Technological foundations of software development

Debug, log, test, profile, analyze your software

Objectives of the session

This session aims to familiarize you with the methods and tools for debugging, logging, testing, profiling, analyzing your software.
In particular, we will cover tools for Java, Python, JavaScript.

Technological foundations of software development

Debug, log, test, profile, analyze your software

Part 1: Big Bang development

ICM – Computer Science Major – Course unit on Technological foundations of computer science
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/

... write code the old-fashioned way

... possibly with some precautions

- try-catch

```
try { ... } catch (Exception ex) { ... }
```

- multicatch

```
try { ... } catch (NullPointerException |IOException ex) { ... }
```

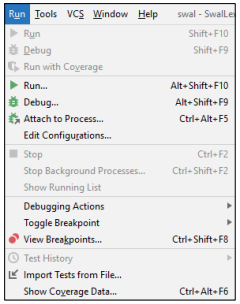
- finally

```
try { ... } catch (Exception ex) { ... } finally { ... }
```

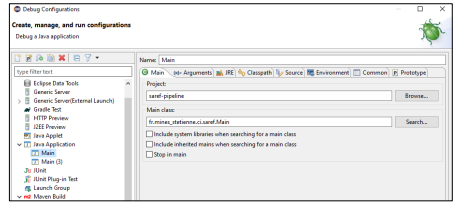
- try with resource:

```
try (InputStream fis = new FileInputStream(file) ) { ... }
catch (...) { ... }
```

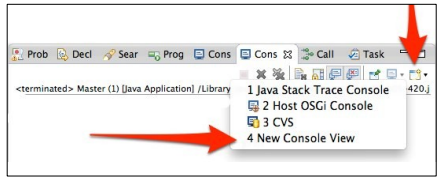

Run your code...



Run Menu in IntelliJ

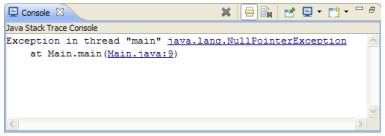


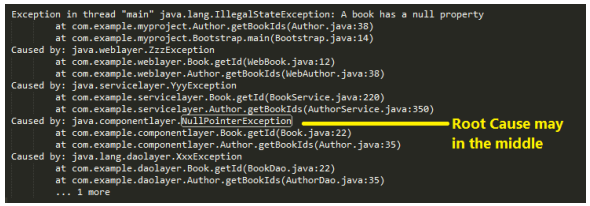
Run configuration in Eclipse



Console in Eclipse

... and stumble upon an error





Stacktrace

Print values ...

```
try{
    System.out.println("Testing PTree constructor setup");
    PTree ret =  new PTree(true);
    System.out.println("PASSED setup test");
    sysout
    return ret;
}
```

Ctrl + Space

```
try{
    System.out.println("Testing PTree constructor setup");
    PTree ret =  new PTree(true);
    System.out.println("PASSED setup test");
    System.out.println();
    return ret;
}
```

and we comment ... and we decomment ...

and we start again...

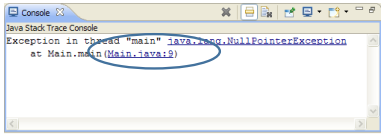
Technological foundations of software development

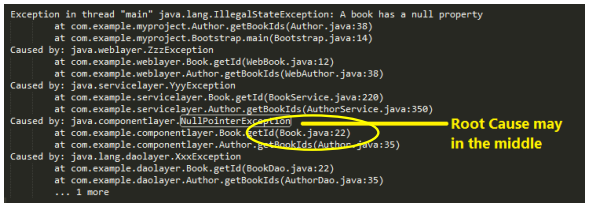
Debug, log, test, profile, analyze your software

Part 2: Troubleshoot a bug

ICM – Computer Science Major – Course unit on Technological foundations of computer science
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/

