# Software is eating the world

# Software is eating the world

## Most businesses and industries are being run on software


Music

Video

Movies


Books


Ads


Finance


Health


Education


Agriculture

...

# Software is eating the world

## Most businesses and industries are being run on software

Music · Video · Movies · Books · Ads · Finance · Health · Education · Agriculture · ...

## Devastating risks due to software failures

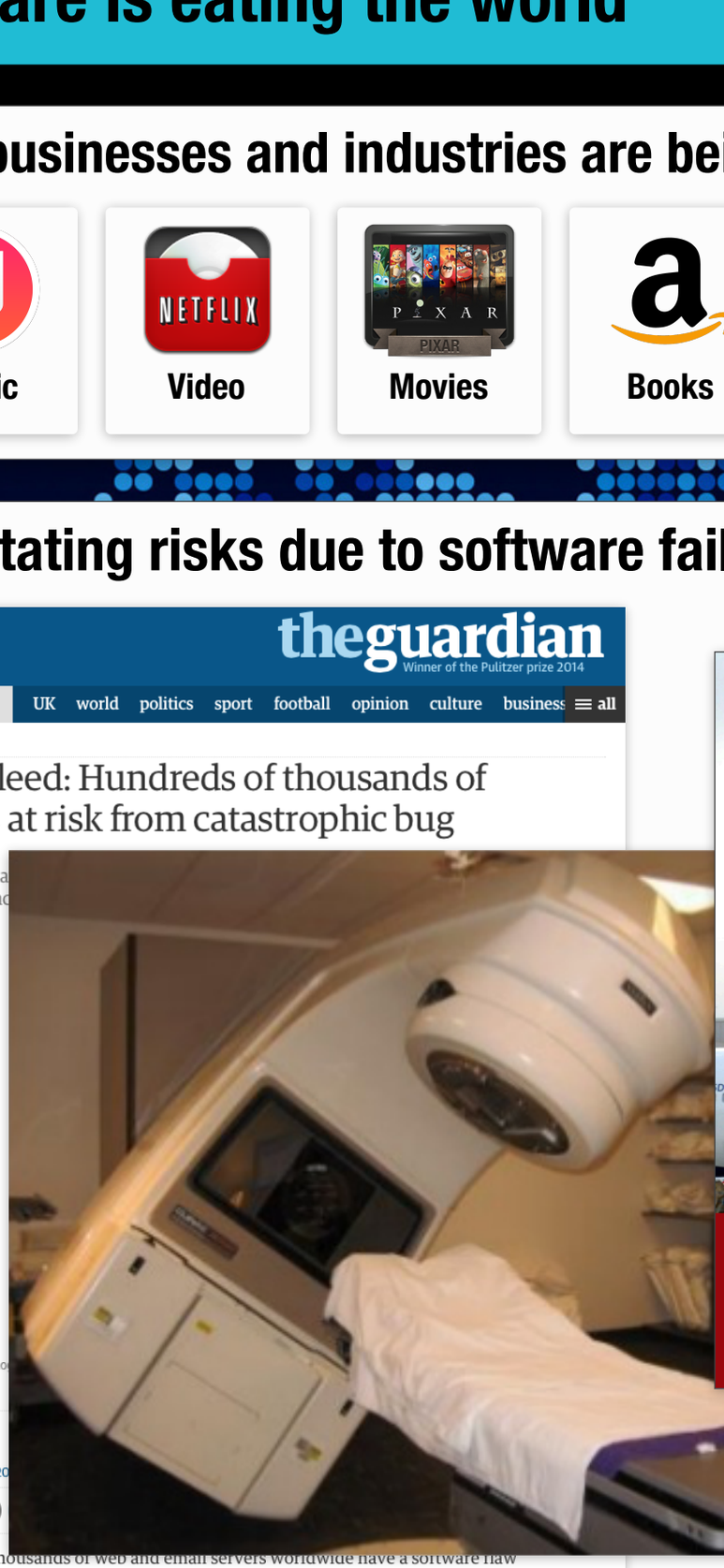
theguardian
Winner of the Pulitzer prize 2014

home › tech | UK | world | politics | sport | football | opinion | culture | business | all

