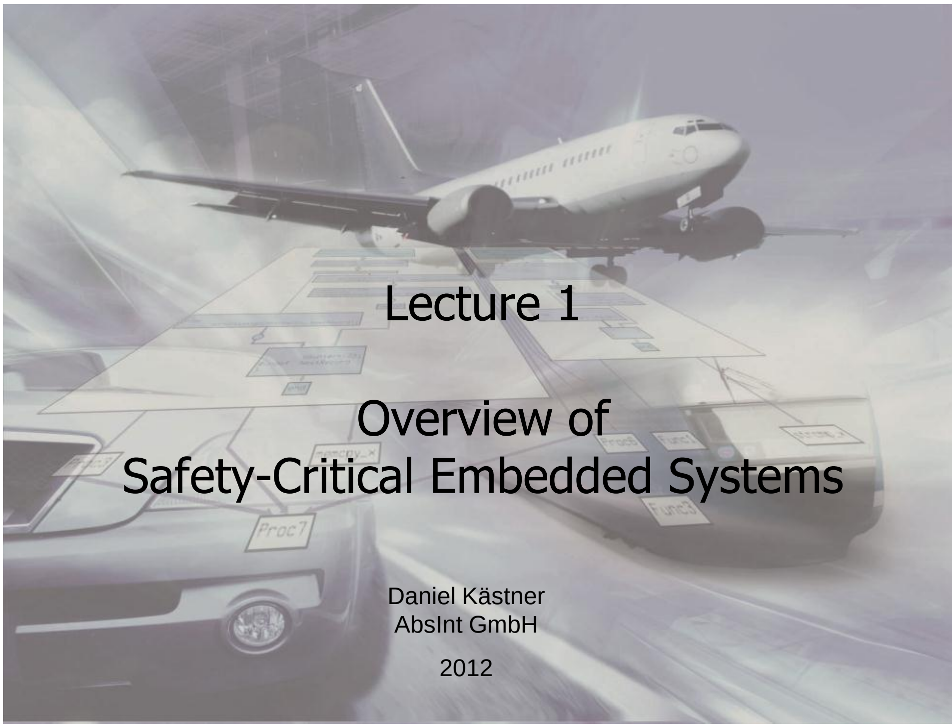


AbsInt Angewandte Informatik GmbH

- Advanced development tools for validation, verification, and certification of safety-critical software.
- Founded in February 1998 by six researchers of Saarland University, Germany, from the group of programming languages and compiler construction of Prof. R. Wilhelm.
- Privately held by the founders.
- Selected customers:

AIRBUS, SIEMENS, TOYOTA, MTU Aero Engines, EADS, BOSCH, Ford, Continental, ZF, TRW, Infineon, SAP, LIEBHERR, OHB, AREVA, EMBRAER, Baxter

- Bachelor/Masters theses, or Hiwi positions available.

Overview

- Daniel Kästner, Reinhard Wilhelm, Florian Martin, Jörg Herter, Sebastian Altmeyer.
- Advanced course (6CP): Fri 10-12, E1.3, HS003. 2 hours exercise. Goal: Working with industry tools for embedded systems development and understanding their theoretical background.
- Contents: Model-based code generation, static program analysis, task scheduling and schedulability analysis.
- Tools used:
  - SCADE: CASE tool for safety-critical embedded systems (avionics).
  - aiT: Static worst-case execution time analysis; StackAnalyzer: static worst-case stack usage analysis; Astrée: static runtime error analysis (avionics & automotive);
  - Symta/S: Task scheduling & schedulability analysis (automotive).
- Practical project with LEGO Mindstorms.

