UNIVERSITÀ DEGLI STUDI FIRENZE

DINFO
Dipartimento di Ingegneria dell'Informazione

Software Engineering for Embedded Systems

Introduction to the course

Laura Carnevali

Software Technologies Lab
Department of Information Engineering
University of Florence
`http://stlab.dinfo.unifi.it/carnevali`
laura.carnevali@unifi.it

Outline

1. Credits
2. Contents of the course
3. Organization of the course

Credits

Credits

- Large part of the materials presented in these slides is taken from:
  - the slides of the course "Real-Time Systems" given by Prof. Giorgio Buttazzo: `http://retis.sssup.it/~giorgio/rts-MECS.html`
  - the slides of the course "Real-Time Systems" given by Prof. Tullio Vardanega: `https://www.math.unipd.it/~tullio/RTS/2019`
- These slides are authorized for personal use only
- Any other use, redistribution, and profit sale of the slides (in any form) requires the consent of the copyright owners

Contents of the course

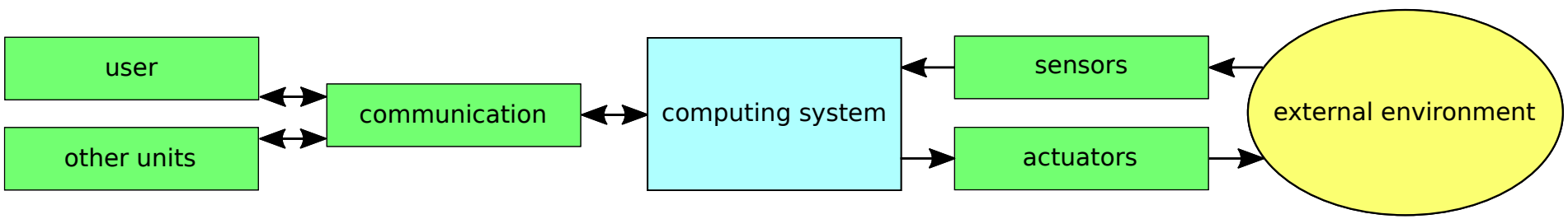
This course is about

- Design, implementation, and testing of **embedded software**
  - What are embedded systems?
  - Where does embedded software go?
  - What are the challenges in software design?
  - What are the guidelines for software programming and implementation?
  - What are the strategies for software testing?

Image from `www.digitalengineering247.com`

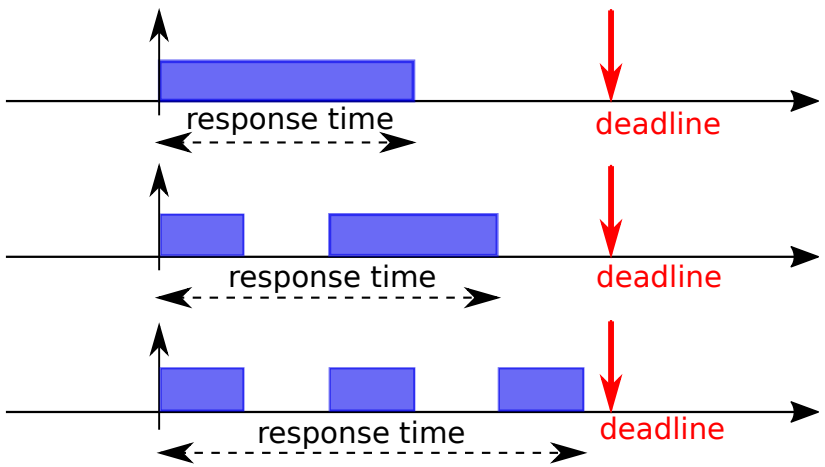
Embedded systems

- An **embedded system** is a dedicated computing system designed for specific functions within a larger system, often with **real-time** computing constraints
  - Real-time, stand-alone, networked, mobile embedded systems
  - Equipped with microcontrollers, microprocessors, custom-designed chips
  - Possibly equipped with (even complex and graphical) user interfaces
- Embedded systems vs general-purpose systems
  - Embedded systems are dedicated to a single purpose or few purposes
  - General-purpose systems are dedicated to multiple purposes
- Embedded systems vs cyber-physical systems
  - Embedded systems are essentially closed systems
  - Cyber-physical systems are essentially open systems



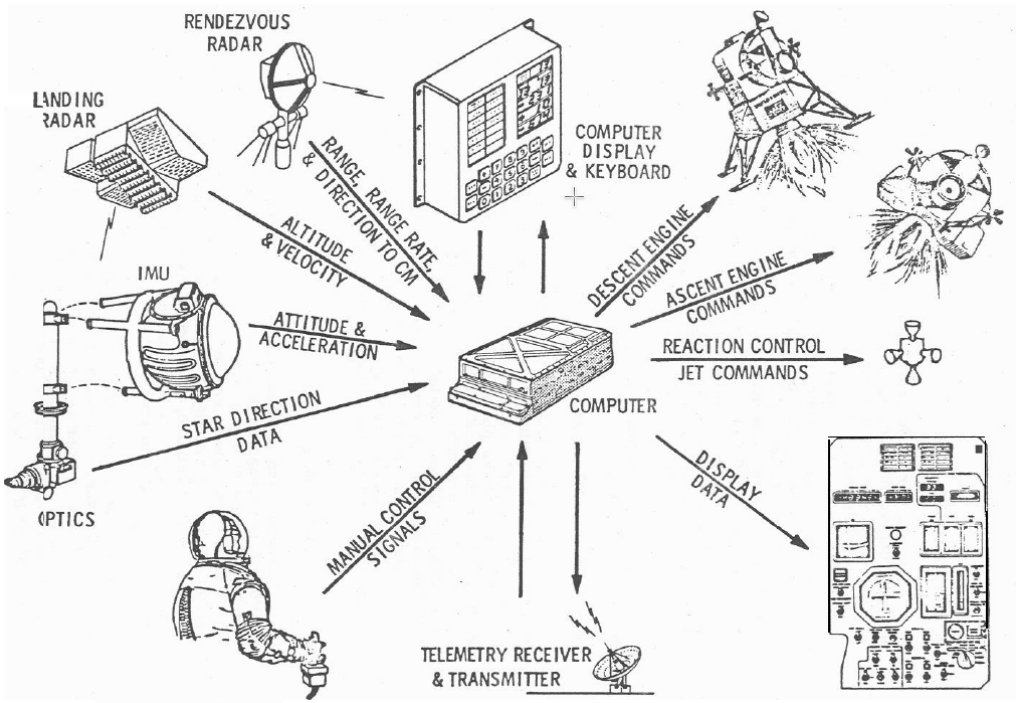
Real-time embedded systems

- A **real-time system** is a computing system subject to **timing constraints**
  - Response times and inter-task interference must be bounded in all scenarios
  - Time is a property not intrinsic but strictly related to the environment
- The satisfaction of timing constraints must be proved
- Correctness depends both on the logical result (**functional** correctness) and on the time at which the result is produced (**non-functional** correctness)
  - A late logically-correct response may be as bad as a logically-wrong response
- A real-time systems is typically **embedded** in a larger system to control its functions, manage the available resources, simplify the user interaction



Some history on embedded systems (1/3)

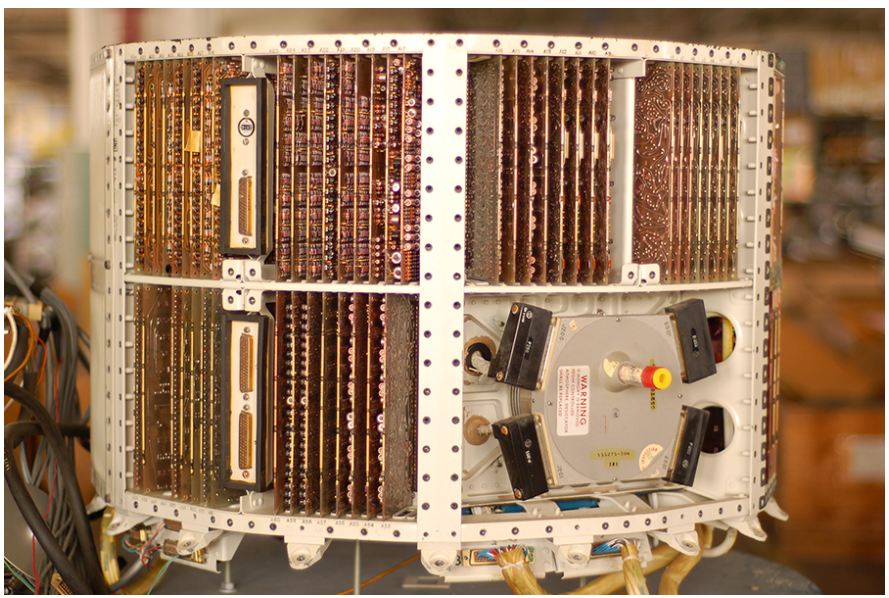
- Early 1960s: Apollo Guidance Computer (by the MIT Instrumentation Lab)
  - The first real-time embedded system, providing spacecraft guidance, navigation, and control for the Apollo Command Module and Lunar Module
  - One of the first computers equipped with integrated circuits
  - Use of software and hardware simulators for software verification and validation



Images from `http://mitmuseum.mit.edu` and `http://www.mit.edu`

Some history on embedded systems (2/3)

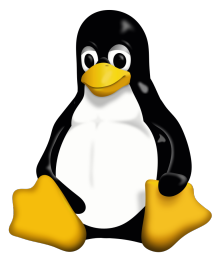
- Mid 1960's: missile guidance systems (by Autonetics, now Boeing)
  - D-17B: the first mass-produced embedded system
  - NS-17: the first embedded system with high-volume use of integrated circuits
- Late 1960's: microprocessor for electronic fuel injection control (by Bosch)
  - The first embedded system released in the automotive industry



Images from `www.minutemanmissile.com` and `www.bosch-presse.de`

Some history on embedded systems (3/3)

