

# Course conclusions

## Advanced Software Engineering

*Department of Computer Science*

*University of Pisa*

*MSc in Computer Science & Networking*

*MSc in Computer Science*

*MSc in Cybersecurity*

(from course intro)

Why?



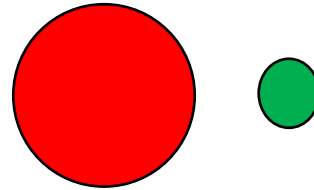
# Job interview questions (examples)

(from course intro)

What is a container?



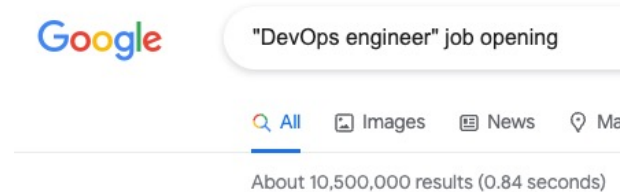
What is a microservice?



Can you develop agile?



Are you a DevOps engineer?



Can you deploy securely?



(from course intro)

Why?  
What?

(from course intro)

**what**

reliability

security

sw products

user stories

sw architecture

microservices

cloud-based sw

Agile

testing

DevOps

*cloud-edge  
continuum*

*BPM*



Course introduction. Software products.
Agile software engineering.
<i>Product vision and marshmallow challenge.</i>
Features, scenarios, and stories.
<i>Features and stories for Tic Tac Toe.</i>
<i>Prototyping Tic Tac Toe with Python and GitHub.</i>
Software architecture.
Enterprise integration patterns.
<i>Enterprise Integration Patterns in Apache Camel.</i>
Cloud-based software.
Cloud-based software.
<i>Docker Compose.</i>
Kubernetes.
Microservices architecture.
<i>Kubernetes and Docker Swarm.</i>
<i>In-class test. Microservices architecture.</i>
Architectural smells and refactorings for microservices. <i>Discussion of in-class test.</i>
<i>Architectural smells and refactorings for microservices.</i>
Security and privacy.
Security and privacy. Security and microservices.
<i>Static security analysis with Bandit.</i>
<i>API vulnerabilities and penetration testing.</i>
Security and microservices. DevOps.
DevOps.
<i>CICD with Jenkins.</i>
Business process modelling.
<i>Business process modelling with Camunda.</i>
<i>Business process modelling with WoPeD.</i>
Testing.
<i>Testing.</i>
Cloud-Edge continuum.
Explaining cascading failures in microservices. Course conclusions.
<i>In-class test. Hands-on test simulation.</i>

1/3 of hours in lab  
2 in-class tests  
90% passed first test  
0 homework assignments

## Advanced Software Engineering - Syllabus

University of Pisa<sup>1</sup> - a.y. 2022/2023

<b>Software Products</b> The product vision. Software product management. Product prototyping.	ch. 1 of [1], slides <sup>2</sup>
<b>Agile Software Engineering</b> Agile methods. Extreme Programming. Scrum.	ch. 2 of [1], slides
<b>Features, Scenarios, and Stories</b> Personas. Scenarios. User stories. Feature identification.	ch. 3 of [1], slides
<b>Software Architecture</b> Importance of architecture. Architectural design. System decomposition. Distribution architecture. Technology issues. Enterprise Integration Patterns.	ch. 4 of [1], [2], slides
<b>Cloud-based Software</b> Virtualization, containers, Docker and Docker Compose. Cloud service models. Software as a Service. Multi-tenant and multi-instance systems. Cloud software architecture. Kubernetes.	ch. 5 of [1], [3], slides
<b>Microservices Architecture</b> Microservices, microservices architecture. RESTful services. Microservice deployment. Architectural smells and refactorings.	ch. 6 of [1], [4] slides
<b>Security</b> Attacks and defenses. Authentication. Authorization. Encryption. Privacy. Challenges of securing microservices. Smells and refactorings for microservices security.	ch. 7 of [1], slides
<b>DevOps</b> DevOps principles, Continuous Integration, Continuous Delivery and Deployment, Infrastructure as Code, DevOps measurement.	ch. 10 of [1], slides
<b>Business Process Modelling</b> Business process models, BPMN, workflow nets.	sect. 5 of [5], [6], slides
<b>Testing</b> Functional testing, test automation, test-driven development, security testing, code reviews.	ch. 9 of [1], slides
<b>Cloud-Edge Continuum</b> Objectives. Declarative application management and continuous reasoning.	slides
<b>Explaining cascading failures in microservices</b> Objectives. Declarative failure root cause analysis.	slides