Heartbleed

### Heartbleed: Hundreds of thousands of servers at risk from catastrophic bug

Code error mea
'heartbeat' func



The Heartbleed lo

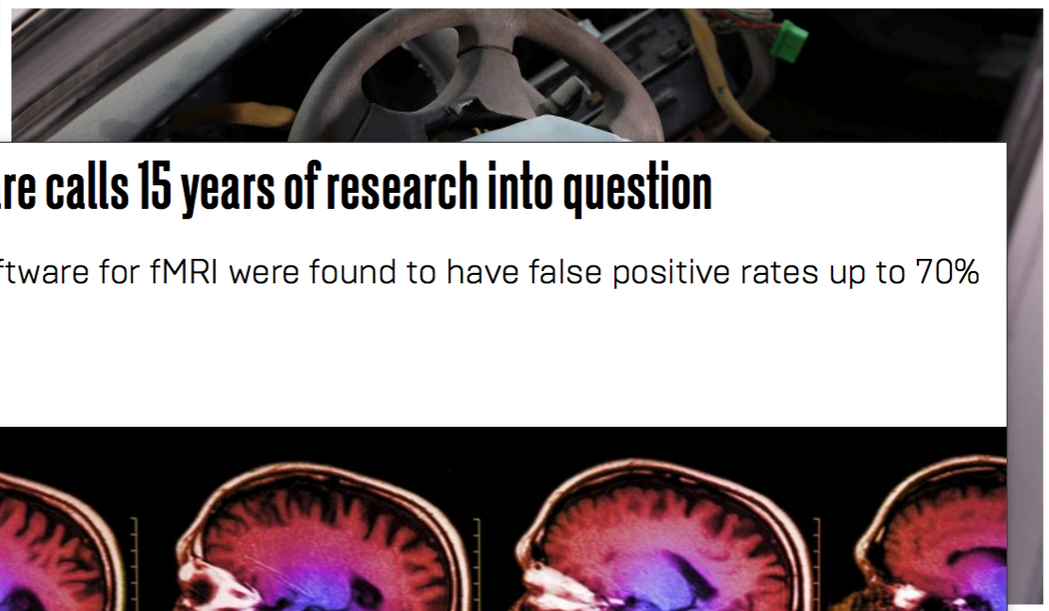
Alex Hern
@alexhern
Wednesday 9 April 20

Hundreds of thousands of web and email servers worldwide have a software flaw that lets attackers steal the cryptographic keys used to secure online commerce and web connections, experts say.

B
NE

### GM to Recall 4.3 Million Vehicles for Software-Related Airbag Defect

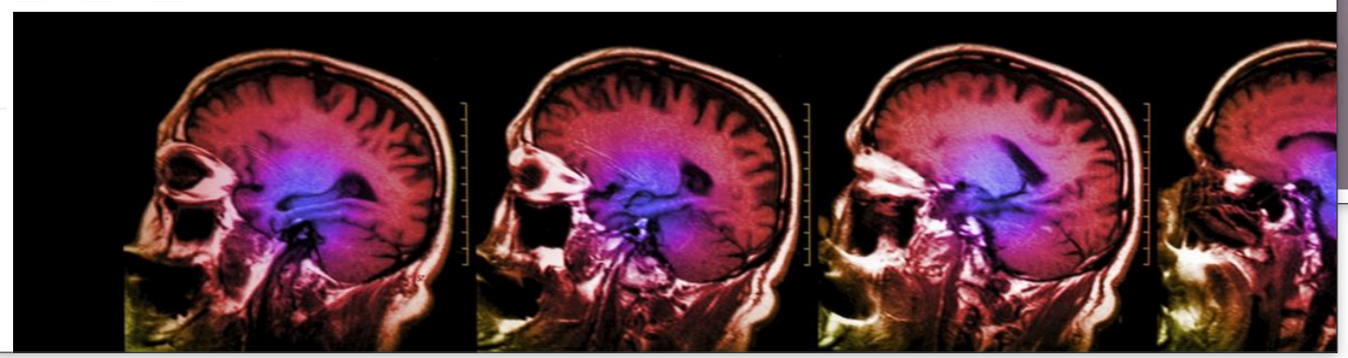
by Reuters SEPTEMBER 9, 2016, 12:37 PM EST



### Bug in fMRI software calls 15 years of research into question

Popular pieces of software for fMRI were found to have false positive rates up to 70%

*By* **EMILY REYNOLDS**
WIRED BRAINS



# Software is eating the world

## Most businesses and industries are being run on software

Music · Video · Movies · Books · Ads · Finance · Health · Education · Agriculture · ...

## Devastating risks due to software failures

theguardian
Winner of the Pulitzer prize 2014

home › tech | UK | world | politics | sport | football | opinion | culture | business | all

Heartbleed

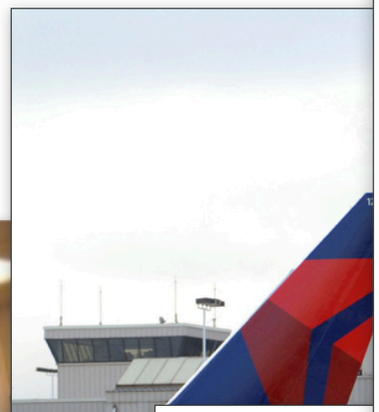
### Heartbleed: Hundreds of thousands of servers at risk from catastrophic bug

Code error mea
'heartbeat' func

The Heartbleed lo

Alex Hern
@alexhern
Wednesday 9 April 20

Hundreds of thousands of web and email servers worldwide have a software flaw that lets attackers steal the cryptographic keys used to secure online commerce and web connections, experts say.



