Análisis Estático de Código Fuente

Dr. Rafael Martínez Gasca
Grupo de Investigación **IDEA**,
Tecnologías Inteligentes y de Seguridad de los Sistemas de Información
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Sevilla

Indice

- **Introducción**
- Ejemplo de Análisis Estático de Código
- Herramientas de análisis estático de código y sus limitaciones

- **Defecto (Defect):** Relacionados a un requisito erróneo.
  - Relaciones: ¿Se cumplen todos los asertos e invariantes?
- **Bug:** desviación del resultado esperado del software. Una acción humana nos lleva a un resultado erróneo.
- **Nasty Bug**: se manifiestan cuando el código se ejecuta
  - **NullPointerException**
  - **Bloques catch vacíos**
  - **Java Memory Leaks.**
  - **Acceso Concurrente a datos compartidos**

- **Manual:** que incluye:
  - Comprensión del Código
  - Revisión del Código
  - Inspecciones del código
- **Automático Estático**: Técnicas sin ejecutar el código:
  - Análisis del Data-flow (DFA)
  - Ejecución Simbólica
  - Análisis de dependencia
- **Automático Dinámico**: Técnicas que rigurosamente examinan un programa basado en algún criterio durante run-time:
  - Code Coverage Analysis
  - Error-seeding y mutation testing, regression testing, …
  - Program slicing y Assertions

- Análisis estático automático de código sin ejecutarlo para encontrar vulnerabilidades de seguridad se puede realizar con:
  - ***source code weakness analyzers*,**
  - ***source code security analyzers*,**
  - ***static application security testing* (SAST) ,**
  - ***static analysis code scanners*, o**
  - ***code weakness analysis tools*.**
- Usan varias clases de técnicas mediante matching de diversos patrones (e.g., ellos podrían hacer ***taint checking*** para el seguimiento de datos desde fuentes no confiables para ver si están enviando operaciones potencialmente peligrosas).

Tipos de análisis estático de código.
Vulnerabilidades identificadas

- **Vulnerabilidades Semánticas:** incluye la fuga de información a través del acceso de un usuario de un programa dado.
- **Vulnerabilidades Dataflow:** entradas o datos que pueden producir efectos no deseados en la ejecución del programa. Ataque SQL-Injection
- **Vulnerabilidades Estructurales:** ocurren cuando por ejemplo un password se codifica en el programa o el nombre de algún recurso que contenga información relevante claves, ficheros de hashing, … entonces cualquiera que pueda acceder al código puede ganar el acceso a este tipo de información tan relevante.
- **Vulnerabilidades de Integración**: Debido a la incorrecta integración de los diferentes componentes software.

Indice

- Introducción
- **Ejemplo de Análisis Estático de Código**
- Herramientas de análisis estático de código y sus limitaciones

Ejemplo de Análisis Estático de Código

- **El ejemplo que sigue se toma del artículo:**
  - ***Testing for Software Security: A Case Study on Static Code Analysis of a File Reader Java Program*** *Natarajan Meghanathan and Alexander Roy Geoghegan Jackson State University USA. Advances in Computer Science, Eng. & Appl., AISC 166, pp. 529–538. 2012*