- Late 1960's - early 1970's: drop in price of integrated circuits, rise in usage
  - TMS1000: the first commercially available microcontroller (by Texas Instrum.)
  - 4004: the first commercially available microprocessor (by Intel)
  - ...
- Late 1980's - 1990's
  - VxWorks: the first real-time embedded operating system (by WindRiver)
  - Windows Embedded CE: real-time embedded operating system (by Microsoft)
  - Embedded Linux: embedded operating systems based on the Linux kernel
  - RTLinux: real-time operating system based on the Linux kernel



Images from `www.windriver.com`, `http://embeddedcdmx.com`, and `https://en.wikipedia.org`

Evolution of embedded systems (1/3)

- Number of transistors on integrated circuits (1971-2018)

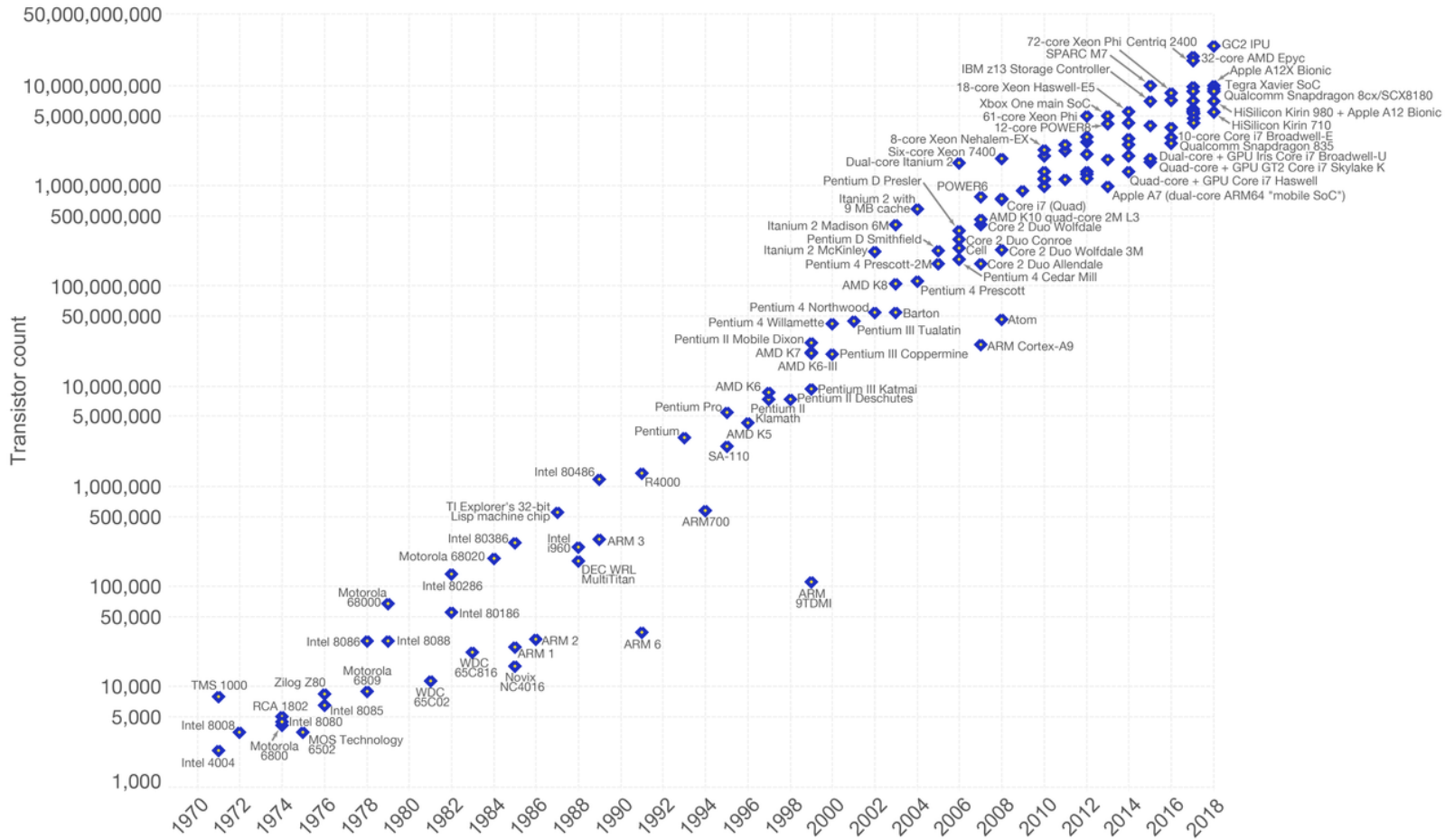


Image from `https://en.wikipedia.org`

Evolution of embedded systems (2/3)

- Number of connected devices (1992-2020)

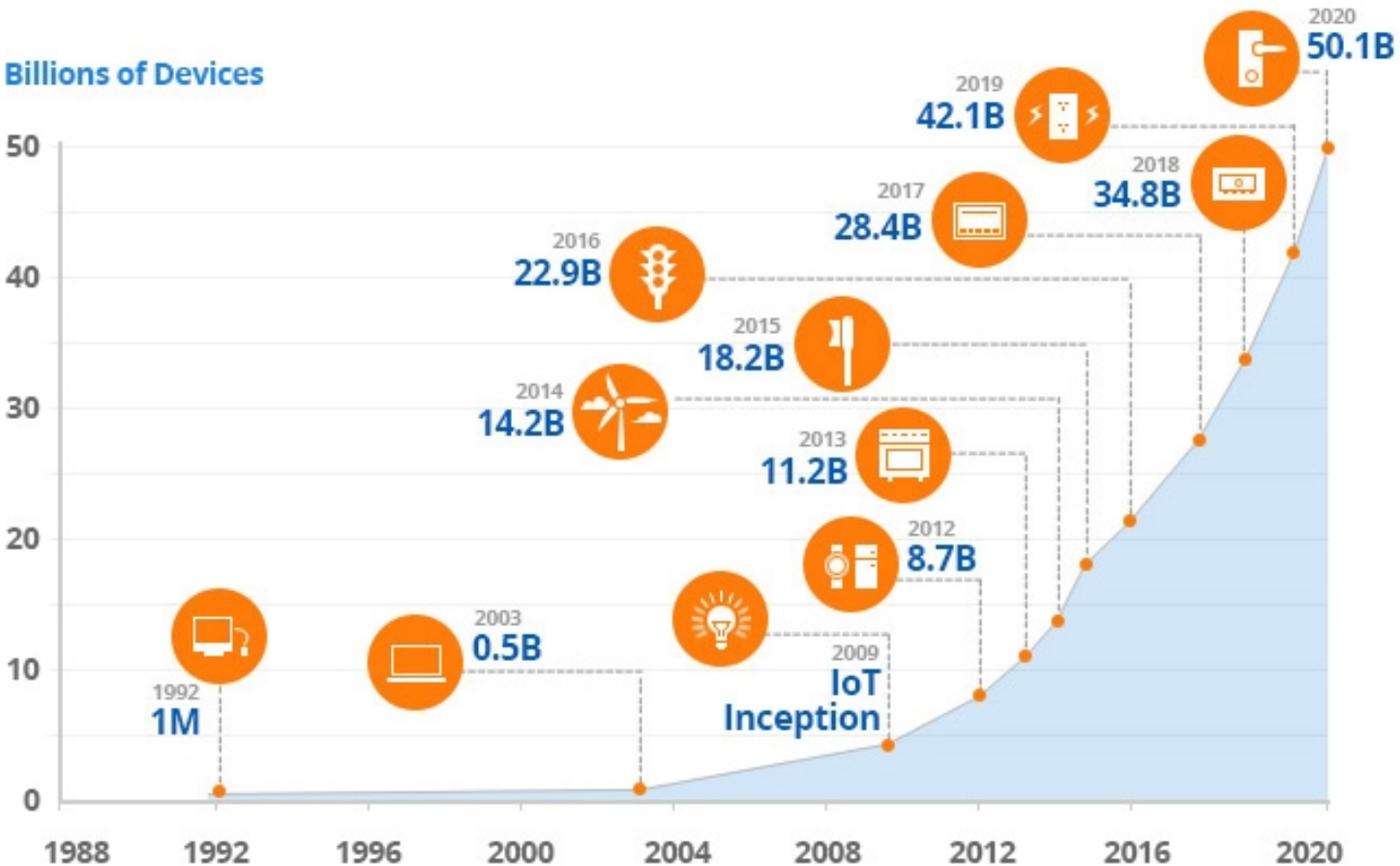


Image from `https://dzone.com` (source: CSCO)

Evolution of embedded systems (3/3)

- Global embedded system market revenue (2015-2021)

