Course Requirements

Software Engineering for Dependable Systems

Tomas Bures
bures@d3s.mff.cuni.cz
Organisation

- Seminar – every other week
  - 4 teaching hours each

- Tomáš Bureš
  - bures@d3s.mff.cuni.cz

- Course website
  - https://d3s.mff.cuni.cz/teaching/nswi054/
Requirements for passing the course

- Students have to subscribe to the course and the group in the Student Information System

- If you cannot attend, let me know (by email) in advance
Requirements for passing the course

- 80% attendance

- Completed homework/report
  - To be prepared by the next meeting

- These will cover the topics covered during the course:
  - Requirement modeling and specification
  - Test specification
  - Various design models
Resources

• Ian Sommerville: Software Engineering (10th edition)

• Online resources indicated in each lecture
Introduction

Software Engineering for Dependable Systems

Tomas Bures
bures@d3s.mff.cuni.cz
Dependability

“The ability to provide services that can defensibly be trusted within a time-period”

- **Availability**: The ability of the system to deliver services when requested
- **Reliability**: The ability of the system to deliver services as specified
- **Safety**: The ability of the system to operate without catastrophic failure
- **Security**: The ability of the system to protect itself against deliberate or accidental intrusion
- **Resilience**: The ability of the system to resist and recover from damaging events
Principal properties

- **Availability**
  - The probability that the system will be up and running and able to deliver useful services to users.

- **Reliability**
  - The probability that the system will correctly deliver services as expected by users.

- **Safety**
  - A judgment of how likely it is that the system will cause damage to people or its environment.

- **Security**
  - A judgment of how likely it is that the system can resist accidental or deliberate intrusions.

- **Resilience**
  - A judgment of how well a system can maintain the continuity of its critical services in the presence of disruptive events such as equipment failure and cyberattacks.
Related system properties

• Repairability
  ▪ Reflects the extent to which the system can be repaired in the event of a failure

• Maintainability
  ▪ Reflects the extent to which the system can be adapted to new requirements;

• Error tolerance
  ▪ Reflects the extent to which user input errors can be avoided and tolerated.
Causes of failure

- **Hardware failure**
  - Hardware fails because of design and manufacturing errors or because components have reached the end of their natural life.

- **Software failure**
  - Software fails due to errors in its specification, design or implementation.

- **Operational failure**
  - Human operators make mistakes. Now perhaps the largest single cause of system failures in socio-technical systems.

Adopted from slides by Ian Sommerville for Software Engineering (10th edition)
Cost/dependability curve

Adopted from slides by Ian Sommerville for Software Engineering (10th edition)
Sociotechnical systems stack

- Systems perspective is essential for dependability
- Necessary to understand how the layers influence the software
- ... and how faults and failures propagate and multiply

Adopted from slides by Ian Sommerville for Software Engineering (10th edition)
Layers in the STS stack

- **Equipment**
  - Hardware devices, some of which may be computers. Most devices will include an embedded system of some kind.

- **Operating system**
  - Provides a set of common facilities for higher levels in the system.

- **Communications and data management**
  - Middleware that provides access to remote systems and databases.

- **Application systems**
  - Specific functionality to meet some organization requirements.

- **Business processes**
  - A set of processes involving people and computer systems that support the activities of the business.

- **Organizations**
  - Higher level strategic business activities that affect the operation of the system.

- **Society**
  - Laws, regulation and culture that affect the operation of the system.

Adopted from slides by Ian Sommerville for Software Engineering (10th edition)
Dependable processes

- Software process to produce dependable software
- Explicitly defined process
  - Data are collected during the process to show that the development team has followed the process
- Repeatable process
  - Does not rely on individual interpretation and judgement. Can be repeated across projects with different team members. Particularly important for critical systems which are developed over long period of time and team may change a lot.

<table>
<thead>
<tr>
<th>Process characteristic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auditable</td>
<td>The process should be understandable by people apart from process participants, who can check that process standards are being followed and make suggestions for process improvement.</td>
</tr>
<tr>
<td>Diverse</td>
<td>The process should include redundant and diverse verification and validation activities.</td>
</tr>
<tr>
<td>Documentable</td>
<td>The process should have a defined process model that sets out the activities in the process and the documentation that is to be produced during these activities.</td>
</tr>
<tr>
<td>Robust</td>
<td>The process should be able to recover from failures of individual process activities.</td>
</tr>
<tr>
<td>Standardized</td>
<td>A comprehensive set of software development standards covering software production and documentation should be available.</td>
</tr>
</tbody>
</table>

Adopted from slides by Ian Sommerville for Software Engineering (10th edition)
Dependable processes