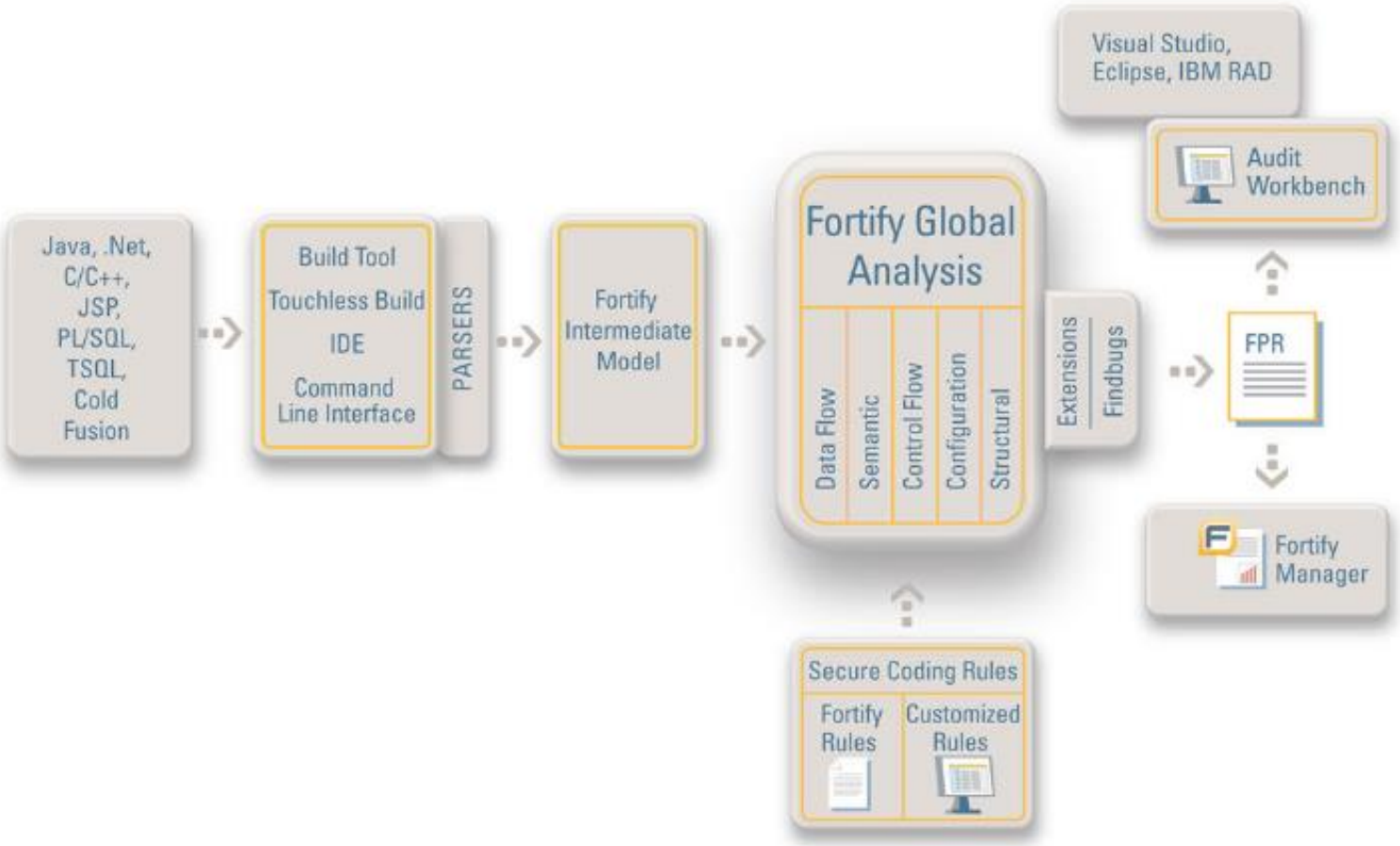
Caso de estudio en Java

```
1     import java.io.*;
2     class testFileRead{
3
5       public static void main(String[ ] args) throws IOException{
6
7       try{
8
9         FileReader fr = new FileReader(args[0]);
10        BufferedReader br = new BufferedReader(fr);
11
12        String line = null;
13
14        while ( (line = br.readLine() ) != null){
15          System.out.println(line);
16        }
17
18
19        br.close();
20        fr.close();
21
22
23      }// try block
24      catch(IOException ie){
25        ie.printStackTrace( );
26        }
27      }
28    }
```

Ejemplo de Analizador Estático de Código

- **Fortify Static Code Analyzer** (SCA) es un conjunto de analizadores de seguridad del software
  - Tecnología que permite localizar y priorizar las violaciones a partir de reglas de seguridad que hay establecidas.
  - Contiene 5 analizadores distintos: ***de data flow, de control flow, semántico, estructural, y de configuración***.
  - Cada analizador acepta diferentes tipos de reglas, que son definiciones de los elementos del código fuente que pueden originar vulnerabilidades de seguridad.

