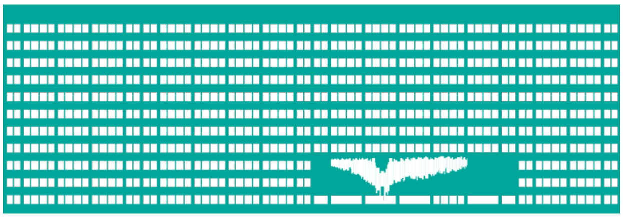
VŠB TECHNICKÁ
UNIVERZITA
OSTRAVA

VSB TECHNICAL
UNIVERSITY
OF OSTRAVA



www.vsb.cz

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

An Introduction to Software Engineering

Author:

Ing. Svatopluk Štolfa, Ph.D.

Department of Computer Science

VSB - Technical University of Ostrava

www.cs.vsb.cz/stolfa

Presenter:

Ing. David Ježek, Ph.D.

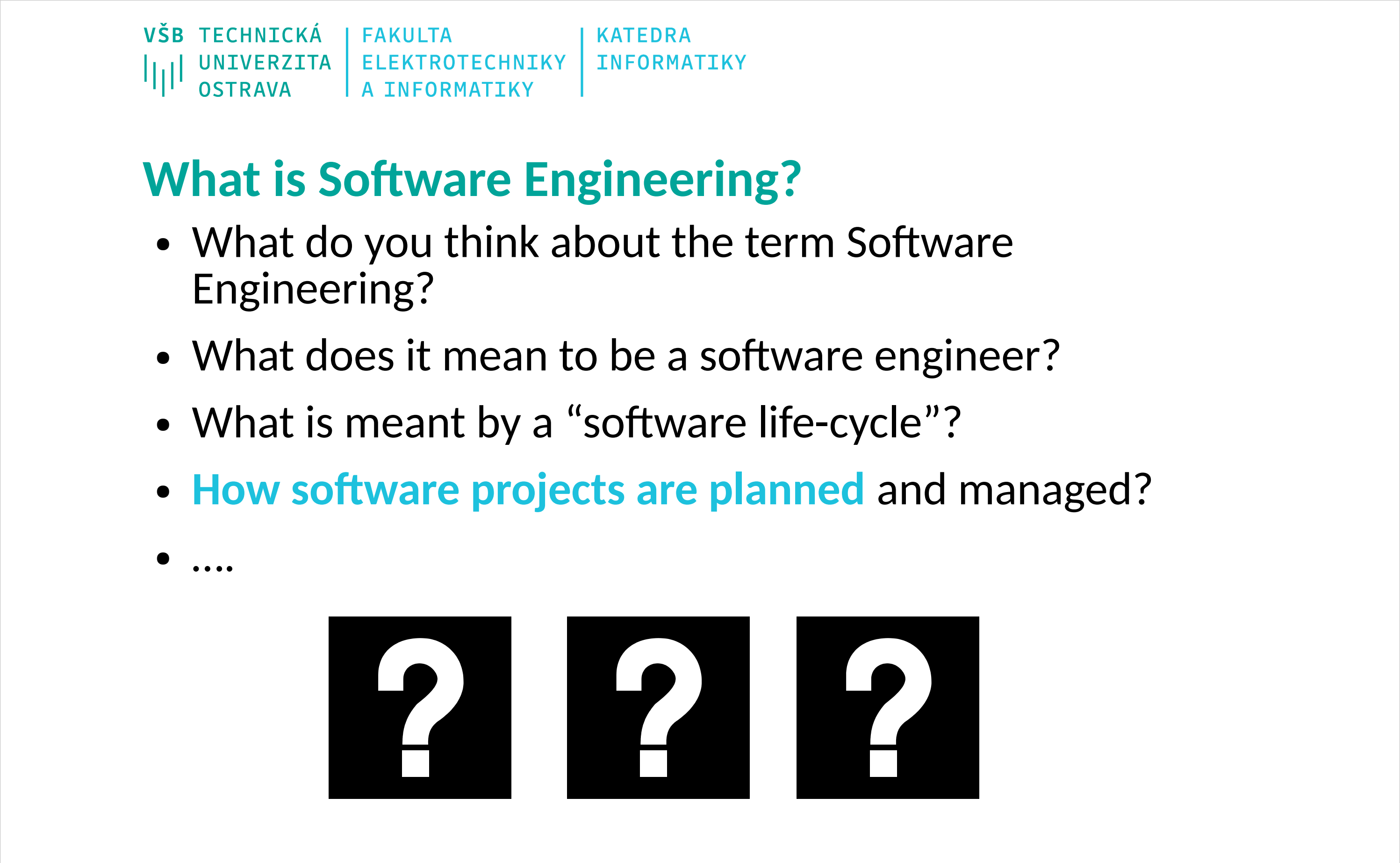
https://swi.cs.vsb.cz/en/jezek/student-information/swi.html

Objectives

- To introduce software engineering and to explain its importance
- To set out the answers to key questions about software engineering
- To introduce ethical and professional issues and to explain why they are of concern to software engineers
- To gain a basic knowledge to become a successful software engineer

References

- Pfleeger, Shari Lawrence, and Joanne M. Atlee. 2009. *Software Engineering: Theory and Practice: Prentice Hall*, ISBN 0136061699.
- Pressman, Roger S. 2010. *Software Engineering : A Practitioner's Approach*. 7th ed. New York: McGraw-Hill Higher Education, ISBN 9780073375977.
- Sommerville, Ian. 2010. *Software Engineering*. 9th ed, International Computer Science Series. Harlow: Addison-Wesley, ISBN 978-0137035151.



