Logiciel Système : détail des enseignements

Table des matières

Chapitre 1

Introduction

Ce document détaille les enseignements prodigués en 3IF et 4IF par l'équipe enseignante "Logiciel Système". Dans ce but, il ne fait que faire référence aux enseignements listés dans le *Computer Science Curricula* (CSC dans la suite) maintenu par l'ACM et l'IEEE-Computer Society. Le CSC est un descriptif détaillé des notions et compétences enseignables dans une formation post-bac de 3 ans en informatique.
Se servir du CSC comme référence a plusieurs avantages :
- Ne pas réinventer la roue ;
- Se servir du travail de gens qui savent faire ;
- Séparer facilement notions et compétences ;
- Obtenir une vue objective de nos enseignements fondamentaux en informatique, le moins possible sujette à désaccord ;
- Obtenir une vue bien plus complète que ce qu'on pourra jamais faire par nous même de nos enseignements.

Le CSC sur lequel est basé ce document est le draft 0.8, disponible à `http://ai.stanford.edu/users/sahami/CS2013` ou encore `http://listes.insa-lyon.fr/sympa2/d_read/if.ls/Programme/ACM_curricula.pdf`.

1.1 Lecture du document

Ce document est structuré de la façon suivante. Chaque chapitre ci-après liste les enseignements prodigués au sein d'un de nos module. Au sein de chaque chapitre sont reportés entièrement les sous-thématique du CSC dont **au moins une notion** est enseignée dans le module considéré. Cela comprend à la fois les cours, TDs, TPs et projets du module. Une sous-thématique est notée <**thématique**>/<**sous-thématique**>. La liste des acronymes pour les thématiques est donnée en section 1.3. Par exemple, *AL/Algorithmic Strategies* fait référence à la sous-thématique *Algorithmic Strategies* de la thématique *Algorithms and Complexity*.
Ainsi, les notions et les compétences associées à chaque sous-thématique sont toutes listées – cela permet de ne pas avoir besoin de se référer au CSC continuellement lors de la lecture de ce document. Pour chaque notion, il est précisé si cette notion est enseignée ou non dans notre module. Et pour chaque compétence, il est indiqué le niveau atteint par nos étudiants parmi : **Non acquis**, **Familiarité**, **Usage**, **Expert**. Ces termes sont définis en section 1.2.
Dans le CSC, les notions ou compétences marqués *Elective* sont considérées comme des spécialisations. Pour les thématiques que l'on aborde indiquées comme telles dans le CSC, on a laissé cette mention dans le présent document. Ceci permet d'être conscient que l'on enseigne quelque chose de vraiment pointu.
Dernière précision, les pré-requis ne sont pour l'instant détaillés qu'au niveau de la sous-thématique. Le détails des notions et compétences réellement pré-requises n'est pas indiqué. Cela demandera éventuellement un travail ultérieur mais pour le moment, charge à chacun de se renseigner sur les collègues qui enseignent cela et d'aller discuter avec eux si besoin.

1.2 Définition des niveaux de compétence

Non acquis L'étudiant n'a pas conscience du concept, même s'il se peut que ce concept soit vaguement apparenté à un autre avec lequel il est familier.

Familiarité L'étudiant comprend ce qu'est le concept, ce qu'il veut dire. Ce niveau de maîtrise est plutôt une connaissance vague d'un concept plutôt qu'un réel savoir-faire. Un étudiant à ce niveau peut répondre à la question "Qu'est-ce que vous savez sur <ceci> ?"

Usage L'étudiant est capable d'utiliser ou d'appliquer un concept d'une manière concrète. Utiliser un concept peut inclure son utilisation appropriée dans un programme, utiliser une technique particulière de preuve, ou encore savoir effectuer une analyse d'un type précis. Ce niveau de maîtrise permet de répondre à la question "Comment faire <ceci> ?".