Ejemplo Analizador Estático de Código



(http://www.fortifysoftware.com/products/sca/scaHowItWorks.jsp)

Ejemplo de Analizador Estático de Código.

- ***Fortify SCA*** se usa para las vulnerabilidades software que podrían causar:
  - **Buffer Overflow, Command Injection,**
  - **Cross-Site Scripting, Denial of Service,**
  - **Format String, Integer Overflow,**
  - **Log Forging, Password Management,**
  - **Path Manipulation, Privacy Violation,**
  - **Race Conditions, Session Fixation, SQL**
  - **Injection, System Information Leak, and Unreleased Resource**

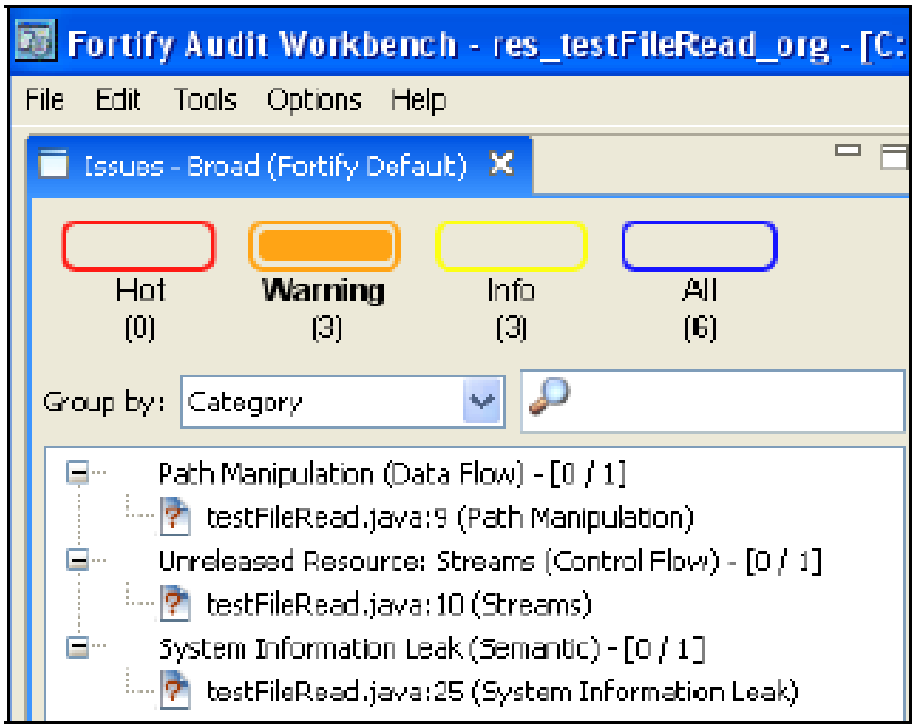
Caso de estudio en Java

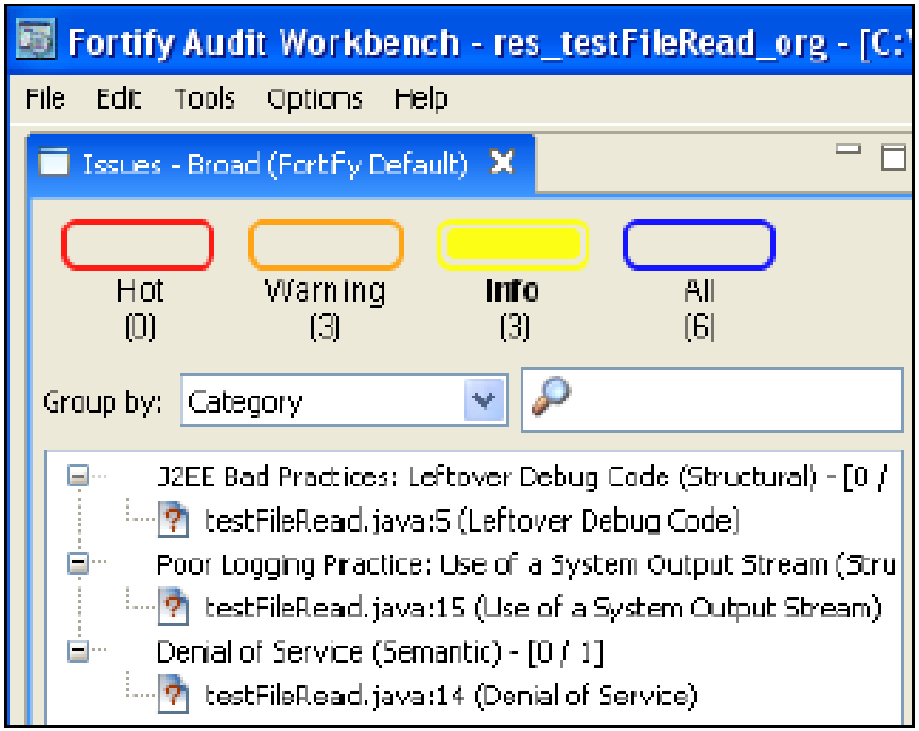
- Debemos asegurarnos que el programa compila antes de usar herramientas automáticas de análisis de código Java.
- Este programa cuando se le pasa el analizador presenta:
  - 3 vulnerabilidades de nivel medio
  - 3 vulnerabilidades de nivel bajo