- Activities
  - Requirements review
  - Requirements management
  - Formal specification
  - System modeling
  - Design and program inspections
  - Static analysis
  - Test planning and management
Formal methods and dependability

- Formal methods – mathematical approaches to verification of software
- Typically based on formal models of software behavior
- Effective for discovering or avoiding:
  - Specification and design errors and omissions
  - Inconsistencies between a specification and a program

More at:
- NSWI132 (Program Analysis and Code Verification)
  - Summer term – 2/2 Zk+Z
  - Lecture: Wed 10:40 S1
  - Lab: Wed 12:20 SU1
- NTIN043 (Formal Foundations of Software Engineering)
  - Winter term – 2/2 Zk+Z
- NSWI101 (System Behavior Models and Verification)
  - Winter term – 2/2 Zk+Z
Dependability in timing

- Dependable systems are often real-time and embedded
- Requires special design and reasoning to give guarantees about timing

More at NSWE001 (Embedded and Real-time Systems)
Why Dependability?
Arianne 5

- Exploded on June 4, 1996
  - only 39 seconds after launch
  - loss of about US$ 370 million
- A 64-bit float was truncated to 16-bit integer in a “non-critical software component”
- This caused unhandled hardware exception
- The erroneous component (a method) was inherited/reused from Ariane 4 and had no practical use in Ariane 5
Patriot – Failure at Dhahran

- February 25, 1991, an Iraqi Scud hit the barracks in Dhahran killing 28 soldiers
- The area was protected by Patriot aerial interceptor missiles
- Due to drift of system's internal
  - by one third of a second in 100 hours
  - amounted to miss distance of 600 meters
- The system detected the missile but due to the time skew, it disregarded it as spurious
Therac-25

- Computer controller radiation therapy machine
- 6 accidents 1985-1987
  - three people died as the direct consequence of radiation burns
- Race condition as the primary cause
- Other causes included
  - Poor design, no review of the software
  - Bad man-machine interface
  - Overconfidence in the software
  - Not understanding safety
- The software was in use previously, but different hardware design covered its flaws
2003 blackout in 8 states of USA and in Ontario (in total 55 mil. people affected), primary culprit was a SW problem (race condition) which delayed notifications
2007 F22 Raptor software bug
navigation, communication and fuel systems crashed after crossing the international date line
2014 Jeep Cherokee wirelessly hacked
security flaw in the infotainment system which allowed an attacker to control almost everything (including throttle, brakes and steering) over internet
Systems Engineering
Professional disciplines involved

- Architecture
- Electrical engineering
- Electronic engineering
- Civil engineering
- Systems engineering
- Software engineering
- Mechanical engineering
- Ergonomics
- User interface design

Adopted from slides by Ian Sommerville for Software Engineering (10th edition)
The system development process

- **Requirements engineering**
  - The process of refining, analysing and documenting the high-level and business requirements identified in the conceptual design

- **Architectural design**
  - Establishing the overall architecture of the system, identifying components and their relationships

- **Requirements partitioning**
  - Deciding which subsystems (identified in the system architecture) are responsible for implementing the system requirements
The system development process

• Subsystem engineering
  ▪ Developing the software components of the system, configuring off-the-shelf hardware and software, defining the operational processes for the system and re-designing business processes

• System integration
  ▪ Putting together system elements to create a new system

• System testing
  ▪ The whole system is tested to discover problems

• System deployment
  ▪ the process of making the system available to its users, transferring data from existing systems and establishing communications with other systems in the environment
Software engineering
Software process

- A structured set of activities required to develop a software system.

- Many different software processes but all involve:
  - Specification – defining what the system should do;
  - Design and implementation – defining the organization of the system and implementing the system;
  - Validation – checking that it does what the customer wants;
  - Evolution – changing the system in response to changing customer needs.
The waterfall model

1. Requirements definition
2. System and software design
3. Implementation and unit testing
4. Integration and system testing
5. Operation and maintenance

Adopted from slides by Ian Sommerville for Software Engineering (10th edition)
Requirements engineering process

- Requirements elicitation and analysis
  - System descriptions
  - User and system requirements
- Requirements specification
- Requirements validation
  - Requirements document

Adopted from slides by Ian Sommerville for Software Engineering (10th edition)
Software design process

Design inputs

- Platform information
- Requirements specification
- Data description

Design activities

- Architectural design
- Interface design
- Component design

Database design

Design outputs

- System architecture
- Database specification
- Interface specification
- Component specification

Adopted from slides by Ian Sommerville for Software Engineering (10th edition)
System implementation

- The software is implemented either by developing a program or programs or by configuring an application system.
- Design and implementation are interleaved activities for most types of software system.
- Programming is an individual activity with no standard process.
- Debugging is the activity of finding program faults and correcting these faults.
Software validation

- Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.
- Involves checking and review processes and system testing.
- System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.
- Testing is the most commonly used V & V activity.