### References

- [1] Ian Sommerville. [Engineering Software Products - An Introduction to Modern Software Engineering](#). Pearson, 2021. <sup>3</sup>
- [2] The Apache Software Foundation. [Enterprise Integration Patterns](#). 2022.
- [3] Jeremy Jordan. [An introduction to Kubernetes](#). 2019.
- [4] D. Neri, J. Soldani, O. Zimmermann, A. Brogi. [Design principles, architectural smells and refactorings for microservices: A multivocal review](#). SICS Software-Intensive Cyber-Physical Systems, 2020.
- [5] OMG. [BPMN 2.0 by example](#). 2010.
- [6] A. Brogi, S. Forti. Workflow nets. [Teaching note](#). 2022.

<sup>1</sup> Course offered for the MSc in Computer Science and Networking, for the MSc in Computer Science and for the MSc in Cybersecurity.

<sup>2</sup> All the slides used in the course are available in the Moodle page of the course.

<sup>3</sup> Two copies of the book will be available in [UNIPi computer science library](#).

### Advanced Software Engineering Lab: Syllabus with Hands-on Test Examples

This document lists the skill sets that you are expected to master by the end of the laboratory module of Advanced Software Engineering. For each group of skills, we list some examples of hands-on-tests. Note that the hands-on test can also combine (simpler) examples from different skill sets.

#### Programming skills

- Understand, write, and modify Python code (classes, functions, Flask microservices).
- Use Git from the Command Line Interface to clone, commit, pull and push code.
- Use GitHub to perform a fork and/or a pull request to an existing repository.

#### Some test examples:

- Given a skeleton repository from GitHub, clone it, extend it with some functionality and push the result on GitHub.
- Fork and perform a pull request to a repo.

#### Enterprise Integration Patterns

- Order and compose (given) patterns to implement a business mission via the Apache Camel Java DSL.

#### Some test examples:

- Given a list of Apache Camel DSL instructions in Java compose them into a program to implement a specific behaviour. The list might contain unnecessary instructions.

#### Docker

- Docker commands to build, tag and run an image.
- Understand, write and modify a Dockerfile and issue commands to build and run a Docker image from such file.
- Understand, write and modify a docker-compose.yml and issue the commands to build and run a multi-service application with Docker Compose.
- Understand, write and modify a stack.yml file and issue the commands to launch a multi-service application in swarm mode.

#### Some test examples:

- Write a Dockerfile using a specific base image, cloning some code from GitHub, installing dependencies, and launching a Python or bash script. Build and run the image from a Dockerfile.
- Complete a docker-compose.yml to launch a simple multi-service application. Build and run the specified orchestration.
- Starting from a docker-compose.yml, complete it and convert it into a stack.yml file to launch your applications in swarm mode.

#### Kubernetes

- Issue the commands to start a minikube cluster with K nodes.
- Understand, write and modify deployment.yaml file and a service.yaml files.
- Understand the difference between different type of Object in K8S.

#### Some test examples:

- None. But there can be questions in the oral part.

### Advanced Software Engineering Lab: Syllabus with Hands-on Test Examples

#### Smells and refactorings

- Launch and use MicroFreshener to identify and fix smells in microservice-based architectures.

#### Some test examples:

- Analyse a given application and apply refactorings suggested from MicroFreshener. Simplify, if possible, the suggested refactorings.

#### Security analyses

- Launch static security analysis with bandit and understand the output.
- Launch penetration with OWASP Zap

#### Some test examples:

- Analyse a given application with bandit and fix a simple security issue (e.g. hardcoded password).
- Launch a microservice-based application with Docker Compose and analyse it with OWASP Zap in automated/manual mode.

#### Jenkins

- Use Jenkins to automate simple CI/CD pipelines (build, test, deliver).
- Understand, write and modify a simple Jenkinsfile.

#### Some test examples:

- Complete/modify a given Jenkins file so to perform CI/CD on a given code repository.

#### BPMN

- Use the Camunda Modeler to write/modify BPMNs.
- Deploy and run process instances with the Camunda Engine.
- Understand the two usage patterns of Camunda and code simple workers for pattern B.
- Create/modify a WF-net and perform automated analyses of WF-nets with WoPed.

#### Some test examples:

- Deploy a BPMN file to Camunda from the Camunda Modeler and complete the code of a worker to accomplish a specific task. Launch a process instance and the worker.
- Transform a BPMN process into a WF-net and analyse it with WoPed.

#### Unit and Performance Testing

- Write and run unit tests with the pytest module, also computing coverage.
- Write and run locustfile.py to perform load testing.

