Topics for state final examination

The goal of this document is to specify the topics for the state final examinat in a way to make it clear (1) what knowledge is expected in particular topics, (2) to what extent this knowledge will be required, and (3) what courses cover it.

This version is prepared for the final examination taking place in June 2022.

Software systems: Dependable systems

This focus tests knowledge and skills related to the design of dependable software systems, examined in the extent of the following profiling courses.

Detailed requirements

NSWI132 Program Analysis and Code Verification

- Explain various methods for verification and analysis of behavior of programs and software systems (model checking, static analysis, deductive methods, their combinations)
- Describe the advantages and disadvantages of methods of verification and analysis of program behavior, especially with respect to testing
- Important aspects: accuracy, scalability, performance (speed)
- Methods for error detection, especially in multithreaded programs
  - Describe methods based on state-space traversal: bounding the number of preemptions, selected heuristics
  - Describe methods based on dynamic program analysis (lock-set analysis, happens-before relation)
  - Define the state of the program and its representation, state space (its components)
  - Define the concept of correctness of multithreaded programs (data-race freedom, serializability, linearizability, ABA problem)
  - Describe relaxed / weak memory models from the perspective of correct program behavior and troubleshooting
- Symbolic execution (explain the basic principle, describe important practical applications)
- Abstraction of program behavior
  - Give examples of some abstractions, explain the concept of predicate abstraction
  - Explain the concepts of over-approximation and under-approximation in the context of program behavior abstraction
  - Explain the implications of using over / under-approximation with respect to property verification and error finding
- Deductive methods of program behavior verification
  - Describe the method of program behavior representation and its properties (invariants, contracts, preconditions, and postconditions) in the form of a logical formula
  - Explain the term “verification condition (VC)” and the basic process of generating a VC
- Static program analysis
  - Explain basic concepts (control-flow graph, how information about the analyzed program is represented, representation of semantics of statements / instructions and their influence on analysis results, worklist algorithm)
  - Basic classification of static analyzes (direction, scope, level of approximation (“may” versus “must”), flow-sensitivity, context-sensitivity)

NTIN043 Formal Foundations of Software Engineering

- Compare individual methods of formal specification of requirements, behavior, and architecture of software systems with respect to expressive power, methodology, and appropriate application domain

  - Methods: algebraic specification languages, model-oriented specification languages (Z, VDM, Alloy), UML and OCL, Petri nets
- General characteristics of formal methods (purpose, advantages, disadvantages) and the way of their use
- Algebraic specification techniques
  - Describe the basic principles and sub-elements of the specification (sort, carrier set, operations and functions, equations)
  - Define the concept of semantics of algebraic specification (at what level these specifications describe the behavior of the software, what are the expressable properties)
  - Explain related terms: executable specifications, rewriting system (including important and necessary features such as "confluence", "termination" and "canonical form")
- Model-oriented specifications
  - Explain the basic principles: how to create specifications, important elements, and features
  - Describe the methodology of creating the specification
  - Describe iterative refinement in the context of model-oriented languages
- UML and OCL
  - Describe the main types of UML diagrams (class, sequence, activity, use cases)
  - Describe basic OCL expressions types (expressions, preconditions, postconditions, invariants)
- Petri nets
  - Describe basic concepts (place, transition, arc, token, configuration / marking, firing of enabled transitions)
  - Create an example model for a simple system (for example, mutual exclusion of two threads)
- Application of formal methods in software development practice
  - Explain the terms Model-driven engineering, model-based testing, generating code from specification (iterative refinement)

NSWI164 Model-Driven Development

- Explain (with examples) what model-driven development is.
- Explain what a model is and give an example. Explain what concrete syntax is and give an example (text, graphic)
- Explain what a meta-model is and what it is used for. Be able to draw a meta-model (using UML class diagrams) for a specific task (e.g., modeling of classes and interfaces of a chosen OO language)
- Explain and give an example of a meta-modeling hierarchy (model, meta-model, meta-meta-model,…)
- Explain what a domain-specific language is and give an example. Show how DSL can be specified (e.g. using Xtext)
- Explain what the transformation of models is and give an example in a chosen language
- Explain what code generation is and how it can be implemented using dedicated tools