... and stumble upon an error

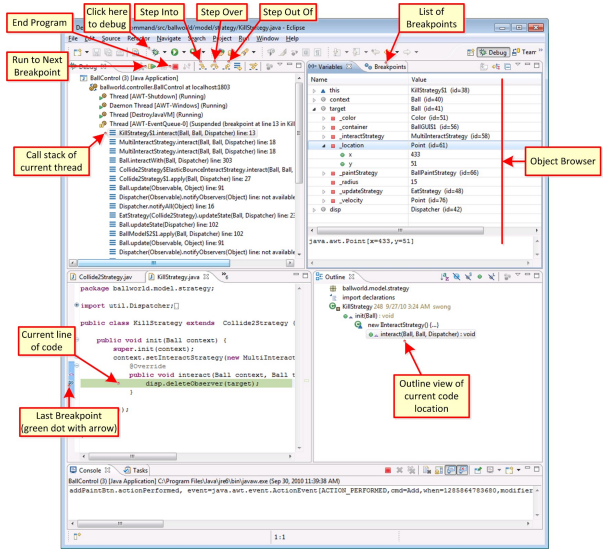




Stacktrace

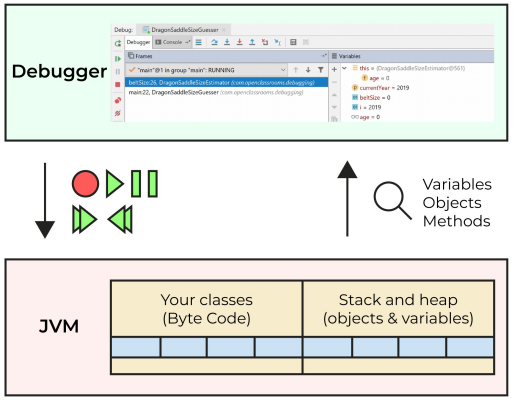
Debugging

- Controlled execution of the program
- Monitoring and/or modification of the content of variables during execution
- Allows to trace bugs



Debugging

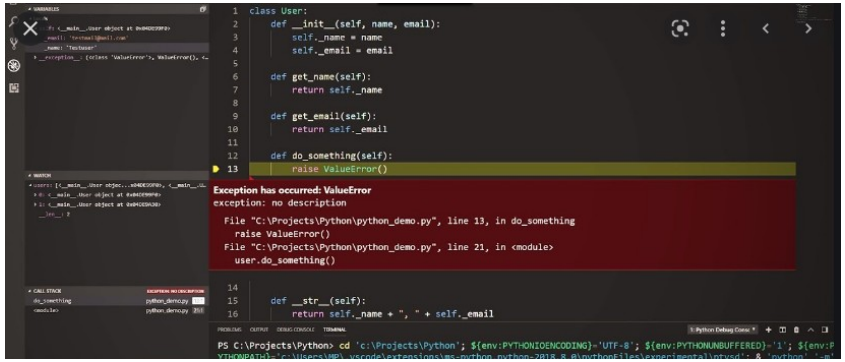
- Uses the **Java Platform Debugger Architecture (JPDA)** framework
- All Java IDE debuggers will have approximately the same functionalities



JPDA

https://docs.oracle.com/javase/8/docs/technotes/guides/jpda/

Debugging with Python



https://www.youtube.com/watch?v=w8QHoVam1-I&ab_channel=Jayanam

Debugging with Python

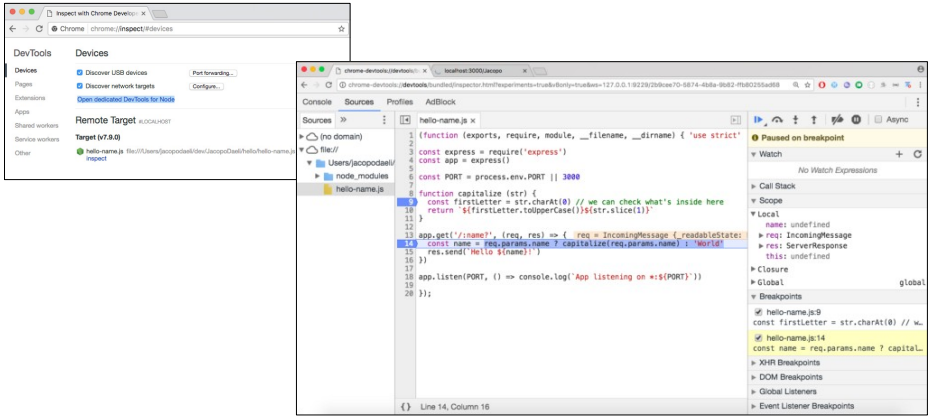
- Python debuggers rely on the standard **pdb** module

The debugger's prompt is `(Pdb)`. Typical usage to run a program under control of the debugger is:

```
>>> import pdb
>>> import mymodule
>>> pdb.run('mymodule.test()')
> <string>(0)?()
(Pdb) continue
> <string>(1)?()
(Pdb) continue
NameError: 'spam'
> <string>(1)?()
(Pdb)
```

https://docs.python.org/3/library/pdb.html

Debugging with node.js



https://medium.com/the-node-js-collection/debugging-node-js-with-google-chrome-4965b5f910f4