Expert L'étudiant est capable de considérer un concept sous plusieurs points de vue et de justifier le choix d'une approche donnée pour résoudre un problème. Ce niveau de maîtrise demande plus que simplement savoir utiliser un concept ; il implique de connaître plusieurs méthodes et de choisir celle la plus appropriée. Un étudiant à ce niveau peut répondre à la question "Pourquoi faire comme <ceci> ?"

Voyons un exemple sur la notion d'itération. Au niveau “Non acquis”, il ne sait pas expliquer cette notion, même s'il sait éventuellement qu'on peut exécuter plusieurs fois la même instruction dans un programme. Au niveau “Familiarité”, un étudiant peut donner une définition de la notion d'itération pour un programme, et sait pourquoi c'est une technique utile. Au niveau “Usage”, il sait écrire un programme utilisant une forme d'itération. Au niveau “Expert”, cela veut dire que l'étudiant comprend différentes méthodes d'itération (boucles for, boucles while, itérateurs) et est capable de choisir la bonne en fonction du cas d'utilisation.

1.3 Thématiques d'enseignement du CSC

AL Algorithms and Complexity

AR Architecture and Organization

CN Computational Science

DS Discrete Structures

GV Graphics and Visual Computing

CI Human-Computer Interaction

AS Information Assurance and Security

IM Information Management

IS Intelligent Systems

NC Networking and Communications

OS Operating Systems

BD Platform-based Development

PD Parallel and Distributed Computing

PL Programming Languages

DF Software Development Fundamentals

SE Software Engineering

SF Systems Fundamentals

SP Social Issues and Professional Issues

Chapitre 2

3IF-EP : Environnement de programmation

2.1 Introduction

Le rôle de ce module est d'amener les étudiants à savoir se servir des machine du département pour compiler et exécuter un programme simple. Il s'agit donc en premier lieu d'avoir de connaître l'environnement de travail au département. En second lieu, il s'agit de comprendre le rôle d'un système d'exploitation, savoir prendre en main les OS Linux du département et de savoir s'en servir pour compiler et exécuter un programme simple grâce à l'usage d'un éditeur de texte, d'un compilateur (gcc) et d'un outil de gestion de la chaîne de compilation (Makefile).

2.2 Pré-requis

Aucun.

2.3 Enseignements

2.3.1 OS/Overview of Operating Systems

Notions abordées

– Role and purpose of the operating system	**[Enseigné]**
– Functionality of a typical operating system	**[Enseigné]**
– Mechanisms to support client-server models, hand-held devices	**[Enseigné]**
– Design issues (efficiency, robustness, flexibility, portability, security, compatibility)	**[Enseigné]**
– Influences of security, networking, multimedia, windows	**[Non enseigné]**

Compétences acquises

1.	Explain the objectives and functions of modern operating systems	**[Familiarité]**
2.	Analyze the tradeoffs inherent in operating system design	**[Familiarité]**
3.	Describe the functions of a contemporary operating system with respect to convenience, efficiency, and the ability to evolve	**[Familiarité]**
4.	Discuss networked, client-server, distributed operating systems and how they differ from single user operating systems	**[Non acquis]**
5.	Identify potential threats to operating systems and the security features design to guard against them	**[Non acquis]**

2.3.2 OS/Operating System Principles

Notions abordées

– Structuring methods (monolithic, layered, modular, micro-kernel models)	**[Enseigné]**
– Abstractions, processes, and resources	**[Enseigné]**
– Concepts of application program interfaces (APIs)	**[Enseigné]**

- Application needs and the evolution of hardware/software techniques **[Enseigné]**
- Device organization **[Non enseigné]**
- Interrupts : methods and implementations **[Non enseigné]**
- Concept of user/system state and protection, transition to kernel mode **[Non enseigné]**

Compétences acquises

1. Explain the concept of a logical layer **[Familiarité]**
2. Explain the benefits of building abstract layers in hierarchical fashion **[Familiarité]**
3. Defend the need for APIs and middleware **[Non acquis]**
4. Describe how computing resources are used by application software and managed by system software **[Familiarité]**
5. Contrast kernel and user mode in an operating system **[Non acquis]**
6. Discuss the advantages and disadvantages of using interrupt processing **[Familiarité]**
7. Explain the use of a device list and driver I/O queue **[Non acquis]**

