TELECOM SudParis

IP PARIS

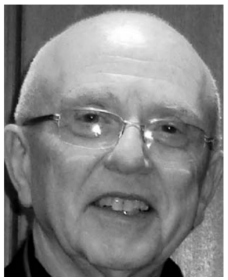
CSC4102 : Introduction au génie logiciel pour applications orientées objet

Denis Conan

Janvier 2024

1 Contexte : Qui demande une ingénierie logiciel ?

David L. Parnas, ACM SIGSOFT[1] Outstanding Research Award, 1998 [Parnas, 2010]



- *We are caught in a catch-22 situation*[2] *:*
  - *Until customers demand evidence that the designers were qualified and disciplined, they will continue to get sloppy software.*
  - *As long as there is no better software, we will buy sloppy software.*
  - *As long as we buy sloppy software, developers will continue to use undisciplined development methods.*
  - *As long as we fail to demand that developers use disciplined methods, we run the risk —nay, certainty— that we will continue to encounter software full of bugs.*
- *Much of the fault lies with our teaching.*
  Computer science students are not taught to work in disciplined ways.

1. *ACM, Special Interest Group on SOFTware engineering*
2. En français, « situation sans issue »

2 Intérêt — Pourquoi une méthodologie ?

2.1 Du programme (monolithique) correct...

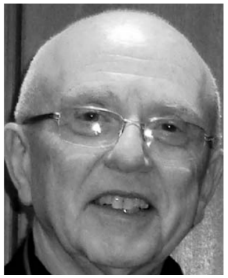
C. Antony R. Hoare, ACM A.M. Turing Award, 1980 [Hoare, 2009]

- *The basic questions common to all branches of science*
  1. *What [should] this program do ?*
  2. *How [is it doing] it ?*
  3. *Why does it work ?*
  4. *What is the evidence for believing the answers to all these questions ?*
- *[When writing programs,] we know in principle how to answer them*[3]
  1. *It is the specifications that describes what [we want] a program [to do]*
  2. *It is programming language semantics that explains how it works*
  3. *It is assertions and other internal interface contracts between component modules that explain why it works*
  4. *It is mathematical and logical proof [and testing], nowadays constructed and checked by computer, that ensures mutual consistency of specifications, interfaces, programs, and their implementations*

3. *Hoare Triple :* $\{P\}S\{Q\}$ *reading "if precondition* P *is true before statement* S *is executed, and if the execution of* S *terminates, then postcondition* Q *is true afterwards".*

2.2 ...À l'ingénierie d'un logiciel

- Définition du terme « ingénierie logiciel » selon [ISO/IEC, 2017, ISO/IEC, 2015] : « *Software engineering : systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software* »



David L. Parnas, ACM SIGSOFT [Parnas, 2010]

- *Experiences remind [us] that the activity we call software engineering does not come close to deserving a place among the traditional engineering disciplines [e.g. civil, electrical, and mechanical].*

J.H. Saltzer, M.F. Kaashoek [Saltzer and Kaashoek, 2009]

- *There are at least two significant ways in which computer systems differ from every other kind of system with which [engineers] have experience*
  - *The complexity of a computer [model] is not limited by physical laws.*
  - *The rate of change of computer system technology is unprecedented.*

