   fclose(f_ctrl);
314:
315:   process_success();
316:   exit(0);
317: }
     ...
```

Case 5. Uncommitted Transaction

218라인에서 시작된 **transaction**에 대해서 **437**라인에서는 **rollback**으로, **623**라인에서는 **commit**으로 종료하지만, **443**라인에서는 해당 **transaction**에 대해서 종료하지 않고 **return**한다.

✓ **commit** 문이 사용되지 않으면, 트랜잭션이 시작된 이후의 변경된 사항에 대해서 모두 **rollback** 될 수 있다.

A-max

```
      ...
178: int com_nms_recv(in_buf, out_buf, status)
179: i_trbf_t *in_buf;
180: o_trbf_t *out_buf;
181: long *status;
182: {
      ...
218:    tx_begin();
219:    if (s_tm) {
220:       EXEC SQL set lock mode to wait 10;
      ...
374:    }
      ...
436:    if (st_code == 0) {
437:       tx_rollback();
438:       edit_md_proc(st_code,  in_buf, out_buf);
439:       return  (0);
440:    }
441:    else if (st_code < 0) {
442:       message(st_code);
443:       return (rcd);
444:    }
      ...
623:    tx_commit();
      ...
629: }
      ...
```

Case 6. Unreachable Code

7421라인에서 **return** 코드에 의해 뒤에 따르는 함수 및 **CURSOR** 해제가 수행되지 않는다.

- ✓ 작성된 코드가 수행되지 않음으로 원하지 않는 결과를 초래할 수 있다.
- ✓ 불필요한 코드가 실행 프로그램에 존재할 수 있다.

A-max

```
        ...
7371: while(1) {
          ...
7377:    if (josa->gy_cnt > GYA_CNT ) {
             ...
7395:       return -1;
7396: }
          ...
7402:    if (SQLCODE == SQLNODATA) {
             ...
7406:       break;
7407:    } else if (SQLCODE != SQLOK) {
             ...
7421:       return -1;
7422:
7423:       fda_error_process (SQLCODE);
7424:
7425:        EXEC SQL close c_bj;
7426:        EXEC SQL free c_bj;
7427:       return SQLCODE;
7428:    }
          ...
7490: }
```

Case 7. Cursor/Prepare Declaration in a Loop

1068라인에서 동일한 내용의 커서 선언과 해제가 **for**문 내에서 반복적으로 수행되고 있다.

- ✓ 반복적으로 리소스의 할당 및 해제가 발생함으로써 실행 속도가 저하될 수 있다.
- ✓ 서비스 다운 및 기능 정지 등의 요인이 될 수 있다.

A-max

```
      ...
946:  for(size_t i=0; I < rec_cnt; i++) {
      ...
1068:    EXEC SQL DECLARE cust_curs
1069:                    CURSOR FOR
1070:             select a.cust_no, a.account_no, b.*
1071:             from info_db:master_info a,
1072:                  info_db:master      b,
1073:                  info_db:gt_setting c
1074:             where a.mng_no = acs_mng_no
1075:               and a.bkn_code = 1
1076:               and a.cust_no = b.cust_no
1077:               and a.account_no = b.account_no
1078:               and a.kbn_code = b.kbn_code
1079:               and b.acs_yn in ('1','2')
1080:               and b.cust_no = c.cust_no
1081:               and b.cust_no = c.ret_cust_no
1082:               and c.reject_yn = '1';
      ...
1097:    EXEC SQL FREE cust_curs;
      ...
1204: }
       ...
```

Application Maximizer A-max

Case 8. Unused Cursor or Prepare

692라인에서 **CURSOR**를 **OPEN**하고 **700**라인에서 **CLOSE**하지만, 해당 커서를 사용하지 않고 **return**한다.

- ✓ 사용하지 않는 **Cursor** 및 **Prepare**에 의해 **DBMS**의 **Operation**을 유발하게 되어 실행 속도가 저하될 수 있다.
- ✓ 불필요한 자원 낭비를 초래한다.

A-max