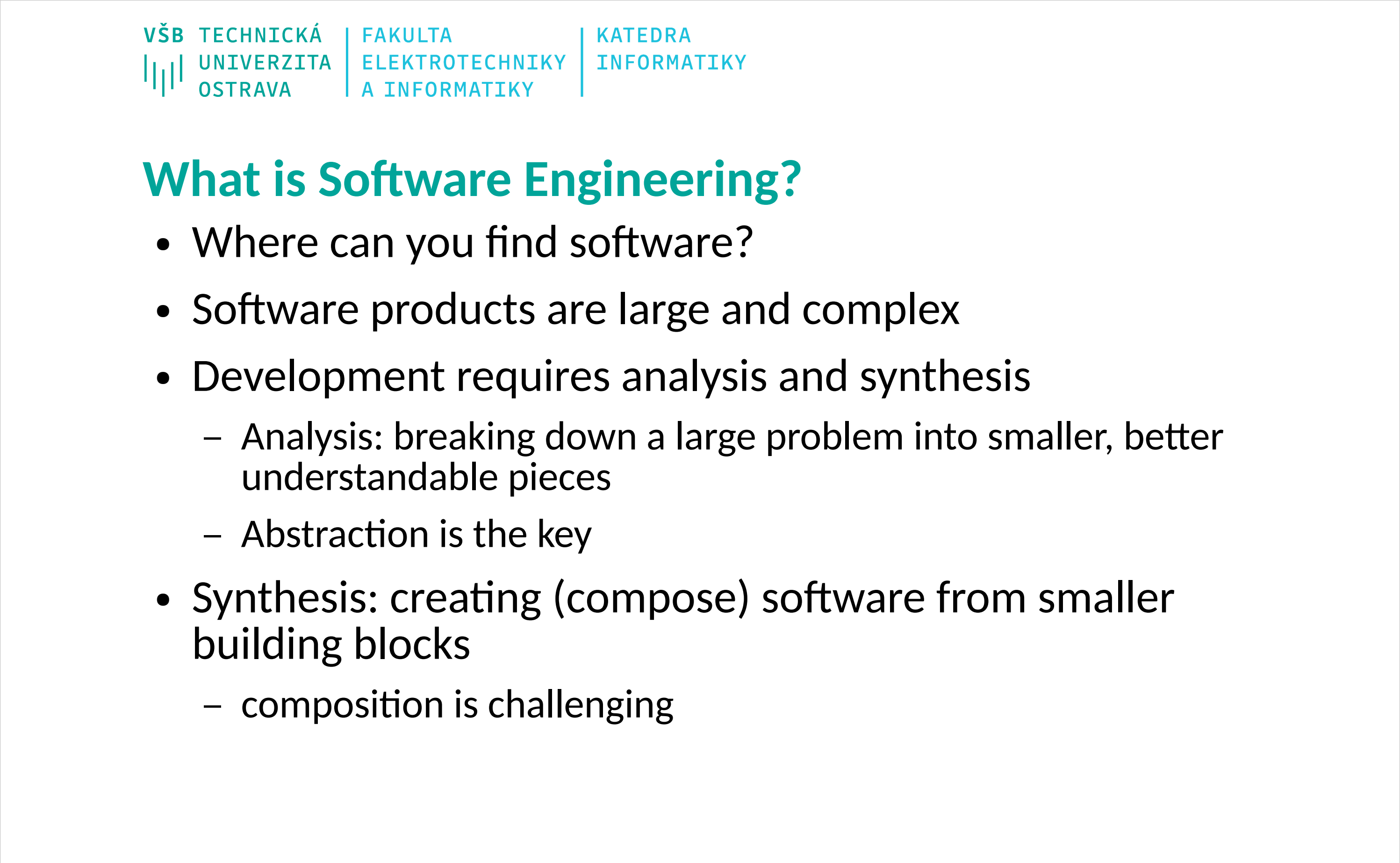
What is Software Engineering?

- What do you think about the term Software Engineering?
- What does it mean to be a software engineer?
- What is meant by a "software life-cycle"?
- **How software projects are planned** and managed?
- ….

Since we are going to talk about software engineering it is good to know what it means. Don't you think?

Are you able to answer these few important questions? What do you think?

Do not worry if you are not sure about the right answer. This is the reason why we are all here. To try to answer these questions, to gain some basic knowledge about a field of study called software engineering and finally, to become a successful software engineer. Ok let's start.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

What is Software Engineering?