Bug in fMRI

Popular piece

By EMILY REYNOL

WIRED

## Need for more of ICT professionals

TRUSTED INSIGHTS FOR COMPUTING'S LEADING PROFESSIONALS | ACM.org | Join ACM | About Communications | ACM Resources | Alerts & Feeds

COMMU

Home / News

ACM TECHNE

IDC S

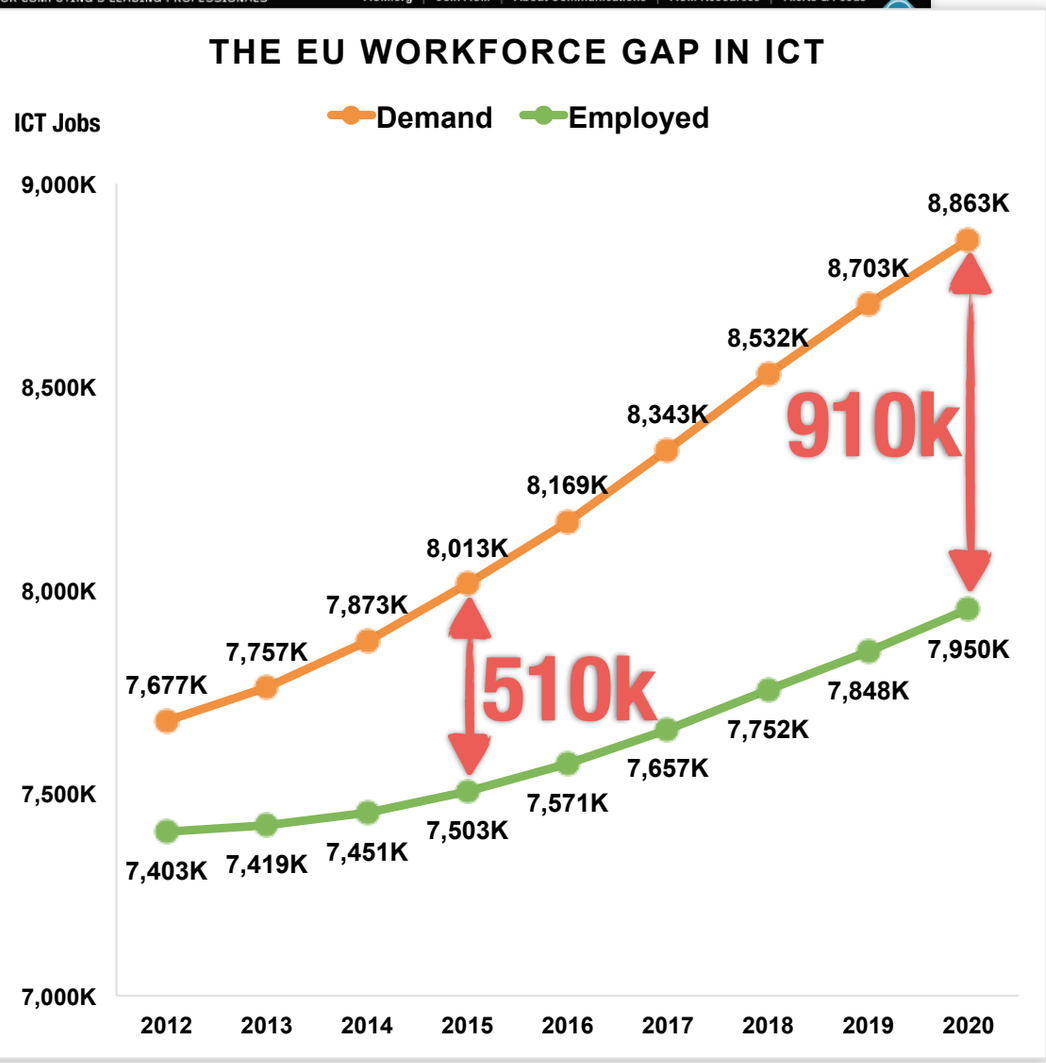
By Applicatio
January 21, 2
Comments

VIEW AS:

IDC's 2014 W
and ICT-Skille
there are abou
developers in
million hobbyis

Credit: Samsu

developer p
of the hobby



# Software is eating the world

# Software is eating the world

# Software Dependability

**Fabio Palomba**
**ZEST @ Ifi**
**University of Zurich**

# Who am I?


## 2013
## Master's Degree in Computer Science

- 
-

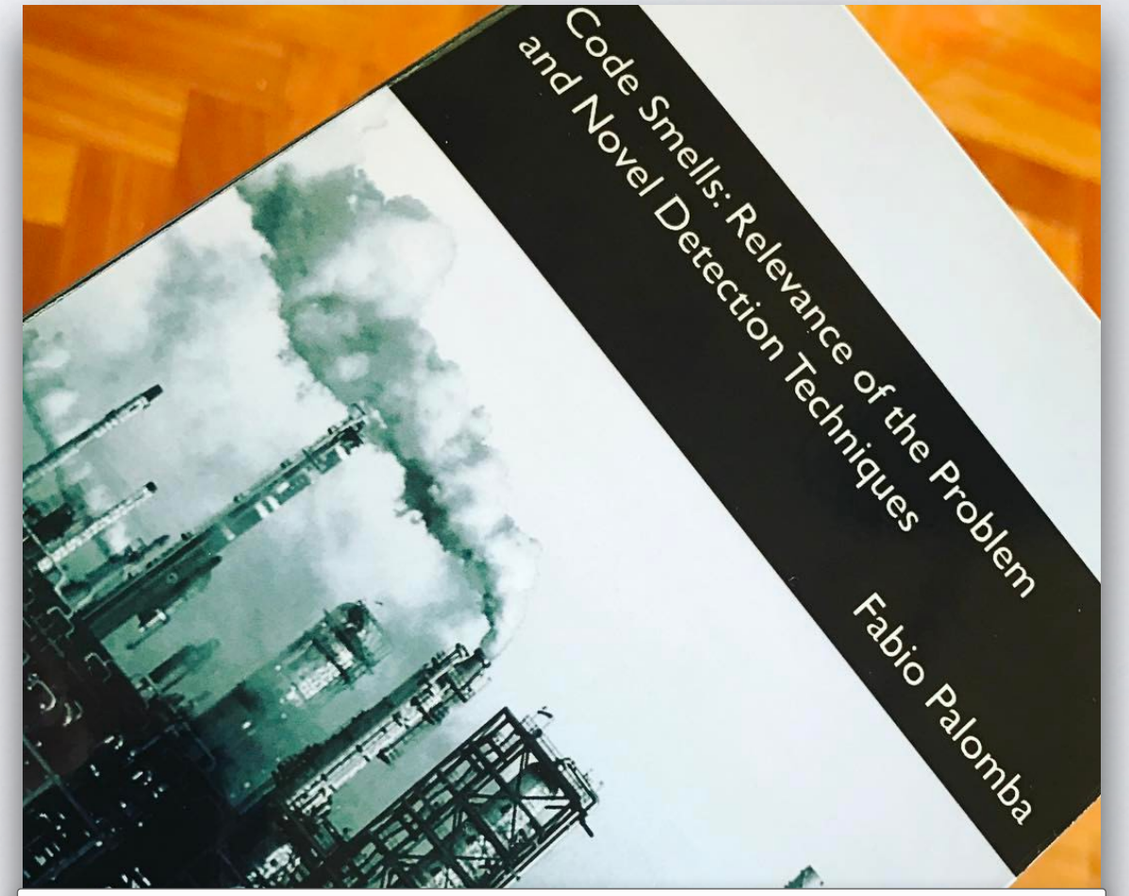
# Who am I?

## 2017
## European PhD Degree in Management and Information Technology

-
-
-

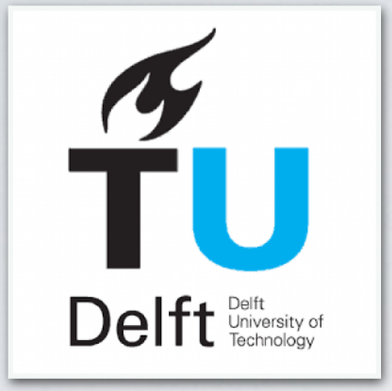
# Who am I?

## 2017
## European PhD Degree in Management and Information Technology




**IEEE Computer Society**
**Best PhD Thesis Award**

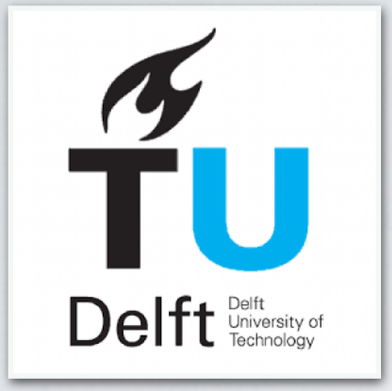
# Who am I?




## 2017
## Post-doctoral Researcher

-
-

# Who am I?




## 2017
## Post-doctoral Researcher

- 
- 
- 


## 2018 - current
## Senior Research Associate

# Who are we?

## ZEST

The **Z**urich **E**mpirical **S**oftware Engineering **T**eam
at the University of Zurich

### Education

- Software construction
- Data science for Software Engineering
- Advanced Software Engineering
- Software Testing

- B.Sc. and M.Sc. Theses
- M.Sc. projects

### Research

- Software quality
  - Peer code review
  - Software testing
  - Technical debt management
- Fundamentals of Data Science for Software Engineering
  - Predictive Analytics
  - Mining Unstructured Data
  - Large-scale API Mining

# Who are we?


Google ABB HUAWEI AdNovum

Microsoft moz://a Software Improvement Group

# Who are we?

**Domenico Serra** — “On the Effectiveness of Manual and Automatic Unit Test Generation: Ten Years Later” Published at MSR’19.