2.3.3 OS/File Systems

Notions abordées

- Files : data, metadata, operations, organization, buffering, sequential, nonsequential **[Non enseigné]**
- Directories : contents and structure **[Enseigné]**
- File systems : partitioning, mount/unmount, virtual file systems **[Enseigné]**
- Standard implementation techniques **[Enseigné]**
- Memory-mapped files **[Non enseigné]**
- Special-purpose file systems **[Non enseigné]**
- Naming, searching, access, backups **[Enseigné]**
- Journaling and log-structured file systems **[Non enseigné]**

Compétences acquises

1. Summarize the full range of considerations in the design of file systems **[Non acquis]**
2. Compare and contrast different approaches to file organization, recognizing the strengths and weaknesses of each **[Non acquis]**
3. Summarize how hardware developments have led to changes in the priorities for the design and the management of file systems **[Non acquis]**
4. Summarize the use of journaling and how log-structured file systems enhance fault tolerance **[Non acquis]**

2.3.4 SE/Tools and Environments

Notions abordées SE/Tools and Environments

- Software configuration management and version control ; release management **[Non enseigné]**
- Requirements analysis and design modeling tools **[Non enseigné]**
- Testing tools including static and dynamic analysis tools **[Non enseigné]**
- Programming environments that automate parts of program construction processes (e.g., automated builds) **[Enseigné]**

  NB : SE aborde makefile et shell mais n'aborde pas les IDE intégrés (eclipse)
- Continuous integration **[Non enseigné]**
- Tool integration concepts and mechanisms

Compétences acquises

1.	Describe the difference between centralized and distributed software configuration management	**[Non acquis]**
2.	Identify configuration items and use a source code control tool in a small team-based project	**[Non acquis]**
3.	Describe the issues that are important in selecting a set of tools for the development of a particular software system, including tools for requirements tracking, design modeling, implementation, build automation, and testing	**[Non acquis]**
4.	Demonstrate the capability to use software tools in support of the development of a software product of medium size	**[Usage]**

Note : certains points enseignés en 3IF ne figurent pas dans le CSC car liés à l'utilisation d'un système d'exploitation (par opposition à sa programmation ou sa compréhension) : utilisation d'un shell [familiarité], scripts shell [familiarity], droits d'accès [familiarité], chemin absolu/relatif [familiarity]

Chapitre 3

3IF-PC : Programmation concurrente

3.1 Introduction

Le rôle de ce module est d'amener les étudiants à savoir programmer de manière concurrente. On y voit donc différents moyens de programmer et synchroniser plusieurs processus communiquants. Attention, nous ne voyons dans ce module aucune notion d'algorithmique parallèle.

3.2 Pré-requis

Modules IF : 3IF-EP
Thématiques CSC :
– SE/Tools and Environments
– OS/Operating System Principles
– OS/Overview of Operating Systems

3.3 Enseignements

3.3.1 OS/Concurrency

Notions abordées

– States and state diagrams (cross reference SF/State-State Transition-State Machines)	**[Non enseigné]**
– Structures (ready list, process control blocks, and so forth)	**[Enseigné]**
– Dispatching and context switching	**[Non enseigné]**
– The role of interrupts	**[Non enseigné]**
– Managing atomic access to OS objects	**[Enseigné]**
– Implementing synchronization primitives	**[Non enseigné]**
– Multiprocessor issues (spin-locks, reentrancy) (cross reference SF/Parallelism)	**[Non enseigné]**