Technological foundations of software development

Debug, log, test, profile, analyze your software

Part 3: Smart Logging

ICM – Computer Science Major – Course unit on Technological foundations of computer science
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/

Logging

Print messages with varying degrees of severity

- fatal : messages concerning an unexpected stop of the application
- error : error messages requiring analysis e.g. exceptions thrown
- warn : warning message
- info : information messages for example the start or stop of the application, the connection to a resource, ...
- trace : messages to follow the execution flow
- debug : debugging messages for example to get the value of variables, ...

only during development

Logging

Example for Java: log4j framework

```java
package com.jmdoudoux.test.log4j;

import org.apache.log4j.Logger;

public class TestLog4j1 {

  private static Logger logger = Logger.getLogger(TestLog4j1.class);

  public static void main(String[] args) {
    logger.debug("msg de debogage");
    logger.info("msg d'information");
    logger.warn("msg d'avertissement");
    logger.error("msg d'erreur");
    logger.fatal("msg d'erreur fatale");
  }
}
```


Logging

Exemple for Java: log4j framework (with Maven)

- add `log4j:log4j:jar:1.2.x` in the list of dependencies in the `pom.xml`
- configuration file `src/main/resources/log4j.properties`

```
log4j.rootLogger=DEBUG, stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d [%-5p] (%F:%M:%L) %m%n
```

an example of a log4j.properties file

- different appenders (console, files, ...)
- choice of the pattern https://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html

Logging

```java
package com.jmdoudoux.test.log4j;

import org.apache.log4j.Logger;

public class TestLog4j1 {

  private static Logger logger = Logger.getLogger(TestLog4j1.class);

  public static void main(String[] args) {
    logger.debug("msg de debogage");
    logger.info("msg d'information");
    logger.warn("msg d'avertissement");
    logger.error("msg d'erreur");
    logger.fatal("msg d'erreur fatale");
  }
}
```

```
log4j.rootLogger=DEBUG, stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d [%-5p] (%F:%M:%L) %m%n
```

```
2008-06-08 10:16:21,546 [DEBUG] (TestLog4j1.java:main:13) msg de debogage
2008-06-08 10:16:21,546 [INFO ] (TestLog4j1.java:main:14) msg d'information
2008-06-08 10:16:21,546 [WARN ] (TestLog4j1.java:main:15) msg d'avertissement
2008-06-08 10:16:21,546 [ERROR] (TestLog4j1.java:main:16) msg d'erreur
2008-06-08 10:16:21,546 [FATAL] (TestLog4j1.java:main:17) msg d'erreur fatale
```

Logging

Limit the cost of logging

```
logger.debug("valeur="+valeur+" , i="+i+" ,next="+next);
```

```
if (logger.isDebugEnabled()) {
  logger.debug("valeur="+valeur+" , i="+i+" ,next="+next);
}
```

```
private static Logger LOGGER = Logger.getLogger(MaClasse.class);
```

http://www.jmdoudoux.fr/java/dej/chap-logging.htm#logging-1

Logging

Same principles for Python

```
FORMAT = '%(asctime)-15s %(clientip)s %(user)-8s %(message)s'
logging.basicConfig(format=FORMAT)
d = {'clientip': '192.168.0.1', 'user': 'fbloggs'}
logger = logging.getLogger('tcpserver')
logger.warning('Protocol problem: %s', 'connection reset', extra=d)
```

would print something like

```
2006-02-08 22:20:02,165 192.168.0.1 fbloggs  Protocol problem: connection reset
```

Level	Numeric value
CRITICAL	50
ERROR	40
WARNING	30
INFO	20
DEBUG	10
NOTSET	0

https://docs.python.org/3/library/logging.html

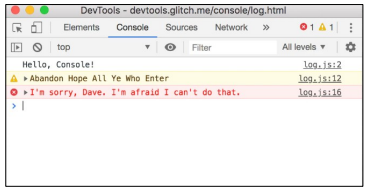
Logging

Same principles for JavaScript

```
console.log('hello world');
// Prints: hello world, to stdout
console.log('hello %s', 'world');
// Prints: hello world, to stdout
console.error(new Error('Whoops, something bad happened'));
// Prints error message and stack trace to stderr:
//   Error: Whoops, something bad happened
//     at [eval]:5:15
//     at Script.runInThisContext (node:vm:132:18)
//     at Object.runInThisContext (node:vm:309:38)
//     at node:internal/process/execution:77:19
//     at [eval]-wrapper:6:22
//     at evalScript (node:internal/process/execution:76:60)
//     at node:internal/main/eval_string:23:3

const name = 'Will Robinson';
console.warn(`Danger ${name}! Danger!`);
// Prints: Danger Will Robinson! Danger!, to stderr
```

https://nodejs.org/api/console.html



https://developer.chrome.com/docs/devtools/console/log/

To read for MCQ test

- Python Basic Logging Tutorial
  https://docs.python.org/3/howto/logging.html#basic-logging-tutorial