Adopted from slides by Ian Sommerville for Software Engineering (10th edition)
Stages of testing

- Component testing
  - Individual components are tested independently;
  - Components may be functions or objects or coherent groupings of these entities.

- System testing
  - Testing of the system as a whole. Testing of emergent properties is particularly important.

- Customer testing
  - Testing with customer data to check that the system meets the customer’s needs.
Testing phases in a V-model

V-model – a plan driven development process

Requirements specification → System specification → System design → Detailed design

- Acceptance test plan
- System integration test plan
- Sub-system integration test plan

Module and unit code and test

- Service
- Acceptance test
- System integration test
- Sub-system integration test

Adopted from slides by Ian Sommerville for Software Engineering (10th edition)
Requirements Engineering 1

Software Engineering for Dependable Systems

Tomas Bures
bures@d3s.mff.cuni.cz
Requirements Engineering

- The process of establishing the services that a customer requires from a system and the constraints under which it operates and is developed.

- The system requirements are the descriptions of the system services and constraints that are generated during the requirements engineering process.
Types of Requirements

- **User requirements**
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.

- **System requirements**
  - A structured document setting out detailed descriptions of the system’s functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

Adopted from slides by Ian Sommerville for Software Engineering (10th edition)
User and Systems Requirements

User requirements definition

1. The Mentcare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g. 10mg, 20mg, etc) separate reports shall be created for each dose unit.
1.5 Access to drug cost reports shall be restricted to authorized users as listed on a management access control list.
User and Systems Requirements

User requirements

Client managers
System end-users
Client engineers
Contractor managers
System architects

System requirements

System end-users
Client engineers
System architects
Software developers

Adopted from slides by Ian Sommerville for Software Engineering (10th edition)
Functional and non-functional requirements

- **Functional requirements**
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
  - May state what the system should not do.

- **Non-functional requirements**
  - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
  - Often apply to the system as a whole rather than individual features or services.

- **Domain requirements**
  - Constraints on the system from the domain of operation.
Types of non-functional requirements

Non-functional requirements

Product requirements
- Efficiency requirements
- Dependability requirements

Organizational requirements
- Security requirements
- Regulatory requirements
- Ethical requirements
- Environmental requirements
- Operational requirements
- Development requirements
- Legislative requirements

External requirements
- Accounting requirements
- Safety/security requirements

Adopted from slides by Ian Sommerville for Software Engineering (10th edition)
Examples of nonfunctional requirements

**Product requirement**
The Mentcare system shall be available to all clinics during normal working hours (Mon–Fri, 0830–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

**Organizational requirement**
Users of the Mentcare system shall authenticate themselves using their health authority identity card.

**External requirement**
The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.
Requirements engineering processes

• Requirements elicitation & analysis
  ▪ Requirements discovery
  ▪ Requirements classification and organization
  ▪ Requirements prioritization and negotiation
  ▪ Requirements specification

• Requirements validation
  ▪ Ensures
    • Validity – does the system provide the functions which best support the customer’s needs?
    • Consistency – are there any requirements conflicts?
    • Completeness – are all functions required by the customer included?
    • Realism – can the requirements be implemented given available budget and technology
    • Verifiability – can the requirements be checked?
  ▪ By requirement reviews, prototyping, test-case generation

• Requirements management
Spiral view of the RE process

Adopted from slides by Ian Sommerville for Software Engineering (10th edition)
Goal-based elicitation and analysis of requirements
Goals

- Goal – a stakeholder objective for the system
- Model of goals hierarchy and their relation
  - Answers the question “Why?”
- Guides requirement elaboration
- Results into concrete requirements which can be turned to Requirements Specification document

Early RE -> Late RE -> Design -> Code -> Test

KAOS

• A methodology for goal-based requirements engineering


• Models:
  - Goal model
  - Object model
  - Agent responsibility model
  - Operation model
KAOS – Goal model

Goal

Transportation requests satisfied in a safe, efficient, usable, and cheap way

AND - decomposition

Cheap elevator system
Usable elevator system
Efficient elevator system
Safe elevator system
Transportation requests satisfied
KAOS – Goal model