NSWI101 System Behaviour Models and Verification

- Formalisms for the representation of computer systems models and their properties
  - Define the terms Kripke structure, LTS, State-transition system, give examples of these structures
  - Name and describe some languages for the specification of these structures, describe the selected approach together with practical aspects (in which cases it is appropriate to use the given formalism) - Promela, Timed Automata, Parallel-assignment language (SMV)
- Temporal logic for property specification
  - Define syntax and semantics of CTL, LTL, TimedCTL, ICTL, …
  - Compare the logical strength of LTL and CTL, examples proving the incomparability of these logics
- Algorithms for model checking
  - Describe the algorithm for model checking LTL, CTL, TimedCTL (Timed Automata), describe the selected algorithm
  - Show the asymptomatic complexity of these algorithms
- Symbolic model checking
  - Define OBDD and describe this modeling approach; describe the complexity of logical operations over OBDD
  - Describe an algorithm for symbolic CTL checking model

- Specific approaches to verification and model checking
  - Explain the terms Infinite and Bounded model checking, abstraction, probabilistic model checking – describe particular methods and the motivation for their existence
  - Provide application examples along with examples of properties that can be verified using these logics

NSWE001 Embedded and real-time systems

- Characteristics of embedded and real-time systems
  - Explain what is specific to the embedded systems
  - Explain what is specific to the real-time systems
  - Characterize different types of real-time systems using utility functions
- Tasks and scheduling
  - Explain what a real-time task is and divide them by types (time / event triggered, periodic / aperiodic)
  - Name and explain task parameters (arrival time, computation time, relative / absolute deadline, lateness, jitter,…)
  - Define the planning problem (including constraints – blocking, precedent)
- Non-periodic planning
  - Explain and demonstrate scheduling algorithms for synchronous / asynchronous activation, preemptive / non-preemptive scheduling, with / without precedent constraints
  - Always list and explain algorithm assumptions
- Periodic planning
  - Explain and demonstrate algorithms for offline / online planning, for online planning further with static / dynamic priorities
  - Explain how schedulability guarantee is done for individual algorithms - search for offline algorithm, Liu-Leyland bound and response time analysis for RM and DM, Liu-Leyland bound and processor demand analysis for EDF
  - Explain what optimality means for RM / DM and what it means for EDF
  - Explain how response time analysis is calculated with system jitter, blocking and overhead
- Scheduling with blocking
  - Explain what blocking is and how it arises
  - Explain how response time analysis is calculated when there is a blocking system
  - Explain and demonstrate (draw a schedule) priorities inversion problem
  - Explain and demonstrate PIP and PCP algorithms. For both algorithms, say what problems they solve, what they do not solve, and what guarantees they provide.
- Design
  - Explain system modes
  - Describe / demonstrate how real-time task requirements are obtained from a specification
- Controllers
  - Explain what PID is, its components, and the practical importance of each component for management
- Communication
  - Difference between time-triggered and event-triggered communication
  - Describe the operation of CAN and the provision of real-time guarantees
  - Explain and demonstrate how schedulability analysis is done in a distributed system with CAN
- Servers
  - Explain what servers in real-time systems are good for
  - Difference between static and dynamic priority servers
  - Explain and demonstrate algorithms for static and dynamic priority servers - polling server, deferrable server, priority exchange server, sporadic server, dynamic priority exchange server, dynamic sporadic server, total bandwidth server, constant bandwidth server, IPE

Covering courses

- NSWI132 Program Analysis and Code Verification
- NTIN043 Formal foundations of software engineering
- NSWI164 Model-driven Development
- NSWI101 System Behaviour Models and Verification
- NSWE001 Embedded and real-time systems

Software systems: System programming

This focus tests knowledge and skills related to system programming and internal functions of software systems, examined in the extent of profiling courses.