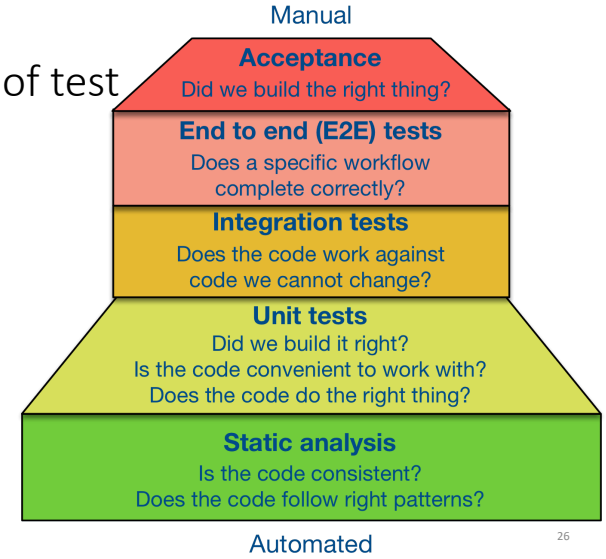
Technological foundations of software development

Debug, log, test, profile, analyze your software

Part 4: Writing and running tests

ICM – Computer Science Major – Course unit on Technological foundations of computer science
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/

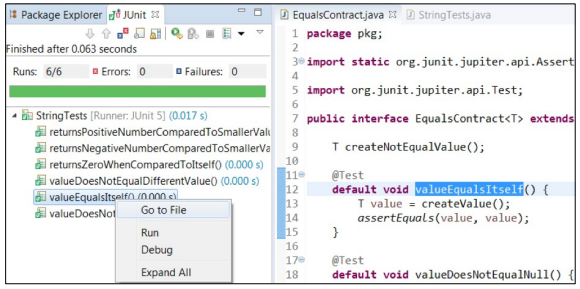
Different types of test



https://survivejs.com/books/maintenance/code-quality/testing/

Unit tests- JUnit for Java

- open source framework for the development and execution of automatable unit tests



http://www.jmdoudoux.fr/java/dej/chap-junit.htm

Unit tests- JUnit for Java

- open source framework for the development and execution of automatable unit tests

```java
public class MaClasse {

  public static int calculer(int a, int b) {
    int res = a + b;

    if (a == 0) {
      res = b * 2;
    }

    if (b == 0) {
      res = a * a;
    }
    return res;
  }
}
```

```java
import junit.framework.*;

public class MaClasseTest extends TestCase {

  public void testCalculer() throws Exception {

    assertEquals(2,MaClasse.calculer(1,1));
  }
}
```

http://www.jmdoudoux.fr/java/dej/chap-junit.htm

Unit tests – Multiple frameworks for Python

- Unit Testing Tools
- Mock Testing Tools
- Fuzz Testing Tools
- Web Testing Tools: Browser simulation, Browser automation
- Acceptance/Business Logic Testing Tools
- GUI Testing Tools
- Source Code Checking Tools
- Code Coverage Tools
- Continuous Integration Tools

https://wiki.python.org/moin/PythonTestingToolsTaxonomy

Unit tests – Multiple frameworks for Python

```
"""
This is the "example" module.

The example module supplies one function, factorial().  For example,

>>> factorial(5)
120
"""

def factorial(n):
    """Return the factorial of n, an exact integer >= 0.

    >>> [factorial(n) for n in range(6)]
    [1, 1, 2, 6, 24, 120]
    >>> factorial(30)
    265252859812191058636308480000000
    >>> factorial(-1)
    Traceback (most recent call last):
        ...
    ValueError: n must be >= 0

    Factorials of floats are OK, but the float must be an exact integer:
    >>> factorial(30.1)
    Traceback (most recent call last):
        ...
    ValueError: n must be exact integer
    >>> factorial(30.0)
    265252859812191058636308480000000

    It must also not be ridiculously large:
    >>> factorial(1e100)
    Traceback (most recent call last):
        ...
    OverflowError: n too large
    """

    import math
    if not n >= 0:
        raise ValueError("n must be >= 0")
```

