

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

```
poisnn (~git-py/poissn) .../backgrad_fake/main.py [poisnn]
Project + main.py x
poisnn ~/git-py/poissn
AdversarialLib2
aisec18_paper
backgrad_fake
  configs
    backgrad_attack.py
    backgrad_lr.py
  datasets.py
  main.py
  models.py
  plot_navy.py
  poisoner.py
  projections.py
  script_fn1.py
  script_fn2.py
  script_lr.py
  script_sb.py
  cachm
  paper
  .gitignore
  README.md
External Libraries
Scratches and Consoles

149 print(pois_x_init.shape, pois_x_initmax(), pois_x_initmin(), pois_y_init)
150
151 x = np.concatenate((trn_x, pois_x_init))
152 y = np.concatenate((trn_y, pois_y_init))
153 #model.train(in_mx, in_yy, batch_size=50, epochs=200) # mnist
154 model.train(in_mx, in_yy, batch_size=50, epochs=50) # spbase
155 #model.set_train_dict('batch_size', epochs=train_dict['epochs']) # spbase
156 noopt_err_trn = model.error(x, y)
157 noopt_err_tst = model.error(ts_x, ts_y)
158 #pois_y_init[:] = [1,0] # for visualizing
159
160 model.close()
161 tf.reset_default_graph()
162 m_seed, s_seed = model._seed + args.change_init, sgd._seed + args.change_sgd
163 model = models.NeuralNet(config_dict['ckpt_dir'], config_dict['name'], model_dict, init_seed=m_seed, sgd._seed = s_seed)
164 print(model)
165 model.set_datasets(trn_x, trn_y, trn_x, trn_y, ts_x, ts_y, ts_x, ts_y, val_x, val_y, val_x, val_y)
166 model.train(batch_size=train_dict['batch_size'], epochs=50)
167 print_model_norms(model)
168 poiser = poisoner.BackGradPoisoner(model, pois_dict['log'],
169                                     trn_x, trn_y, ts_x, ts_y, val_x, val_y)
170
171 print(pois_x_init)
172
173 pois_x_cur, pois_y_cur, loss = poiser.poison(pois_x_init, pois_y_init, count, epoch_ct = pois_dict['bg_epoch'], learn_rate = pois_dict['lr'], max_it = pois_dict['max_it'])
174 print("normdiff")
175 diff = (pois_x_cur - pois_x_init).reshape((count,-1))
176 print(np.linalg.norm(diff, axis=0))
177 if config_dict['data']['dataset']=='spbase':
178     pois_x_cur[pois_x_cur < 0.5] = 0
179     pois_x_cur[pois_x_cur >= 0.5] = 1
180     print(np.linalg.norm(pois_x_cur - pois_x_init))
181     print_model_norms(model)
182     if loss>best_loss:
183         best_loss = loss
184         pois_x, pois_y, best_loss = pois_x_cur, pois_y_cur, loss
185         x = np.concatenate((trn_x, pois_x))
186         y = np.concatenate((trn_y, pois_y))
187
188 model.close()
189 tf.reset_default_graph()
190 m_seed, s_seed = model._seed + 2*args.change_init, sgd._seed + 2*args.change_sgd
191 model = models.NeuralNet(config_dict['ckpt_dir'], config_dict['name'], model_dict, init_seed=m_seed, sgd._seed = s_seed)
192 print_model_norms(model)
193 model.set_datasets(trn_x, trn_y, vvv
```

How far can we get?