- Where can you find software?
- Software products are large and complex
- Development requires analysis and synthesis
  - Analysis: breaking down a large problem into smaller, better understandable pieces
  - Abstraction is the key
- Synthesis: creating (compose) software from smaller building blocks
  - composition is challenging

Do you think that it is easy to build software? Can you think of some software products?

MS Office, Car computer unit, airplane computer, mobile phone, …

Try to answer this questions:

Aren’t this programs and solutions quite big to do it at once? Where would you start do build such software?

Are you able to code it yourself. How long will it take? Is it not better to cooperate with other people?

Software products are large and complex

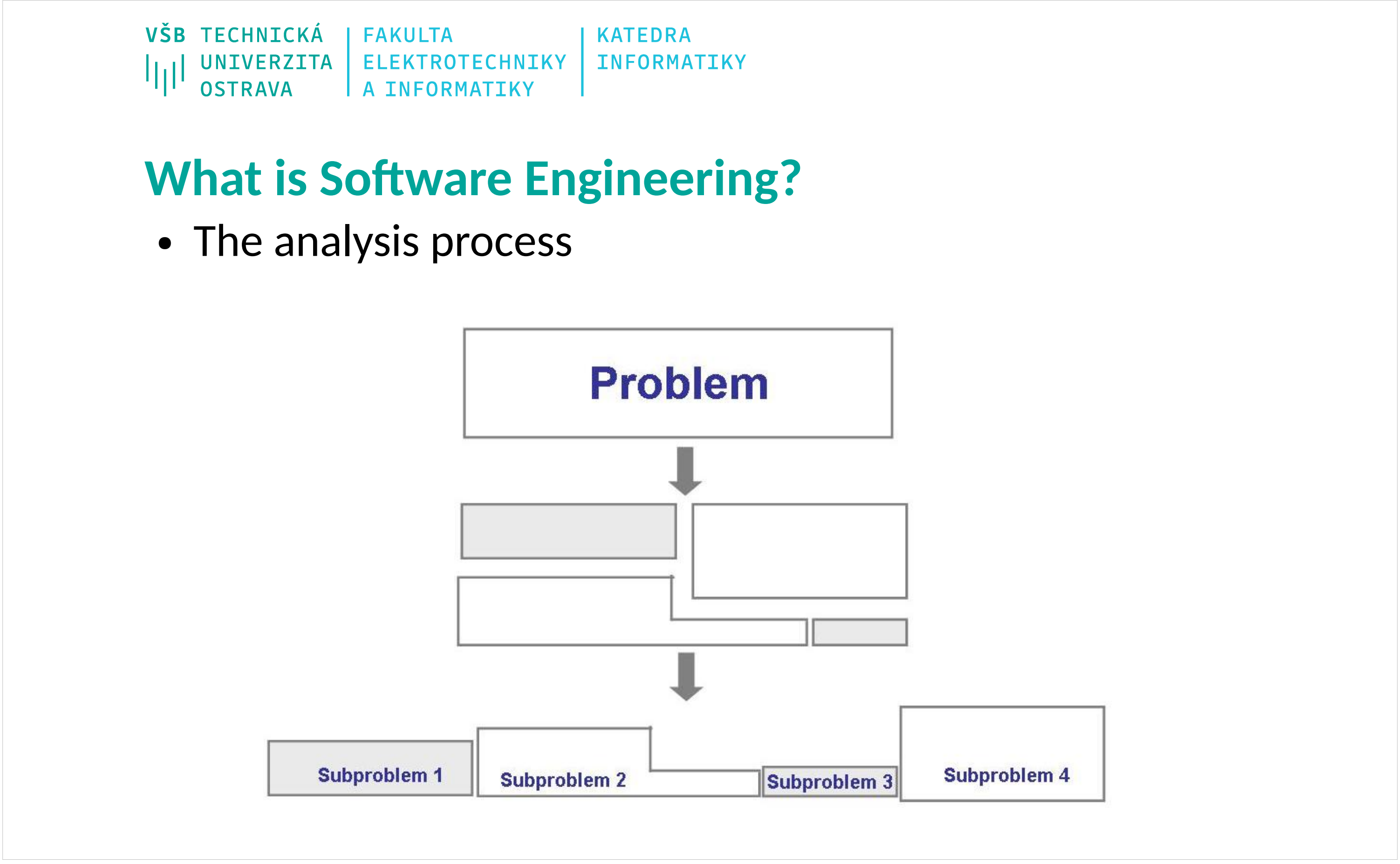
Development requires analysis and synthesis

Analysis: break down a large problem into smaller, understandable pieces

abstraction is the key

Synthesis: build (compose) a software from smaller building blocks

composition is challenging



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

What is Software Engineering?