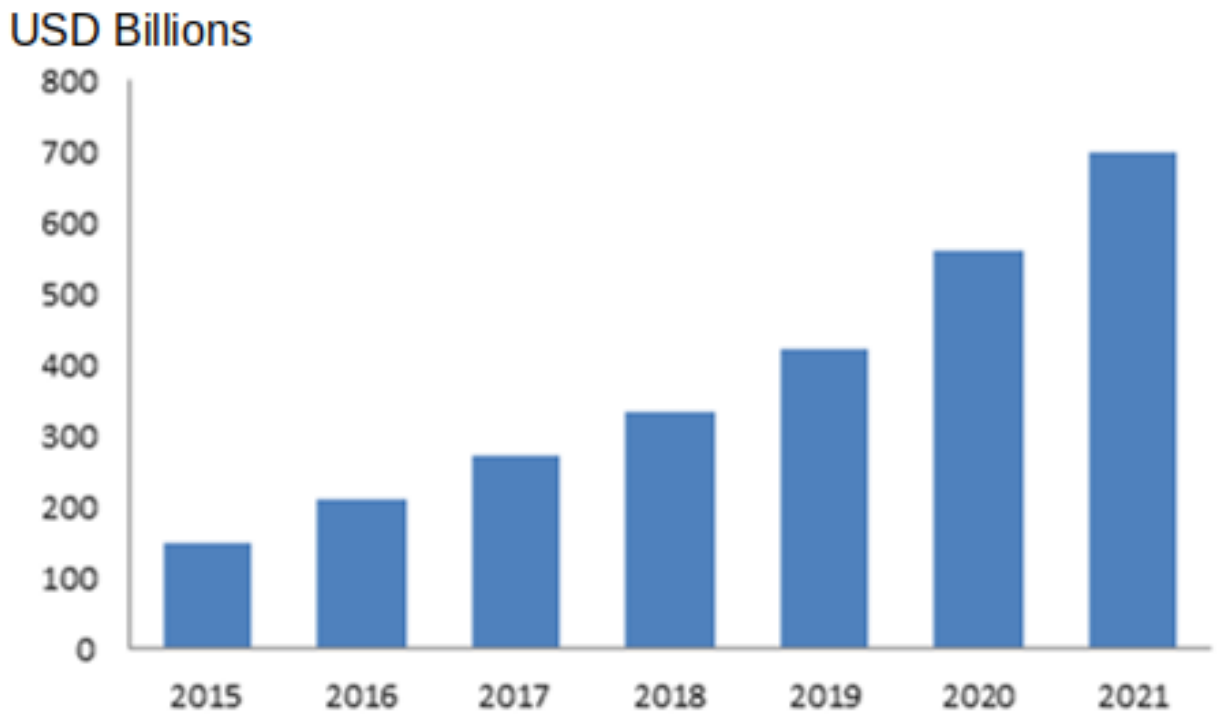
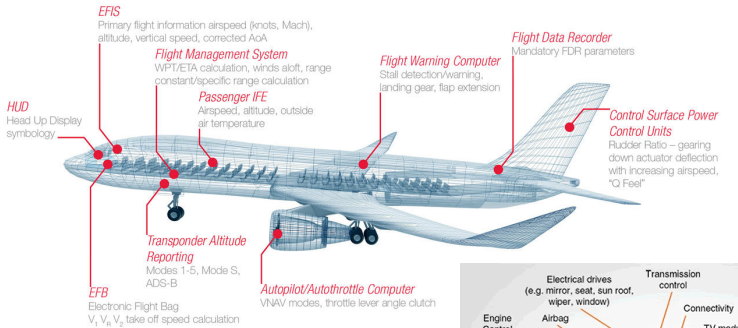
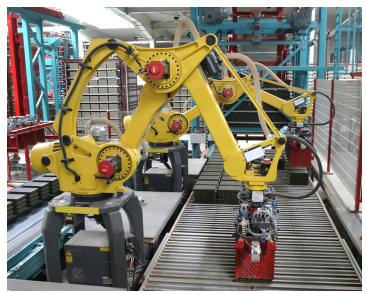


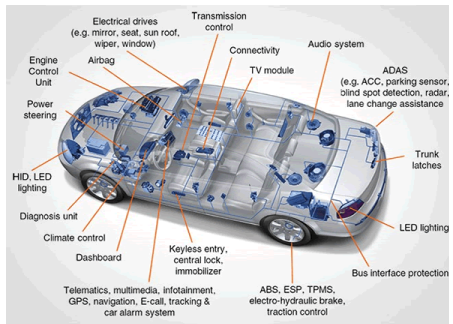
Image from `https://semiengineering.com` (source: Zion Research Analysis 2016)

Embedded systems are present everywhere

- Nowadays, 98% of processors in the world are used in embedded systems
  - Avionics applications
  - Automotive applications
  - Military and aerospace applications
  - Robotics
  - Industrial automation
  - Healthcare applications
  - Consumer electronics
  - Telecommunications
  - Multimedia applications
  - Intelligent transportation systems
  - ...



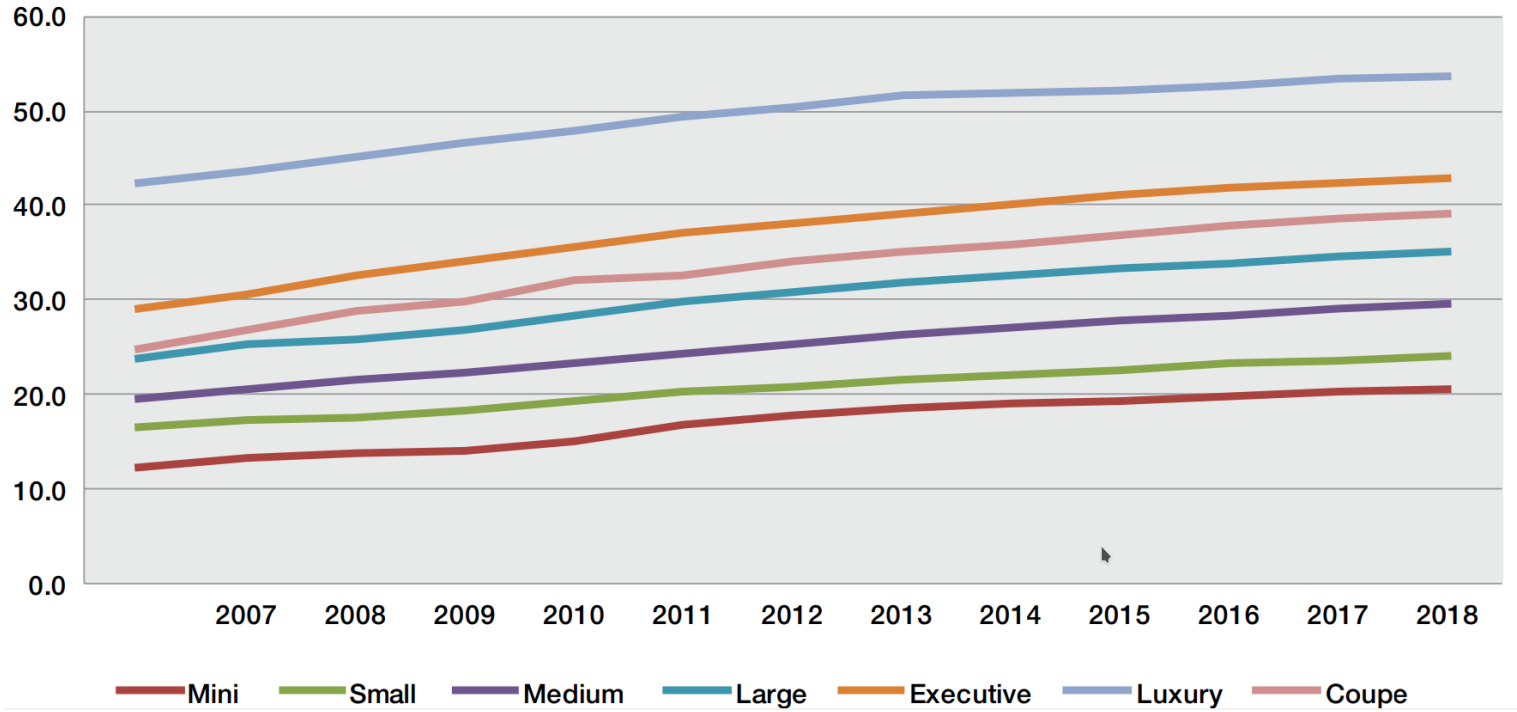




Images from `http://interactive.aviationtoday.com`, `www.connectorsupplier.com`, `www.lg.com`, `http://satcommunity.com.au`, `www.agi-automation.com`, `http://wearablecomputergear.com`

Increasing complexity of embedded systems

- Average number of Electronic Control Units (ECUs) in a car
  - There can be up to 100 ECUs in a modern luxury car



Images from `https://www.nxp.com` (source: Strategy Analytics)

Where does embedded software go?

- Embedded software controls almost everything in a car

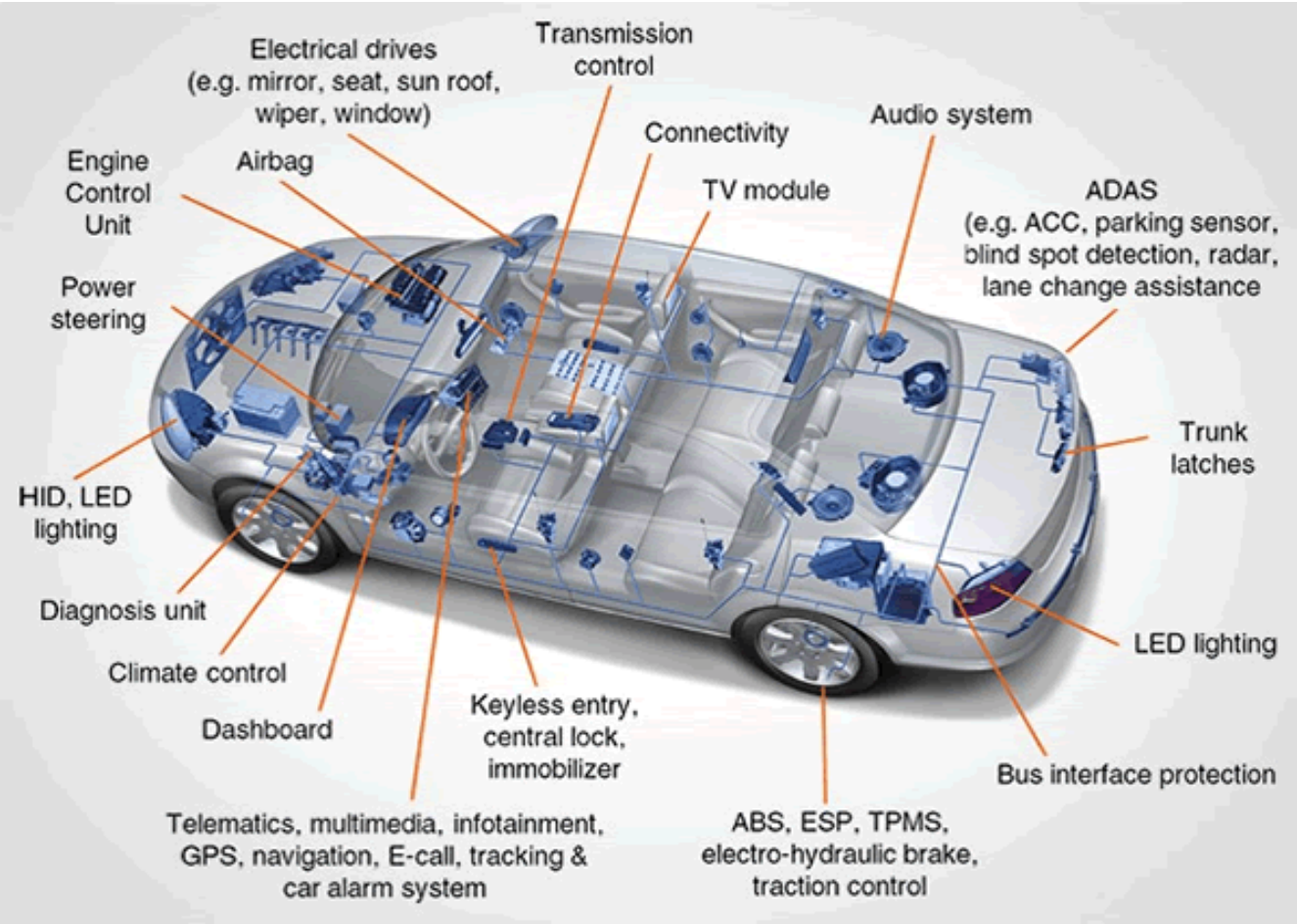


Image from `https://www.connectorsupplier.com`

Complexity of embedded software (1/2)

