WHY DO WE HAVE TO PREPARE A SOFTWARE SCHEDULABILITY ANALYSIS?

SOFTWARE PRODUCT ASSURANCE WORKSHOP 2023, ESAC, SPAIN

ANDREAS WORTMANN, 25-28.09.2023

AGENDA

WHY DO WE HAVE TO PREPARE A SOFTWARE SCHEDULABILITY ANALYSIS?

- INTRODUCTION
  - What is a Schedulability Analysis?
- QUICK OVERVIEW
  - Preconditions: Requirements, Software Architecture
  - The Analyses: Essential Steps, Methods, Tools
- THEORY AND PRACTICE
  - Lessons Learned
  - Technical and Process specific Considerations
- BENEFITS
  - Proof of timing constraints and requirements
  - Incidental Catch

WHAT IS A SCHEDULABILITY ANALYSIS?

INTRODUCTION

- Focus on Onboard Flight Software (FSW, OBSW, SCSW, ICSW …)
  - Software subsystems and central software item that **glues together subsystems** (hardware and software)
  - **Control loops**, including Attitude and Orbit Control Thermal Control
  - Telecommand and Telemetry Interfaces: Full **commandability and visibility**
  - **Robustness and Reliability**: FDIR and redundancy
  - Payload operation (very specific needs)

- Real-time needs & Timing requirements
  - Hardware access
  - Control algorithms

- Schedulability Analysis focuses on the (non-functional) **timing aspects** of the software (or entire systems)
  - increase confidence in correct timing
  - formally proof certain aspects correct under all circumstances
  - SA is required for flight software qualification by the [ECSS-E-40-1]

WHAT IS A SCHEDULABILITY ANALYSIS?

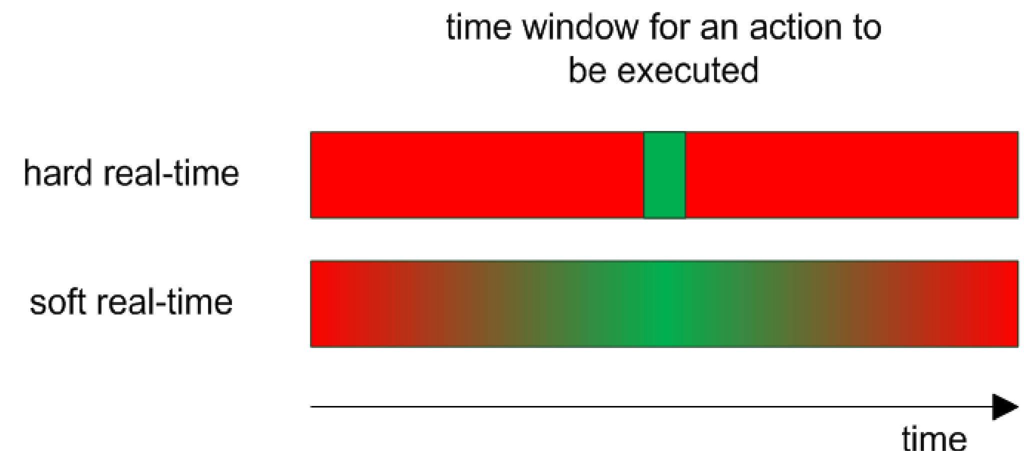
INTRODUCTION

- **Hard** deadline violation: loss of system integrity
  - “The s/w shall retrieve a data packet from the input buffer within 100ms after reception.”
  - “The AOCS loop including the data acquisition and equipment commanding shall be executed at a period of 1Hz”

  → **crucial deadline, a violation might have serious impact**
    - Loss of data or invalid calculation results
  - In general these requirements are also tight
    (small timing window for execution)

- **Soft** deadline violation: loss of performance but does not harm the system stability and safety
  - “The Telecommand TC(154,3) shall switch on the gyro within 10 seconds.
    Note: An additional delay of up to 5 seconds is tolerated.”