Detailed requirements

NPRG014 Concepts of modern programming languages

- Explain and illustrate the difference between class-based languages with static typing, dynamic typing and prototyping languages (i.e., languages without classes)
- Class-based languages – explain and illustrate the following in a chosen language:
  - Concepts of classes, inheritance, interfaces, visibility
  - Static methods / fields vs. singletons (object vs. class in Scala)
  - Closures in the type hierarchy (Any, Null, Nothing). Show where they are suitable.
  - Type parameters (in the class and method declarations) and type variables (i.e., within classes). Show how they are used together with lower and upper limits.
  - Covarians, contravarians and invariants. Show for each one where it is suitable.
  - Concepts of traits, mixins and linearization
  - Extension methods and implicit conversions. What are they good for?
  - Path-dependent types
  - Default parameters (typeclasses)
  - Meta-classes
  - Concepts of functional programming in imperative OO languages (currying, partial application, functors, monoids, tail recursion)
  - Language features that allow for internal DSL
  - Case-classes, pattern matching, extractors (apply and unapply methods)
  - Comprehensions
  - Static meta-programming (macros / compiler plugins)
- Explain and show an example of high-level abstraction for concurrency - dataflow, promises, actors, fork / join, geometric decompositions of collections, agents
- Prototyping Languages - Explain and illustrate the following in a chosen language
  - Concept of an object without a class, slots, parent slots
  - Implementation of language concepts with classes in a prototype language (i.e., draw a hierarchy of objects) - inheritance, static members / fields, field instances

NSWI080 Middleware

- Remote Function Call (RPC)
  - Demonstrate the mechanism of calling remote custom functions on code examples (e.g. RMI, gRPC, Thrift…)
  - Explain the motivation for the existence of interface definition languages
  - Explain and demonstrate the function of the thread pool mechanism on code examples
  - Explain the function of passing references in object variants of remote function calls
- Messaging
  - Explain the motivation for the existence of languages for the definition of data formats
  - Discuss variants of technical parameters of data representation (binary vs text, no vs with schema, no vs with type information)
  - Demonstrate custom-subscribe mechanism for code choices on code examples (e.g. ActiveMQ, RabbitMQ, 0MQ…)
  - Demonstrate messaging communication mechanism on code examples (Kafka)
  - Demonstrate collective communication mechanism (MPI) on code examples
- REST
  - Explain the motivation for the specific rules of the REST architectural style (representation of resources, manipulation with resources, absence of status)
  - Demonstrate interface definition (OpenAPI) on code examples
- SOA and microservices
  - Explain motivation

NSWI161 Advanced Operating Systems

- Operating system architecture
  - Explain the role of the operating system and name and explain the role of its basic components
  - Explain the role of privileged processor mode and give examples of privileged operations
  - Microkernels
  - Explain the motivation of microkernel and uniquernel architectures
  - Virtualization
  - Explain the motivation of operating system virtualization
  - Explain the difference between a virtual machine and a container
  - Demonstrate the use of namespace objects on a running container to achieve isolation (lsns)
- Memory management
  - Heap
  - Explain and demonstrate memory allocation issues on multiprocessor computers using code examples
  - Illustrate solutions to these problems on a specific modern allocator of your choice (e.g. jemalloc, tcmalloc, mimalloc…)
  - Explain the motivation of the slab allocator and explain its use on a given code example
  - Paging
  - Explain the address translation procedure in virtual machines on a given diagram on processors with hardware virtualization support (Intel EPT, AMD NPT)
- File systems
  - Explain mirroring and striping mechanisms and the role of parity in RAID configurations and their impact on performance and reliability
  - Use standard tools to set the specified RAID configuration (mdraid, btrfs)
  - Use standard tools to create a new volume and volume image (btrfs)
  - Explain the use of the copy-on-write concept when creating volume images
- Network management
- Service management
  - Explain the role of service management in the operating system
  - Use standard tools to identify and control running services (systemctl)
  - Create a definition of the service and its dependencies (systemd service unit file) for the given program
- Operating system management and development
  - Use standard tools to monitor events within the operating system (perf)
  - Create short programs (5-10 lines) to monitor operating system operation (bpftrace)