#### Some test examples:

- Extend a given testset for a specific microservice written in Flask and fix a bug.
- Write a new (simple) microservice for the microase application and 3 tests for such a service.
- Launch an application with Docker [Compose] and write a locustfile.py to perform load testing against it. Specify that an endpoint is called 2 times more than others.



(from course intro)

Why?  
What?  
How?





Wednesday at 2 pm  
Thursday at 2 pm  
Friday at 11 am

«cum tempore»

(from course intro)



classes



+ labs (**BYOD**)



Ian Sommerville.

[Engineering Software Products - An Introduction to Modern Software Engineering.](#)

Pearson, 2021

All other course material on 





News



List of classes



Exam rules



Syllabus



Lab Syllabus (with Hands-on Test Examples)



Results of MCQ #1

## Classes



Course introduction



Software products



Agile software engineering



Features, scenarios, and stories



Software architecture



Cloud-based software



Microservices architecture



Security



DevOps



Business Process Modelling



Testing



Cloud-Edge Continuum



Explaining cascading failures in microservices



Course conclusions

## Labs



Lab 01 - Product vision and marshmallow challenge



Lab 02 - Prototyping TicTacToe with Python and GitHub



Lab 03 - EIP with Apache Camel



Lab 04 - Docker Compose



Lab 05 - K8S and Docker Swarm



Lab 06 - Architectural smells and refactorings



Lab 07 - Static security analysis with bandit



Lab 08 - API vulnerabilities and penetration testing



Lab 09 - CI/CD pipelines with Jenkins



Lab 10 - BPMN with Camunda



Lab 11 - Unit and performance testing



(from course intro)

Why?  
What?  
How?  
Who?



**Antonio Brogi**

Full Professor

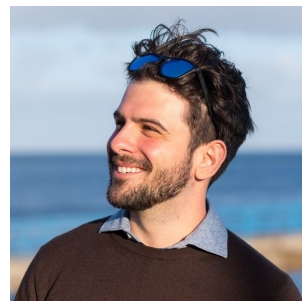
antonio.brogi@unipi.it

(from course intro)

**Stefano Forti**

Assistant Professor

stefano.forti@unipi.it



**Marco Gaglianese**

Teaching Assistant

**Jacopo Massa**

Teaching Assistant



**Alessandro  
Bocci**



**Jacopo  
Soldani**

guest speakers

Why?

What?

How?

Who?

Exam rules?

# How to get the credits

(from course intro)

## **The exam consists of two parts**

1. A hands-on pass/fail test on the technologies used in the labs  
*followed by*
2. Questions on the topics of syllabus

## ***Continuous assessment (optional)***

*The continuous assessment will consist of in-class tests during the term.*

*For students that will profitably participate in the continuous assessment:*

- *The evaluation will be 1/3 from the continuous assessment activities and 2/3 from the questions on the syllabus during the exam*
- *The questions on the syllabus during the exam will be taken from a list that will be published at the end of the course*

# Second in-class test

Tomorrow's lab will start with the second in-class MCQ test.

Room allocation for second in-class test

- If your\_id ("matricola") ends with 0/1/2/3/4/5/6 then go in room A1
- If your\_id ("matricola") ends with 7/8/9 then go in room M1

[After the in-class test, the lab will be held on room A1 and it will focus on a simulation on the hands-on test of the final exam.]



# Exam dates

(from course intro)

5 exam dates

- 2 in January-February
- 2 in June-July
- 1 in September

+

2 “extraordinary” exam dates (only for some categories of students)

- 1 in October-November
- 1 in March-April



Recall that to participate in the exams  
**you MUST register on [esami.unipi.it](https://esami.unipi.it)**  
by the set deadlines

# First exam date

First exam date scheduled for January 18th at 2 pm

To participate you must register on [esami.unipi.it](https://esami.unipi.it) by January 13th

- Part I: Hands-on test on the technologies used on the labs
- Part II: Oral interview, consisting of questions on the syllabus

Once the list of students registered for the exam will be available, we will publish the schedule of the oral interviews (January 18<sup>th</sup> and 19<sup>th</sup>)

Why?

What?

How?

Who?

Exam rules?

Q&A

Q: What does the CAP theorem tell us?

A: *"In presence of a network **P**artition, you cannot have both **A**vailability and **C**onsistency."*

In other words, in a distributed system (where network can lose arbitrarily messages), we can have either Consistency (every read returns the value of the last write) or Availability (every request is responded).

Q: What is the percentage associated to a refactoring (of an architectural smell)?

A: The percentage of the considered literature sources suggesting that refactoring for that smell.

Q: What is the percentage associated to a security smell?