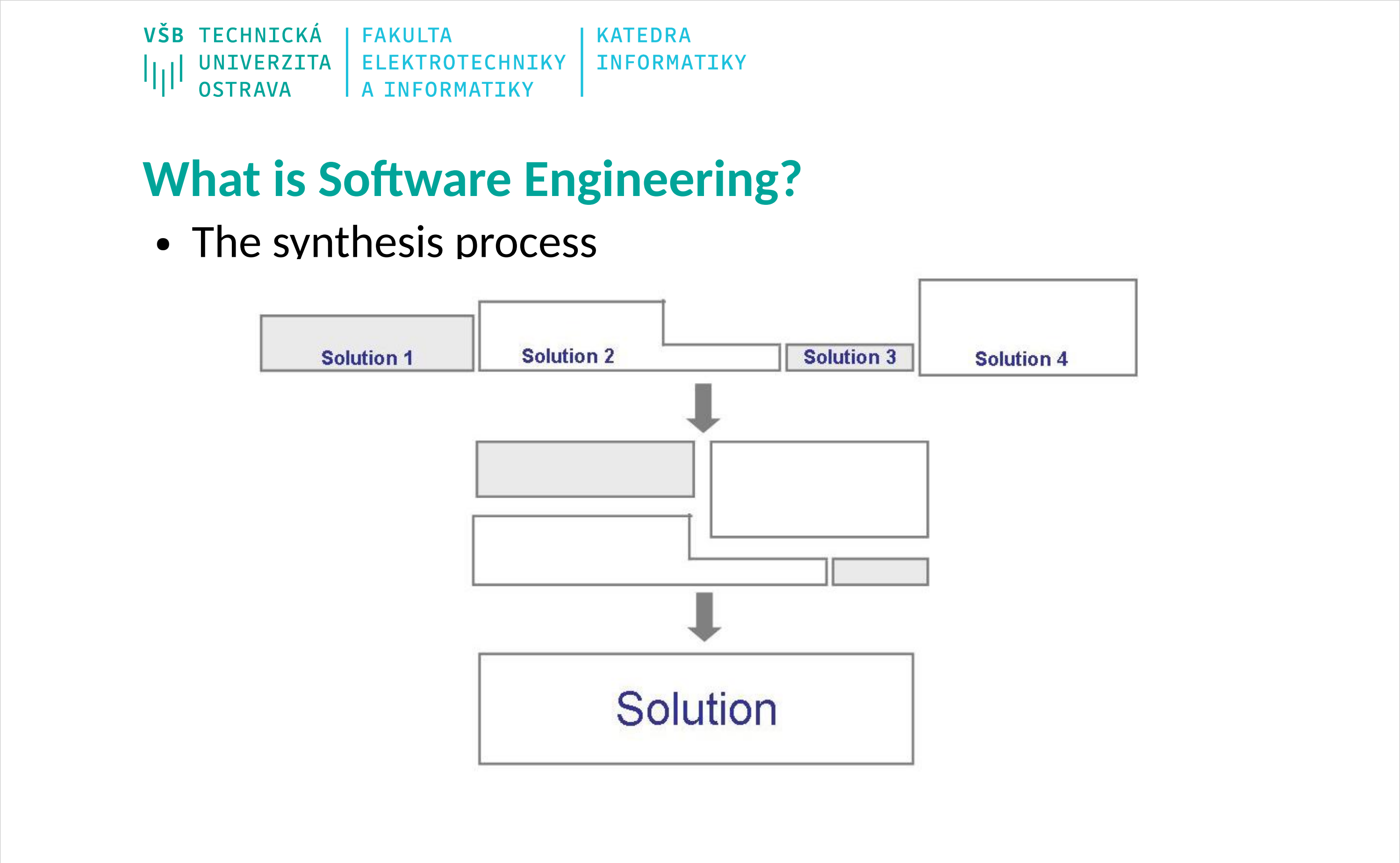
- The analysis process

Most problems are large and sometimes tricky to handle. So we must begin investigating the problem by analyzing it. Break it down into pieces that we can understand and try to deal with them. The larger problems can then be described as a collection of small subproblems.

VŠB TECHNICKÁ
UNIVERZITA
OSTRAVA

FAKULTA
ELEKTROTECHNIKY
A INFORMATIKY

KATEDRA
INFORMATIKY



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

What is Software Engineering?

- The synthesis process

Once you have analyzed the problem, you must find out the solution to the subproblems. Then the issue it to construct the solution of the large problem that consists of these small subproblems.

What is Software Engineering?

- Building a home myself.

Let’s imagine that you want to build a home.

Suppose that you get lost in the forest and you have a moment to build some type of a shelter before it gets dark, cold and possibly before it starts to rain. This will give you some small amount of time and you do not need much experience in building houses nor somebody else’s help. In case that your first attempt to pitch a roof fails, there is little damage (unless you let a heavy branch hit your head or break your bones…) and little effort wasted (money or time). You can easily rebuild the shelter in a few minutes. If you are not experienced at all, you will probably need more time and more attempts to finally build something like that. Anyway, you are able to do it yourself.

Next example – the house. Ok, suppose that some of us are able to build the house themselves. Sometimes you ask your friend to do some work for you. Definitely, this way you spend some money to buy building material and you need to plan ahead - what to build, what type of work you need do first, second, next, …
Of course, you can still be in charge of everything and do it yourself.