  → **soft deadline can be missed without seriously compromising the system**
    - some minor things might be effected
      (user interaction slow, limited data rate …)
  - In general these requirements are also relaxed
    (large timing window for execution)



PRECONDITION: REQUIREMENTS

QUICK OVERVIEW

Precondition: **Timing Requirements must be identified and characterized !**

Timing requirements are extracted from Device manuals, user expectation, engaged algorithms

Decomposition of System according to temporal needs

- Periodic
- Aperiodic
- Hard
- Soft

Temporal Characteristics		
	Aperiodic	Periodic
Hard	TM transmission TC reception Memory handling (e.g. EEPROM) UART handling	TM HK acquisition Monitoring of HK data AOC execution and related s/s commanding Thermal loop Physical interface handling (MIL-STD-1553B, CAN, SpaceWire, SpaceFiber)
Soft	TM generation TC execution Maintenance functionality	TM HK packet generation Monitoring of HK data

Selected Software Architecture needs to support these needs.

Use the appropriate Real-time Software Architecture to realize the temporal needs.

Combination of different architectures is feasible and common.

PRECONDITION: SOFTWARE ARCHITECTURE

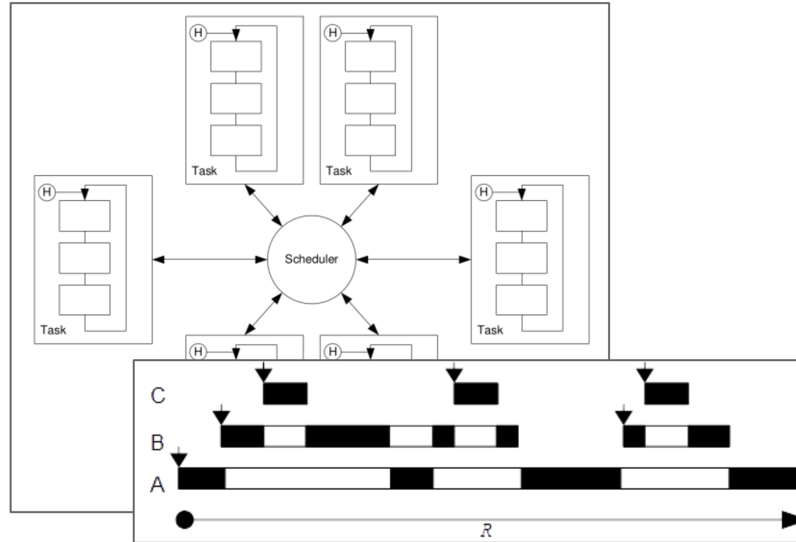
QUICK OVERVIEW

Precondition: **Software Architecture must be analyzable !**

Fixed Time Slice

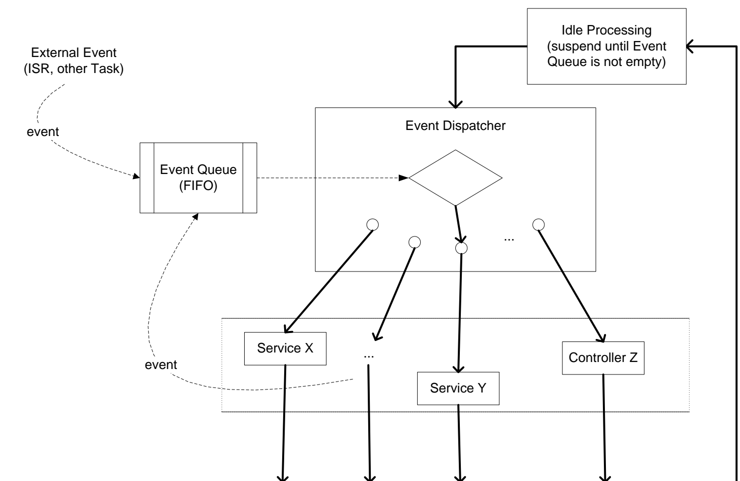
Inflexible, hard to maintain, lack of compositionality, used in hypervisors etc.