Organization

- Website: http://rw4.cs.uni-saarland.de/teaching/dses12/
- Mailing Lists:
  - contact all lecturers and tutors: dses12-team@gigasun.cs.uni-sb.de
  - Mailing list address: dses12@gigasun.cs.uni-sb.de
- **Important**: send email with name, matriculation number, and tutorial date to jherter@cdl.uni-saarland.de
- Exercises
  - No teams for theoretical exercises
  - First tutorial in week 43 (22.10.-26.10.)
  - Tutorial dates: tbd at end of today's lecture

Organization

- Written examination: 22.02.2013
  - At least 50% of total exercise points
  - Successful participation in project
  - Final grade composed from examination result and potential bonus points from project.
- Project phase:
  - Teams of up to 3 students
  - Start in week 44
  - Submission & Presentation: week 6 2013

Motivation

- Embedded systems have revolutionized everyday life and have become an integral part of it.
- Without embedded systems:
  - no energy supply
  - no transportation
  - scarce food supply
  - degraded medical service
  - no electronic communication
  - …
- Market, application range and complexity of embedded systems are growing and impose new challenges.

Motivation

- Excellence cluster at Saarland University:
  Multi-modal Computing and Interaction
- Goal: develop computing systems that can interact with humans in a natural way. They should be
  - able to process different kinds of information: speech, images, videos, graphics, ...
  - pervasive: be available anytime, anywhere
  - reactive: analyze their environment, react to speech, text, gestures.
- ➢ Embedded systems all over.
- Selected challenges: distributed systems, real-time processing, safety.

[Source: Press release of Saarland University Computer Science Dptmt]

Embedded Systems: Definition

- Embedded systems are
  - embedded in a physical environment and interact with it for measuring or controlling purposes.
  - Information processing systems embedded into a larger product; main reason for buying is not information processing.

- Characteristics of embedded systems:
  - complex interaction with environment
  - usually dedicated towards a certain application
  - typically reactive systems
  - high availability and reliability required
  - often safety-critical
  - often real-time processing required
  - often limited resource availability

Application Areas of Embedded Systems

- Avionics: Pilot information systems, braking & steering systems, cabin pressure and air conditioning control, anti-collision systems, fly-by-wire, UAVs (unmanned airborne vehicles), …
- Space: Autonomous vehicles, satellite control, …
- Automotive: engine control, airbag, air-conditioning, electronic brakes, active suspension, blind-angle alert systems, adaptive cruise control, lane assistant, steer-by-wire, brake-by-wire, drive-by-wire, …
- Consumer electronics: AV-R Receivers, CD-/DVD-/MP3-/bluray-players, washing machines, microwave ovens, PC peripherals,
- Infrastructure & automation: smart home, smart grid, roboters, …
- Telecommunications: network switches, cell phones / smartphones, fax & answering machines, IPTV, …
- Healthcare Technology: infusion pumps, defilibrators, diagnostic imaging (CT, MRI, …), pacemaker, artificial eye, …

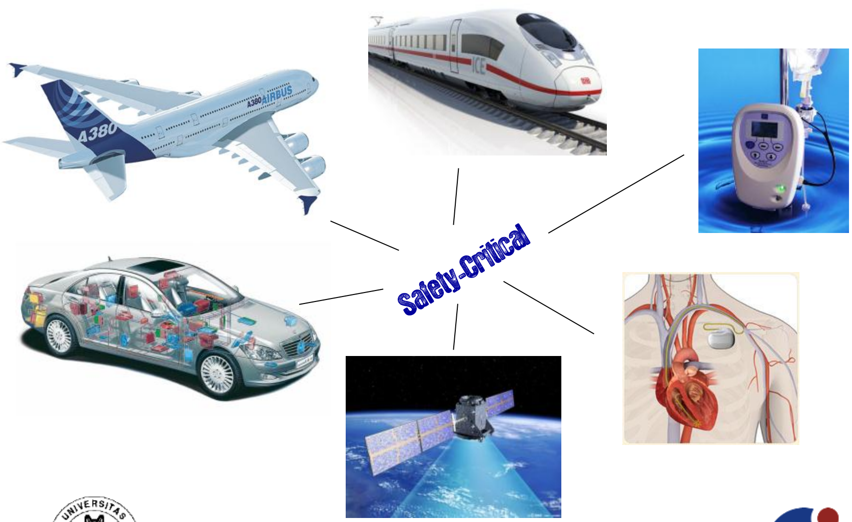
Embedded Systems Market

- Global market volume for embedded systems estimated between 68 and 138 billion EUR (2009).
- German market for embedded systems is third largest behind USA and Japan; German market volume in 2010 estimated at 19 billion EUR.
- Growth rates have been stable at ~8% in the last years, also future growth rates estimated at 7-10%.
- Perceived as one of the most important industries of the future.
- More than 40.000 jobs in German embedded systems suppliers in 2008, tendancy increasing. More than 250.000 jobs in embedded systems appliances (software development or integration of embedded systems).
- Sources:
  - *Nationale Roadmap Embedded Systems 2009*
  - *Bitkom-Study "Eingebettete Systeme – Ein strategisches Wachstumsfeld für Deutschland".*