Next example – the block of flats. Now, you definitely need a plan, more money and more resources – people and materials to accomplish this.

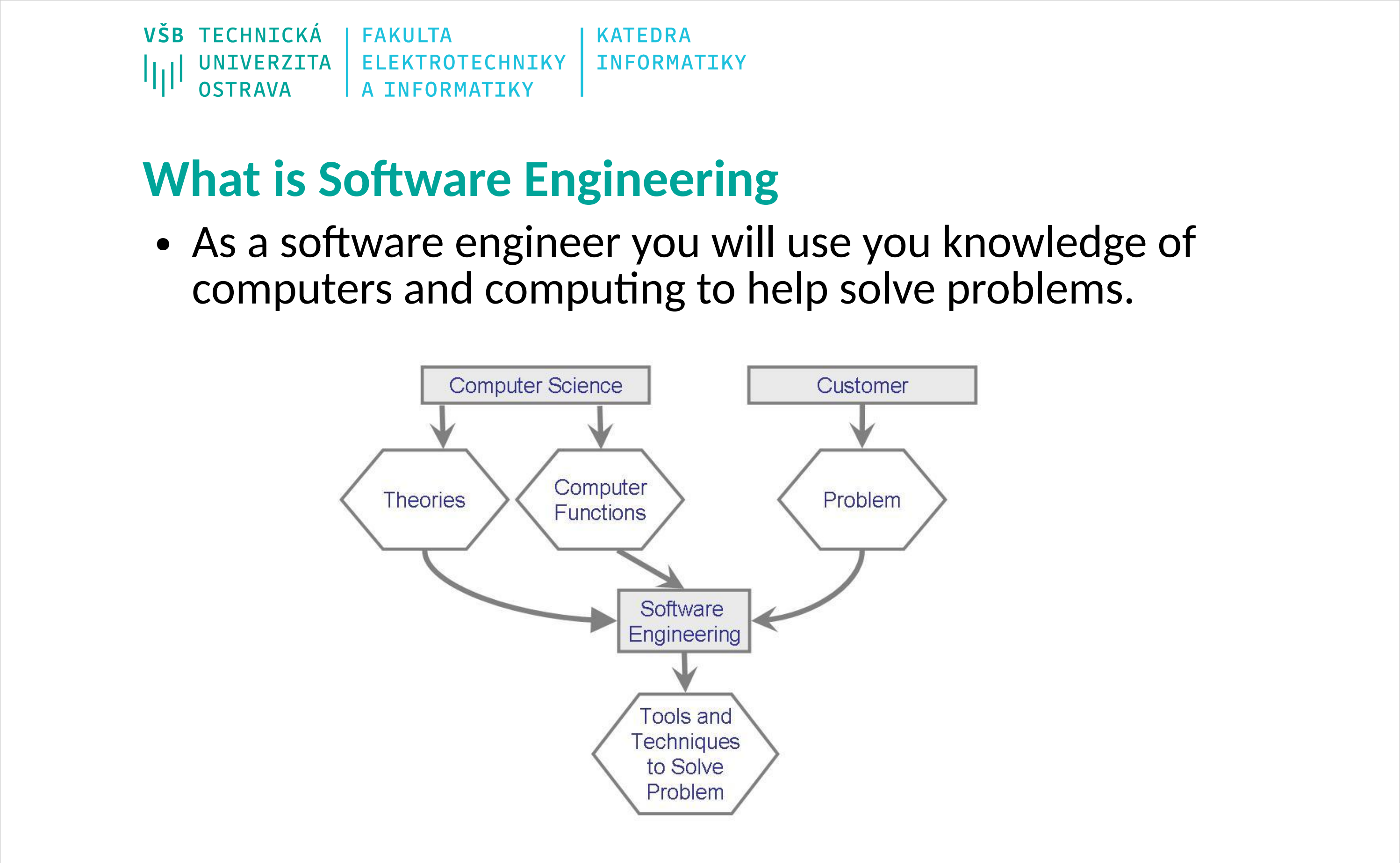
Next example – skyscraper. Can you do it yourself? I don’t think so. No individual can. You might be able to imagine or design its looks, its architecture, draw some basic plans but that’s it. You will definitely need some experienced people to do the rest for you. And I forgot one important thing: you will need a lot of money ;-).

All these examples are metaphors for building software.
The first one – shelter – you want to do your own simple counter, function, that can write some words on the screen etc. Very easy application.

Second one – the house. Let’s imagine for instance an application for your home library, a simple web page etc.

Third one – the block of flats. This could be for instance ERP system for small enterprises, an accounting system, a large web page etc.

Fourth one – the skyscraper. Plane computer software, ERP system for a big company, …



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

What is Software Engineering

- As a software engineer you will use you knowledge of computers and computing to help solve problems.

Computer Science
Customer
Theories
Computer Functions
Problem
Software Engineering
Tools and Techniques to Solve Problem

Ok, going back to the Software Engineering - what will YOU do?
As a software engineer you will use you knowledge of computers and computing to help solve problems.