3 Contenu — Quels sont les objectifs du module ?

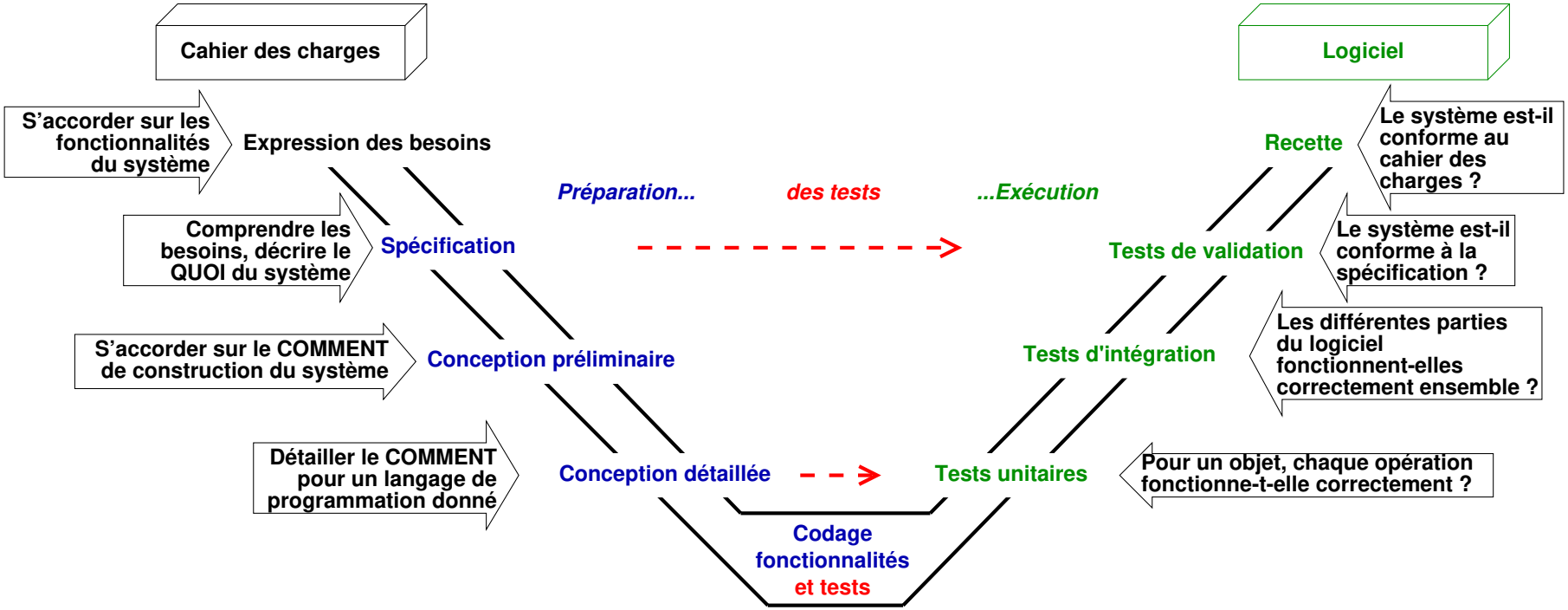
3.1 Acquis d'apprentissage visés

- À l'issue du module, vous serez capables :

1. Thème « spécification, conception et programmation orientées objet »
   - de spécifier, concevoir et programmer entièrement et de manière systématique un logiciel de petite taille, mais réaliste, dont le cahier des charges est fourni, en utilisant la notation UML et le langage JAVA
   - d'appliquer des patrons (motifs) de conception et de programmation (idiomes) donnés dans le but d'améliorer la qualité du logiciel
2. Thème « qualité logicielle et tests »
   - de rédiger des tests de validation et des tests unitaires (les tests d'intégration ne sont ni étudiés ni utilisés)
   - de programmer et d'exécuter (de manière automatique) ces tests
   - d'utiliser des logiciels d'analyse statique de la qualité du code
3. Thème « méthodologie de développement agile »
   - de partager vos artefacts produits tout au long des séances
4. Thème « outillage des activités de développement »
   - d'identifier et d'utiliser des outils représentatifs des activités de dév.