Functional Safety

- Safety-critical systems: a malfunctioning of the system can cause significant damage and may endanger human beings.
- Functional safety: freedom from unacceptable risk of physical injury or of damage to the health of people either directly or indirectly (through damage to property or to the environment).
- Legal regulations require developing and verifying safety-critical systems with due diligence, according to the state of the art.
- Safety standards formalize the minimal processes and requirements for system development with due diligence. Non-compliance is indication for negligence in liability suits.

Safety-Critical Embedded Systems



Safety Standards

- Formulate requirements on system and software development and verification process
- Define minimal requirements for state of the art

- Aerospace: DO-178B/DO-178C (latest revision 2012)
- Automotive: ISO-26262 (latest revision 2011)
- General E&E systems: IEC-61508 (latest revision 2011)
- Railway: CENELEC EN-50128 (latest revision 2012)

- Formal methods and model-based development recognized and recommended – since about 2010 (!)

Demonstrating Functional Safety

- Demonstrating functional correctness
  - Compliance to specified functional requirements
  - ➢ Automated and/or model-based testing
  - ➢ Formal techniques:
    - Model checking
    - Theorem proving
- Satisfaction of non-functional requirements
  - Absence of runtime errors (division by zero, invalid pointer accesses, overflow and rounding errors, …)
  - Availability of sufficient resources
    - satisfying timing requirements (e.g. WCET, WCRT)
    - satisfying memory requirements (e.g. no stack overflow)
  - Testing inappropriate
  - ➢ Formal techniques:
    - Abstract Interpretation

Real-Time Systems

- In a real-time system, the correctness not only depends on the logical results but also on the timing of the applications.

- Distinction:
  - Hard real-time system: It is vital that the system satisfies the timing condition. Failure results in catastrophic consequences, e.g. the loss of lifes. Examples: flight control software, airbag control.

  - Soft real-time system: It is desirable that the system satisfies the timing conditions; otherwise the functioning of the system is negatively affected. Example: MP3-Player, telephone software.

Dependability of Embedded Systems

- High dependability requirements:
  - Reliability $R(t)$: probability of system working correctly provided that is was working at $t=0$.
  - Maintainability $M(d)$: probability of system working correctly d time units after error occurred.
  - Availability $A(t)$: probability of system working at time t.
  - (Functional) Safety: no harm to be caused
  - Security: confidential and authentic communication
  - Even perfectly designed systems can fail if the assumptions about the workload and possible errors turn out to be wrong. Making the system dependable must not be an after-thought, it must be considered from the very beginning.

The Software Challenge

- From daily experience on desktop applications and other (hopefully) non-critical applications:
  erroneous software widely perceived as normal.
  - reset as universal fix
  - abundant patches and updates

- But:
  - software patch for pacemaker?

    SC Magazine Oct 2012: […] have reverse-engineered a pacemaker transmitter to make it possible to deliver deadly electric shocks to pacemakers within 30 feet and rewrite their firmware.
    http://www.scmagazine.com.au/News/319508,hacked-terminals-capable-of-causing-pacemaker-mass-murder.aspx

  - Ctrl-Alt-Delete on brake controller?
- Hence: high-quality software can be developed, but requires well-structured and sound approach.