Caso de estudio en Java

```
C:\res\CCLI-2010\Modules-Meghanathan\Static-Code-Analysis-Examples\Ex1_FileReade
r>sourceanalyzer testFileRead.java


[C:\res\CCLI-2010\Modules-Meghanathan\Static-Code-Analysis-Examples\Ex1_FileRead
er]

[F014B0E28C8E6288784927FC772618FE : low : Denial of Service : semantic ]
testFileRead.java(14) : BufferedReader.readLine()

[EDD1323454D69423D2DD7D4D187D22B7 : medium : System Information Leak : semantic
]
testFileRead.java(25) : Throwable.printStackTrace()

[78FA8236847 1A9D617111E250114E445 : medium : Path Manipulation : dataflow ]
testFileRead.java(9) :   ->new FileReader(0)
    testFileRead.java(5) :   ->testFileRead.main(0)

[865F144B2D584D3CB7CEDB696F190416 : medium : Unreleased Resource : Streams : con
trolflow ]
    testFileRead.java(9) : start -> loaded : fr.new FileReader(...)
    testFileRead.java(10) : loaded -> loaded : fr.new BufferedReader(..., fr, ..
.)
    testFileRead.java(14) : loaded -> end_of_scope : #end_scope(fr) (exception t
hrown)

[423D552C35C67B4A8F045E1C079B74FB : low : J2EE Bad Practices : Leftover Debug Co
de : structural ]
    testFileRead.java(5)

[ADBD437811B82372BC593D8FB94B74B6 : low : Poor Logging Practice : Use of a Syste
m Output Stream : structural ]
    testFileRead.java(15)

C:\res\CCLI-2010\Modules-Meghanathan\Static-Code-Analysis-Examples\Ex1_FileReade
r>
```

Caso de estudio en Java





- **Vulnerabilidad DoS**: El programa no podrá usarse por los legítimos usuarios. (control flow error)
  - El método ***readline( )*** invocado por el objeto *BufferedReader* puede **usarse por el atacante para leer una cantidad no acotada de entrada.**
  - **Consecuencia**: Un atacante puede consumir gran cantidad de memoria o causar una ***OutOfMemoryException*** *tal que el programa.*
- **SANITIZACIÓN: validar las entradas de usuarios asegurando que no causarán la utilización inapropiada de recursos. Acotar la entrada a leer y si es muy grande lanzar una excepcion IOException.**

- Nuevo método readLine

```
36      public static String readLine(BufferedReader br) throws IOException{
37
38        StringBuffer sb = new StringBuffer( );
39        int intC;
40        intC = br.read( );
41        String line = null;
42        do{
43              if (intC == -1)
44                  return null;
45
46          char c = (char) intC;
47
48          if (c = = '\n') {
49            break;
50          }
51          if (sb.length( ) >=  testFileRead.MAX_STR_LEN) {
52            Throw new IOException("input too long");
53          }
54          sb.append(c);
55        } while ( ((intC = br.read( )) != -1) );
56
57        line = sb.toString( );
58
59        return line;
60      }
```