NPRG058 Advanced Programming in Parallel Environment

- Multicore-CPU and NUMA systems
  - Briefly describe the architecture of the CPU and memory (i.e. the classic shared memory system) and critically evaluate them in terms of the efficiency of parallel algorithms (CPU communication, shared cache, MESI protocol, NUMA factor)
  - Show examples of the consequences of basic synchronization procedures on the efficiency of algorithms (context switch overhead, spin-lock starvation, false sharing, and cache-line ping-pong,…)
  - Design the concept of lock-free implementation of a simple data structure (linked list, hash table, growing vector,…).
  - Design a parallelization procedure of a simple algorithm (matrix multiplication, k-means,…) with respect to cache and NUMA factor.
- GPU
  - On a suitable example (algorithm implementation), describe the principle of using the data-parallel paradigm on the GPU with reference to the advantages and disadvantages of lockstep execution
  - On the example of data-aggregation algorithm (e.g. histogram calculation), show suitable optimization techniques for elimination of synchronization collisions (privatization, use of shared memory, parallel reduction)
  - Show a good example of using warp-cooperative or warp-synchronization instructions for code optimization.

    - Design an algorithm (and describe the most important technical means) to optimize data transfers between the guest and the GPU; compare this algorithm with alternatives such as memory mapping or unified memory
    - Show the possible advantages of dynamic parallelism on a suitable algorithm (e.g. Mandelbrot); explain the necessary technical resources that the GPU must support (context block of the fiber block)
- MPI

NSWI035 Principles of Distributed Systems

- Synchronization algorithms
    - Explain the concepts of physical and logical clocks, describe the coordinator's choice algorithm and the concepts of causal dependence, describe delivery protocols
- Distributed consensus
    - Define the concept of the global state, describe its detection and its applications, describe algorithms for reaching a distributed consensus, Paxos, RAFT
- Distributed shared memory
    - Describe consistency models, describe and explain distributed paging
- Resource and process management
    - Define and explain distributed deadlock detection algorithms

Covering courses

- NPRG014 Concepts of modern programming languages
- NSWI080 Middleware
- NSWI161 Advanced Operating Systems
- NPRG058 Advanced Programming in Parallel Environment
- NSWI035 Principles of Distributed Systems

Software systems: High Performance Computing

This focus tests knowledge and skills related to software design systems with high computing power, examined in the extent of the following profiling courses.

Detailed requirements:

NSWI109 Compiler Design

- Describe the basic concepts of compilers and representations of the translated program
  - Control flow, data flow, dependencies, aliases, scopes
  - Sequential and non-sequential intermediate codes, SSA form
- Describe the compiler architecture, the order of important compilation steps
- Register allocation
  - Describe and explain graph coloring algorithm, spill-code solution
- Scheduling
  - Describe processor model, list scheduling, branch-and-bound approach
  - Describe Unroll-and-compact algorithms, modulo scheduling, trace scheduling
- Parallelization and vectorization by the compiler
  - Describe polyhedral compilation
- Analysis of indicators and aliases
  - Describe the terms Andersen's and Steensgaard's method

NPRG058 Advanced Programming in Parallel Environment

- Multicore-CPU and NUMA systems
  - Briefly describe the architecture of the CPU and memory (i.e. the classic shared memory system) and critically evaluate them in terms of the efficiency of parallel algorithms (CPU communication, shared cache, MESI protocol, NUMA factor)
  - Show examples of the consequences of basic synchronization procedures on the efficiency of algorithms (context switch overhead, spin-lock starvation, false sharing, and cache-line ping-pong,…)
  - Design the concept of lock-free implementation of a simple data structure (linked list, hash table, growing vector,…).
  - Design a parallelization procedure of a simple algorithm (matrix multiplication, k-means,…) with respect to cache and NUMA factor.
- GPU
  - On a suitable example (algorithm implementation), describe the principle of using the data-parallel paradigm on the GPU with reference to the advantages and disadvantages of lockstep execution
  - On the example of data-aggregation algorithm (e.g. histogram calculation), show suitable optimization techniques for elimination of synchronization collisions (privatization, use of shared memory, parallel reduction)
  - Show a good example of using warp-cooperative or warp-synchronization instructions for code optimization.
  - Design an algorithm (and describe the most important technical means) to optimize data transfers between the guest and the GPU; compare this algorithm with alternatives such as memory mapping or unified memory
  - Show the possible advantages of dynamic parallelism on a suitable algorithm (e.g. Mandelbrot); explain the necessary technical resources that the GPU must support (context block of the fiber block)
- MPI