Standard **doctest** module searches for pieces of text that look like interactive Python sessions, and then executes those sessions to verify that they work exactly as shown.

```
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

you run example.py directly from the command line

```
$ python example.py
$
```

https://docs.python.org/3/library/doctest.html

Unit tests – Multiple frameworks for Python

Standard **unittest** module,

- inspired by Junit,
- similar flavor as major unit testing frameworks in other languages.
- Concepts: *test fixture, test case, test suite, test runner*

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

https://docs.python.org/3/library/unittest.html

Unit tests – Multiple frameworks for Python

pytest module,
One of the most widely used Python testing frameworks

- Use `assert` statements (simpler than unittest)
- Auto-discovery of tests modules and functions
- Modular fixtures for managing small or parametrized long-lived test resources
- Can run **unittest** test suites out of the box

https://docs.pytest.org/en/stable/getting-started.html

```
# content of test_sample.py
def inc(x):
    return x + 1


def test_answer():
    assert inc(3) == 5
```

To execute it:

```
$ pytest
============================= test session starts ==============================
platform linux -- Python 3.x.y, pytest-8.x.y, pluggy-1.x.y
rootdir: /home/sweet/project
collected 1 item

test_sample.py F                                                        [100%]

================================= FAILURES =================================
_____________________________ test_answer _____________________________

    def test_answer():
>       assert inc(3) == 5
E       assert 4 == 5
E        +  where 4 = inc(3)