Preemptive Multitasking (Real-time Operating System)



Flexible set of independent tasks, scheduling policy, preemption, blocking, shared resource with arbitration protocol used for **hard (+soft) real-time**

Event-Driven Cooperative Scheduling (Asynchronous Message Passing, XF)



Flexible composition of independent loosely coupled components, single task run-to-completion, cooperative sched., used for **soft real-time**

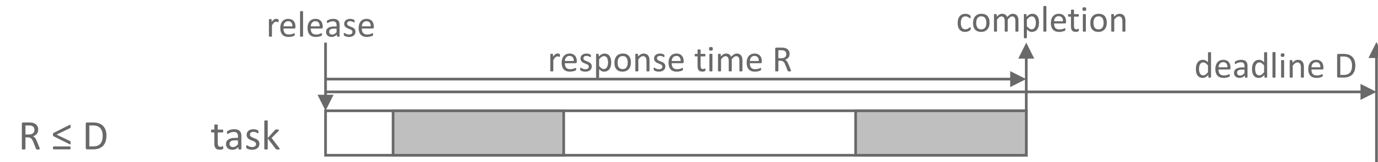
ESSENTIAL STEPS

QUICK OVERVIEW

Analysis comprises 2 independent parts

- **Formally proof all hard real-time constraints** to hold under all (incl. worst case) conditions
  - Established methodology for preemptive multitasking RTOS
  - Mathematical model that allows formal reasoning for each timing constraint
  - Response Time Analysis

R ≤ D

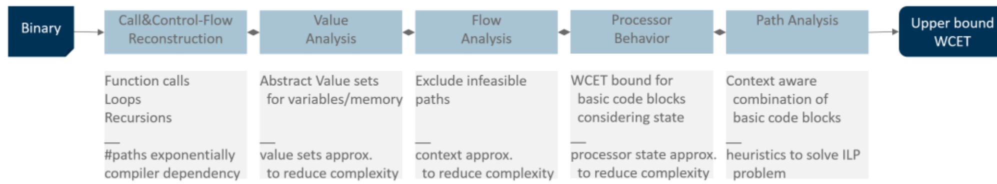


- **Argue all soft real-time constraints** to be met
  - Systematic Reasoning depends on the software architecture
  - A formal proof aiming at a 100% confidence in general is infeasible as it is
    - prohibitively expensive to carry out and
    - very likely to require the entire system to be highly inefficient
  - CPU Load Analysis based on measurements and statistics
  - Long-term system observation and monitoring

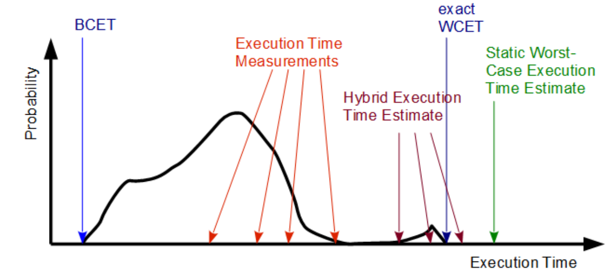
METHODS: RESPONSE TIME ANALYSIS

VERIFY HARD REAL-TIME CONSTRAINTS

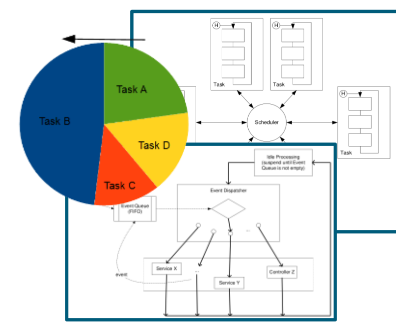
1) Timing analysis of code
   - sequential code snippet
   - Safe upper bound of WCET (Worst Case Execution Time)
   - (Static) Code Analysis
     → all execution paths & all system/memory/cache states