**Fernando Petrulio** — “The Developer’s Perspective of Lambdified APIs” Submitted to TSE.

**Roberta Guadagni** — “The Secret Life of Software Vulnerabilities: An Empirical Study”

---

**Fabiano Pecorelli** — Working on improving test code quality

# administrative information: general information

| | |
|---|---|
| **lecturer** | Fabio Palomba |
| **teaching assistant** | Yours truly |
| **lectures** | Mon, Tue, Thu, Frid - This week;<br>Mon, Tue, Thu - Next week. |
| **GitHub repository** | https://github.com/fpalomba/SWDependability-Unisa2019 |
| **email** | palomba@ifi.uzh.ch<br>fpalomba@unisa.it |
| **email hours** | Mon—Fri 12:30 - 13:30 |
| **office hours** | By appointment |

# administrative information: Table Of Contents

| Content | Title |
|---|---|
| Lecture 1 | Prologue & Dependability Requirements |
| Lecture 2 | Bad Code Smells |
| Lecture 3 | Software Security |
| Lecture 4 | Machine Learning for Dependability |
| Lecture 5 | Software Defect Prediction |
| Lecture 6 | Software Metrics, Traceability and Impact Analysis |
| Lecture 7 | Software Dependability Hands-on |
| Lecture 8 | DevOps and Organisation Aspects |

# administrative information: Table Of Contents

| Content | Title |
|---|---|
| Lecture 1 | Prologue & Dependability Requirements |
| Lecture 2 | Bad Code Smells |
| Lecture 3 | Software Security |
| Lecture 4 | Machine Learning for Dependability |
| **Lecture 5** | **Software Defect Prediction [Dr. Pascarella - TU Delft]** |
| Lecture 6 | Software Metrics, Traceability and Impact Analysis |
| Lecture 7 | Software Dependability Hands-on |
| **Lecture 8** | **DevOps and Organisation Aspects [Dr. Tamburri - TU Eindhoven]** |

# administrative information: Table Of Contents

| Content | Title |
|---|---|
| Lecture 1 | Prologue & Dependability Requirements |
| Lecture 2 | Bad Code Smells |
| ***Lecture 3*** | ***Software Security [Flipped Classroom]*** |
| Lecture 4 | Machine Learning for Software Dependability |
| **Lecture 5** | **Software Defect Prediction [Dr. Pascarella - TU Delft]** |
| Lecture 6 | Software Metrics, Traceability and Impact Analysis |
| ***Lecture 7*** | ***Software Dependability Hands-on [Practicum]*** |
| **Lecture 8** | **DevOps and Organisation Aspects [Dr. Tamburri - TU Eindhoven]** |

# administrative information: Table Of Contents

| Content | Title |
|---|---|
| Lecture 1 | Prologue & Dependability Requirements |
| Lecture 2 | Bad Code Smells |
| ***Lecture 3*** | ***Software Security [Flipped Classroom]*** |
| Lecture 4 | Machine Learning for Dependability |
| **Lecture 5** | **Software Defect Prediction [Dr. Pascarella - TU Delft]** |
| Lecture 6 | Software Metrics, Traceability and Impact Analysis |
| ***Lecture 7*** | ***Software Dependability Hands-on [Practicum]*** |
| **Lecture 8** | **DevOps and Organisation Aspects [Dr. Tamburri - TU Eindhoven]** |

**Midterm exam - Multiple Choice + Open Questions + Oral (optional)**

# Software dependability

## Definition

# Software dependability

## Definition

The dependability of a system reflects the extent of the user's confidence when operating and interacting with the the system.

# Software dependability

## Definition

The dependability of a system reflects the extent of the user's confidence when operating and interacting with the the system.

A system can be considered to be dependable if it operates without interruption, delivers the services that are expected by the stakeholders, does not have adverse effects on the system's environment and does not damage its data or the system itself.

# Software dependability

**Definition**

The dependability of a system reflects the extent of the user's confidence when operating and interacting with the the system.

A system can be considered to be dependable if it operates without interruption, delivers the services that are expected by the stakeholders, does not have adverse effects on the system's environment and does not damage its data or the system itself.

**Dependability covers:**

- *Software maintainability*: the extent to which a system can be adapted/modified to accomodate new change requests;

# Software dependability

**Definition**

The dependability of a system reflects the extent of the user's confidence when operating and interacting with the the system.