Well-known Software Faults

- Ariane 5 – Flight 501
- Patriot missile software problem
- Airbus A-320 Flight to Warsaw 1993
- USS Yorktown incident
- Infusion pumps software problems

Ariane 5 – Flight 501

- Ariane 5
  - Satellite launcher
  - Successor of Ariane 4 with more payload capacity and lower cost
  - Explosion on maiden flight on June, 4th, 1996.
  - Complete report: http://esamultimedia.esa.int/docs/esa-x-1819eng.pdf
- Course of events
  - regular start, nominal behavior for 36 seconds
  - T + 36.7s: deviation from flight path, partial disintegration
  - T + 39s: self destruction by automated flight termination system
- Consequences
  - satellite loss: >370M US $
  - launcher unavaible for more than 1 year
  - reputation of Ariane rockets severely damaged (Ariane 4 considered very reliable)

Ariane 5 – Flight 501

- Ariane 5 Flight Control System (FCS):
  - Inertial Reference System (SRI): measures attitude of launcher and its movements in space.
    - Inertial platform: sensors, e.g., laser gyros and accelerometers.
    - Internal computer: calculates angles and velocities from sensor input
  - On-Board Computer (OBC): executes the flight program, i.e., controls actuators to follow intended trajectory.
- Hardware redundancy:
  - Two OBCs
  - Two SRIs operating in parallel, with identical hardware and software.

Ariane 5 – Flight 501

- Detailed chain of events:
  - Disintegration caused by angle of attack of more than 20 degrees.
  - Angle was commanded by OBC on basis of data transmitted by SRI2.
  - SRI2 showed a diagnostic bit pattern instead of correct attitude data because the unit had declared a failure due to a software exception.
  - The OBC could not switch to SRI1, because that unit had already ceased to function during the previous data cycle (72 milliseconds period) for the same reason as SRI2.
  - The internal SRI software exception was caused by an overflow during execution of a data conversion from 64-bit floating point to 16-bit signed integer value.
    - 64-bit floating point numbers in interval $[-3.6 \cdot 10^{308}, 3.6 \cdot 10^{308}]$
    - 16-bit signed integers in interval $[-32768, 32767]$
  - The unexpected high value occurred in the Horizontal Bias, BH, related to the horizontal velocity sensor input.

Ariane 5 – Flight 501

- Detailed chain of events (c'ed):
  - The error occurred in the alignment software reused from the Ariane 4 launcher. The value of BH was much higher than expected because Ariane 5 could reach considerably higher horizontal velocity values than Ariane 4.
  - The execution of the alignment software is not required after lift-off; it was introduced on Ariane-4 to avoid restarting the alignment (45 min) after a hold in the count-down.
  - This requirement does not apply to Ariane 5, which has a different preparation sequence; it was maintained for commonality reasons (don't change software which worked well on Ariane 4) although it served no purpose.

The Patriot Missile Software Problem

- Patriot: Surface-to-air defense missile system used, e.g. during Operation Desert Storm in the Gulf War in the early 1990's.
- On the night of the 25th of February, 1991, a Patriot missile system operating in Dhahran, Saudi Arabia, failed to track and intercept an incoming Scud. The Iraqi missile impacted into an army barracks, killing 28 U.S. soldiers and injuring another 98.
- The cause of the missile system failing to defend against the incoming Scud was traced back to a bug in Patriot's radar and tracking software.
- Full Report at http://www.gao.gov/assets/220/215614.pdf

The Patriot Missile Software Problem

- The bug occurs in the calculation of the next location of the incoming target. The prediction is calculated based on the target's velocity and the time of the last radar detection.
  - Velocity is stored as a whole number and a decimal.
  - Time is a continuous integer or whole number measured in tenths of a second.
  - The algorithm used to predict the next air space to scan by the radar requires that both velocity and time be expressed as real numbers.
  - The Patriot's computer only has 24-bit fixed-point registers.
  - The value 1/10 used to count tenth seconds has a non-terminating binary expansion and was chopped at 24 bits after the radix point.