test_sample.py:6: AssertionError
========================= short test summary info ==========================
FAILED test_sample.py::test_answer - assert 4 == 5
============================ 1 failed in 0.12s =============================
```

To read for MCQ test

- **doctest** documentation (beginning only)
  https://docs.python.org/3/library/doctest.html
- **pytest** getting started guide
  https://docs.pytest.org/en/stable/getting-started.html

Technological foundations of software development

Debug, log, test, profile, analyze your software

Part 5: Profiling the execution of your software

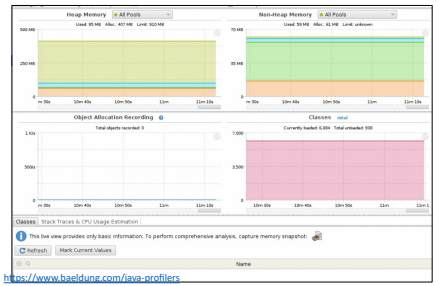
ICM – Computer Science Major – Course unit on Technological foundations of computer science
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/

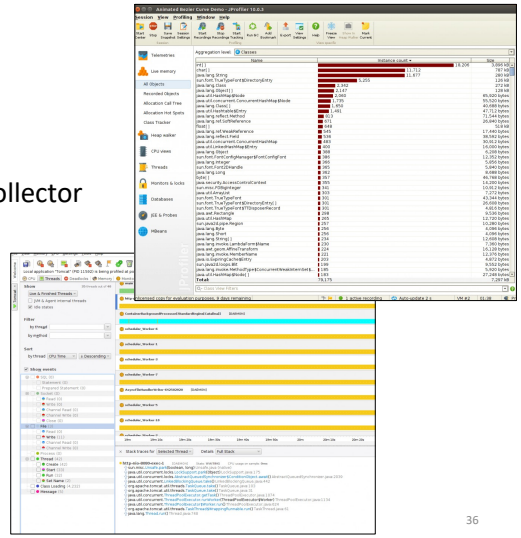
Profiling the execution of your software: Why ?

- detect memory leaks
- detect application congestion points
- identify possible improvements

Java

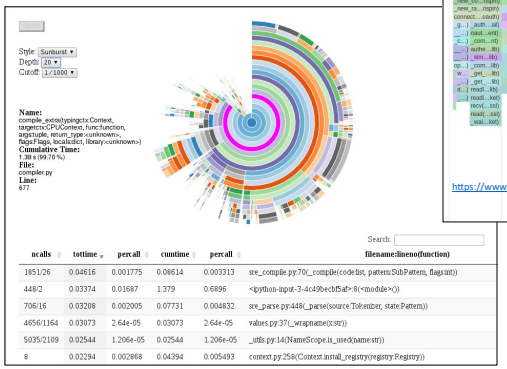
- threads, function calls
- number of objects, garbage collector
- connections to databases

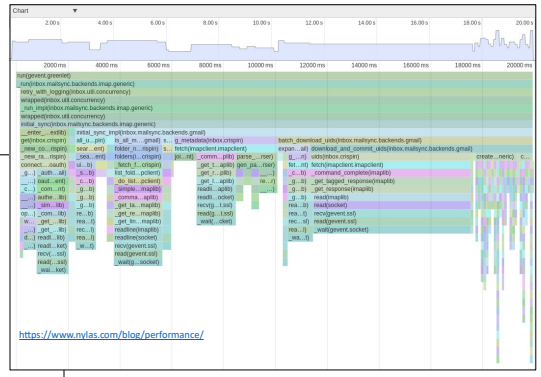




https://www.baeldung.com/java-profilers

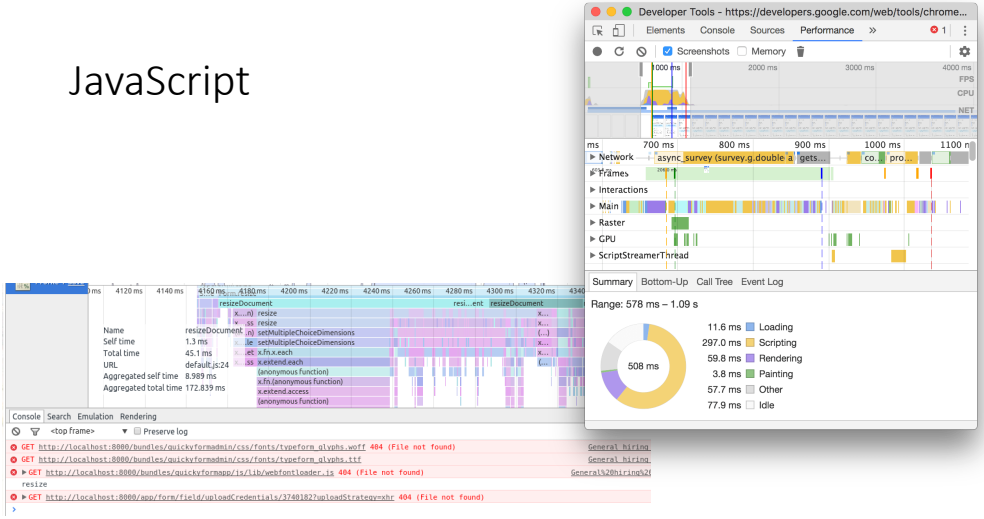
Python





https://docs.anaconda.com/accelerate/profiling/

JavaScript



https://developer.chrome.com/docs/devtools/

Technological foundations of software development

Debug, log, test, profile, analyze your software

Part 6: Static code analysis

ICM – Computer Science Major – Course unit on Technological foundations of computer science
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/

Dynamic vs. static analysis

Pros of dynamic analysis

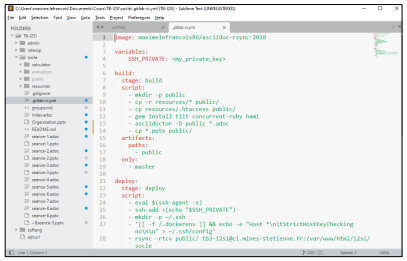
- is able to detect dependencies that cannot be detected in static analysis. Ex: dynamic dependencies by reflection, dependency injection, polymorphism.
- can collect temporal information.
- processes real input data. During static analysis it is difficult to impossible to know which files will be passed as input, which Web requests will come, which user will click, etc.

Cons of dynamic analysis

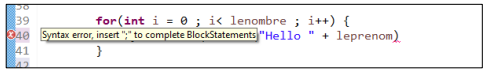
- may have a negative impact on the performance of the application.