```
      ...
651: static int
652: ctx_logger_write(_ctx_logger_t *ctx)
653: {
      ...
690:     bzero(&in, sizeof(app_logger_f001_t));
691:
692:     EXEC SQL OPEN acc_f001_cur;
693:
694:     ret = write_logger(ctx, &in);
695:     if (ret != SUCCESS) {
696:         error_message(ret, ctx, msg);
697:         return RC_FAIL;
698:     }
699:
700:     EXEC SQL CLOSE acc_f001_cur;
701:
702:     return RC_SUCCESS;
703: }
      ...
```

Case 9. DB Query not in a transaction

183라인에서 **Transaction**이 시작되어 **314**라인에서 종료되었지만, 중간에 **SQL**문을 사용하고 있지 않다.

- ✓ 사용하지 않는 트랜잭션 모드를 만들게 됨으로써 불필요한 **DBMS operation**이 수행된다.
- ✓ 불필요한 자원 낭비를 초래한다.

A-max

```
    ...
165:  bool alg_set_rects(char *pszFile)
166:  {
    ...
183:     EXEC SQL BEGIN WORK;
184:
    ...
191:     strcpy( szLoadFile, ALG_DATA_FILE);
192:     strcat( szLoadFile, pszFile);
193:     strcat( szLoadFile, ".dat");
194:     if((fp = fopen(szLoadFile, "r")) == NULL) {
195:        error_print();
196:        return FALSE;
197:     }
    ...
309:     fclose(fp);
    ...
314:     EXEC SQL COMMIT WORK;
315:
316:     return TRUE;
317: }
    ...
```

Case 10. DB Connection with no queries

398라인에서 **DB connection** 함수를 호출하지만, 해당 프로그램에서는 **DB** 관련된 작업을 수행하지 않는다.

- ✓ **DB**작업을 하지 않는 프로그램에서 **DB connection**을 생성함으로써, 프로그램의 실행 속도가 저하될 수 있다.
- ✓ 불필요한 자원 낭비를 초래한다.

A-max

```
    ...
286: bool ConnectDB(const char *pszServerName)
287: {
    ...
312:     EXEC SQL CONNECT :szUserID
313:                 IDENTIFIED BY :szPassword;
    ...
321: }
    ...
346: int main(argc, argv)
347: int argc;
348: char **argv;
349: {
        ...
398:     if (!ConnectDB(COMM_DB_W2))
399:     {
400:         cgiError(E_DB_CONN_S2);
401:         exit(CGI_NOCONN)
402:     }
        ...
683:     return 0;
684: }
```

Case 11. Functions with an incorrect number of parameters

1355라인에서 호출한 함수에서 인자는 **4**개가 사용되었지만 실제 해당 함수가 정의된 **753**라인에서 **5**개의 인자를 가지는 함수로 선언되어 있다.

✓ 함수 호출 시 인자를 잘 못 전달함으로써 원하지 않는 결과를 초래할 수 있다.

A-max

```
      ...
753:  static long check_goods (
754:          idx_no, flag, ms_goods_in,
755:          ms_goods_out, err_msg)
756:  int                 idx_no,
757:  unsigned short  flag,
758:  char *              ms_goods_in,
759:  char *              ms_goods_out,
760:  char *              err_msg
761:  {
          ...
829:      return 0;
830:  }
      ...
1097:  long goods_select(idx, kind, …)
      ...
1104:  {
      ...
1355:      rt = check_goods (idx, flag, in, errmsg);
      ...
1450: }
      ...
```

Case 12. XA operation in Non-XA mode or Vice Versa

388라인에서 호출한 함수를 보면 **774**라인에서 **Non-XA mode**에서 **XA operation**을 사용하였음을 알 수 있다.

- ✓ **DB**를 처리하는데 있어서 **XA**와 **Non-XA**의 혼용은 **DBMS** 무결성에 위배 된다.
- ✓ 데이터의 질적 저하를 초래 한다.

A-max

```
          ...
374:      EXEC SQL OPEN dml_cur;
          ...
387:      if (sdt->type == DML_TARGET) {
388:          ret = dml_check_target(sdt);
          ...
717: }
718:
719: static int
720: dml_check_target(stp_dml_target_t *sdt)
721: {
        ...
747:        if (sdt->in == 0) {
748:            print_message(std, err_cd, msg);
749:            return RC_FAIL;
750:        }
          ...
774:      tx_open();
          ...
798:      return RC_SUCC;
799: }
    ...
```

Case 13. Use for Standard API Function in C

191라인에서 사용한 **strcpy, strcat** 함수는 **size**를 포함하는 **strncpy, strncat** 함수를 사용하도록 한다.

- ✓ 문자열과 관련된 함수를 사용할 때에는 가능한 **Size**를 포함하는 함수를 사용하도록 권장한다.
- ✓ 이는 정의된 버퍼 크기를 초과하여 사용하게 될 경우 발생할 수 있는 데이터의 손실에 대해서 사전에 코드의 안정성을 유지하기 위한 방법이다.

A-max