Fixed-Point Numbers

- The base-b representation is extended by a radix point and a number of fractional bits. With two's complement notation:

$$z = z_{n-1} \dots z_0 . z_{-1} \dots z_{-m}$$
$$:= -z_{n-1} b^{n-1} + \dots + z_0\, b^0 + z_{-1} b^{-1} + \dots + z_{-m} b^{-m}$$

- Representable numbers for $\mathrm{b} = 2$:

$$-2^{n-1}, -2^{n-1} + 2^{-m}, \dots, 2^{n-1} - 2^{-m}$$

- Some binary numbers with $n = 9, m = 23$ (Q8.23 format):

Number (decimal)	Bit representation
0	000000000.00000000000000000000000
0.5	000000000.10000000000000000000000
-0.5	111111111.10000000000000000000000
-255.25	100000000.11000000000000000000000

The Patriot Missile Software Problem

- $\frac{1}{10}$ has a non-terminating binary representation. Using fixed-point numbers with 23 bits after the radix point causes the value to be truncated:

$$\frac{1}{10} = 0.0001100110011001100110011001100110\dots$$
$$\approx 0.00011001100110011001100$$

- The rounding error is 0.000000095_{10}.
- The error in precision grows as the time value increases.
- After 100 consecutive hours in continuous operation the resulting inaccuracy was roughly 0.34 seconds:
  $100 * 60 * 60 * 10 * 0.000000095_{10} \approx 0.34$
- The Scud travels at roughly 1.7km/sec.
- The computed target location was more than half a kilometer away from the missile.

Airbus A-320 Flight to Warsaw 1993

- Landing in Warsaw:
  - Bad weather conditions (rain, wind, wet runway)
  - Aquaplaning on touchdown at 300 km/h
  - Delayed operation of braking system
  - The aircraft departed the runway at a speed of 133 km/h and rolled 90 m before it hit the embankment and another airplane, causing a fire in the passenger cabin.

- Consequences
  - 2 dead, 56 injured (from 70 occupants)
  - Complete destruction of aircraft
- Complete report at
  http://www.rvs.uni-bielefeld.de/publications/Incidents/DOCS/ComAndRep/Warsaw/warsaw-report.html

Airbus A-320 Flight to Warsaw 1993

- Principle cause of crash:
  - weather conditions were not correctly evaluated by flight crew
  - incorrect decision of flight crew: abandonment of landing and go around was necessary
- Secondary cause: significant delay of braking systems

Time	Distance from THR11*	Situation
T0	770m	RLG on ground
T0+3	1030m	NLG on ground, braking command issued (reverser levels full)
T0+6s	1525m	LLG on ground
T0+12s	1680m	spoilers fully deployed
T0+14s	1800m	full reversers achieved
T0+31s	2700m	end of runway

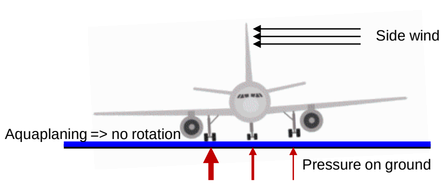
*beginning of runway

Airbus A-320 Flight to Warsaw 1993

- Software design considerations:
  - ➢ Prevent activating braking system in mid-air
  - spoilers: high aerodynamic forces
  - thrust reversal: danger of in-flight destruction (example: accident of a 767-300 ER Lauda Air, 1991, 223 casualties).
- Specification:
  - activate spoilers and reversers only if
    - throttle is at minimum
    - AND one of the following conditions:
      - EITHER pressure of more than 6t on left and right landing gears
      - OR wheel rotation at speed above 130 km/h at both main landing gears

Airbus A-320 Flight to Warsaw 1993

- Conditions at landing:



- Spoilers / reversers: inhibited
  - not enough pressure on left main landing gear (side wind)
  - insufficient wheel rotation speed
- Consequence: change of specification; new condition: $P_{left} + P_{right} \geq P_{min}$

USS Yorktown Incident

- USS Yorktown was a cruiser in the United States Navy from 1984 to 2004.
- On 21 September 1997, while on maneuvers off the coast of Cape Charles, Viriginia, a crew member entered a zero into a database field causing a divide by zero error in the ship's Remote Data Base Manager which brought down all the machines on the network, causing the ship's propulsion system to fail.
- According to witness reports (revoked later) *Yorktown* had to be towed back to Norfolk Naval Station.