Compétences acquises

1.	Describe the need for concurrency within the framework of an operating system	**[Familiarité]**
2.	Demonstrate the potential run-time problems arising from the concurrent operation of many separate tasks	**[Usage]**
3.	Summarize the range of mechanisms that can be employed at the operating system level to realize concurrent systems and describe the benefits of each	**[Familiarité]**
4.	Explain the different states that a task may pass through and the data structures needed to support the management of many tasks	**[Non acquis]**
5.	Summarize techniques for achieving synchronization in an operating system (e.g., describe how to implement a semaphore using OS primitives)	**[Familiarité]**
6.	Describe reasons for using interrupts, dispatching, and context switching to support concurrency in an operating system	**[Non acquis]**
7.	Create state and transition diagrams for simple problem domains	**[Non acquis]**

3.3.2 OS/Scheduling and Dispatch

Notions abordées

- Preemptive and non-preemptive scheduling (cross reference SF/Resource Allocation and Scheduling, PD/Parallel Performance) **[Enseigné]**
- Schedulers and policies (cross reference SF/Resource Allocation and Scheduling, PD/Parallel Performance) **[Enseigné]**
- Processes and threads (cross reference SF/computational paradigms) **[Enseigné]**
- Deadlines and real-time issues **[Non enseigné]**

Compétences acquises

1. Compare and contrast the common algorithms used for both preemptive and non-preemptive scheduling of tasks in operating systems, such as priority, performance comparison, and fair-share schemes **[Familiarité]**
2. Describe relationships between scheduling algorithms and application domains **[Familiarité]** .
3. Discuss the types of processor scheduling such as short-term, medium-term, long-term, and I/O **[Non acquis]**
4. Describe the difference between processes and threads **[Non acquis]**
5. Compare and contrast static and dynamic approaches to real-time scheduling **[Non acquis]**
6. Discuss the need for preemption and deadline scheduling **[Non acquis]**
7. Identify ways that the logic embodied in scheduling algorithms are applicable to other domains, such as disk I/O, network scheduling, project scheduling, and problems beyond computing **[Non acquis]**

3.3.3 PD/Communication and Coordination

Notions abordées

- Shared Memory **[Enseigné]**
- Consistency in shared memory models **[Non enseigné]**
- Message passing - Blocking versus non-blocking styles for sending and receiving messages **[Enseigné]**
- Message passing - Message buffering (cross-reference PF/Fundamental Data Structures/Queues) **[Non enseigné]**
- Atomicity - Granularity of atomic accesses and updates, and the use of constructs such as critical sections or transactions to describe them **[Enseigné]**
- Atomicity - Mutual Exclusion using locks, semaphores, monitors, or related constructs **[Enseigné]**
- Potential for liveness failures and deadlock (causes, conditions, prevention) **[Enseigné]**

Compétences acquises

1. Use mutual exclusion to avoid a given race condition **[Usage]**
2. Give an example of an ordering of accesses among concurrent activities that is not sequentially consistent **[Familiarité]**
3. Give an example of a scenario in which blocking message sends can deadlock **[Familiarité]**
4. Explain when and why multicast or event-based messaging can be preferable to alternatives **[Non acquis]**
5. Write a program that correctly terminates when all of a set of concurrent tasks have completed **[Usage]**
6. Use a properly synchronized queue to buffer data passed among activities **[Usage]**

7.	Explain why checks for preconditions, and actions based on these checks, must share the same unit of atomicity to be effective	**[Non acquis]**
8.	Write a test program that can reveal a concurrent programming error ; for example, missing an update when two activities both try to increment a variable	**[Non acquis]**
9.	Describe at least one design technique for avoiding liveness failures in programs using multiple locks or semaphores	**[Non acquis]**
10.	Describe the relative merits of optimistic versus conservative concurrency control under different rates of contention among updates	**[Non acquis]**
11.	Give an example of a scenario in which an attempted optimistic update may never complete	**[Non acquis]**

Chapitre 4

4IF-SEA : Gestion des ressources physiques par les systèmes informatiques

4.1 Introduction

Il s'agit dans ce module de comprendre comment un système informatique partage les ressources disponibles (ici : temps et mémoire) entre plusieurs applications, de savoir modéliser et évaluer les ressources utilisées par des applications concurrentes s'exécutant au-dessus d'un système d'exploitation.