- **Vulnerabilidad de Fuga de Información del Sistema (semántico)**: Se trata que se revelará al usuarios no legítimos datos del sistemas o información de depuración.
  - El método ***printStackTrace( )*** , llamado por los objetos de la clase IOException del programa, podría favorecer esta potencial fuga de información.
- **SANITIZACIÓN:** Se elimina la llamada a ie.printStackTrace() y se coloca:
  - ***System.println.out("Excepción se ha producido");***
  - ***Programar para que estos errores vayan a un fichero de log***

- **Vulnerabilidad por recursos no liberados (DoS) (control flow)**
  - Los dos flujos '*fr*' de ***FileReader*** y *'br' de* ***BufferedReader*** *según el programa podrían no ser liberados nunca al haber un fallo en el bloque try hasta que Sistema Operativo* explícitamente fuerza la liberación al finalizar el programa.
- **SANITIZACIÓN: Poner en el código *finally* y declarar fr y br fuera del bloque try**

```
finally {
        If(br !=null)
            br.close();
        If (fr( !=null)
            fr(close();
        }
```

- **Vulnerabilidad por Path Manipulation**: ocurre cuando una entrada de usuario se permite aunque viole los controles establecidos en las operaciones sobre los recursos.
  - Esto permite al atacante acceder o modificar recursos protegidos.
- **SANITIZACIÓN:**
  1. **Codificar una lista de valores válidos para el usuario y que el usuario no pueda escoger otros. Por ejemplo, en el programa presentar al usuario una lista de ficheros que podrían ser leidos y el usuario tiene que seleccionar uno entre ellos.**
  2. **Tener listas blancas de caracteres permitidos al usuario en las entradas.**

- Implementación de la Sanitización 2

```
63      public static int sanitize(String filename){
64
65        if (filename.indexOf( (int) '/') != -1){
66          System.out.println(" invalid argument... You cannot read from a directory other than the current one");
67              return -1;
68          }
69
70          if ( !filename.endsWith(".txt") ){
71              System.out.println(" you can read only a text file with a .txt extension..");
72              return -1;
73          }
74
75          return 0;
76      }
```

- Uso de un método Scanner para recoger el nombre

```
12        try{
13
14        Scanner sc = new Scanner(System.in);
15        String filename = sc.next( );
16
17        if (sanitize(filename) != -1){
18
19          fr = new FileReader(filename);
20          br = new BufferedReader(fr);
21          String line = null;
```