Infusion Pumps Software Problems

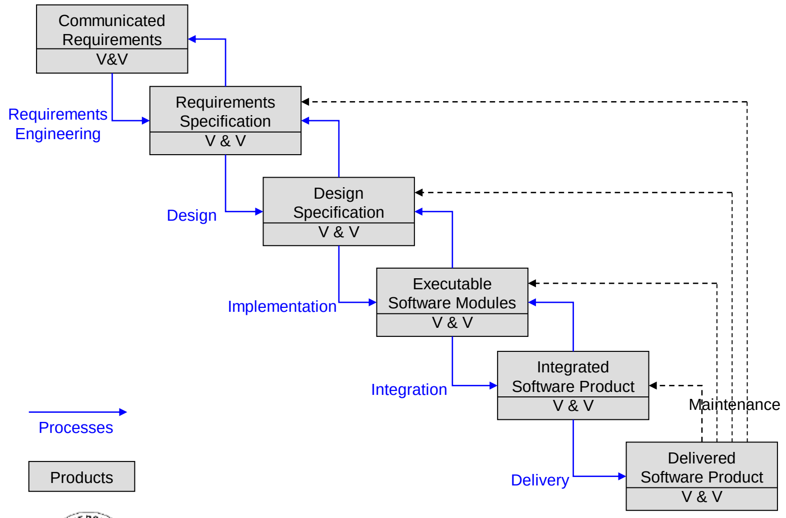
- External infusion pumps are medical devices that deliver fluids, including nutrients and medications, into a patient’s body in a controlled manner.
- From 2005 through 2009, FDA received approximately 56,000 reports of adverse events associated with the use of infusion pumps, including numerous injuries and deaths.
- From 2005 through 2009, 87 infusion pump recalls were conducted by firms to address identified safety problems.
- Many of the problems that have been reported are related to software malfunctions. Examples:
  - failure to activate alarm when problems occur
  - activated alarm in the absence of a problem
  - over- or under-infusion.
- Source: *U.S. Food and Drug Administration, Center for Devices and Radiological Health. White Paper: Infusion Pump Improvement Initiative, 2010.*

Software Development

- Waterfall model
  - Classic software life cycle model; until early 1980s the only widely accepted life cycle model.
  - Represents the software life cycle using processes and products.
  - Each process transforms a product to produce a new product as output. Then the new product becomes the input of the next process.
  - Important characteristics: processes are iterative.

- V-Model
  - Regulates "who", "when", "what" in a software development project.
  - Development standard for IT systems of the German Federation for the entire civil and military area.
  - Basics: hierarchical decomposition of system into smaller parts until realization becomes possible.
  - Verification and validation is done on each construction stage.
  - No strict temporal ordering imposed.

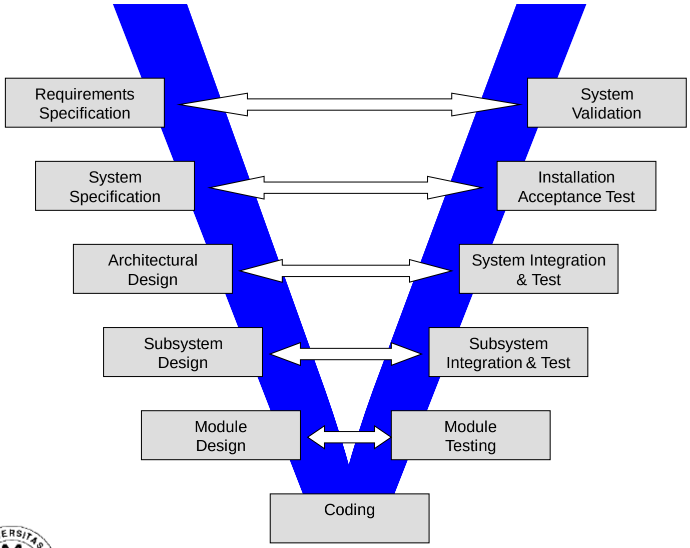
Waterfall Model



Software Development

- Waterfall model
  - Classic software life cycle model; until early 1980s the only widely accepted life cycle model.
  - Represents the software life cycle using processes and products.
  - Each process transforms a product to produce a new product as output. Then the new product becomes the input of the next process.
  - Important characteristics: processes are iterative.
- V-Model
  - Regulates "who", "when", "what" in a software development project.
  - Development standard for IT systems of the German Federation for the entire civil and military area.
  - Basics: hierarchical decomposition of system into smaller parts until realization becomes possible.
  - Verification and validation is done on each construction stage.
  - No strict temporal ordering imposed.