A system can be considered to be dependable if it operates without interruption, delivers the services that are expected by the stakeholders, does not have adverse effects on the system's environment and does not damage its data or the system itself.

**Dependability covers:**

- *Software maintainability*: the extent to which a system can be adapted/modified to accomodate new change requests;
- *Software reliability*: the extent to which a system is reliable with respect to the expected behaviour;

# Software dependability

**Definition**

The dependability of a system reflects the extent of the user's confidence when operating and interacting with the the system.

A system can be considered to be dependable if it operates without interruption, delivers the services that are expected by the stakeholders, does not have adverse effects on the system's environment and does not damage its data or the system itself.

**Dependability covers:**

- *Software maintainability*: the extent to which a system can be adapted/modified to accomodate new change requests;
- *Software reliability*: the extent to which a system is reliable with respect to the expected behaviour;
- *Software availability*: the extent to which a system remains available for its users;

# Software dependability

## Definition

The dependability of a system reflects the extent of the user's confidence when operating and interacting with the the system.

A system can be considered to be dependable if it operates without interruption, delivers the services that are expected by the stakeholders, does not have adverse effects on the system's environment and does not damage its data or the system itself.

## Dependability covers:

- *Software maintainability*: the extent to which a system can be adapted/modified to accomodate new change requests;
- *Software reliability*: the extent to which a system is reliable with respect to the expected behaviour;
- *Software availability*: the extent to which a system remains available for its users;
- *Software security*: the extent to which a system can keep data of its users safe and ensures the safety of their users.

# Software dependability

## Definition

The dependability of a system reflects the extent of the user's confidence when operating and interacting with the the system.

A system can be considered to be dependable if it operates without interruption, delivers the services that are expected by the stakeholders, does not have adverse effects on the system's environment and does not damage its data or the system itself.

## Dependability covers:

- *Software maintainability*: the extent to which a system can be adapted/modified to accomodate new change requests;
- *Software reliability*: the extent to which a system is reliable with respect to the expected behaviour;
- *Software availability*: the extent to which a system remains available for its users;
- *Software security*: the extent to which a system can keep data of its users safe and ensures the safety of their users.

# Software dependability

## Definition

The dependability of a system reflects the extent of the user's confidence when operating and interacting with the the system.

A system can be considered to be dependable if it operates without interruption, delivers the services that are expected by the stakeholders, does not have adverse effects on the system's environment and does not damage its data or the system itself.

**Dependability covers:**

- Safe system operation depends on the system being available and operating reliably;
- A system might be unreliable because its data has been corrupted by an external attack or because of poor implementation solutions applied by developers.

# Software dependability

## Dependability is subjective

- The dependability of a system is not absolute but depends on the judgement of the stakeholders, e.g., what seems to be a failure to one stakeholder represents a normal behaviour to another;

# Software dependability

## Dependability is subjective

- The dependability of a system is not absolute but depends on the judgement of the stakeholders, e.g., what seems to be a failure to one stakeholder represents a normal behaviour to another;

## Dependability types

- Functional vs non-functional requirements;
- In case a stakeholder explicitly refers to dependability aspects, you have a functional dependability requirement, e.g., all data must be maintained on a local disk;
- In case you have requirements that specify dependability properties of the system, you have a non-functional dependability requirement, e.g., the probability of failure should be 0.001.

# Software dependability

## Dependability is subjective

- The dependability of a system is not absolute but depends on the judgement of the stakeholders, e.g., what seems to be a failure to one stakeholder represents a normal behaviour to another;

## Dependability types

- Functional vs non-functional requirements;
- In case a stakeholder explicitly refers to dependability aspects, you have a functional dependability requirement, e.g., all data must be maintained on a local disk;
- In case you have requirements that specify dependability properties of the system, you have a non-functional dependability requirement, e.g., the probability of failure should be 0.001.

## Dependability is part of the business

- Dependability issues should be considered alongside other business goals, as they can automatically lead to organisational problems.

# Software dependability - A closer look

## Software maintainability

- During software maintenance and evolution, developers apply continuous changes to implement new features and fix defects. Because of close deadlines and/or stress, they often do not have time or are not willing to spend effort in designing code changes;

# Software dependability - A closer look

## Software maintainability

- During software maintenance and evolution, developers apply continuous changes to implement new features and fix defects. Because of close deadlines and/or stress, they often do not have time or are not willing to spend effort in designing code changes;
- This may lead to the emergence of technical debt, i.e., *not-quite-right* code that you make right to complete a task;

# Software dependability - A closer look

## Software maintainability

- During software maintenance and evolution, developers apply continuous changes to implement new features and fix defects. Because of close deadlines and/or stress, they often do not have time or are not willing to spend effort in designing code changes;
- This may lead to the emergence of technical debt, i.e., *not-quite-right* code that you make right to complete a task;
- There are several forms of technical debt, depending on the development phase in which it is introduced, e.g., design-debt, requirement-debt, test-debt.