What terms do we need to know to explain this?
Method: refers to a formal procedure; a formal “recipe” for accomplishing a goal that is typically independent of the tools used
Tool: an instrument or automated system for accomplishing something in a better way
Procedure: a combination of tools and techniques to produce a product
Paradigm: philosophy or approach for building a product (e.g., OO vs. structured approaches)
Computer science: focusing on computer hardware, compilers, operating systems, and programming languages
Software engineering: a discipline that uses computer and software technologies as a problem-solving tools

As you can see on the figure, and you are certainly aware of - a computer science is about theories, hardware, functions etc.
On the other hand there is a customer who wants to solve a problem. Customers might not be experienced (they are usually not), in computers and so it is your job to bring-the knowledge of computers and the problem together. Finding solution is your domain.

The joining point that tries to combine the problem resolution with computer knowledge is called Software Engineering and the people who do the job are generally called Software Engineers.

Definition of Software Engineering

- The IEEE Computer Society (The Institute of Electrical and Electronics Engineers) defines software engineering as:

1)The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

2)The study of approaches as in (1).

Definition of Software Engineering

Software engineering is an engineering discipline concerned with practical problems of developing large software systems.

Software engineering involves:

- Technical and non-technical issues
- Knowledge of specification, design and implementation techniques
- Human factors
- Software management

Software Engineering Code of Ethics and Professional Practice 1/2

1. PUBLIC - Software engineers shall act consistently with the public interest.

2. CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.

3. PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

4. JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.

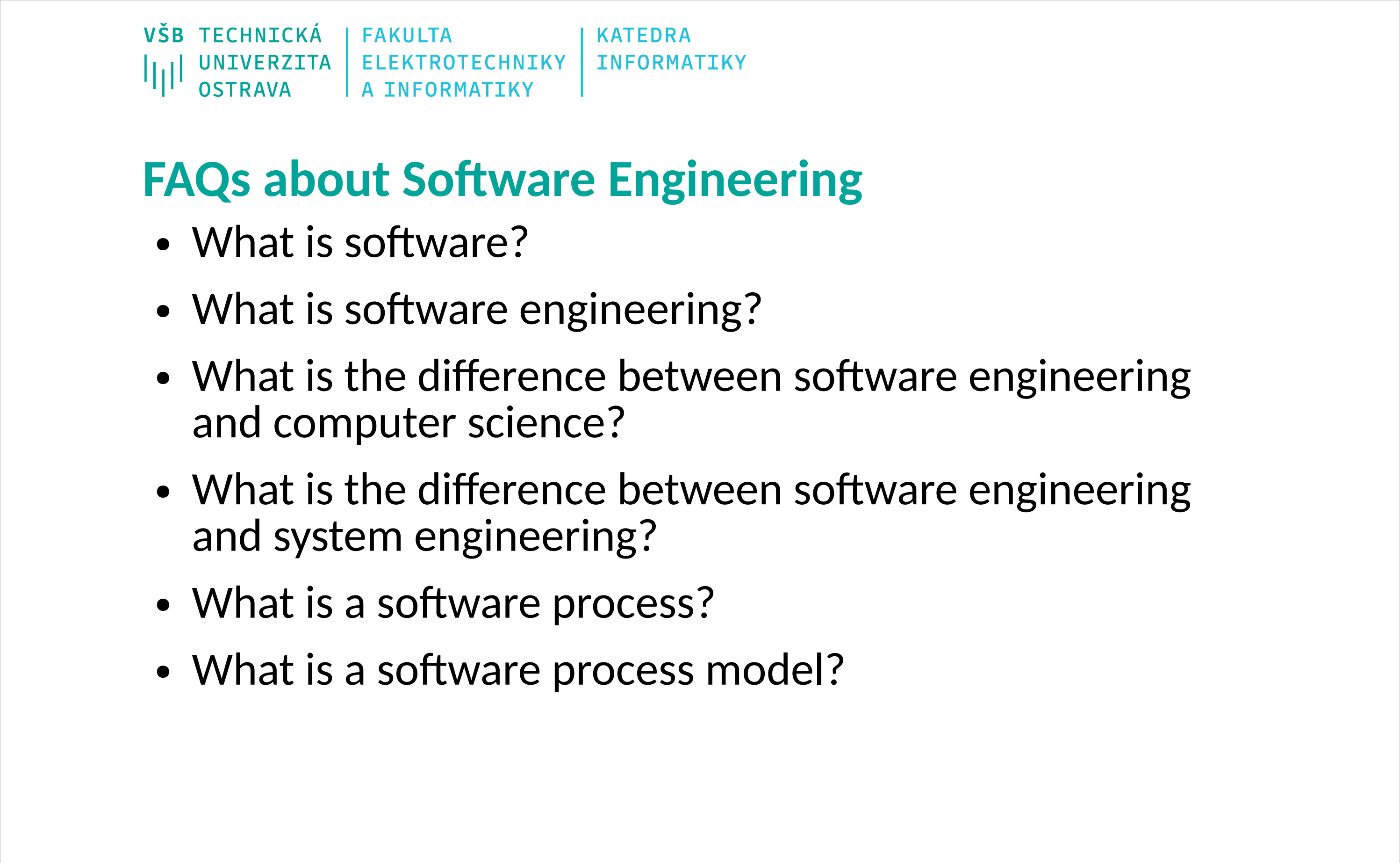
Software Engineering Code of Ethics and Professional Practice 2/2

5. MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.

6. PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

7. COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.

8. SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

FAQs about Software Engineering

- What is software?
- What is software engineering?
- What is the difference between software engineering and computer science?
- What is the difference between software engineering and system engineering?
- What is a software process?
- What is a software process model?