- Average number of Lines of Code (LOC) in a car

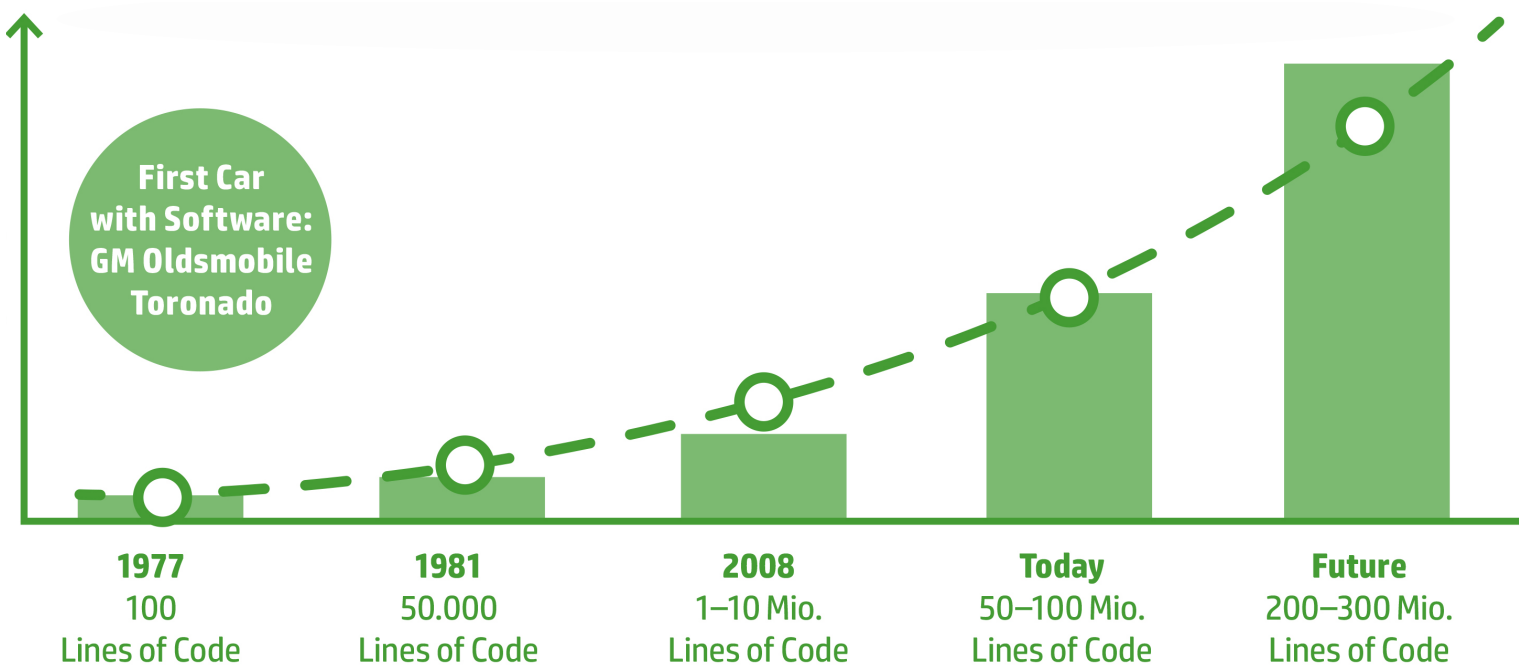


Image from `http://plattform-maerkte.de` (sources: projekt-race, spectrum ieee)

Complexity of embedded software (2/2)

- Comparing software complexity: cars are the most software-intensive systems

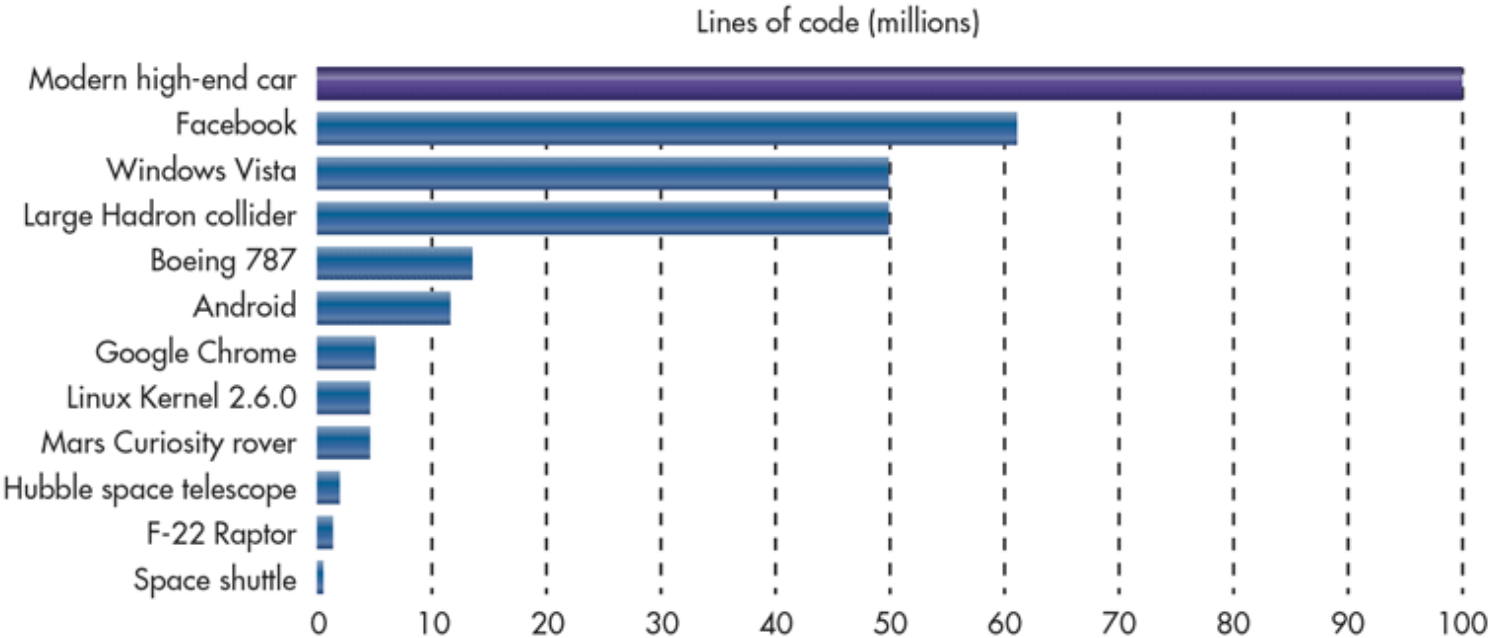
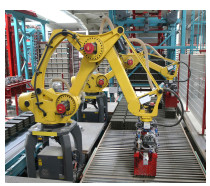


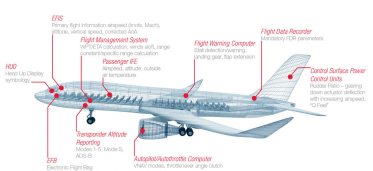
Image from `www.electronicdesign.com`
(sources: NASA, IEEE, Wired, Boeing, Microsoft, Linux Foundation, Ohioh)

What's special in embedded systems?

- Characteristics
  - **High heterogeneity** of components and processing activities
  - **High variability** in size and scope
  - **Scarce resources** (space, weight, time, memory, energy, . . .)
  - **High concurrency** and resource sharing (high task interference)
  - **Interaction with the environment**: event-driven and time-driven
  - **High variability** in workload and resource demand
- Requirements
  - **Proven dependability** (reliability, availability, maintainability, safety, . . .)
  - **Continuity of operation** without (constant) human supervision
  - **High efficiency** in resource management
  - **Timeliness** (and also temporal isolation to limit task interference)
  - **High predictability** in responding to environment-events and time-events
  - **Adaptivity** (robustness) to handle overload conditions





Sources of non-determinism in embedded systems (1/2)

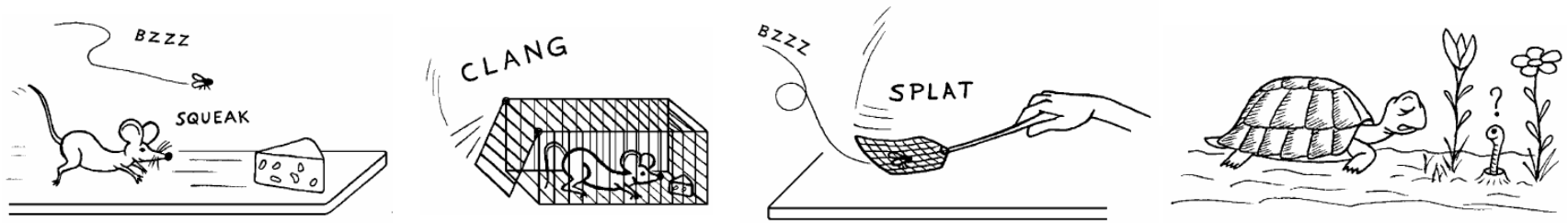
- Multitasking (system calls supporting concurrent programming)
  - May introduce unbounded delays on the execution time of tasks
- Priority-based scheduling
  - Supports the implementation of various approaches for task management
  - Task arrivals may require remapping of (a limited number of) priorities
- Fast response to external interrupts
  - Obtained by setting interrupt priorities larger than task priorities and by executing most of the code with interrupts enabled
  - Improves the system reactivity to external events
  - May introduce unbounded delays on the execution time of tasks
- Direct Memory Access (DMA)
  - Data transfer from devices to main memory independently of the CPU
  - Cycle stealing (CPU blocked if DMA is performing a data transfer): the number of cycles waited by the CPU becomes unpredictable
  - Time-slice method (dedicated time slots): less efficient, more predictable