# Software dependability - A closer look

## Software maintainability

- During software maintenance and evolution, developers apply continuous changes to implement new features and fix defects. Because of close deadlines and/or stress, they often do not have time or are not willing to spend effort in designing code changes;
- This may lead to the emergence of technical debt, i.e., *not-quite-right* code that you make right to complete a task;
- There are several forms of technical debt, depending on the development phase in which it is introduced, e.g., design-debt, requirement-debt, test-debt.
- We are going to specifically study design debt, and in particular bad code smells, i.e., symptoms of the presence of poor design or implementation choices applied by developers when evolving a software system;

# Software dependability - A closer look

## Software maintainability

- During software maintenance and evolution, developers apply continuous changes to implement new features and fix defects. Because of close deadlines and/or stress, they often do not have time or are not willing to spend effort in designing code changes;
- This may lead to the emergence of technical debt, i.e., *not-quite-right* code that you make right to complete a task;
- There are several forms of technical debt, depending on the development phase in which it is introduced, e.g., design-debt, requirement-debt, test-debt.
- We are going to specifically study design debt, and in particular bad code smells, i.e., symptoms of the presence of poor design or implementation choices applied by developers when evolving a software system;
- We will also take a look on program refactoring, namely the actions to make in order to remove code smells from source code;

# Software dependability - A closer look

## Software maintainability

- During software maintenance and evolution, developers apply continuous changes to implement new features and fix defects. Because of close deadlines and/or stress, they often do not have time or are not willing to spend effort in designing code changes;
- This may lead to the emergence of technical debt, i.e., *not-quite-right* code that you make right to complete a task;
- There are several forms of technical debt, depending on the development phase in which it is introduced, e.g., design-debt, requirement-debt, test-debt.
- We are going to specifically study design debt, and in particular bad code smells, i.e., symptoms of the presence of poor design or implementation choices applied by developers when evolving a software system;
- We will also take a look on program refactoring, namely the actions to make in order to remove code smells from source code;
- This will also be the subject of the hands-on.

# Software dependability - A closer look

## Software reliability

- The introduction of defects is, unfortunately, unavoidable. Software reliability measures how robust a system is to defects;

# Software dependability - A closer look

## Software reliability

- The introduction of defects is, unfortunately, unavoidable. Software reliability measures how robust a system is to defects;
- Besides testing, that you are going to further explore during the second part of this course, other techniques can be applied to control for defects;

# Software dependability - A closer look

## Software reliability

- The introduction of defects is, unfortunately, unavoidable. Software reliability measures how robust a system is to defects;
- Besides testing, that you are going to further explore during the second part of this course, other techniques can be applied to control for defects;
- We are going to investigate defect prediction, i.e., the set of golden-rules for defining and validating supervised machine learning model that can predict the existence of defects in source code;

# Software dependability - A closer look

## Software reliability

- The introduction of defects is, unfortunately, unavoidable. Software reliability measures how robust a system is to defects;
- Besides testing, that you are going to further explore during the second part of this course, other techniques can be applied to control for defects;
- We are going to investigate defect prediction, i.e., the set of golden-rules for defining and validating supervised machine learning model that can predict the existence of defects in source code;
- We are also going to study source code metrics, that can be used to establish the portions of source code that are less maintainable and reliable;

# Software dependability - A closer look

## Software reliability

- The introduction of defects is, unfortunately, unavoidable. Software reliability measures how robust a system is to defects;
- Besides testing, that you are going to further explore during the second part of this course, other techniques can be applied to control for defects;
- We are going to investigate defect prediction, i.e., the set of golden-rules for defining and validating supervised machine learning model that can predict the existence of defects in source code;
- We are also going to study source code metrics, that can be used to establish the portions of source code that are less maintainable and reliable;
- This will also be the subject of the hands-on.

# Software dependability - A closer look

## Software availability

- Naturally, to be used and usable, a software system must be available, i.e., a software must run and be available to users most of the time;

# Software dependability - A closer look

## Software availability

- Naturally, to be used and usable, a software system must be available, i.e., a software must run and be available to users most of the time;
- This dimension is closely connected to fault tolerance and reliability: if a software does not fail, it is available;

# Software dependability - A closer look

## Software availability

- Naturally, to be used and usable, a software system must be available, i.e., a software must run and be available to users most of the time;
- This dimension is closely connected to fault tolerance and reliability: if a software does not fail, it is available;
- Availability can be stimulated and addressed through data redundancy techniques and fault recovery approaches;

# Software dependability - A closer look

## Software availability

- Naturally, to be used and usable, a software system must be available, i.e., a software must run and be available to users most of the time;
- This dimension is closely connected to fault tolerance and reliability: if a software does not fail, it is available;
- Availability can be stimulated and addressed through data redundancy techniques and fault recovery approaches;
- System availability can be measured by counting the number of times a system is functioning. A widely-known metric is (1 - (down-time/total-time)).