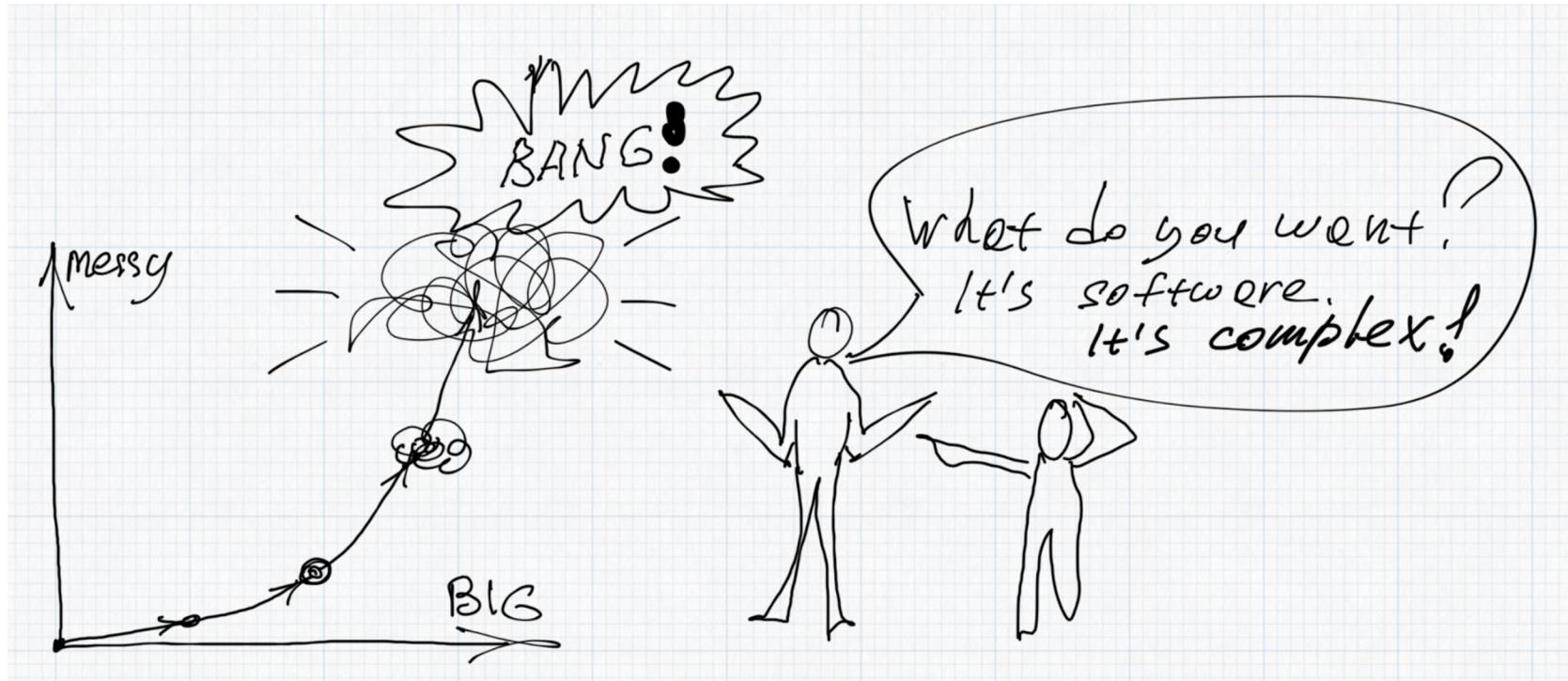


Image credit: <https://medium.com/@gaperton/software-managing-the-complexity-caff5c4964cf>

Why Industrial Software Development (ISDe)?

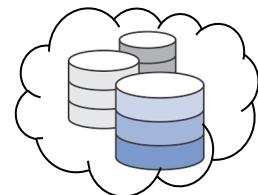
- Complexity of modern, industrial and open-source software projects cannot be managed as an unstructured process
 - it requires significant planning and managing work before writing even a single line of code!
 - (... and we have not yet even considered interactions with customers / stakeholders ...)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

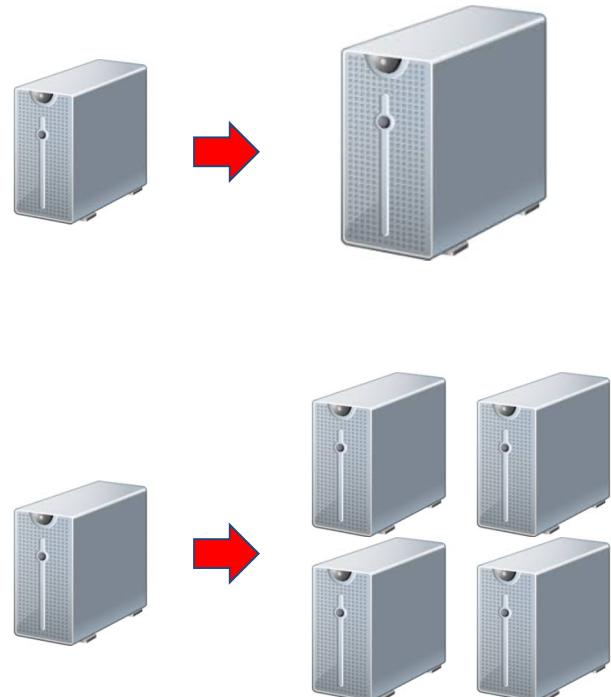
- AI has shown that very valuable information can be extracted from the vast amount of data available nowadays in many applications

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 ~ 12-16 hours
 - Basic data structures and algorithmic design
 - Object-oriented programming (OOP)
- Design Patterns and UML (Unified Modeling Language) ~ 20 hours
- Test-Driven Development (TDD) ~ 20 hours
- Practical Sessions and Exercises ~ 6-12 hours
- Seminars ~ 6-8 hours
 - Git (for versioning and collaborative development)
 - Developing applications in the cloud (using Amazon Web Services, AWS)

Time to poll (via MentiMeter)

- Students' age and passed exams
- Programming skills and tools



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
- **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

- No intermediate assessments!
- **Written exam after the course**
 - First part (~30 mins): methodological aspects in software engineering / development
 - Second part (~1h 30 mins): OOP, design patterns and testing (Python programming)