4.2 Pré-requis

- AR/Memory system organization and architecture
- AR/Assembly level machine organization
- SDF/Fundamental Programming Concepts
- SDF/Fundamental Data Structures
- AR/Memory system organization and architecture
- AR/Assembly level machine organization
- SDF/Fundamental Programming Concepts
- SDF/Fundamental Data Structures
- SDF/Development Methods
- HCI/Designing Interaction
- NC/Networked Applications
- PD/Parallelism Fundamentals

4.3 Enseignements

4.3.1 OS/Real Time and Embedded Systems

Notions abordées

- Process and task scheduling **[Enseigné]**
- Memory/disk management requirements in a real-time environment **[Non enseigné]**
- Failures, risks, and recovery **[Non enseigné]**
- Special concerns in real-time systems **[Non enseigné]**

Compétences acquises

1. Describe what makes a system a real-time system **[Familiarité]**
2. Explain the presence of and describe the characteristics of latency in real-time systems **[Non acquis]**
3. Summarize special concerns that real-time systems present and how these concerns are addressed **[Familiarité]**

.

4.3.2 AR/Memory system organization and architecture

Notions abordées

- Storage systems and their technology **[Non enseigné]**
- Memory hierarchy : importance of temporal and spatial locality **[Non enseigné]**
- Main memory organization and operations **[Non enseigné]**
- Latency, cycle time, bandwidth, and interleaving **[Non enseigné]**
- Cache memories (address mapping, block size, replacement and store policy) **[Non enseigné]**
- Multiprocessor cache consistency/Using the memory system for inter-core synchronization/atomic memory operations **[Non enseigné]**
- Virtual memory (page table, TLB) **[Enseigné]**
- Fault handling and reliability **[Non enseigné]**

Compétences acquises

1. Identify the main types of memory technology **[Familiarité]**
2. Explain the effect of memory latency on running time **[Non acquis]**
3. Describe how the use of memory hierarchy (cache, virtual memory) is used to reduce the effective memory latency **[Non acquis]**
4. Describe the principles of memory management **[Familiarité]**
5. Explain the workings of a system with virtual memory management **[Familiarité]**
6. Compute Average Memory Access Time under a variety of memory system configurations and workload assumptions **[Non acquis]**

4.3.3 OS/Concurrency

Notions abordées

- States and state diagrams (cross reference SF/State-State Transition-State Machines) **[Enseigné]**
- Structures (ready list, process control blocks, and so forth) **[Enseigné]**
- Dispatching and context switching **[Enseigné]**
- The role of interrupts **[Enseigné]**
- Managing atomic access to OS objects **[Enseigné]**
- Implementing synchronization primitives **[Enseigné]**
- Multiprocessor issues (spin-locks, reentrancy) (cross reference SF/Parallelism) **[Non enseigné]**

Compétences acquises

1. Describe the need for concurrency within the framework of an operating system **[Non acquis]**
2. Demonstrate the potential run-time problems arising from the concurrent operation of many separate tasks **[Non acquis]**
3. Summarize the range of mechanisms that can be employed at the operating system level to realize concurrent systems and describe the benefits of each **[Non acquis]**
4. Explain the different states that a task may pass through and the data structures needed to support the management of many tasks **[Usage]**
5. Describe reasons for using interrupts, dispatching, and context switching to support concurrency in an operating system **[Usage]**
6. Create state and transition diagrams for simple problem domains **[Familiarité]**

4.3.4 OS/Scheduling and Dispatch

Notions abordées

- Preemptive and non-preemptive scheduling (cross reference SF/Resource Allocation and Scheduling, PD/Parallel Performance) **[Enseigné]**
- Schedulers and policies (cross reference SF/Resource Allocation and Scheduling, PD/Parallel Performance) **[Enseigné]**
- Processes and threads (cross reference SF/computational paradigms) **[Enseigné]**