Let’s try to answer some FAQs about software engineering. Usually, somebody not experienced in the software engineering asks these basic questions:

What is software?

What is software engineering?

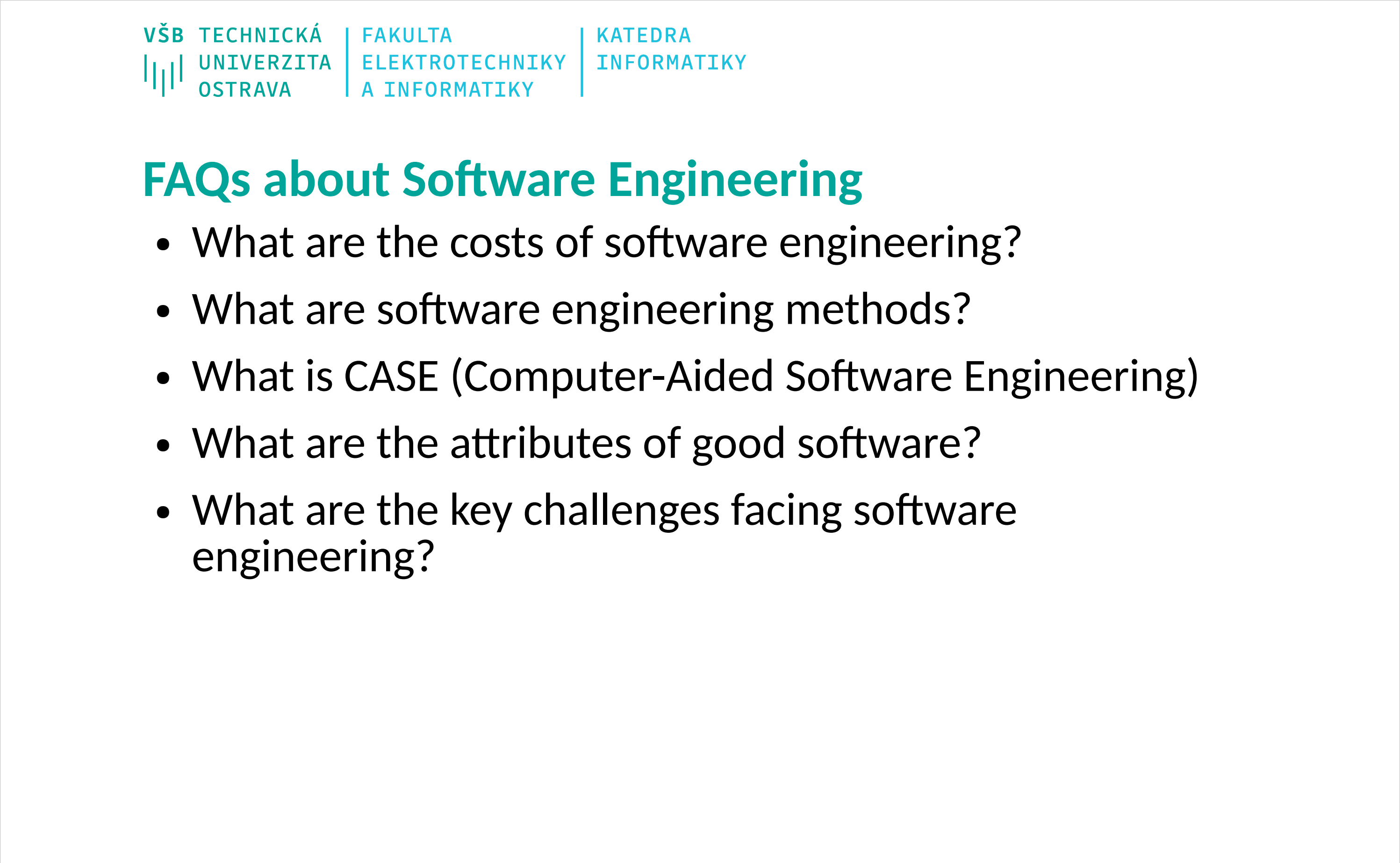
What is the difference between software engineering and computer science?

What is the difference between software engineering and system engineering?

What is a software process?

What is a software process model?

And of course, we will learn how to answer them in more detail later in this course.



FAQs about Software Engineering

- What are the costs of software engineering?
- What are software engineering methods?
- What is CASE (Computer-Aided Software Engineering)
- What are the attributes of good software?
- What are the key challenges facing software engineering?