3.2 Concernant le développement logiciel agile

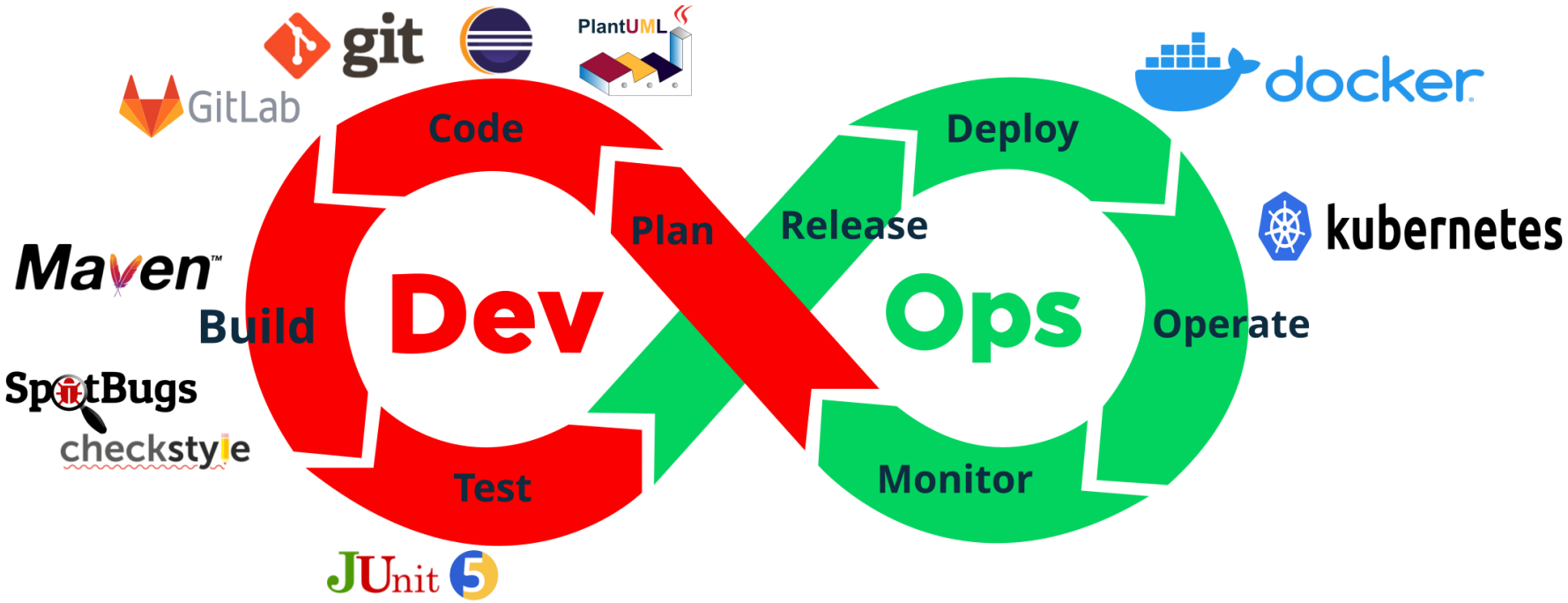
- Activités de dév. logiciel d'un binôme-projet « qui fait tout le logiciel »
- Quelques éléments fournis au départ
- Cycle en V guidé dans les premières séances, et ensuite plus d'autonomie



3.2.1 Modèle en « cascade » : Avertissement

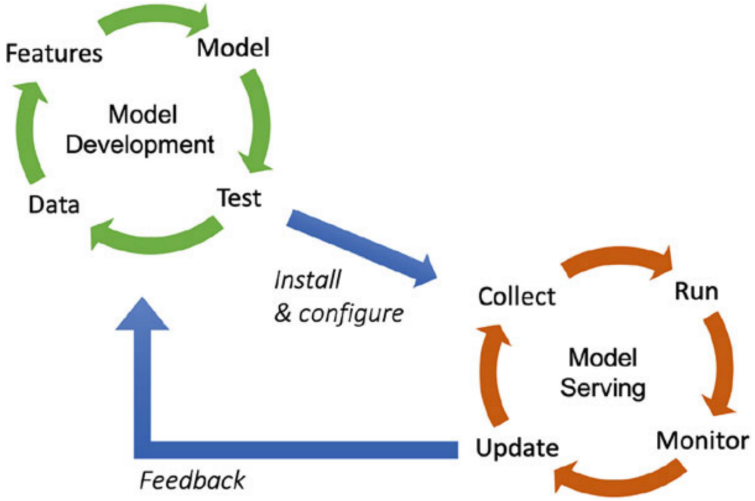
- Le modèle dit « en cascade » est un modèle pédagogique [Meyer, 2021]
  - *« Dans l'édition anniversaire de son livre fameux « The Mythical Man-Month » de 1975 [Brooks, 1995], Frederik Brooks explique le malentendu :*
    - *" The waterfall model derives from a Gantt chart layout of a staged process."*
    - *"The waterfall model, which was the way most people thought about software projects in 1975, unfortunately got enshrined into DOD-STD-2167, the Department of Defense specification for all military software. This ensured its survival well past the time when most thoughtful practitioners had recognized its inadequacy and abandoned it."*
- Dans le module, utilisé à des fins pédagogiques dans les premières séances
  - Pour avoir une vision large des différentes activités
  - Pour mettre en valeur la traçabilité entre les artefacts produits
- **En dehors de l'apprentissage, personne n'utilise le modèle en cascade**