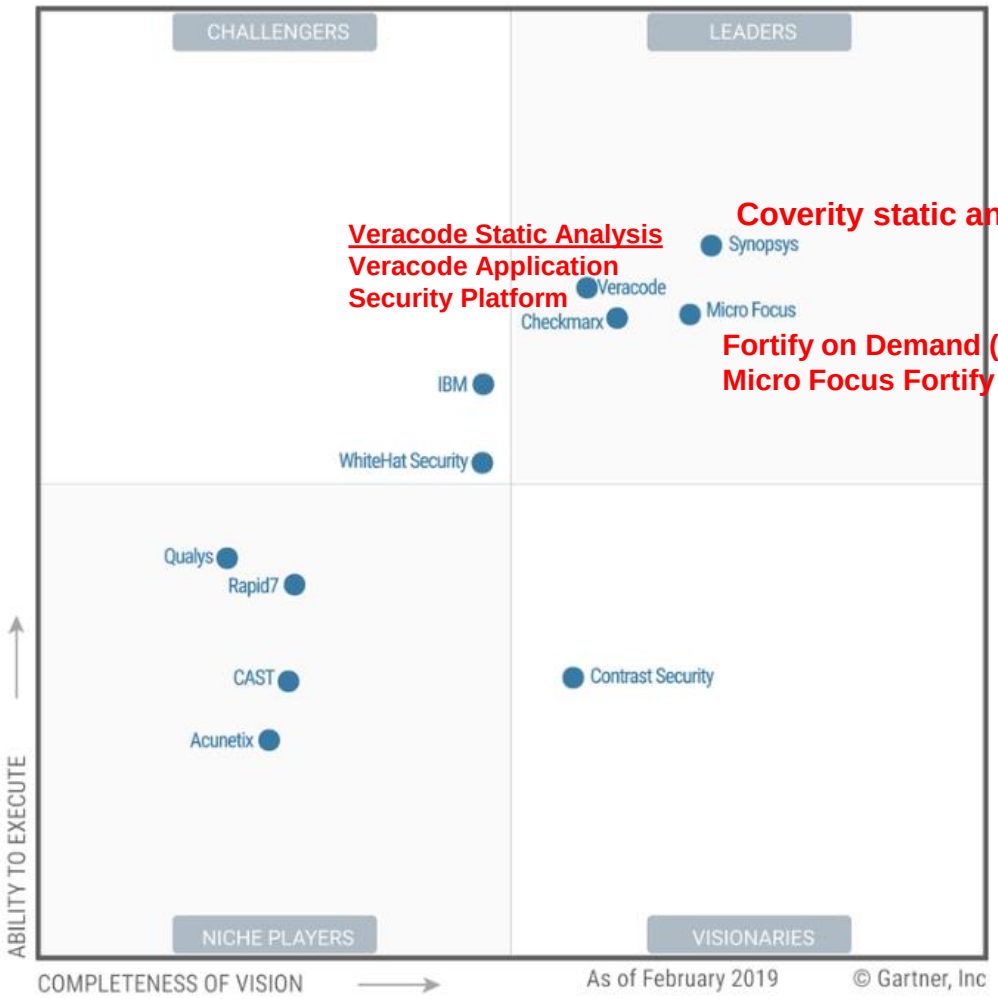
- 3. Tener una lista negra de caracteres que no se permiten introducir al usuario como argumento/variable. Por ejemplo, si el usuario no se le permite leer ficheros que no estén en el directorio del programa que está ejecutándose.

Indice

- Introducción
- Ejemplo de Análisis Estático de Código
- **Herramientas de análisis estático de código y sus limitaciones**

Herramientas de Análisis Estático de Código

Figure 1. Magic Quadrant for Application Security Testing



Source: Gartner (April 2019)

RIPS Static Code Analysis
Yasca
Kiuwan Static Code Analysis

Herramientas de Análisis Estático de Código

- Código en Microsoft .Net: FxCop (F), StyleCop (F), CodeIt.Right(C)
- Código en Java: FindBugs (F), PMD (F). CheckStyle(F), Jlint (F)
- Código en C/C++: Lint, CodeSonar(C), HP Code Advisor (C). Splint (S)
- Multi-lenguajes y multipropósito: Coverit Prevent, Klockwork Insight, Hummurapi, RATS, Understand (C), SonarQube, SQuoRE

Herramientas de Análisis Estático de Código.
Limitaciones

- Las salidas de las herramientas automáticas de análisis estático de código **requieren una evaluación humana final, pues**:
  - Resulta muy complejo que una herramienta conozca exactamente y de forma automática cúales problemas son más o menos importantes para alcanzar un nivel de riesgo de seguridad aceptable.
  - Se pueden producir **falsos negativos** (el código contiene bugs que la herramienta no detecta) o **falsos positivos** (la herramienta informa sobre bugs que no contiene).
  - Una Buena **herramienta de análisis estático es aquella que, aunque algunas veces muestre falsos positivos, nunca permite falsos negativos.**

- No detectar los **errores semánticos**

```
int calculaAreaRectangulo(int length, int width)
{
        return (length + width);
}
```

  - Un herramienta de análisis estático podría detectar un posible *overflow* en el cálculo, pero no determinar que la función no proporciona el valor que se espera de ella.

- Las **limitaciones de espacio** en las funciones que implementan los lenguajes de alto nivel.