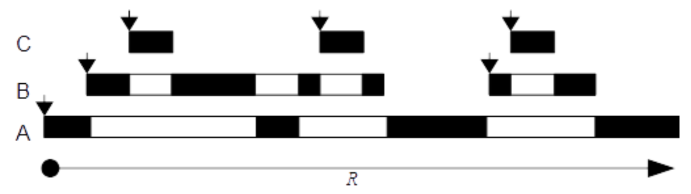



```
uint32_t  execTime_myFunction(uint32_t l)
{
  Osal_TaskTimeConsumption __ttc;
  Osal_StartTaskTimeMeasurement(&__ttc);

  uint32_t x = 0;
  if ( l > 1000 )
  {
    l = 1000;
  }
  for ( uint16_t i = 0 ; i < l; i++ )
  {
    x = x + i;
  }
  uint32_t duration = Osal_StopTaskTimeMeasurement(&__ttc);
  if ( duration > execTime_etm.max )
  {
    execTime_etm.max = duration;
  }
  if ( duration < execTime_etm.min )
  {
    execTime_etm.min = duration;
  }
  execTime_etm.lst = duration;

  return x;
}
```



2) Schedulability calculation
   - Tasking Model (e.g. preemptive multitasking, RTOS)
   - Mathematical Theory
   - WCET serves as input data for calculating Response Time





$$R_i = CS + B_i + C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot (C_j + 2CS) + CI(R_i)$$

$$CI(\Delta t) = \sum_{r \in ISR} \left\lceil \frac{\Delta t}{T_r} \right\rceil \cdot (C_r + 2CIS)$$

3) Check with requirements

Req. AOCS-210:
Execution of the AOCS step function and its pre- and postprocessing shall be completed within 200ms after the 1Hz activation pulse.

$$\forall i \in \tau : R_i \leq D_i$$

TOOLS: RESPONSE TIME ANALYSIS

VERIFY HARD REAL-TIME CONSTRAINTS

WCET - Tools:
- AbsInt a³
- Rapita Systems
- Bound – T
- …

Response-Time Tools:
- OHB Excel, DSL
- MAST, Cheddar
- … commercial …

Determined WCET

```
@staticWCET
exported uint32 myFunction2(uint32 l) {
  uint32 x = 0;
  if (l > 1000) {
    l = 1000;
  } if
  for (uint32 i = 0;i < l; i++ ) {
    x = x + i;
  } for
  return x;
} myFunction2 (function)
```

```
#compiler
compiler "sparc-gcc" ;

#clock rate
clock exactly 65.536 MHz;

#analysis precision
interproc flexible, max-length

#includes
include "../../../ResponseTimeAnalyses/Annotations/DHSW.ais";

# section properties
area section ".text" is read-only;
area "hkAccessTable" contains data and is read-only;

# memory accesses
global accesses restrict to 0x40000000 .. 0x5fffffff;

# function specific annotations
instruction "HandleThermostatAlert@static@TcsController.c" is entered
area "itsHkRepositoryAccessService" + 0x024 bytes .. "itsHkRepositoryA
loop "SwThermostat_SetMode" + 1 loop default-unroll = 6;
loop "SwThermostat_HasActiveSets" + 1 loop default-unroll = 6;
flow "EventReportingServicePUS05_SendErrorEvent" <= 1;
loop "GetMinMaxFloat" + 1 loop max 7;
snippet routine "log" is infeasible;
```

Timing Requirements

Req. AOCS-210:
Execution of the AOCS step function and its pre- and postprocessing shall be completed within 200ms after the 1Hz activation pulse.

Tasking Model

```
Tasking Model : SCSW
------------------------------------------------------------------------
Analysis:
RTA1 : Response Time Analysis (Immediate Ceiling Priority Protocol)
          Context Switch Penalty 0.0001 ms

------------------------------------------------------------------------
// Protect shared memory from concurrent access.
Monitor DataReportingServicePUS03  -> ceiling Priority: 60 (AOCS)

// External cyclic stimulus from timer
Cyclic Stimulus periodic_10Hz T= 100 ms -> 10.0 Hz

// ISR is triggered by the timer and divides frequency for releasing periodic tasks
ISR Periodicals Stimulus: periodic_10Hz
                Offset  = 0 ms
                WCET    = 0.158 ms
                Releases Job:
                PeriodicLOOP with period of 10 counts and an offset of 0 counts
                PeriodicAOCS with period of 5 counts and an offset of 0 counts

// Periodic AOCS step function
Task AOCS Priority: 60 -> 2.0 Hz

// Execute AOCS step function
Job PeriodicAOCS Task     : AOCS                                  -> 2.0 Hz
                 Priority = 60
                 WCET     = 98.68 ms
                 Monitors used:
                 DataReportingServicePUS03 : wcet= 5.322 ms
                 Deadlines:
                 internal : WCET of 62.245 ms until deadline of 70 ms
                 end : WCET of 98.68 ms until deadline of 500 ms

// Main loop of the software for reporting and checks
Task LOOP Priority: 50 -> 1.0 Hz

// Execute reporting and checks
Job PeriodicLOOP Task     : LOOP                                  -> 1.0 Hz
                 Priority = 50
                 WCET     = 192.0 ms
                 Monitors used:
                 TmPacketPool : wcet= 0.016 ms
                 Deadlines:
                 end : WCET of 192.0 ms until deadline of 1000 ms
```