NSWI035 Principles of Distributed Systems

- Interprocess communication
  - Define and describe the concepts of reliability, RPC, group communication

- Synchronization algorithms
  - Explain the concepts of physical and logical clocks, describe the coordinator's choice algorithm and the concepts of causal dependence, describe delivery protocols
- Distributed consensus
  - Define the concept of the global state, describe its detection and its applications, describe algorithms for reaching a distributed consensus, Paxos, RAFT
- Distributed shared memory
  - Describe consistency models, describe and explain distributed paging
- Resource and process management
  - Define and explain distributed deadlock detection algorithms

NSWI150 Virtualization and cloud computing

- Explain the motivation for virtualization, describe the taxonomy of virtualization technologies.
- Describe the concepts of hardware-assisted virtualization, paravirtualization, emulation. Describe the communication between virtual machines.
- Describe virtualization support on today's hardware architectures and operating systems.
- Explain container technologies, Linux namespaces, cgroups, docker, different philosophy of using containers.
- Describe various hardware solutions for data centers.
- Describe cluster, load balancing, high availability, and fault tolerance.
- Describe cloud technologies, components, and services. Explain the terms IaaS, PaaS, and SaaS.
- Describe execution models – virtual machines, cloud services, containers, microservices, serverless computing, mobile services, and high-performance computing. Describe cloud management, orchestration, and monitoring.
- Describe methods of data management, storage and processing, no-sql database in the cloud, map / reduce, and related technologies.
- Describe communication and synchronization in cloud platforms, CDN, caching.

NSWI131 Performance Evaluation of Computer Systems

- Measuring tools
  - Explain the problems associated with overhead and perturbation of measurements and illustrate them for time measurement
  - Explain the basic functions of hardware performance event counters (events, counters, PEBS, latency sampling)
  - Use common tools (e.g. perf, PAPI, VTune…) to obtain information from hardware performance event counters
  - Describe the basic functions of common mechanisms for monitoring other metrics (e.g. SNMP, JMX…)
  - Describe the mechanism of profiling the program, discuss its features and limitations, use basic profiling tools (e.g. perf) to identify performance-sensitive program locations
  - Describe the mechanisms of instrumentation of basic forms of the program (source code, bytecode, binary code), discuss their limitations, use basic instrumentation tools (at least one selected from Coccinelle, AspectJ, DiSL, Pin, Valgrind, DynamoRio, bpftrace…) for program instrumentation
- Measuring techniques
  - Describe and use basic experimental design techniques (single factor, one factor at a time, n-fold validation)
- Performance metrics
  - Define common performance metrics (throughput, latency, jitter, availability, CPI / IPC, miss rate, MIPS, FLOPS, wall clock time, thread time) and discuss their properties (such as portability or comparability)
  - Explain the motivation for the construction of performance benchmarks and give examples of benchmarks together with their metrics (e.g. SPEC CPU, SPEC JBB, TPC-C…)
- Measurement processing
  - Define and describe the use of basic techniques for filtering remote observations (winsorization)
  - Define and describe the use of basic statistics (mean, median, quantiles, variance) on measured data
  - Define confidence intervals and use parametric methods and bootstrap to calculate the confidence interval of the average of measured data (no knowledge of the formulas of specific calculations is required, but the ability to find and use such a formula if necessary is expected)
  - Use statistical hypothesis testing to evaluate the measured data (knowledge of the formulas of specific calculations is not required, but the ability to find and use such formulas if necessary is expected)

  - Use common chart types (scatterplot, histogram, Q-Q, box-and-whiskers, violins) to visualize measurement results

Covering courses

- NSWI109 Compiler Design
- NPRG058 Advanced parallel programming
- NSWI035 Principles of Distributed Systems
- NSWI150 Virtualization and cloud computing
- NSWI131 Performance evaluation of computer systems