Sources of non-determinism in embedded systems (2/2)

- Cache
  - Speeds up processor execution and reduces conflicts with other devices
  - Affected by the number of preemptions (which destroy program locality)
- System calls
  - Should be preemptable and with bounded execution time
- Mechanisms for task communication and synchronization
  - May have undesirable effects (e.g., priority inversion, blocking, deadlock) if resource access protocols are not used, introducing unbounded delays
- Memory management
  - Static allocation improves predictability but reduces flexibility
- Small kernel and fast context switch
  - Reduces the average response time of tasks
  - Does not guarantee that timing requirements are met
- Scarce support for the specification of timing constraints on tasks

Meeting requirements: false myths to dispel

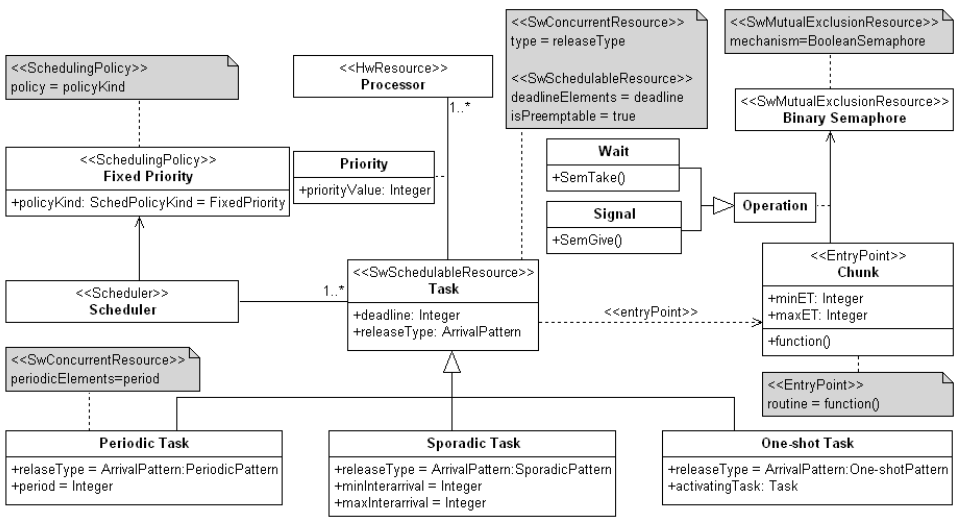
- Real-time programming is **not empirical**
  - Control software produced by empirical techniques can be highly unpredictable
- Real-time programming is **not low-level**
  - Low-level programming is tedious and yields code difficult to maintain
  - Verification of requirements is easier if programming is high-level
- Minimizing the **average response time** is not enough
  - Increase in CPU power will not be enough to satisfy real-time requirements
  - Real-time computing is not fast computing (speed is not the point)
- Software **testing** is essential but by itself is not enough
  - Software behavior depends on input data $\Rightarrow$ testing provides partial verification
- Hardware/software **simulation** is useful but by itself is not enough

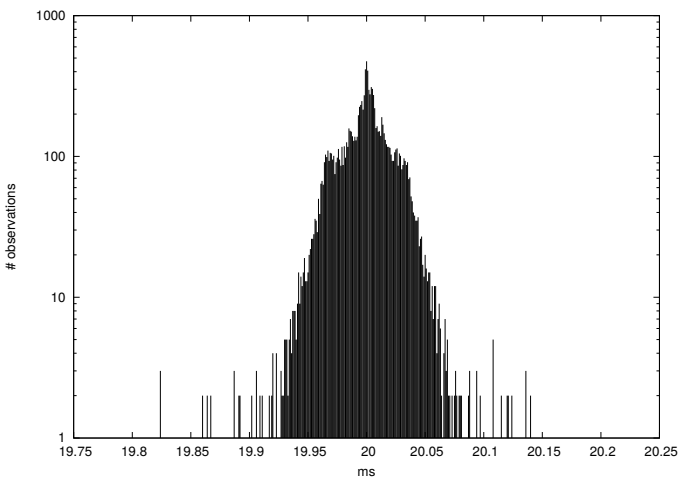


Images from "Hard real-time computing systems" by Prof. G. Buttazzo

Meeting requirements: predictability

- System-level **predictability** is needed
  - Ability to predict task behavior and guarantee timing requirements in advance
- How can predictability and other requirements be met?
  - Novel solutions are needed for software design and testing
  - Efficient schedulability and feasibility tests are needed
  - Estimating the **Worst-Case Execution Time (WCET)** of tasks is essential
  - Novel solution are needed also for hardware design and testing (out of scope)





Meeting requirements: WCET estimation is not easy (1/3)

- Estimation by analysis
  - Bound the number of loop cycles
  - Compute the longest path
  - Bound the number of cache misses
  - Compute the execution time for each instruction in the longest path

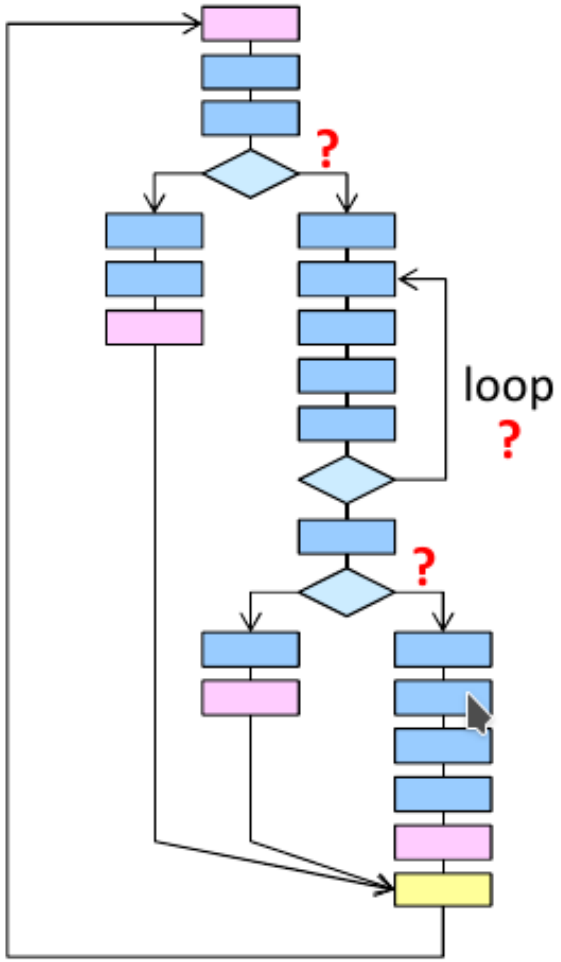


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Meeting requirements: WCET estimation is not easy (2/3)

- Estimation by measurements
  - Execute the task several times with different input data
  - Collect statistics on the task execution time

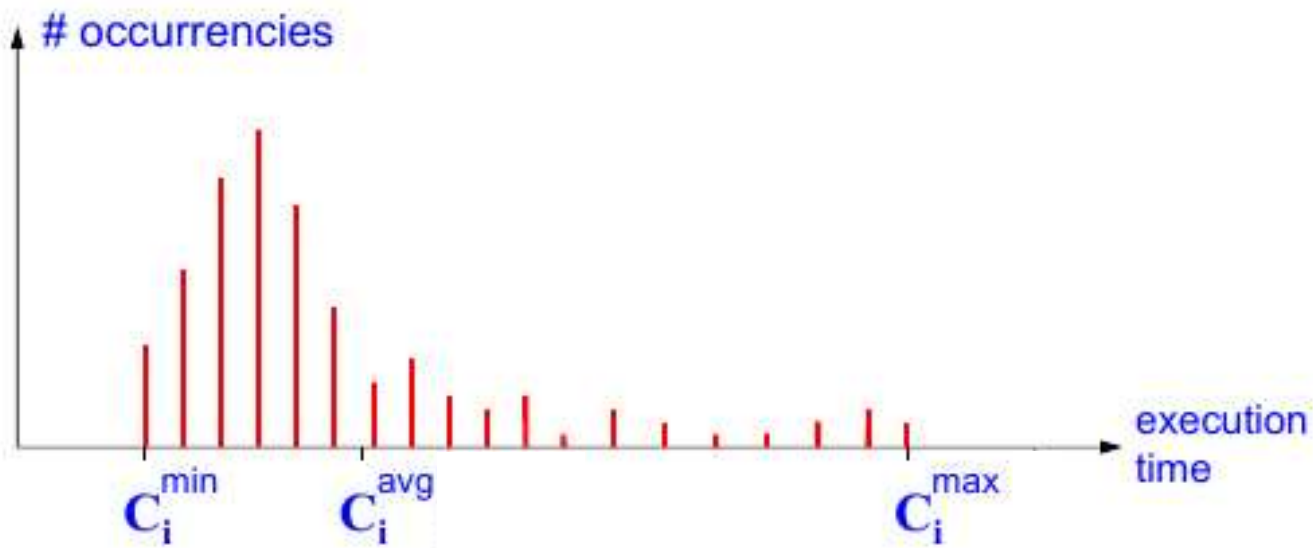


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Meeting requirements: WCET estimation is not easy (3/3)

- Combining estimation by analysis and by measurements
  - **BOET**: Best Observed Execution Time
  - **AOET**: Average Observed Execution Time
  - **WOET**: Worst Observed Execution Time
  - **WCET**: Worst Case Execution Time