3.2.2 Concernant DevOps



- À gauche : les outils utilisés dans le module CSC4102
- À droite : des outils utilisés dans les VAP ASR et DSI

3.2.3 Concernant les logiciels basés sur des techniques d'apprentissage

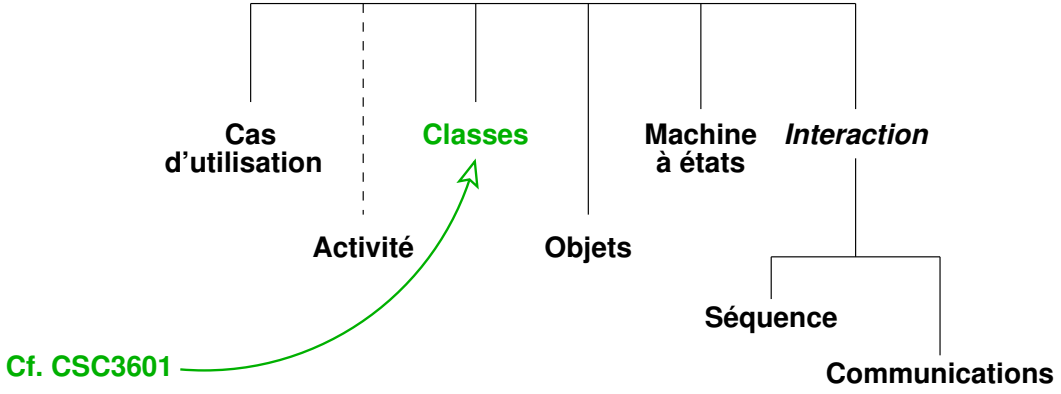


- Hors périmètre du module CSC4102
- Comportement non spécifié par les développeurs mais appris par une machine à partir de données
- À gauche, développement de modèles
  - Par des *data scientists*
  - Prétraitement de données brutes nettoyées et transformées en caractéristiques
  - Caractéristiques pour entraîner un modèle + choix d'un algorithme d'apprentissage
- À droite, sur une machine d'exécution cible
  - Machine d'exécution développée par des ingénieurs logiciel
  - Prédiction sur des données réelles non annotées
- Logiciels construits avec des cycles de vie différents de CSC4102

Image extraite de P. Lalanda. Edge Computing and Learning. In M. Kirsch Pinheiro *et al* (editors), The Evolution of Pervasive Information Systems, Springer, 2023

3.3 Concernant la modélisation avec UML

- Connaissances — savoirs
  - Différents types de diagrammes avec leurs notations
  - Rôles complémentaires des types de diagrammes
  - Cohérence entre diagrammes de même type ou de types différents



- Pratique — savoir-faire
  - Présentation dans le cours d'une première étude de cas
  - Mise en pratique lors des séances avec une autre étude de cas

