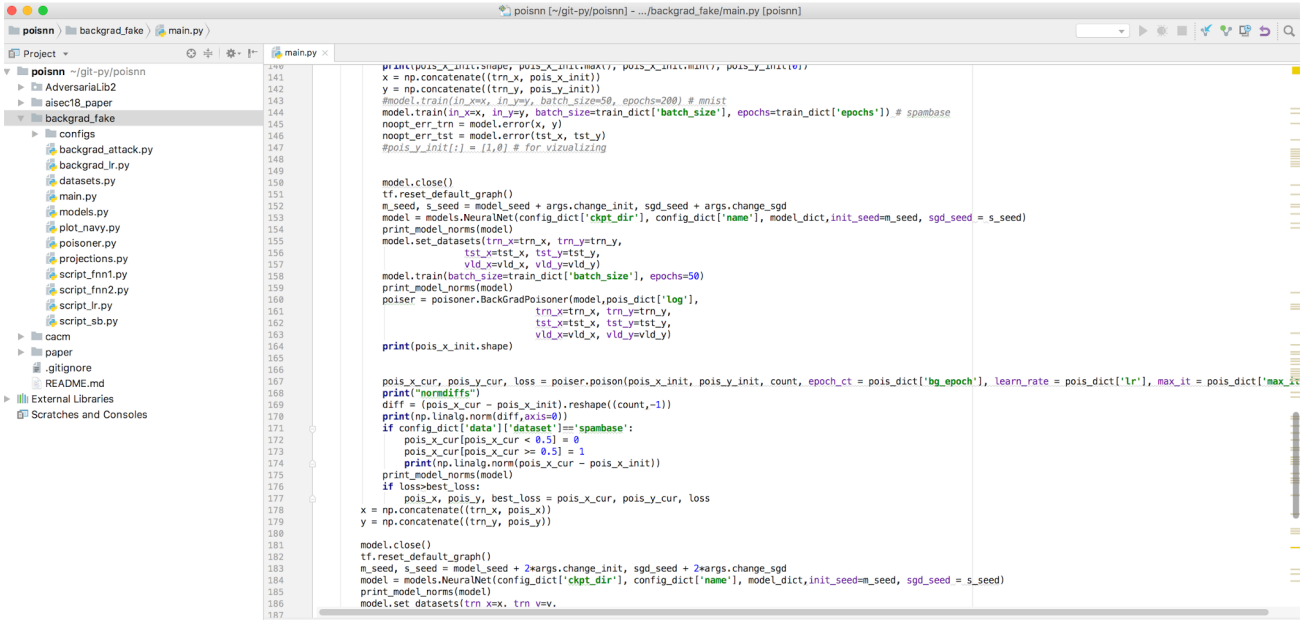
Industrial Software Development (ISDe)

Instructors
Battista Biggio and **Luca Didaci**

M.Sc. in Computer Engineering, Cybersecurity and Artificial Intelligence
University of Cagliari, Italy

Why Industrial Software Development (ISDe)?

- Software development as a (*young*) student / researcher is mostly learning and prototyping ideas in an unstructured manner



How far can we get?

Image credit: https://medium.com/@gaperton/software-managing-the-complexity-caff5c4964cf

Why Industrial Software Development (ISDe)?

- Complexity of modern, industrial and open-source software projects cannot be managed as an unstructured process
  - it requires significant planning and managing work before writing even a single line of code!
  - (... and we have not yet even considered interactions with customers / stakeholders …)

- Some examples (in the area of computer vision, machine/deep learning and AI)
  - OpenCV https://github.com/opencv/opencv
  - Scikit-learn https://github.com/scikit-learn/scikit-learn
  - Tensorflow https://github.com/tensorflow/tensorflow
  - PyTorch https://github.com/pytorch/pytorch

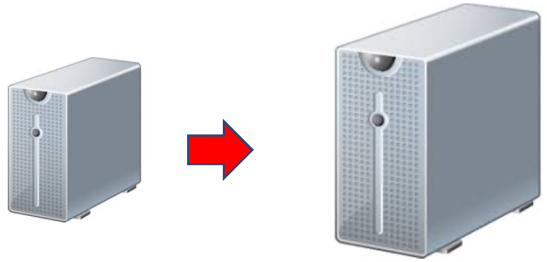
Things are getting even more complex...