Compétences acquises

1. Compare and contrast the common algorithms used for both preemptive and non-preemptive scheduling of tasks in operating systems, such as priority, performance comparison, and fair-share schemes **[Familiarité]**
2. Describe relationships between scheduling algorithms and application domains **[Non acquis]**
3. Discuss the types of processor scheduling such as short-term, medium-term, long-term, and I/O **[Familiarité]**
4. Describe the difference between processes and threads **[Usage]**
5. Compare and contrast static and dynamic approaches to real-time scheduling **[Familiarité]**
6. Discuss the need for preemption and deadline scheduling **[Familiarité]**
7. Identify ways that the logic embodied in scheduling algorithms are applicable to other domains, such as disk I/O, network scheduling, project scheduling, and problems beyond computing **[Non acquis]**

4.3.5 OS/Memory Management

Notions abordées

- Review of physical memory and memory management hardware **[Enseigné]**
- Working sets and thrashing **[Non enseigné]**
- Caching **[Non enseigné]**

Compétences acquises

1. Explain memory hierarchy and cost-performance trade-offs **[Non acquis]**
2. Summarize the principles of virtual memory as applied to caching and paging **[Familiarité]**
3. Evaluate the trade-offs in terms of memory size (main memory, cache memory, auxiliary memory) and processor speed **[Familiarité]**
4. Defend the different ways of allocating memory to tasks, citing the relative merits of each **[Familiarité]**
5. Describe the reason for and use of cache memory (performance and proximity, different dimension of how caches complicate isolation and VM abstraction) **[Familiarité]**
6. Discuss the concept of thrashing, both in terms of the reasons it occurs and the techniques used to recognize and manage the problem **[Familiarité]**

4.3.6 SF/Resource Allocation and Scheduling

Notions abordées

- Kinds of resources : processor share, memory, disk, net bandwidth **[Enseigné]**
- Kinds of scheduling : first-come, priority **[Enseigné]**
- Advantages of fair scheduling, preemptive scheduling **[Enseigné]**

Compétences acquises

1. Define how finite computer resources (e.g., processor share, memory, storage and network bandwidth) are managed by their careful allocation to existing entities [Familiarité]
2. Describe the scheduling algorithms by which resources are allocated to competing entities, and the figures of merit by which these algorithms are evaluated, such as fairness **[Non acquis]**
3. Implement simple schedule algorithms [Usage]
4. Measure figures of merit of alternative scheduler implementations **[Non acquis]**

4.3.7 SF/Virtualization and Isolation

Notions abordées

- Rationale for protection and predictable performance **[Non enseigné]**
- Levels of indirection, illustrated by virtual memory for managing physical memory resources [Enseigné]
- Methods for implementing virtual memory [Enseigné]

Compétences acquises

1. Explain why it is important to isolate and protect the execution of individual programs and environments that share common underlying resources, including the processor, memory, storage, and network access [Familiarité]
2. Describe how the concept of indirection can create the illusion of a dedicated machine and its resources even when physically shared among multiple programs and environments [Familiarité]
3. Measure the performance of two application instances running on separate virtual machines, and determine the effect of performance isolation **[Non acquis]**

4.3.8 SF/Proximity

Notions abordées

- Speed of light and computers (one foot per nanosecond vs. one GHz clocks) [Enseigné]
- Latencies in computer systems : memory vs. disk latencies vs. across the network memory [Enseigné]
- Caches, spatial and temporal locality, in processors and systems [Enseigné]
- Caches, cache coherency in database, operating systems, distributed systems, and computer architecture **[Non enseigné]**
- Introduction into the processor memory hierarchy : registers and multi-level caches, and the formula for average memory access time [Enseigné]

Compétences acquises

1. Explain the importance of locality in determining performance. [Familiarité]
2. Describe why things that are close in space take less time to access. [Familiarité]
3. Calculate average memory access time and describe the tradeoffs in memory hierarchy performance in terms of capacity, miss/hit rate, and access time. [Familiarité]