3.4 Concernant JAVA, la qualité logicielle, l'outillage

- Connaissances — savoirs
  - Approfondissement des concepts de l'orientation objet :
    retours sur l'héritage (classe abstraite, interface, transtypage, redéfinition),
    retours sur les classes paramétrées et les *lambda expressions*
    librairie (collections avec listes et dictionnaires)
  - Introduction aux patrons de conception et aux idiomes JAVA
- Pratique — savoir-faire
  - Gestion de versions avec GIT et GITLABENS
  - Construction de logiciel JAVA avec MAVEN
  - Programmation des tests avec JUNIT
  - Qualité du code avec CHECKSTYLE et SPOTBUGS

3.5 Après le module — À quoi ça sert ensuite ? où ? quand ?

- À titre indicatif, quelques métiers :
  - VAP ASR et JIN
    - architecte, ingénieur développeur, ingénieur R&D, consultant, intégrateur
  - VAP DSI
    - architecte SI, maîtrise d'œuvre des SI, consultant, intégrateur
  - VAP ISI
    - maîtrise d'ouvrage

4 Prérequis

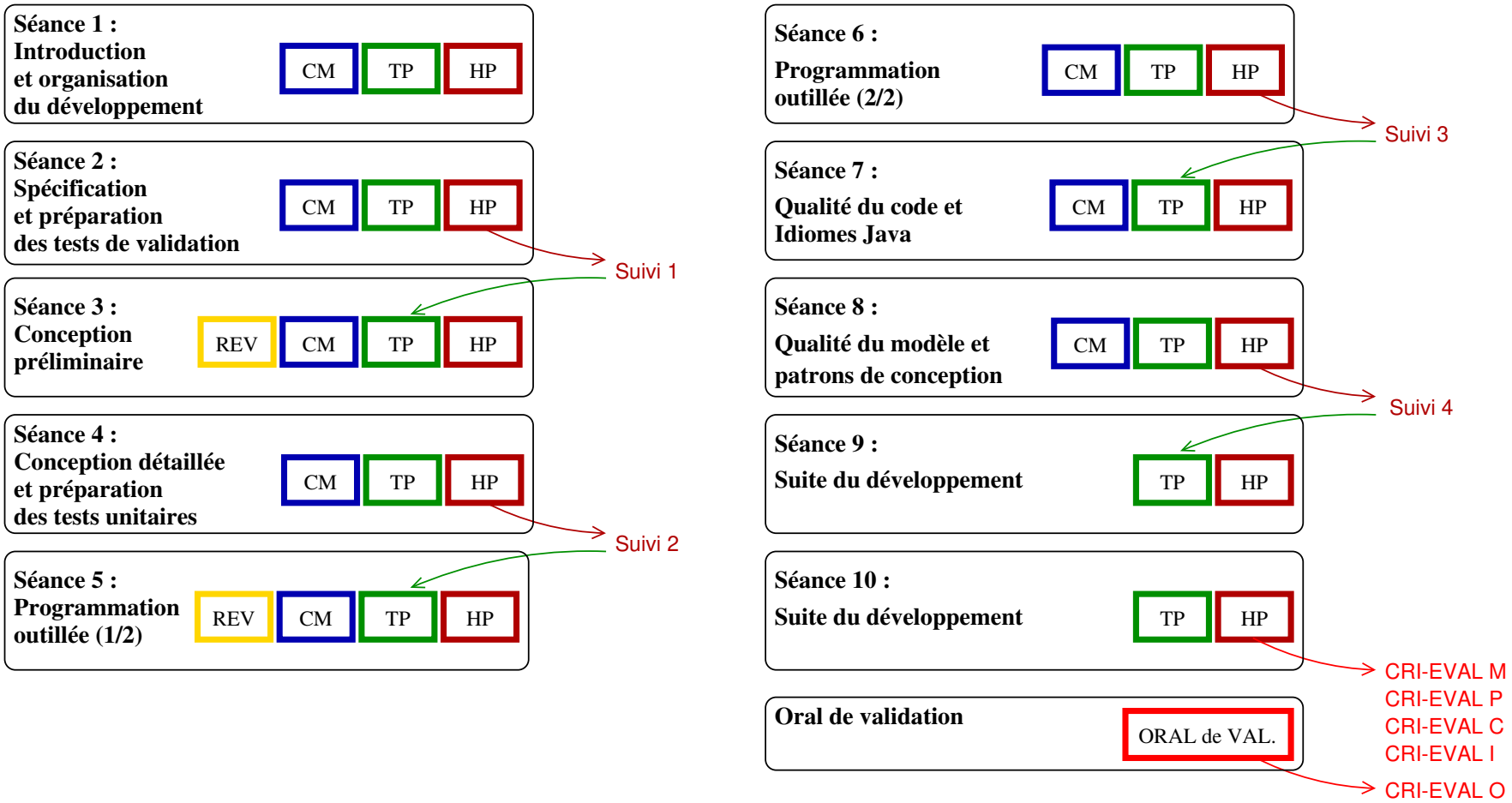
1. À propos de CSC3101
2. À propos de CSC3601
3. À propos de PRO3600 et CSC4101