Efficient elevator system

Waiting time minimized

Elevator moves minimized

Stops at intermediate floors if pending requests

Transportation duration minimized

Closest elevator sent on passenger calls

Number of direction changes minimized

Passengers informed of elevator direction

Passengers do not enter elevators heading in the opposite direction

Elevator direction updated few seconds before next stop

Conflict

Requirement

Expectation
KAOS – Goal model

- Elevator called
  - Button-based interface provided
  - Button depressed
  - Button command detected
  - Passengers informed of their call’s status
  - Selected button’s light on until requested lift arrives
  - All button lights off until an elevator gets called

Responsibility
- Elevator Company
- Passenger
- Elevator Controller

Agent
- Responsibility
- Expectation assignment
KAOS – Goal model

OR - decomposition

Qualitative goals to enable selection of a particular decomposition
KAOS – Goal model

Elaboration of an expectation
Goal patterns

- Achieve – achieve the goal at some point in the future
- Cease – undo a goal at some point in the future
- Maintain – maintain a goal for some time
- Avoid – prevent a goal from becoming true
- Optimize – maximize or minimize some measure (soft-goal)
Tactics for decomposing goals

- Case-driven decomposition
  - E.g. *The goal of the system is to build a system that satisfies all stakeholders’ needs: functional and non-functional ones.*

- Milestone driven decomposition
  - E.g. *System is cheap to build, (then) cheap to run, and (then) cheap to maintain.*
Generic goal patterns

System satisfying stakeholders' needs

- System satisfying functional needs
- System satisfying non-functional needs

- Safe system
- Cheap system
- Usable system
- Efficient system
- Laws in force respected
- Environment preserved
Generic goal patterns

- System cheap to build
- System cheap to run
- System cheap to maintain

- Resource consumption minimized

- Robust and reliable system
- Evolutive system
Generic goal patterns

Safe system

No casualty

Robust and reliable system

Secure system

Privacy preserved

No intrusion
Generic goal patterns

- Usable system
- Fair system
- System easy to use
- Robust and reliable system
- Users informed about request status
Generic goal patterns

Service requested

Service request maintained until executed

Service executed according to request

Infrastructure available

Domain property

Service request satisfied
Agent responsibility model

- Elevator stopped
- Overweight conditions reported to the passengers
- Cage door closed while moving
- Elevator stopped at passenger destination
- Elevator kept on current floor, doors open, until overweight conditions disappear
- Elevator stopped upon power failure
- No door opening while moving
- Weight conditions checked before next move
- Emergency conditions reported
- Passengers informed of their call’s status
- Elevator resumed
- Button command detected
- Moving elevator stopped next floor in case of fire signal
- Door closed when cage not stopped on a floor level
- Emergency lights on when needed
Object model

No passenger locked in forever in case of breakdown

Elevator equipped with a breakdown alarm

Breakdown alarm used

Emergency conditions reported

Guard appointed

Alarm answered by guard

Emergency power available

Alarm cleared by guard’s response

Concerns:

- Alarm bell
- Relationship

Entity
Object model

**Diagram:**

- **Object Model Diagram for Elevator System**
  - **Elevator System**:
    - Year: Date
    - Model: String
    - Make: String
    - Associations as in UML
  - **Alarm Device**
  - **Alarm Bell**
  - **Power Supply**
  - **Floor**
    - 2..n
  - **Control Room**
  - **Elevator Controller**
    - 1

**Note:**
- Associations as in UML
Operation model

Operationalisation of a requirement

Schedule

Input

Concerns

Execute schedule

Perfomance

Elevator Controller

Elevator stopped on calling floor

Process
Operation model

- Cage State change
- Arrival on floor
- Button light on
- Button light off
- Cage scale measure change
- Cage Door Sensor event
- Elevator state change
Obstacle model

- Passengers informed of elevator direction
  - Elevator direction updated few seconds before next stop

Obstacle decomposition

- Elevator direction unreadable
  - Inadequate LED size
    - LEDs at least 2in high
  - Elevator direction unreadable by blind people
    - Elevator direction announced by voice
  - Insufficient display contrast

Obstacle mitigation
Completeness Criteria

- A goal model is said to be complete with respect to the refinement relationship ‘if and only if’ every leaf goal is either an expectation, a domain property or a requirement.

- A goal model is complete with respect to the responsibility relationship ‘if and only if’ every requirement is placed under the responsibility of one and only one agent (either explicitly or implicitly if the requirement refines another one which has been placed under the responsibility of some agent).

- To be complete, a process diagram must specify
  - the agents who perform the operations
  - the input and output data for each operation.

- To be complete, a process diagram must specify when operations are to be executed.

- All operations are to be justified by the existence of some requirements (through the use of operationalization links).