Calculation, Visualization and Documentation

	Task ID τ_i	Description (Class Name)	min Period [s] T_i	WCET C_i	Blocking Term B_i	Deadline D_i	Priority P_i	RTEMS Priority	CRP_i
Real-Time Tasks	ICB_1	ICBController_1 (per)	0,1	XXXXXXX	0,000045	0,00427246	247	8	0,000006025
	ICB_2	ICBController_2 (per)	0,1	XXXXXXX	0,000045	0,00427246	247	8	0,000006025
	ICB_3	ICBController_3 (per)	0,1	XXXXXXX	0,000045	0,00427246	247	8	0,000006025
	ICB_4	ICBController_4 (per)	0,1	XXXXXXX	0,000045	0,02929688	247	8	0,000006025
	ICB_5	ICBController_5 (pre)	0,1	XXXXXXX	0,000045	0,01	247	8	0,000006025
	ICB_6	ICBController_6 (aper)	0,1	XXXXXXX	0,000045	0,01	247	8	0,000006025
	MPL	MilbusController	0,1	XXXXXXX	0,000046	0,085	246	9	0,000011211
	MPF	MilbusController	0,1	XXXXXXX	0,000046	0,085	245	10	0,000005872
	SCHD	OnboardScheduler	0,1	XXXXXXX	0,000977	0,1	230	25	0,000006025
	TC	PacketReceptionService	XXXXXXX	XXXXXXX	0,000977	XXXXXXX	225	30	0,000005796
	OBT	OnBoardTime	1	XXXXXXX	0,002672	0,25	215	40	0,000006025
	LOOP	Loop	1	XXXXXXX	0,001115	1	205	50	0,000016625
	LOOP_1	Loop_1	1	XXXXXXX	0,001115	0,2	205	50	0,000016625

THEORY AND PRACTICE

TECHNICAL CONSIDERATIONS

- Classification: which requirement is a soft and which is a hard timing constraint?
  - For every hard timing constraint there must be a good and valid reason (not just a feeling and sense of urgency)
  - Just because something is important to happen (cf. FDIR) it's not necessarily to be classified "hard"
  - If a function is determined to be subject to hard real-time but the envisaged implementation "breaks" the architecture, maybe the implementation needs to be reconsidered.
    (cf. "this TC is generated onboard as part of the control loop, so its execution needs to start and complete execution within 100ms")
  - Has impact on how to implement the respective functionality in order to be able to verify the requirements