[... Sondage Wooclap]

- Dispositif de révision
  - Nous mettons à disposition des éléments de cours et des exercices de révision
  - Révisions non comptabilisées dans le travail hors présentiel
  - Avant la séance 3 : diagramme de classes UML
  - Avant la séance 5 : concepts de base de la programmation en JAVA

5 Structuration du dispositif pédagogique

À partir d'un cahier des charges, chaque binôme-projet propose sa solution



6 Modalités d'évaluation

- Première session (répartition de la notation à titre indicatif)
  - Travail en binôme avec des suivis/tutorats (10 points)
    - Solution logicielle livrée dans le projet (9 pts)
      - Modélisation + préparation des tests (M = 3 pts)
      - Programmation outillée (P = 4 pts)
      - Cohérence entre la modélisation et la programmation (C = 1 pt)
      - Logiciel complet, intégralité (I = 1 pt)
    - Agilité avec présence active aux séances (A = 1 pt)
      - Présence : 0.1 point par séance
  - Connaissances globales lors d'un oral de validation (O = 10 points)
    - À base de questions de niveaux de difficulté variés, 10 minutes
      - Questions de cours + questions sur votre solution

$\Longrightarrow$ Assister aux cours + hors présentiel sur le projet ($\approx$ 3h + 3h par séance)

  - Note finale = si (O $\geq$ 5) alors $M + P + C + I + A + O$
    sinon $\min(9, M + P + C + I + A + O)$

- Seconde session : contrôle sur table portant sur la totalité du module

7 Équipe enseignante et site Web


Olivier Berger


Georgios Bouloukakis


Élisabeth Brunet


Sophie Chabridon


Denis Conan


Paul Gibson


Chantal Taconet


Michel Simatic

- Site Web : https://www-inf.telecom-sudparis.eu/COURS/CSC4102/
  - Contenu des séances
  - Menu > Organisation > Grilles d'auto-évaluation

    https://www-inf.telecom-sudparis.eu/COURS/CSC4102/?page=grilles_auto_evaluation
  - Glossaire, liste des tâches, aides et foires aux questions

Références I

Brooks, F. (1995).
The Mythical Man-Month : Essay on Software Engineering, Anniversary Edition.
Addison-Wesley.

Hoare, C. (2009).
Retrospective : An Axiomatic Basis for Computer Programming.
Communications of the ACM, 52(10) :30–32.

ISO/IEC (2015).
Information technology—Vocabulary.
International Standard ISO/IEC 2382:2015, ISO/IEC Joint Technical Committee 1, Information technology.

ISO/IEC (2017).
Systems and software engineering—Vocabulary.
International Standard ISO/IEC 24765:2017, ISO/IEC Joint Technical Committee 1, Information technology.

Références II

Meyer, B. (2021).
The four PEGS of requirements engineering.
ACM Learning Webinar, https://learning.acm.org/techtalks/PEGS.

Parnas, D. (2010).
Risks of Undisciplined Development.
Communications of the ACM, 53(10) :25–27.

Saltzer, J. and Kaashoek, M. (2009).
Principles of Computer System Design : An Introduction.
Morgan Kaufmann.