  Por ejemplo para la función ***BufferedReader*** en java se dice:

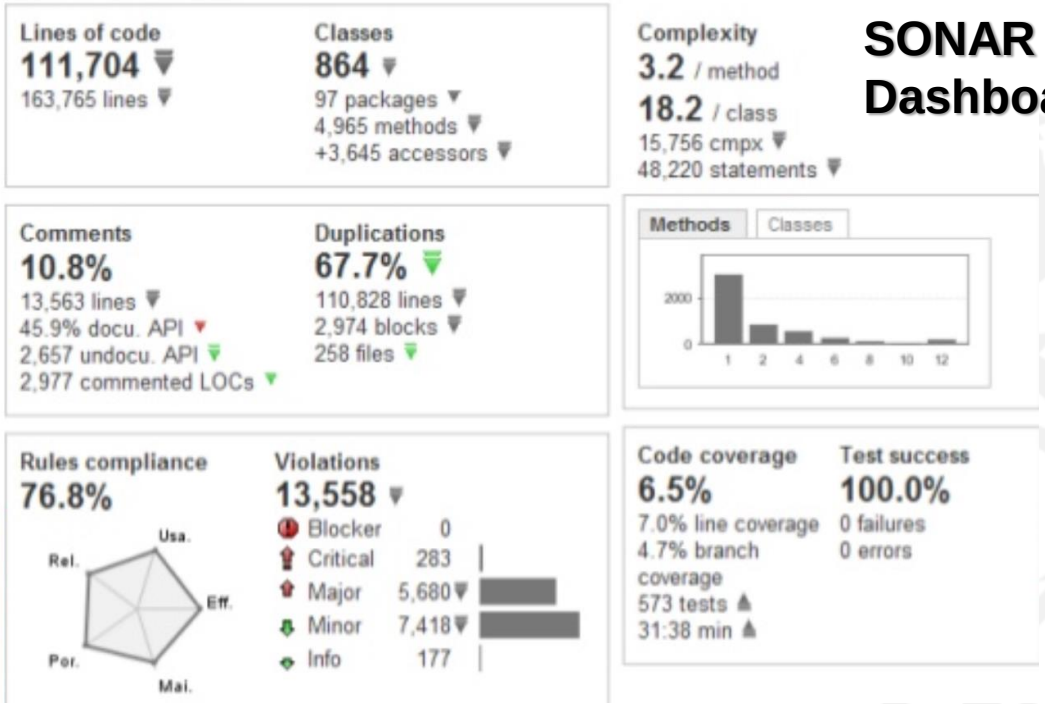
The buffer size may be specified, or the default size may be used. The default is large enough for most purposes.

Herramientas de Análisis Estático de Código.
Limitaciones

- Las **limitaciones en tiempo** en las funciones que implementan los lenguajes de alto nivel.
  - La implementación sobre el tratamiento de eventos para interfaces gráficas que se realizó en las primeras versiones de awt en Java
- **Imprecisiones en la aritmética de punto flotante**.
- Adecuado **grado de aleatoriedad de los números** aleatorios generados.

Herramientas de Análisis Estático de Código.
Métricas para el dashboard

- Muchas de las herramientas permiten a partir del análisis del código fuente visualizar un dashboard



SONAR Application Dashboard

Otras métricas podrían ser:

Error Detection Efficiency (Errors found by a inspection/ Total errors en el producto antes de la inspección) x 100 según Fagan,1976

Prioridad de los diferentes bugs encontrados que son clasificados según diferentes tipos.

Certificaciones de software de seguridad

- Certificaciones de seguridad de productos software
  - **Common Criteria** for Information Technology Security Evaluation
    - **Uso de Evaluation Assurance Level (EAL 1- funcionalmente verificada al EAL 7 -formalmente verificada)**
  - **FIPS 140-2 del NIST y el Programa de Validación de módulos criptográficos**(CAVP) **(la implementación de todo ello se ha revisado Dic2019)**
    - Testing de **funciones de Seguridad Aprobadas**
    - Generadores de ***Números aleatorios aprobados***
    - Técnicas de ***establecimiento de claves aprobados***, que se referencian en los anexos de FIPS 140-2.

Bibliografía

- *Gray Hat Hacking. The Ethical Hacker's Handbook, Daniel Regalado et al. 2015*
- **Información sobre analizadores de seguridad de código fuente: *https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html***
- ***https://www.intertech.com/Blog/top-10-nasty-java-bugs/***