```
      ...
165:  bool alg_set_rects(char *pszFile)
166:  {
      ...
183:      EXEC SQL BEGIN WORK;
184:
      ...
191:      strcpy( szLoadFile, ALG_DATA_FILE);
192:      strcat( szLoadFile, pszFile);
193:      strcat( szLoadFile, ".dat");
194:      if((fp = fopen(szLoadFile, "r")) == NULL) {
195:          error_print();
196:          return FALSE;
197:      }
      ...
309:      fclose(fp);
      ...
314:      EXEC SQL COMMIT WORK;
315:
316:      return TRUE;
317: }
      …
```

7. 지원 환경

Application Maximizer A-max

Operating System	HP-UX IBM AIX Sun Solaris FreeBSD Linux Windows	**Middleware**	Tuxedo ATMI in C Tmax XATMI & TX in C
Database	Oracle Pro*C Informix ESQL/C DB2 SQC	**Compiler**	GNU gcc HP-UX aCC IBM xlc Sun cc ANSI C 호환 Comiler

III. 적용 사례

1. 적용 사례

Application Maximizer A-max

적용 사례

- 적용 사례 : 국내 **A** 금융사 차세대 뱅킹 시스템 구축 및 운영에 적용 중
- 적용 대상 : 온라인, 배치 및 **DBIO** 모듈 등 약 **50,000**본 이상의 소스에 적용

적용 방법

- 개발 단계 : 단위별 검증 및 팀별 일괄 검증 방법을 이용하여 개발과정시 적용
- 운영 단계 : 추가 및 수정 후 이행 요청시 소스 검증 보고에 따른 반려 및 이행

적용 시스템

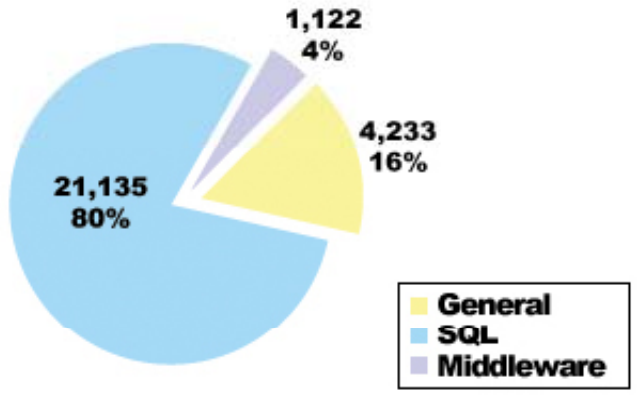
- **OS : HP-UX itanium64 UNIX Operating System**
- **Database : Oracle 10g Database System**
- **Middleware : Tmax v3.14 System**

2. 유형별 진단 결과 요약

Application Maximizer A-max

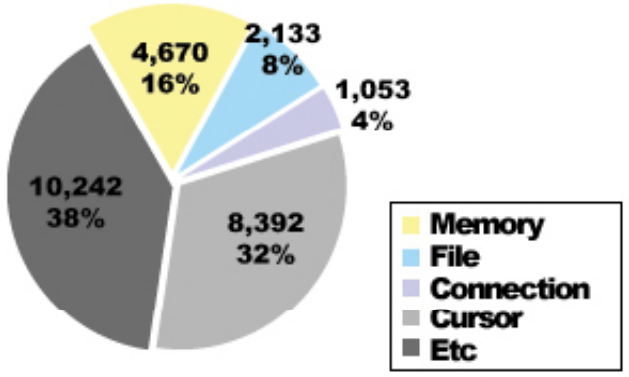
By Code Type

Types	Messages
General	4,233
SQL	21,135
Middleware	1,122
Total	26,490



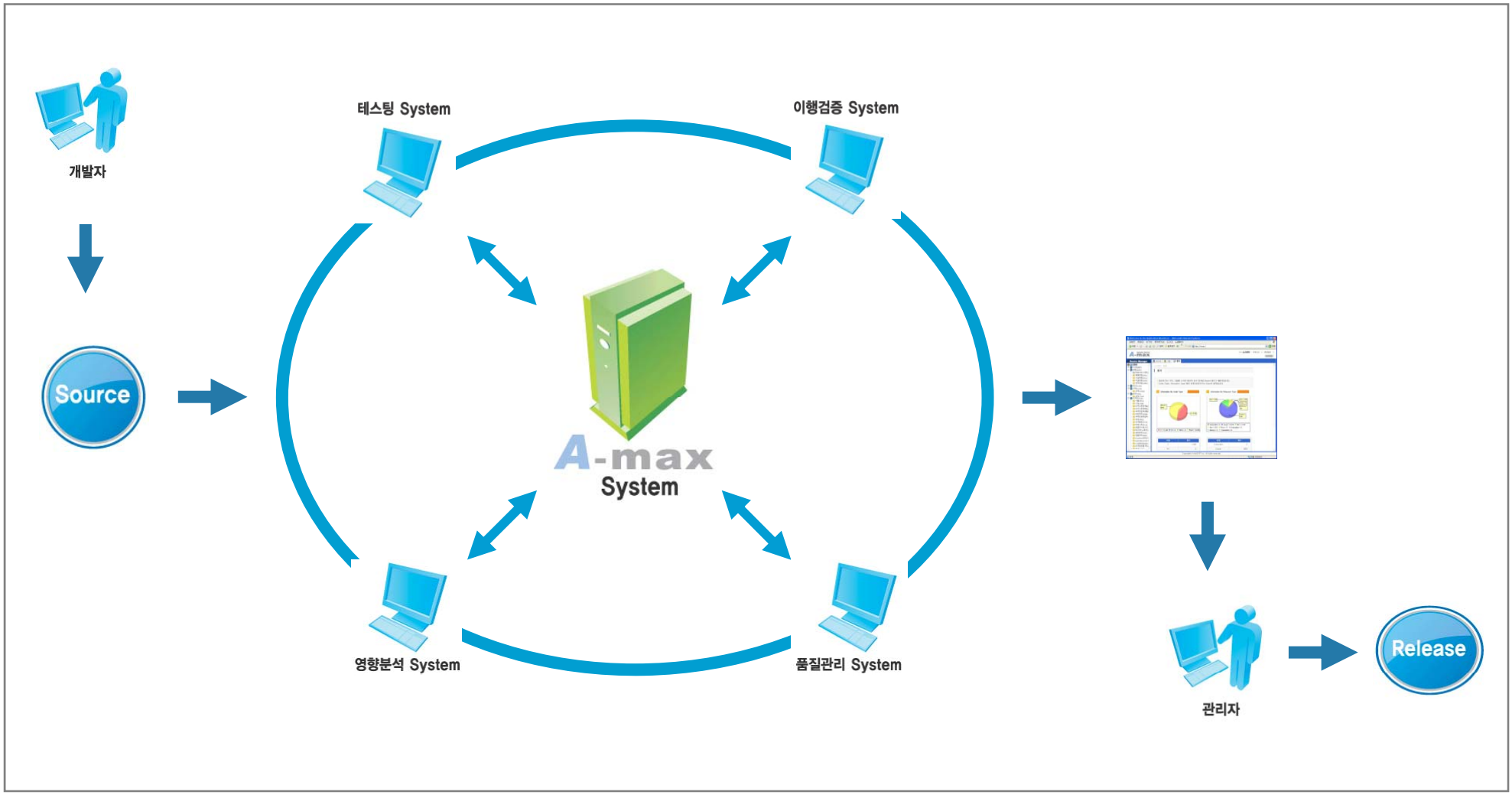
By Resource Type

Types	Messages
Memory	4,670
File	2,133
Connection	1,053
Cursor	8,392
Etc	10,242
Total	26,490



※ 2,352개의 파일을 대상으로 진단한 결과 요약임.

3. 적용 시스템 구조


개발자
Source
테스팅 System
이행검증 System
A-max
System
영향분석 System
품질관리 System
관리자
Release

Thank you !!!

인프라소프트 ㈜
www.infrasoft.co.kr
02-468-2570