- cannot guarantee complete coverage of the source code, as executed, is based on user interaction or automated testing.

Static code analysis

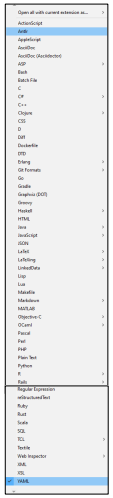
- Syntax checking
- Semantic check
- Linter
- Quick fixes



Syntax highlighting with Sublime Text



Syntactic validation syntaxique with Eclipse



list of available grammars

Static analysis tools

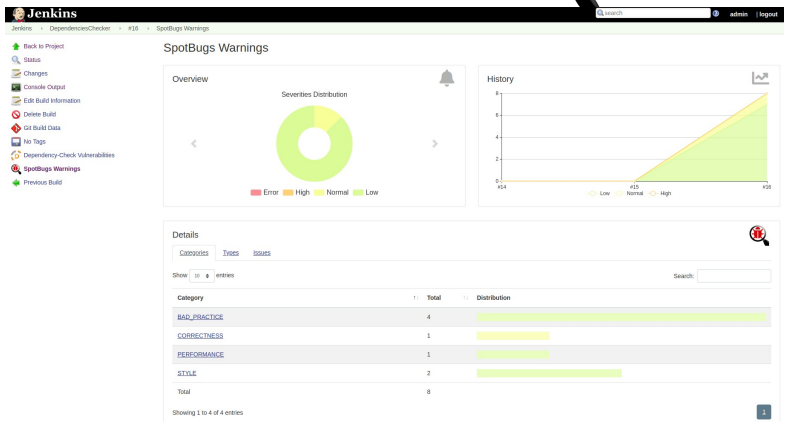
- Example for Java

Java [edit]

Tool	Latest release	Free software	Duplicate code	Notes
Checkstyle	2020-01-26	Yes; LGPL	No	Besides some static code analysis, it can be used to show violations of a configured coding standard. Duplicate code detection was removed[10] from Checkstyle.
Coverity	2017-01-19	No; Proprietary		Coverity is a static analysis and Static Application Security Testing (SAST) platform that finds critical defects and security weaknesses in code as it's written before they become vulnerabilities, crashes, or maintenan
Eclipse	2017-06-28	Yes; EPL	No	Cross-platform IDE with own set of several hundred code inspections available for analyzing code on-the-fly in the editor and bulk analysis of the whole project. Plugins for Checkstyle, FindBugs, and PMD.
FindBugs	2015-03-06	Yes; LGPL		Based on Jakarta BCEL from the University of Maryland. SpotBugs is the spiritual successor of FindBugs, carrying on from the point where it left off with support of its community.
Infer	2017-10-19	Yes; BSD with additional patent clause[9]		Developed by an engineering team at Facebook with open-source contributors. Targets null pointer exceptions, leaks, and thread safety issues.
IntelliJ IDEA	2021-04-06	Yes; ASL 2	Yes	A leading Java IDE with built-in code inspection and analysis. Plugins for Checkstyle, FindBugs, and PMD.
JArchitect	2017-06-11	No; Proprietary		Simplifies managing a complex code base by analyzing and visualizing code dependencies, defining design rules, doing impact analysis, and by comparing different versions of the code.
Jtest	2019-05-21	No; Proprietary	Yes	Testing and static code analysis product by Parasoft.
LDRA Testbed		No; Proprietary		Analysis and testing tool suite.
PMD	2020-10-24	Yes; BSD, ASL 2, LGPL	Yes	A static ruleset based source code analyzer that identifies potential problems.