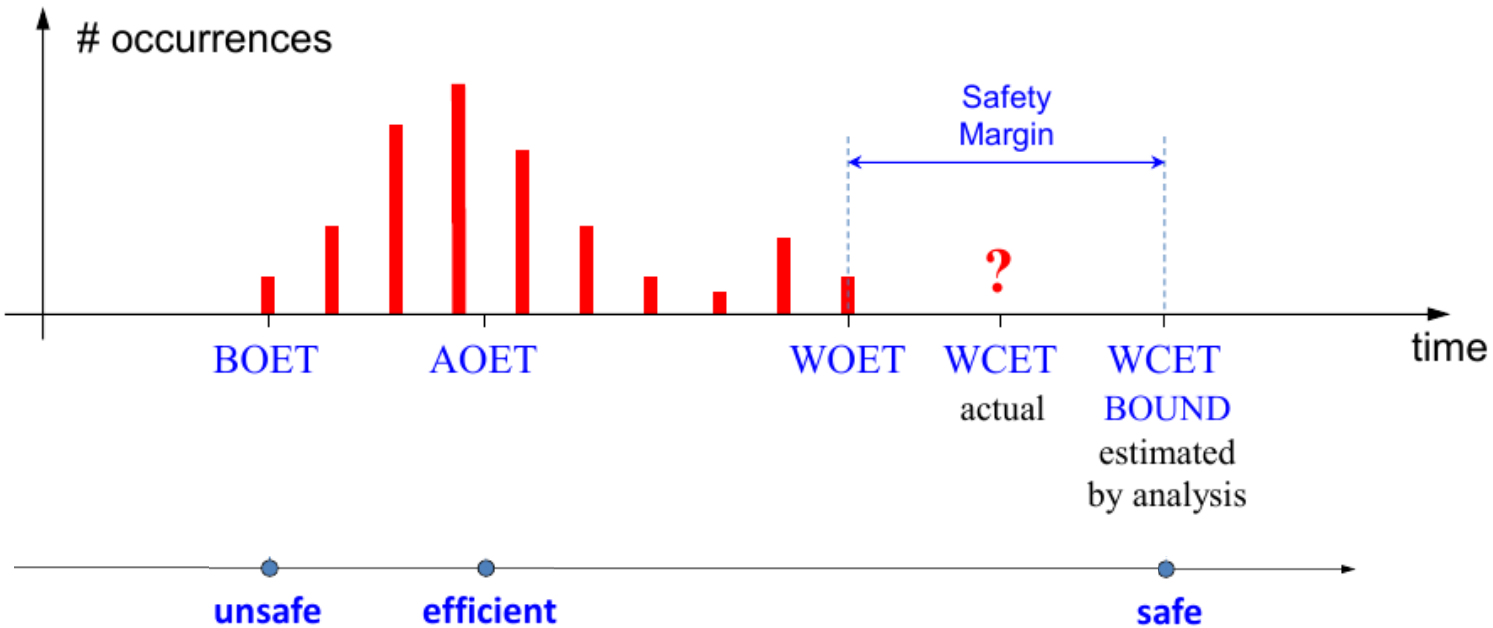
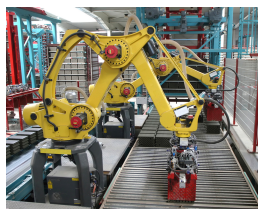
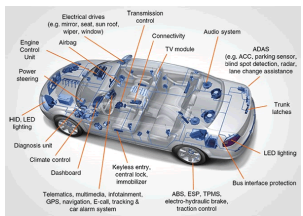
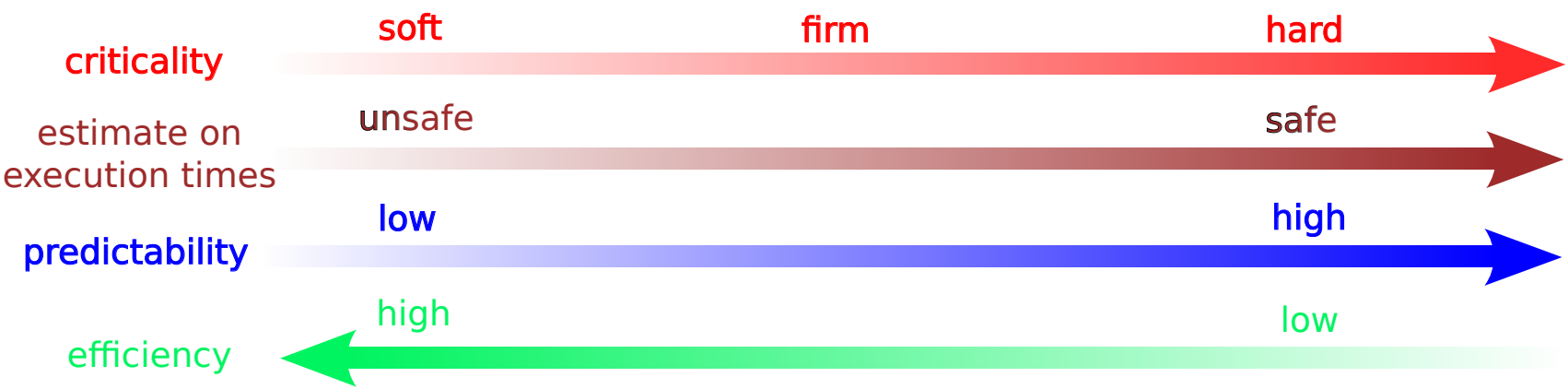


Image by Prof. G. Buttazzo, Scuola Superiore Sant'Anna, Pisa

Meeting requirements: predictability vs efficiency

- Criticality of embedded software (based on the effects of a deadline miss)
  - **Soft**: missing a deadline is not critical but may cause performance degradation
  - **Firm**: missing a deadline has not catastrophic effects but invalidates the task
  - **Hard**: missing a deadline may have catastrophic effects on the overall system

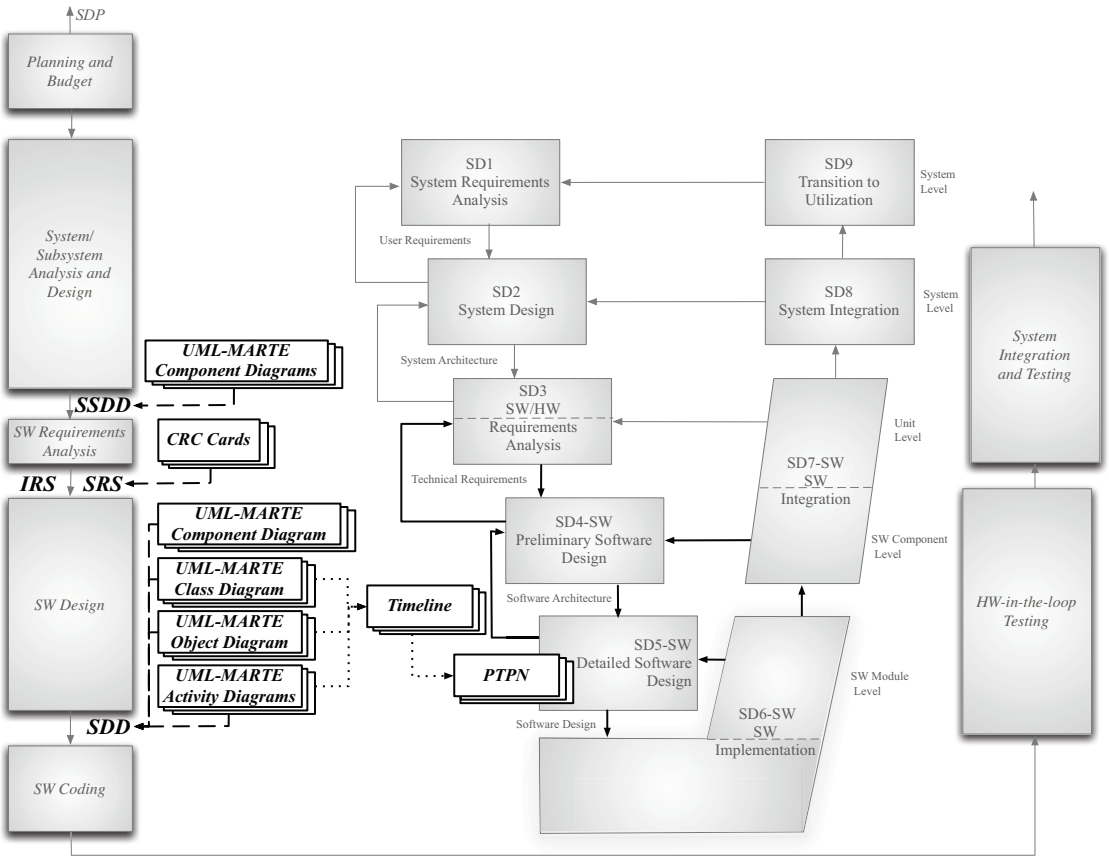




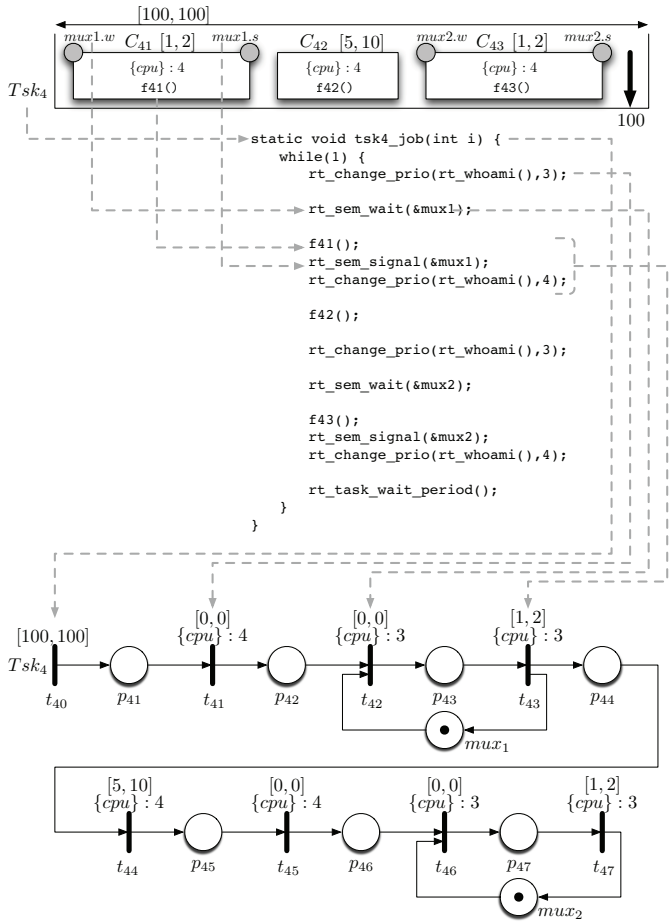


Images from `www.lg.com`, `http://www.agi-automation.com`, `https://www.connectorsupplier.com`