V-Model



Safety Standards – Avionics

- DO-178B Standard: guidelines for the production of software for airborne systems and equipment.
  - Development assurance levels:
    - A: catastrophic failure condition for the aircraft (e.g. aircraft crash)
    - B: Hazardous/severe failure condition for the aircraft (e.g. injured persons)
    - C: Major failure condition for the aircraft (e.g. flight management system down => manual operation by pilot)
    - D: Minor failure condition for aircraft (e.g. pilot-ground communications down)
    - E: No effect on aircraft operation or pilot workload (e.g. entertainment system down)

DO-178B Standard

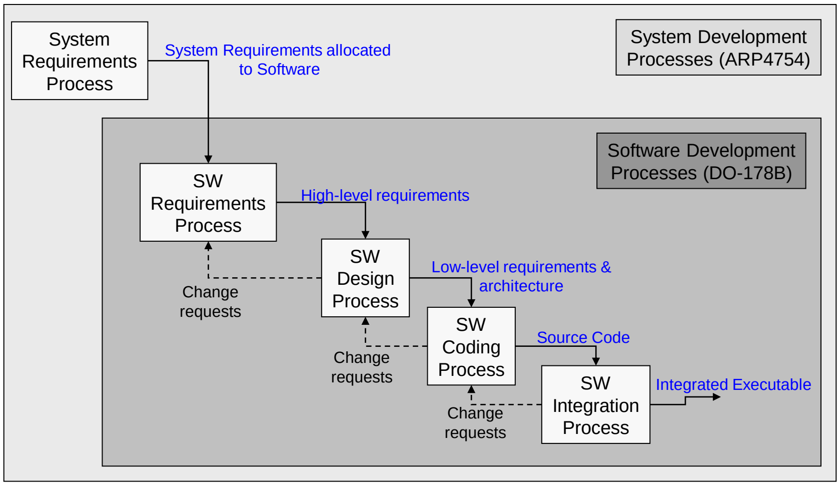
- Essence: formulation of appropriate objectives/requirements and verification that these objectives have been achieved. The ways of achieving an objective may vary.

- Purpose: detect and report errors that may have been introduced during the software development process.

- Important: All requirements have to be verifiable and must be compliant with the requirements of other stages.

- Testing is part of the verification process, but reviews and analyses are also required. Analyses should be reproducible.

DO-178B Development Process



DO-178B Verification Process

- Reviews and Analyses of the High-Level Requirements:
  - Algorithm accuracy
- Reviews and Analyses of the Low-Level Requirements:
  - Compatibility with target computer: no conflict between software requirements and hardware/software features of the target computer, e.g. system response times, input/output hardware
- Reviews and analyses of the source code:
  - Verifiability: the source code does not contain statements and structures that cannot be verified and the code does not have to be altered to test it.
  - Accuracy and consistency: stack usage, resource contention, worst-case execution timing, exception handling, use of non-initialized variables or constants.

DO-178C

- Revision of DO-178B to bring it up to date with current software development and verification techniques, published in 2012.
  - model-based software development
  - object-oriented software
  - use and qualification of software tools and
  - the use of formal methods to complement or replace dynamic testing
    - theorem proving
    - model checking
    - abstract interpretation

Development of Avionics Software

- Airbus A340 contains 115 digital units and 20 MB onboard software.

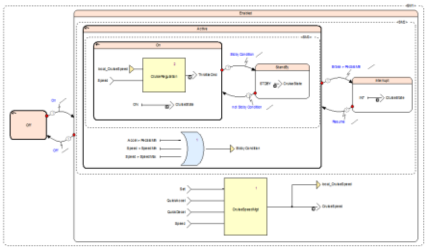
- Development of safety-critical avionics software is very expensive:
  - Avg development and test of 10 KLOC level B software is 16 person-years
  - Cost of minor bug is $100K-$500K
  - Cost of major bug is $1M-$500M
  - Time-to-market 3-4 years
  - For Level A software, the overall verification cost (including testing) may account for up to 80% of the budget

Why so expensive?