A: The percentage of the considered literature sources reporting that security smell.

Q: What is DevOps automation?

A:

**“Everything that can be should be automated”**

#### Continuous integration

Each time a developer commits a change to the project's master branch, an executable version of the system is built and tested.

#### Continuous delivery

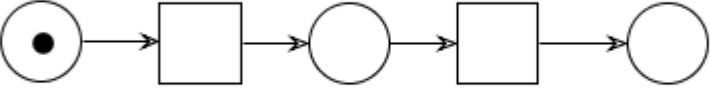
Executable software is tested in a simulated product's operating environment.

#### Continuous deployment

New release of system made available to users every time a change is made to the master branch of the software.

#### Infrastructure as code

Machine-readable models of infrastructure (network, servers, routers, etc.) on which the product executes are used by configuration management tools to build the software's execution platform. The software to be installed, such as compilers and libraries and a DBMS, are included in the infrastructure model.

Q: Does WoPeD consider the net N  sound because its «completion» is both live and bounded?

A: Yes.

WoPeD considers N sound since  $(N^*, \{i\})$  is live and bounded, where  $N^*$  is N extended with a transition from the sink place o to the source place i.

(On the other hand, N itself is not live.)



Q: How to use equivalence partitions for testing?

A:

- (1) Identify “equivalence partitions”: sets of inputs that will be treated the same in your code
- (2) Test your program using several inputs from each equivalence partition

## Q: What does “mock objects” mean?

### Test automation

Good practice: Structure automated tests in three parts

1. Arrange - Set up the system to run the test (define test parameters, **mock objects** if needed)
2. Action - Call the unit that is being tested with the test parameters
3. Assert – Assert what should hold if test executed successfully.

```
#Arrange - set up the test parameters
p = 0
r = 3
n = 31
result_should_be = 0

#Action - Call the method to be tested
interest = interest_calculator (p, r, n)

#Assert - test what should be true
self.assertEqual (result_should_be, interest)
```

A: If necessary, mock objects that emulate the functionality of code that has not yet been developed. (A mock object is a simulated object that mimics the behaviour of code that has not yet been developed.)

Why?

What?

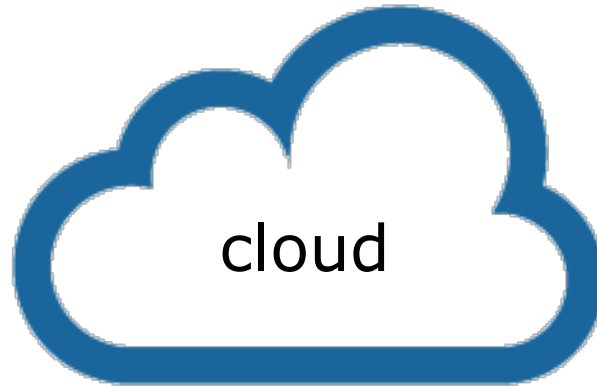
How?

Who?

Exam rules?

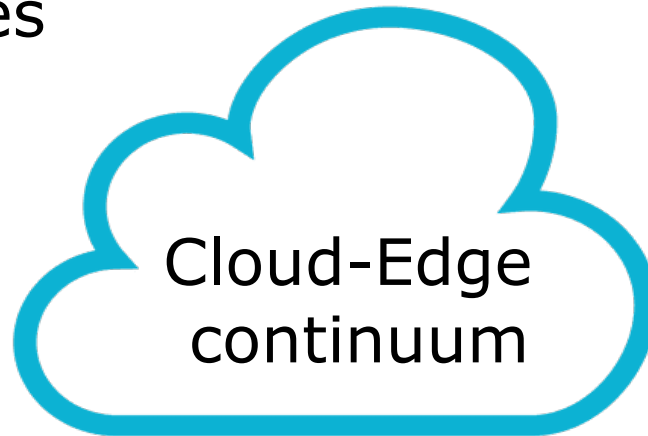
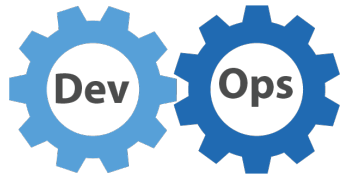
Q&A

Theses?