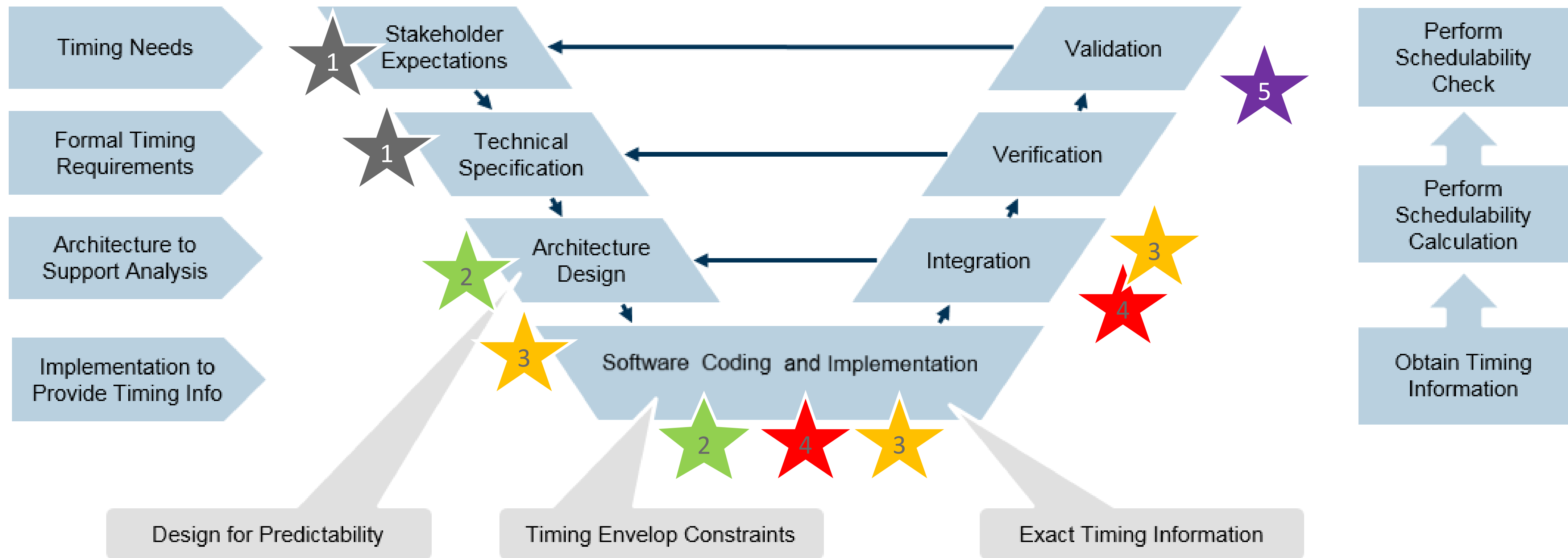
- SA needs to be considered very early in the design and development process
  - SA is tightly coupled with software architecture
  - SA starts prior to the first line of code is written
  - It is essential for all developers to understand the real-time behavior of the software
  - Software modularity is not limited to function, but needs to consider timing and analyzability as well

THEORY AND PRACTICE

PROCESS SPECIFIC CONSIDERATIONS

1 Requirements
2 Design and Implementation
3 SA Calc
4 WCET Analysis
5 Measurements

V-Cycle

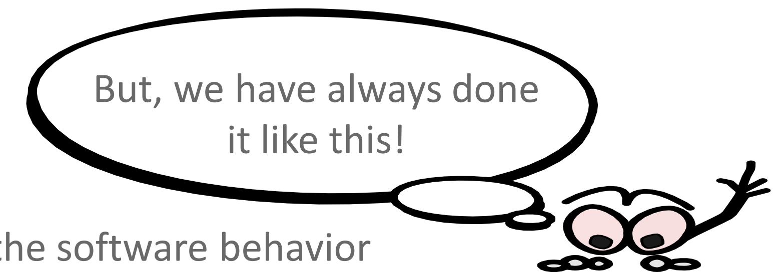


THEORY AND PRACTICE

LESSONS LEARNED

- Timing requirements are essential but often are missing or of insufficient quality
  - But they are needed early to make the correct design decisions and to be able to verify/validate/test your system in the end
  - Deficiencies are in requirement documents, user manual and ICDs (and in how they are processed)
  - Timing requirements need to be as loose and relaxed as possible and not driven by a "feeling of urgency"
  - All timing requirements must have a technical justification

- Some common design/implementation pattern are in conflict with assumptions or efficient application of the theory !
  - Design for analyzability – know the assumptions and show that they are obeyed
  - Great source for discussions !
  - Ex.: misconceived software modularity **vs.** small set of tasks and shared resources
  - Ex.: self-suspending functions in "user-friendly" API with hidden ISR **vs.** full knowledge of the software behavior
  - Ex.: suspensions to implement temporal sequences, hardware access, API call **vs.** each task has a single point of suspension
  - Ex.: Worst Case Time (WCET) **vs.** Average Time (AET) … selecting algorithms developers tend to go for the average execution time
  - Advice: Keep it simple!
    simple code is easy to be analyzed by static code analyzers (avoid dynamic structures and data dependent control flow)
    and: less error-prone