- Multiple descriptions:
  - At each level software is rewritten into another form – traditionally by hand => expensive and error-prone.
- Ambiguity and lack of accuracy of specifications.
- Manual coding
- Late detection of specification and design errors

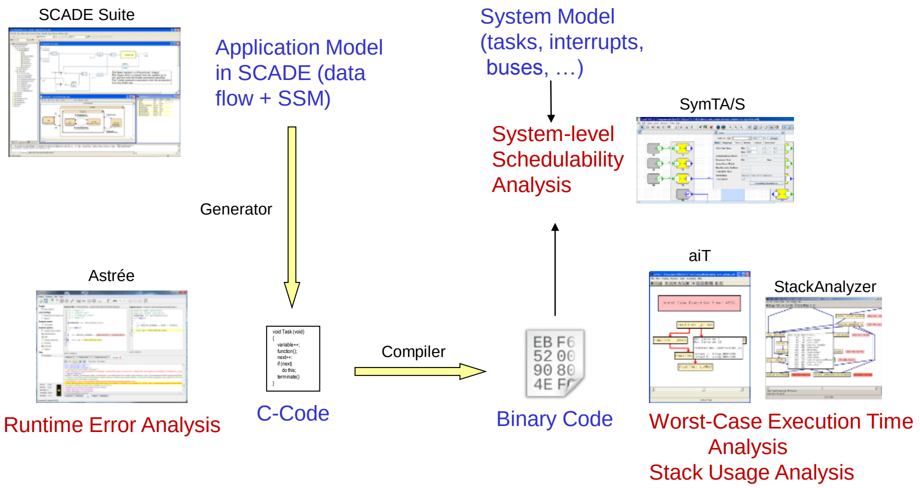
Model-based Software Development

- Application graphically specified by data flow diagrams and/or finite state machines
- Model is software specification and has executable semantics
- Automated & integrated development tools:
  - automatic target code generation (typically C code)
  - automatic simulation
  - formal verification at model level
  - model-based testing



- Examples: Esterel SCADE, Matlab/Simulink + dSPACE TargetLink
- BUT: Higher level of abstraction than with programming in C.
  - timing? memory consumption? runtime errors? system integration?

Model-based Software Development



Contents of Lecture

- SCADE: Data flow kernel (textual representation)
- Basic Automata Theory
- SCADE: SyncCharts / Safe State Machines (SSM)

- Abstract Interpretation: basics & theory, applications to runtime errors, worst-case execution time, stack usage.

- Scheduling & Schedulability Analysis

Lego Mindstorms NXT

- A brief overview of the NXT hardware based on information from LEGO. The LEGO Mindstorm NXT consists of following items:
- **NXT brick**
  - CPU: 32-bit ARM7 micro controller @ 48 MHz
  - Co-Processor: 8-bit AVR micro controller @ 4 MHz
  - 4 input ports / 3 output ports
  - 100 x 64 pixel LCD display
  - USB 2.0 and Bluetooth support
  - Speaker
- **Sensors**
  - Ultrasonic sensor
  - Touch sensor
  - Sound sensor
  - Light sensor
- **Motors**
  - 3 motors with integrated rotation sensors

Practical Project

- Aim: program a Mindstorm robot to
  - automatically drive to a light source
  - avoid obstacles along the way
  - be controllable by clapping
- Organization
  - Groups of 2 to 3 students
  - Several Milestones over the whole semester:
    - Milestone 1 (week 48): Specification as SyncChart
    - Milestone 2 (week 51): Implementation using Scade
    - Milestone 3 (week 5): Timing and stack usage validation with aiT/StackAnalyzer

Tutorials

- Gruppe A (Hahn)
- Fr, 12-14
- Mo, 10-12

- Gruppe B (Haupenthal)
- Mi, 10-12
- Mi, 12-14
- Do, 10-12