microservices

networking



*green computing*

*security*

*AI*



# Many topics available for MS theses, top student satisfaction (ask around)

2020

1. Edoardo Baldini. A system-based perspective on green IoT. MSc. March 2020.
2. Marco Cameriero. Modelling and analysing scalable cloud applications. MSc. March 2020.
3. Giacomo Vallorani. Deploying Applications to Multiple Clouds. BSc. March 2020.
4. Lorenzo Pierotti. Geolocalizzazione di smartphone all'interno di un edificio. BSc. May 2020.
5. Aldo D'Aquino. Design, prototyping, and validation of a Kubernetes operator for an IoT platform based on Red Hat OpenShift. MSc. July 2020.
6. Giacomo De Liberali. AsyncMDSL: a domain-specific language for modeling message-based systems. MSc. July 2020.
7. Leonardo Frioli. Automating the deployment of cloud-native applications over multiple platforms. MSc. July 2020.
8. Giuseppe Bisicchia. Goal-driven Management of IoT Indoor Environments. BSc. July 2020.
9. Samuele Intaschi. Progettazione e sviluppo di una GUI per EdgeUsher. BSc. July 2020.
10. Giuseppe Muntoni. Mining the Architecture of Microservice-Based Applications from their Kubernetes Deployment. BSc. July 2020.
11. Monica Amico. GiòVase 2.0: Un Servizio per la Gestione di Piante Indoor. BSc. October 2020.
12. Riccardo Falco. Allocazione di funzioni virtuali di rete (VNF) con specifiche multiple di servizi in ambienti multi cloud. BSc. October 2020.
13. Andrea Bongiorno. Automating the Generation of Executable Test Cases from Production Events. MSc. October 2020.
14. Massimiliano Fatticcioni. Estensione delle funzionalità di un'applicazione a microservizi in Cloud. BSc. December 2020.
15. Giulio Paparelli. Analysing the Replica-Aware Management of Multi-Component Applications. BSc. December 2020.
16. Giuseppe Montesano. Identificazione automatica di causalità e fallimenti in applicazioni multi-servizio. BSc. December 2020.
17. Martina Trigilia. Sviluppo di un installer multiplatforma. BSc. December 2020.

2021

18. Nicolò Maio. Integrazione dello standard OPC UA con una piattaforma Cloud IIoT. BSc. March 2021.
19. Nicolò Tumino. Evoluzione DevSecOps delle pipeline di Continuous Integration di un'applicazione distribuita in cloud. BSc. May 2021.
20. Francesco Buti. Sperimentazione di Chaos testing per il monitoraggio di un'infrastruttura Fog federata. BSc. June 2021.
21. Riccardo Paoletti. Modelling and Analysing Enterprise Application Integration. MSc. October 2021.
22. Jacopo Massa. Data-aware application placement and routing in the Cloud-IoT continuum. MSc. October 2021.
23. Marco Gaglianese. FogMon 2.0, an Improved Monitoring Tool for Fog Infrastructures. MSc. October 2021.
24. Tommaso Macchioni. Progettazione e realizzazione di un'automazione basata sul paradigma GitOps con OpenShift. BSc. December 2021.
25. Javad Khalili. Mining Microservice-based Architectures: From Kubernetes to microTOSCA. MSc. December 2021.
26. Giovanni Bartolomeo. Enabling Microservice Interactions within Heterogeneous Edge Infrastructures. MSc. December 2021.
27. Tommaso Lencioni. Orchestration and placement of applications in an autonomous-driving environment. BSc. December 2021.

2022

28. Denis Buscaino. Sviluppo del protocollo di comunicazione MQTT per una smart capsule per il delivery di sangue, emocomponenti e medicinali. BSc. February 2022.
29. Marco Spinosa. Designing, prototyping, and assessing a distributed IIoT platform. MSc. April 2022.
30. Edoardo Maione. Ambient Intelligence For Monitoring Indoor Environments. MSc. April 2022.
31. Francois Romain. Modelling microservice-based applications. MSc. June 2022. (U. Namur)
32. Alessio Matricardi. Simulazione di orchestrazione di funzioni FaaS su infrastrutture Cloud-IoT. BSc. July 2022.
33. Giuseppe Bisicchia. Continuous Reasoning for application Management over the Cloud-IoT continuum. MSc. July 2022.
34. Alessio Russo. Development of a capacity tool for a food delivery platform. MSc. July 2022.
35. Maxime Bever. Linda in the Fog. MSc. August 2022. (U. Namur)
36. Giorgio Dell'Immagine. Automated detection of security smells in microservice-based applications. BSc. December 2022.
37. Marco Marinò. Risoluzione automatica di smell in applicazioni a microservizi. BSc. December 2022.
38. Giovanni Neglia, Installazione e configurazione di un cluster Big Data. BSc. December 2022.
39. Francesca Funaioli. Euristiche per problema di piazzamento di catene di VNF. BSc. December 2022.

+ 8 ongoing



<https://www.youtube.com/watch?v=b9434BoGkNQ>