- AI has shown that very valuable information can be extracted from the vast amount of data available nowadays in many applications

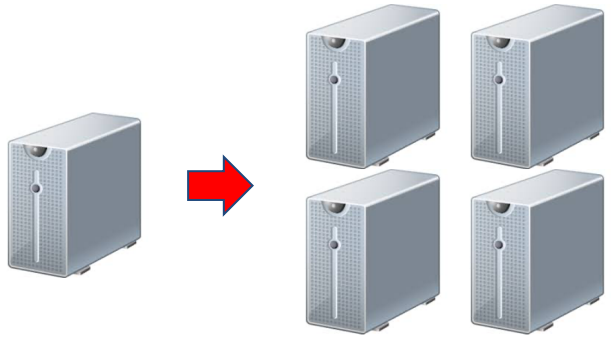
Data created → **Data stored** → **Data analysis / AI** → **Actionable knowledge**

Vertical vs Horizontal Scalability

- **Vertical scalability**: increasing resources on a single machine - RAM, CPU, hard disk,...
  - no longer enough to cope with large data volumes and large predictive models (deep networks)



- **Horizontal scalability**: execute software and scale its execution on multiple machines, depending on the requested processing power
  - software has to be executed on heterogeneous architectures (e.g., cloud, data centers)



ISDe: Main lessons to be learned

- Software can not be designed without following a systematized/managed procedure
  - It requires a massive dose of engineering and planning
  - Accounting for scalability / architectural requirements, etc.

- There cannot be improvisation. No need to reinvent the wheel
  - Be sure to adopt / re-use existing solutions to known problems!

- We will show you how to design software and algorithms in a principled manner
  - Software development / engineering is not just coding/programming!

- It is about defining a set of principles/values and a methodology to:
  - negotiate the required features that a piece of software should have with the customers,
  - create/update it using a well-defined strategy that accounts for cost and effort, and
  - plan its releases / updates in advance.

Why Industrial Software Development (ISDe)?

- Learning and prototyping is different from developing industrial applications
  - Dealing with increasing complexity requires a standardized approach to managing projects

- **Software Development / Engineering**: set of tools and methodological approaches to plan, develop, test and maintain (complex) software projects

Goal of the Course

- **Software Development / Engineering**: set of tools and methodological approaches to plan, develop, test and maintain complex software projects
  - Programming is not a trivial, unstructured process!

- **Methodologies**
  - Agile principles, tools and methodologies for software development/engineering
  - DevOps principles and basic aspects

- **Programming basics**
  - Properly define, write and test algorithms
  - Object-oriented programming (OOP) abstractions
  - Design Patterns
  - Test-driven Development (TDD)

- We will use *Python* as our main programming language in this course, but these principles apply to other languages as well!

Why Python?

- Pros
  - Easy to learn, interpreted language
  - Fast prototyping and execution
  - Many efficient third-party libraries (e.g., image processing, machine learning, etc.)
  - Wrappers to code written in different languages (mostly C/C++ libraries for efficient numerical computations - e.g., LAPACK)

- Cons
  - No strict control on types (sometimes causing unwanted behavior rather than raising error)
  - Memory management (not immediately clear when copying data or not…)

Good trade-off between efficiency and effort (no need to reinvent the wheel)!

Tentative Outline of the Course (8 CFU / 80 hours)

- Software Engineering (Tools and Methodologies) ~ 10 hours
  - Agile manifesto / principles, and main methodologies (Scrum, Kanban)
- Basics of (Python) Programming ~ 20 hours
  - Basic data structures and algorithmic design
  - Object-oriented programming (OOP)
- Design Patterns and UML (Unified Modeling Language) ~ 20 hours
- Test-Driven Development (TDD) ~ 10 hours
- Practical Sessions and Exercises ~ 6-12 hours
- Seminars ~ 6-8 hours
  - Git (for versioning and collaborative development)
  - Developing (web) applications (in the cloud)

What are you expected to know?

- **Prerequisites**
  - Basic knowledge of object-oriented programming (OOP)
  - Basic knowledge / understanding of C, C++ and Java
  - Ability to write simple programs in Python

- No worries. We have prepared a small introduction to **Python** for this course.
  - but please be ready to use it and practice during the course!

What are you expected to learn?

- **Software development / engineering:** set of tools and methodological approaches to plan, develop, test and maintain complex software projects
  - Agile principles and methodology
  - Test-driven development (TDD)
  - DevOps basics and Git

- **Programming**
  - Object-oriented programming (OOP) and abstractions
  - Design patterns
  - Code testing (using TDD as a methodological example)

Tools that will be used in the course

- PyCharm (IDE) with Python 3
  - The required Python libraries will be listed in the next lectures
- Git (for code versioning) - already integrated / supported in PyCharm
- Google Colab (Python notebook to run/edit code in your browser)

How to Pass the Exam

- Home computer-exercise assignment (project) + Oral examination
  - You can do a written assessment at the end of the course (once!) instead of the oral examination
  - You can do the oral examination before the computer exercise
  - Teams of 3 students maximum can do the home computer exercise

- Grading policy = Computer exercise (10/30) + Oral examination (20/30)

- **Written exam** (only once *right after the course)*
  - OOP, design patterns and testing (Python programming)