Aim of the course

- Study software methodologies improving predictability of embedded systems
  - How to model time-critical applications and analyze their timing properties
  - How to design, program, and test time-critical software applications





Program at a glance

- **Part 1: Real-time and embedded systems**
  - Scheduling algorithms for real-time embedded systems (basic concepts, periodic task scheduling, cyclic executive scheduling, rate monotonic scheduling, deadline monotonic scheduling, earliest deadline first scheduling)
  - Resource access protocols for real-time embedded systems (priority inheritance protocol, priority ceiling protocol, extended schedulability tests)
  - Real-time operating systems and standards (RT-POSIX, OSEK/VDX, ARINC-APEX, the real-time operating system VxWorks, development of real-time applications on Raspberry Pi)
- **Part 2: Advanced topics on scheduling and testing**
  - Advanced topics on schedulability analysis (Petri nets, time Petri nets, preemptive time Petri nets, using preemptive time Petri nets in a V-Model SW life cycle subject to MIL-STD-498, timed automata)
  - Software testing (testing methodology, control flow and data flow testing, finite state testing, real-time testing)
- **Part 3: Systems engineering**
  - Elements of model-based systems engineering (SysML)

Organization of the course

Lectures

- Moodle web page
  - `https://e-l.unifi.it/course/view.php?id=42580`
  - Password: SWEES
  - Register to the course web page to receive announcements
- Lectures (first semester, from 16/09/2024 to 13/12/2024)
  - Tuesday 16:00-18:00, room 49, Santa Marta
  - Wednesday 15:00-17:00, room 33, Santa Marta
- The schedule of lectures will be confirmed week by week:
  - Lecture 1 (17/09/2024) - Introduction to the course
  - Lecture 2 (18/09/2024) - Scheduling algorithms
  - ...
- Lab sessions with Dr Imad Zaza (to be announced during the course)
- Student meetings
  - Emails with comments or requests for explanation are welcome
  - Send an email to arrange a meeting in standard office hours or in other hours

Exam

- Assessment of theoretical and practical skills acquired on the course topics
- Exam of "Software Engineering for Embedded Systems" (6 CFU)
  - Consists of an oral test on the course contents
  - Students can individually develop a small self-assignment (optional).
- Exam of "Software Engineering for Embedded Systems Laboratory" (3 CFU)
  - Consists of the discussion of an assignment related to the course contents
  - The assignment can be developed individually or by a group of students (typically 2-3 students, each developing a clearly identifiable contribution)
  - The subject is agreed with the teacher (proposals by students are welcome)
- Exam dates (available also on the course page)
  - Students must register to the exam: `http://sol.unifi.it/prenot/prenot`

Oral test with self-assignment (6 CFU)

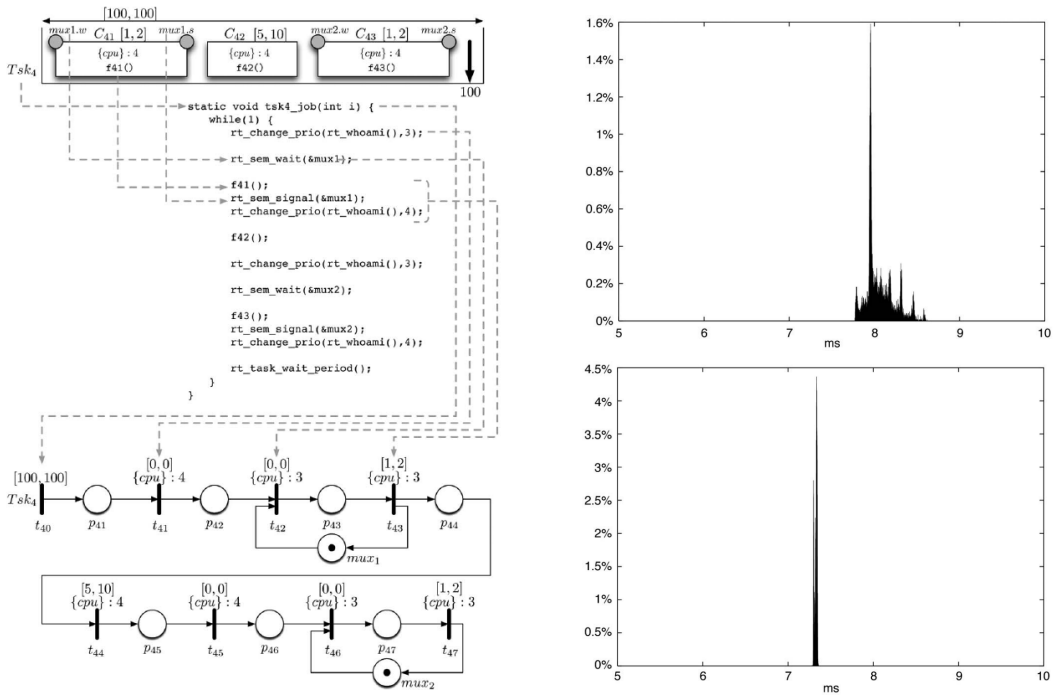
- Developing a self-assignment is not mandatory
- Students developing a self-assignment should be able to define the topic individually and are required to send a short report one week before the exam
- Examples of self-assignment topics
  - Implement a set of real-time tasks managed by fixed-priority preemptive scheduling on the real-time operating system VxWorks
  - Use ORIS 1.0 or Uppaal (or any other tool with comparable expressivity) to verify the requirements of some concurrent timed system, e.g., a set of real-time tasks managed by fixed-priority preemptive scheduling
  - Use Papyrus to develop SysML diagrams of some embedded system
  - ...

Project work (3 CFU)

- Arrange a group of (typically 1 to 3) students and then, **when you are ready to start working**, send an email to agree on the topic of the assignment
- You are required to give a presentation of results during the exam
  - Ensure the contribution of each student to the work is clearly identifiable
- Examples of assignment topics
  - Use ORIS 1 to verify the requirements of a real-time task-set managed by fixed-priority preemptive scheduling; implement the task-set on VxWorks; instrument the code to log the time at which job releases and completions occur; and, analyze the time-stamped log to verify that deadlines are met
  - Instrument the VxWorks code of a real-time task-set to log the time at which any event of the PTPN model of the task-set occurs; and, process the time-stamped log using ORIS 1 to verify sequencing and timing constraints
  - Perform fault injection on the VxWorks code of a real-time task-set and exploit the state class graph of the PTPN model of the task-set to perform test case selection and to support test case execution
  - ...

An assignment on real-time programming on Linux-RTAI

- Implementation and testing of real-time task sets
  - Sets of periodic and sporadic tasks, with mutual exclusion resources, ...
  - Testing conformance to a formal model specified as preemptive time Petri net
- RTAI: Real-Time Application Interface for Linux (https://www.rtai.org)
  - Real-time task-sets implemented as kernel modules
  - Real-time task-sets running on an Intel NUC NUC10i3FNHN



Assignment developed in the academic year 2022/2023

An assignment on test case generation and prioritization

- Test case generation and prioritization for testing cyber-physical systems
  - Simulation of complex cyber-physical systems is computationally expensive
  - Generation of reactive test cases through genetic algorithms

IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, VOL. 14, NO. 3, MARCH 2018 1055

Employing Multi-Objective Search to Enhance Reactive Test Case Generation and Prioritization for Testing Industrial Cyber-Physical Systems

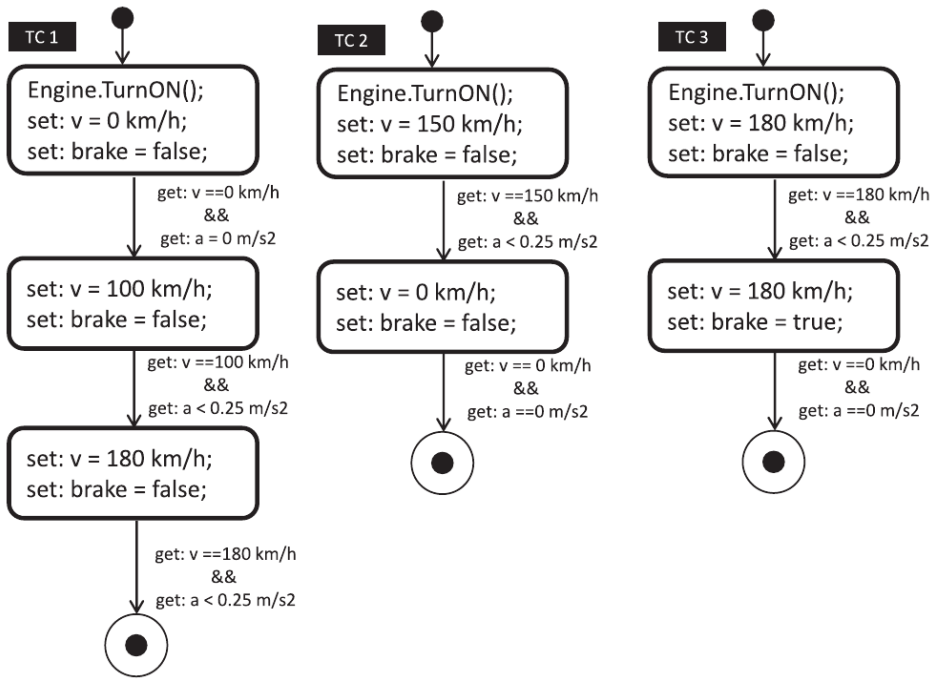
Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Goiuria Sagardui, and Leire Etxeberria

***Abstract*—The test case generation and prioritization of industrial cyber-physical systems face critical challenges, and simulation-based testing is one of the most commonly used techniques for testing these complex systems. However, simulation models of industrial CPSs are usually very complex, and executing the simulations becomes computationally expensive, which often make it infeasible to execute all the test cases. To address these challenges, this paper proposes a multi-objective test generation and prioritization approach for testing industrial CPSs by defining a fitness function with four objectives and designing different crossover and mutation operators. We empirically evaluated our fitness function and designed operators along with five multi-objective search algorithms [e.g., nondominated sorting genetic algorithm (NSGA-II)] using four case studies. The evaluation results demonstrated that NSGA-II achieved significantly better performance than the other algorithms and managed to improve random search for on average 43.80% for each objective and 49.25% for the quality indicator hypervolume.**

***Index Terms*—Automated validation, cyber-physical systems, test generation.**

pick-and-place machines [2], pacemakers [3], and unmanned aerial vehicles (UAVs) [4]. CPSs are starting to have a high impact on industrial automation for different applications such as manufacturing, electronics assembly, continuous processing, or energy management [5]. Complex industrial CPSs often include heterogeneous components such as electromechanical, thermal, computing, and communication elements, interconnected among them and encompassing environmental effects [6].