4.3.9 SF/Evaluation

Notions abordées

- Choosing and understanding performance figures of merit (e.g., speed of execution, energy consumption, bandwidth vs. latency, resource cost)

– Choosing and understanding workloads and representative benchmarks (e.g., SPEC, Dhrystone), and methods of collecting and analyzing performance figures of merit
– CPI equation (Execution time = Nb. of instructions * cycles/instruction * time/cycle) as tool for understanding tradeoffs in the design of instruction sets, processor pipelines, and memory system organizations.
– Amdahl's Law : the part of the computation that cannot be sped up limits the effect of the parts that can

Compétences acquises

1. Explain how the components of system architecture contribute to improving its performance. **[Familiarité]**
2. Describe Amdahl's law and discuss its limitations. **[Familiarité]**
3. Design and conduct a performance-oriented experiment, e.g., benchmark a parallel program with different data sets in order to iteratively improve its performance. **[Familiarité]**
4. Use software tools to profile and measure program performance. **[Familiarité]**

4.3.10 SF/Virtualization and Isolation

Notions abordées

– Rationale for protection and predictable performance **[Non enseigné]**
– Levels of indirection, illustrated by virtual memory for managing physical memory resources **[Enseigné]**
– Methods for implementing virtual memory and virtual machines **[Non enseigné]**

Compétences acquises

1. Explain why it is important to isolate and protect the execution of individual programs and environments that share common underlying resources, including the processor, memory, storage, and network access. **[Non enseigné]**
2. Describe how the concept of indirection can create the illusion of a dedicated machine and its resources even when physically shared among multiple programs and environments. **[Familiarité]**
3. Measure the performance of two application instances running on separate virtual machines, and determine the effect of performance isolation. **[Non enseigné]**

4.3.11 OS/Real Time and Embedded Systems

Notions abordées

– Process and task scheduling **[Enseigné]**
– Memory/disk management requirements in a real-time environment **[Non enseigné]**
– Failures, risks, and recovery **[Non enseigné]**
– Special concerns in real-time systems **[Enseigné]**

Compétences acquises

1. Describe what makes a system a real-time system **[Familiarité]**
2. Explain the presence of and describe the characteristics of latency in real-time systems **[Familiarité]**
3. Summarize special concerns that real-time systems present and how these concerns are addressed **[Familiarité]**

4.3.12 SE/Software Project Management

Notions abordées

- Team participation
  - Team processes including responsibilities for tasks, meeting structure, and work schedule **[Enseigné]**
  - Roles and responsibilities in a software team **[Enseigné]**
  - Team conflict resolution **[Non enseigné]**
  - Risks associated with virtual teams (communication, perception, structure) **[Non enseigné]**
- Effort Estimation (at the personal level) **[Enseigné]**
- Risk **[Non enseigné]**
- Team management **[Enseigné]**
- Project management **[Non enseigné]**
- Software measurement and estimation techniques **[Non enseigné]**
- Software quality assurance and the role of measurements **[Non enseigné]**
- Risk **[Non enseigné]**
- System-wide approach to risk including hazards associated with tools **[Non enseigné]**

Compétences acquises

1. Identify behaviors that contribute to the effective functioning of a team **[Familiarité]**
2. Create and follow an agenda for a team meeting **[Familiarité]**
3. Identify and justify necessary roles in a software development team **[Familiarité]**
4. Understand the sources, hazards, and potential benefits of team conflict **[Non acquis]**
5. Apply a conflict resolution strategy in a team setting **[Non acquis]**
6. Use an ad hoc method to estimate software development effort (e.g., time) and compare to actual effort required **[Non acquis]**
7. List several examples of software risks **[Non acquis]**
8. Describe the impact of risk in a software development life cycle **[Non acquis]**
9. Describe different categories of risk in software systems **[Non acquis]**
10. Tout ce qui est marqué *Elective* **[Non acquis]**