RIPS	2019-01-07	No; Proprietary		Language-specific source code analysis solution with many integration options for accurate detection of complex security and quality issues.
Semgrep	2021-03-16 (0.43.0)	Yes; LGPL v2.1	No	A static analysis tool that helps expressing code standards and surfacing bugs early. A CI service and a rule library is also available.
Soot	2020-10-28	Yes; LGPL		A language manipulation and optimization framework consisting of intermediate languages.
Squale	2011-05-26	Yes; LGPL		A platform to manage software quality.
SourceMeter	2016-02-01	No; Proprietary	Yes	A platform-independent, command-line static source code analyzer.
ThreadSafe	2014-03-28	No; Proprietary		A static analysis tool focused on finding concurrency bugs.

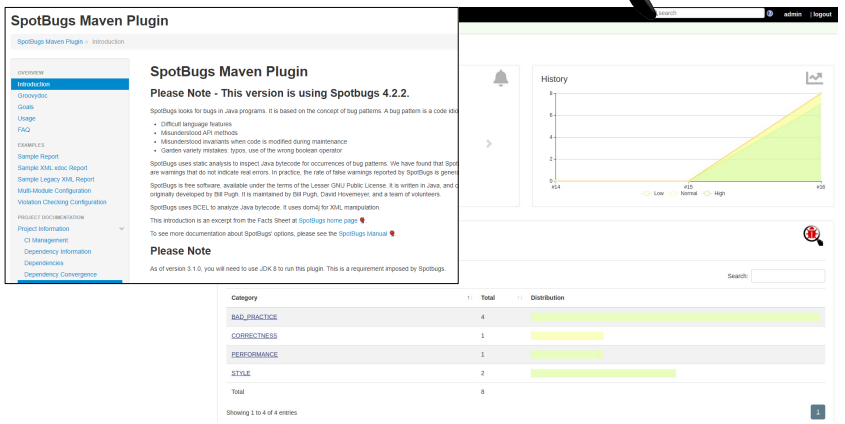
https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis

Static analysis tools- e.g. SpotBugs



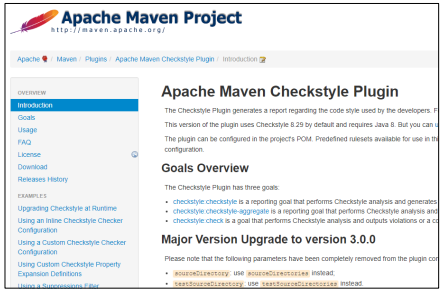
https://mip-cloud.gitlab.io/post/2018/12/spotbugs/

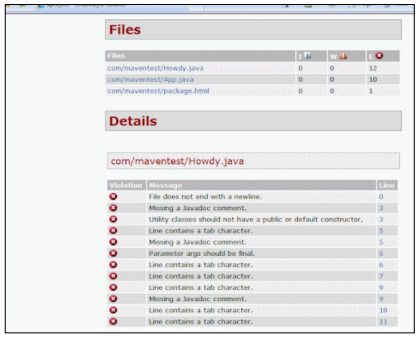
Static analysis tools- e.g. SpotBugs



https://spotbugs.github.io/spotbugs-maven-plugin/

Static analysis tools- e.g. checkstyle

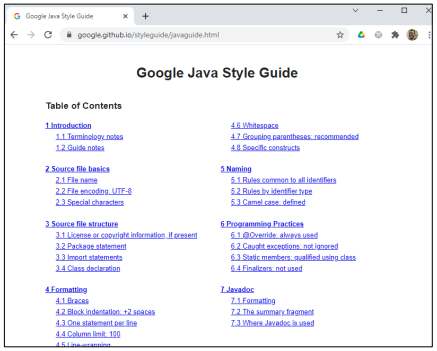


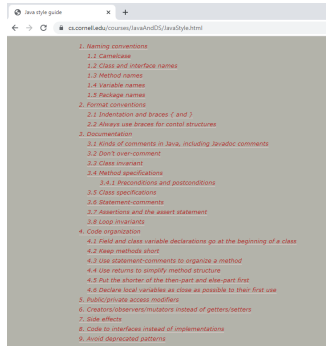


https://checkstyle.sourceforge.io/

Develop with style !

"There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. The other way is to make it so complicated that there are no obvious deficiencies." - C.A.R. Hoare.





Technological foundations of software development

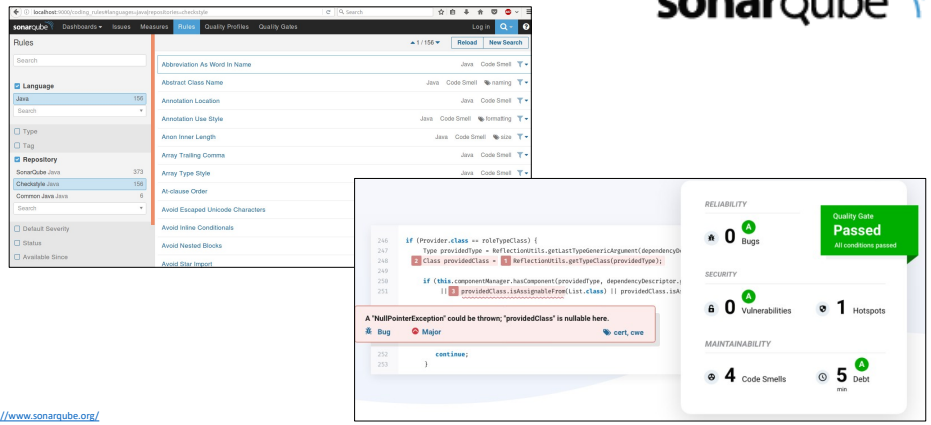
Debug, log, test, profile, analyze your software

Part 7: Integration in platforms

ICM – Computer Science Major – Course unit on Technological foundations of computer science
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/

Code quality and security analysis platform

sonarqube



https://www.sonarqube.org/

Technological foundations of software development

Debug, log, test, profile, analyze your software

ICM – Computer Science Major – Course unit on Technological foundations of computer science
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/

... Your turn

Complete the TODO section:

https://ci.mines-stetienne.fr/cps2/course/tfsd/course-5.html#_todos