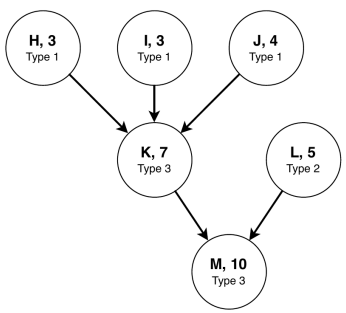
CPSs have been cataloged as untestable systems, and traditional testing techniques [e.g., model-based testing (MBT)] are usually expensive, time-consuming, or infeasible to apply [7]. This is due to the fact that it is challenging for the traditional testing techniques to capture complex continuous dynamics and interactions between the system and its environment (e.g., people walking around when automatic braking systems are in use for automotive systems) [7], [8]. As a result, simulation-based testing has been envisioned as an efficient means to test CPSs in a systematic and automated manner [7]. The use of simulation models permits several advantages such as

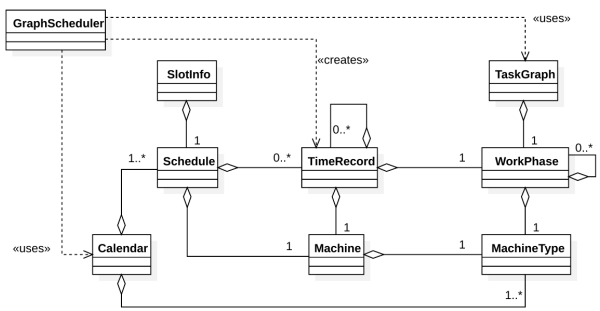


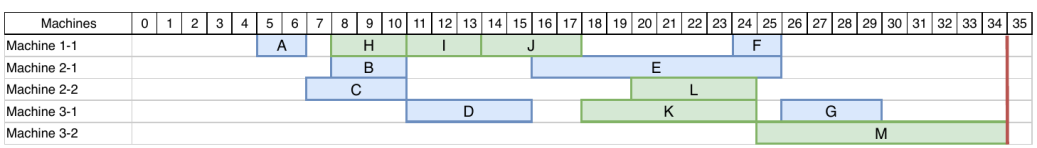
Assignment developed in the academic year 2022/2023

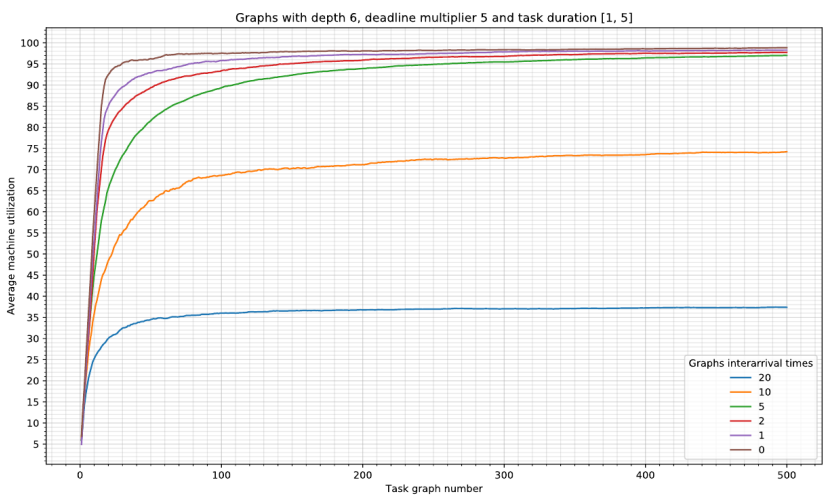
An assignment on scheduling of task graphs

- A scheduling problem referred to the context of the textile industry
  - A **task graph** represents the work phases of a **bill of materials** with deadline
  - Each work phase is executed by a subcontractor on a machine of a specific type
  - Each instance of a machine of a given type has its own working speed
- A **sequential non-preemptive** scheduling algorithm
  - The algorithm schedules one task graph at a time
  - The algorithm avoids task preemption due to practical costs
  - Tasks of each task graph are first scheduled by Earliest Deadline First (EDF) and then maximally postponed without violating precedences and deadlines





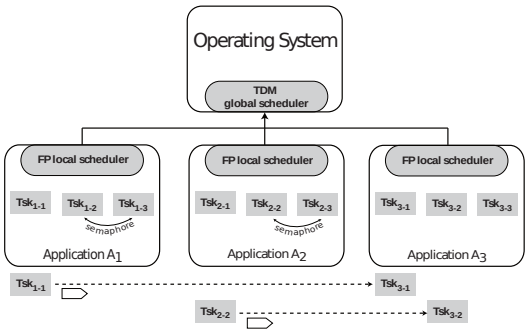




Assignment developed in the academic year 2019/2020

Examples of assignment topics include (but are not limited to):

- Schedulability analysis of real-time systems (e.g., hierarchical scheduling sys.)



Appl.	Slot	Slot length	Task	Release	Offset	Jitter	Deadline	Chunk	Prio	Exec. Time	Sem	Mbx
A_1	T_1	3	Tsk_{11}	[10,10]	0	[0,0]	5	C_{111}	2	[0.6,0.8]	-	-
								C_{121}	3	[1.0,1.2]	-	-
			Tsk_{12}	[40,40]	0	[0,1]	40	C_{122}	3	[0.2,0.4]	-	$mbx_{11}(\ell)$
			Tsk_{13}	[40,40]	10	[0,2]	40	C_{131}	4	[1.8,2.3]	-	-
								C_{132}	4	[0.6,0.9]	-	$mbx_{11}(s)$
			Tsk_{14}	[40,∞)	20	[0,0]	40	C_{141}	5	[1.1,1.4]	-	-
								C_{142}	5	[0.1,0.2]	-	-
A_2	T_2	4	Tsk_{21}	[40,∞)	0	[0,0]	40	C_{211}	2	[0.2,0.3]	mux_{21}	-
								C_{212}	2	[0.4,0.5]	-	-
			Tsk_{22}	[50,50]	0	[0,1]	50	C_{221}	3	[4.6,6.1]	-	-
								C_{222}	3	[0.2,0.3]	mux_{21}	-
			Tsk_{23}	[50,50]	0	[0,2]	50	C_{231}	4	[3.4,4.4]	-	-
								C_{232}	4	[0.2,0.4]	mux_{22}	-
			Tsk_{24}	[50,50]	16	[0,0]	50	C_{241}	5	[4.7,6.1]	-	-
								C_{242}	5	[0.1,0.3]	mux_{22}	-
A_3	T_3	1	Tsk_{31}	[80,80]	2	[0,0]	80	C_{311}	2	[3.6,4.8]	-	-
			Tsk_{32}	[100,∞)	15	[0,0]	100	C_{321}	3	[0.4,0.5]	-	-
A_4	T_4	1	Tsk_{41}	[100,100]	0	[0,2.5]	100	C_{411}	2	[3.4,4.2]	-	-
								C_{412}	2	[0.8,1.4]	mux_{41}	-
			Tsk_{42}	[200,∞)	10	[0,0]	200	C_{421}	3	[0.4,0.5]	-	-
								C_{422}	3	[0.2,0.3]	mux_{41}	-
A_5	T_5	1	Tsk_{51}	[200,200]	10	[0,0]	200	C_{511}	2	[1.2,1.6]	-	-
			Tsk_{52}	[400,∞)	3	[0,0]	400	C_{521}	3	[3.6,4.8]	-	-
			Tsk_{53}	[1000,1000]	0	[0,2]	1000	C_{531}	4	[3.0,4.0]	-	-

- Testing of real-time systems (e.g., test input generation)
- Analysis of performance of railway systems (e.g., physical/virtual coupling)

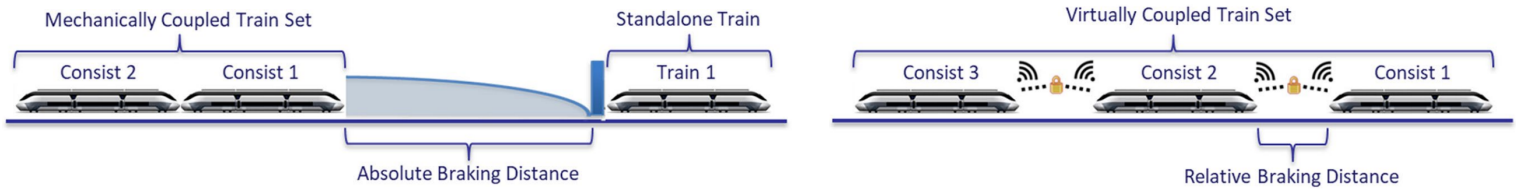


Image from "Technical feasibility analysis and introduction strategy of the virtually coupled train set concept", Scientific Reports, 2022

- Predictive maintenance in networked systems (e.g., edge-cloud comp. sys.)
- Resource allocation in networked systems (e.g., edge-cloud computing sys.)