BENEFITS

EXPECTATIONS: PROOF OF TIMING CONSTRAINTS AND REQUIREMENTS

- A Schedulability Analysis can proof certain timing properties, but
  - The proof is carried out on a mathematical model
  - It is assumed that the mathematical model resembles the implementation (but in general this can’t be shown)

- SA can not formally proof every timing requirement .. and this is not necessary
  - “only” hard timing requirements are formally proven,
  - Soft timing requirements are argued to be met based on measurements and estimates

- SA is only one brick contributing to System/Software correctness
  - Ongoing monitoring (of the timing behavior) still is essential
  - Comprehensive tests still are essential

BENEFITS

INCIDENTAL CATCH

- Ideally timing analysis is performed by a different team / personnel
  - Providing an extra pair of eye for thorough code review and understanding
- Orthogonal view into the software
  - Usually the function is the primary focus of developer and reviewer, now the time perspective is added
- Early considerations wrt. a clean and structured architecture
  - Shared resources and their accessors (+monitors, semaphores etc) are identified by analysis (must adhere the architecture, code review by analysis)
- Static code analysis identifies all feasible execution paths, even those not intended or tested
  - Identify unintended and untested execution paths that may lead to unexpected behavior
- Static code analysis identifies the longest execution path
  - Hot-spot identification early in the process → Optimization of algorithms, ensure scalability
  - Ex.: a data-pool is populated with a limited set of data during development and test such that search inefficiencies are identified late
- Static stack analysis is a low-hanging fruit
  - Worst case stack usage can be calculated by static timing analysis tools as it engages the same technology
- Keep it simple and stupid (KISS)
  - Architecture designed for analysis typically is simple (repetitive design pattern)
  - Simple code is easier to be analyzed by code analyzers (less manual annotations required)
  - Simple code usually is less error-prone code, (Ex.: avoid dynamic structures and data dependent control flow)

OUTLOOK

WHAT ABOUT OTHER ARCHITECTURES?

- More advanced processing architectures (Multi-Core processors, Hypervisors, Distributed systems)
  - Proofing timing/access constraints across multiple processors yields explosion in complexity

    So we'd better avoid that …
  - … and follow the often suggested approach of separating hard real-time sensitive functionality into dedicated Cores or CPUs
  - Ex.: in SMP RTOS dedicated tasks are bound to specific CPU cores (BMP – bound multi-processing), which are analyzable independently

- Realtime bus systems (MIL-STD-1553B, CAN, TTEthernet, SpaceWire, SpaceFibre, RS485 …)
  - Usually restriction to a real-time capable protocol subset: periodic queries "send-list" approach
  - Often the scheduling theory is applicable similarly
    - Ex.: real-time communication via multi-master CAN with proven upper bound for the Response/Transmission Time

SUMMARY

INTRODUCTION & QUICK OVERVIEW

- SA is only one brick contributing to System/Software correctness
  - Ongoing monitoring (of the timing behavior) still is essential
  - Comprehensive tests still are essential
- SA argues about timing and
  - can formally proof “only” hard timing requirements and
  - may provide systematic means to argue that soft timing requirements are met
- SA ranges the entire lifecycle and is specifically important in early phases (requirements, architecture definition ..)
- SA provides incidental benefits
  - insight in the software from a different (non-functional) perspective
  - additional confidence that the software/system works as expected

WHY DO WE HAVE TO PREPARE A SOFTWARE SCHEDULABILITY ANALYSIS?

ANDREAS WORTMANN, SOFTWARE PRODUCT ASSURANCE WORKSHOP 2023, ESAC, SPAIN, 25-28.09.2023

THANK YOU!

OHB SE
Manfred-Fuchs-Platz 2-4
28359 Bremen
Germany

Phone: +49 421 2020 8
Fax: +49 421 2020 700
Email: info@ohb.de
Web: www.ohb.de