# Software dependability - A closer look

## Software availability

- Naturally, to be used and usable, a software system must be available, i.e., a software must run and be available to users most of the time;
- This dimension is closely connected to fault tolerance and reliability: if a software does not fail, it is available;
- Availability can be stimulated and addressed through data redundancy techniques and fault recovery approaches;
- System availability can be measured by counting the number of times a system is functioning. A widely-known metric is (1 - (down-time/total-time)).
- We are NOT going to investigate this further, but we rather focus on other aspects that can indirectly address availability (e.g., defect prediction);

# Software dependability - A closer look

## Software security

- Software vulnerabilities are patterns that can be maliciously used by others to perform attacks that can impact reliability, availability, and safety;

# Software dependability - A closer look

## Software security

- Software vulnerabilities are patterns that can be maliciously used by others to perform attacks that can impact reliability, availability, and safety;
- Security is somehow connected to maintainability, as indicates wrong design or implementation choices applied by developers;

# Software dependability - A closer look

## Software security

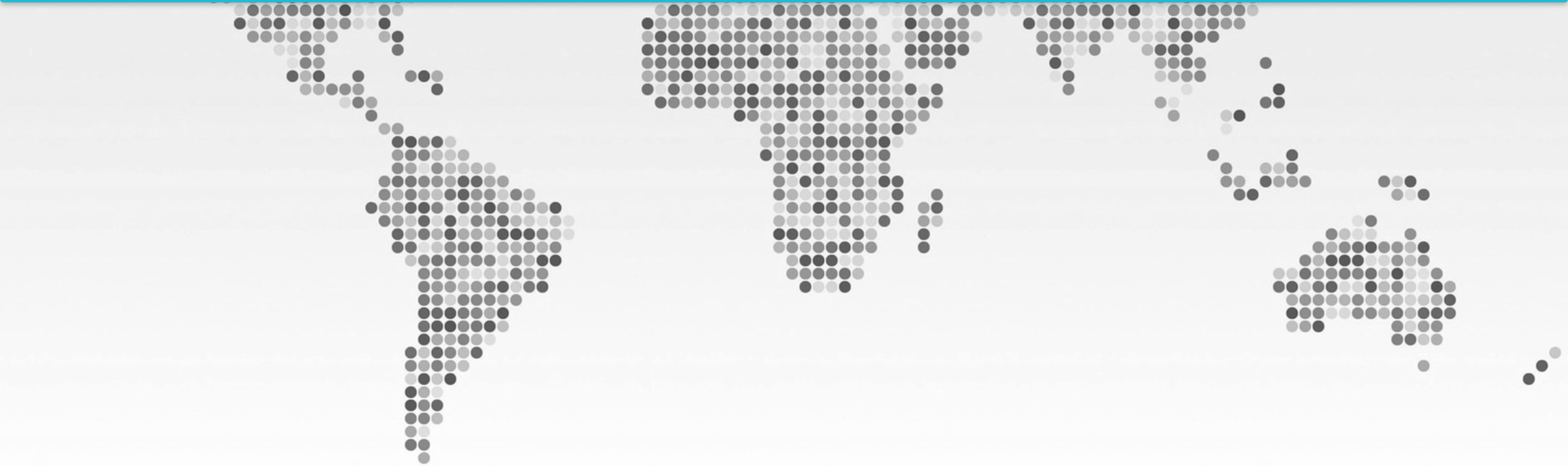
- Software vulnerabilities are patterns that can be maliciously used by others to perform attacks that can impact reliability, availability, and safety;
- Security is somehow connected to maintainability, as indicates wrong design or implementation choices applied by developers;
- The area of vulnerabilities is still full of open challenges and opportunities in research, as there are no fully reliable tools to identify and refactor potentially vulnerable pieces of code;

# Software dependability - A closer look

## Software security

- Software vulnerabilities are patterns that can be maliciously used by others to perform attacks that can impact reliability, availability, and safety;
- Security is somehow connected to maintainability, as indicates wrong design or implementation choices applied by developers;
- The area of vulnerabilities is still full of open challenges and opportunities in research, as there are no fully reliable tools to identify and refactor potentially vulnerable pieces of code;
- We are going to study some of these open challenges through the analysis of the state of the art. You will have a look to empirical studies and potentially useful techniques that you are required to understand and analyse with respect to their limitations and possible improvements.

# End of the first part



# Software dependability - Getting ready for practical stuff

## A simple use case scenario

**https://github.com/SERG-Delft/jpacman-framework**

- Cloning and setting JPacman; Run the test;
- A quick look to Git and its features.

## To run the Linux command line:

- Cygwin: **https://www.cygwin.com**