And other questions about more specific things/questions might come to your mind as well.

Let’s try to answer them briefly in this introduction lesson to give you some thoughts and ideas you may find useful in your future careers.

(As) You know, software and computers are not only about playing games. Somebody has to look after those do not know anything about computers and make their jobs and entertainment easier by providing support.

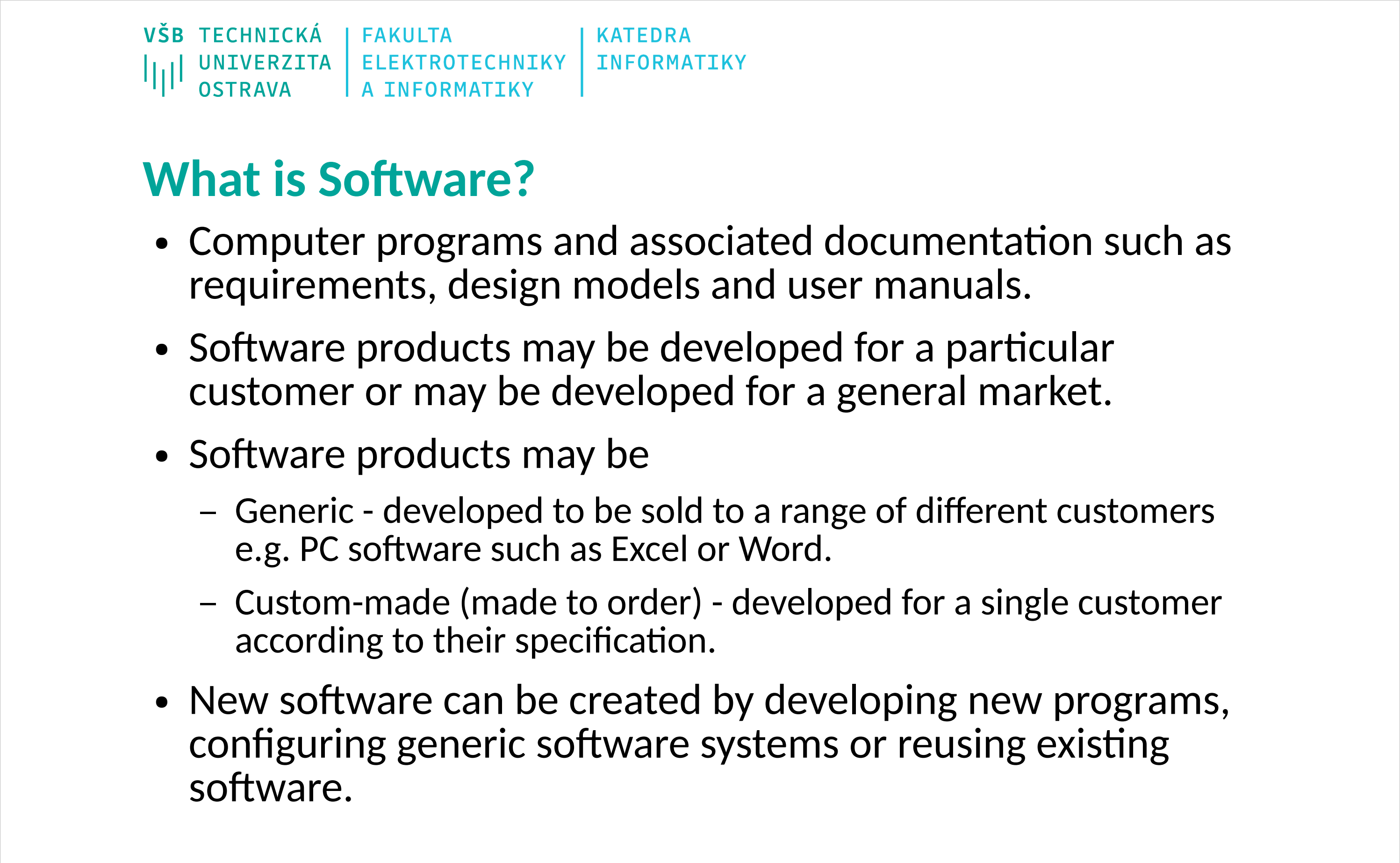
I suppose that you who are sitting here and want to know something about Software Engineering are also those who will challenge the problems of your customers and will lead them to the successful solutions.

Or maybe you just had an idea of developing a similar computer game you used to play in the past because you liked it so much.

Maybe you like computers and just want to help spread out computer-aided solutions and encourage people to use them.

Who knows? It might have something to do with your motivation and your future job.

Let’s continue and answer the questions.



What is Software?

- Computer programs and associated documentation such as requirements, design models and user manuals.
- Software products may be developed for a particular customer or may be developed for a general market.
- Software products may be
  - Generic - developed to be sold to a range of different customers e.g. PC software such as Excel or Word.
  - Custom-made (made to order) - developed for a single customer according to their specification.
- New software can be created by developing new programs, configuring generic software systems or reusing existing software.

Computer programs and associated documentation such as requirements, design models and user manuals.

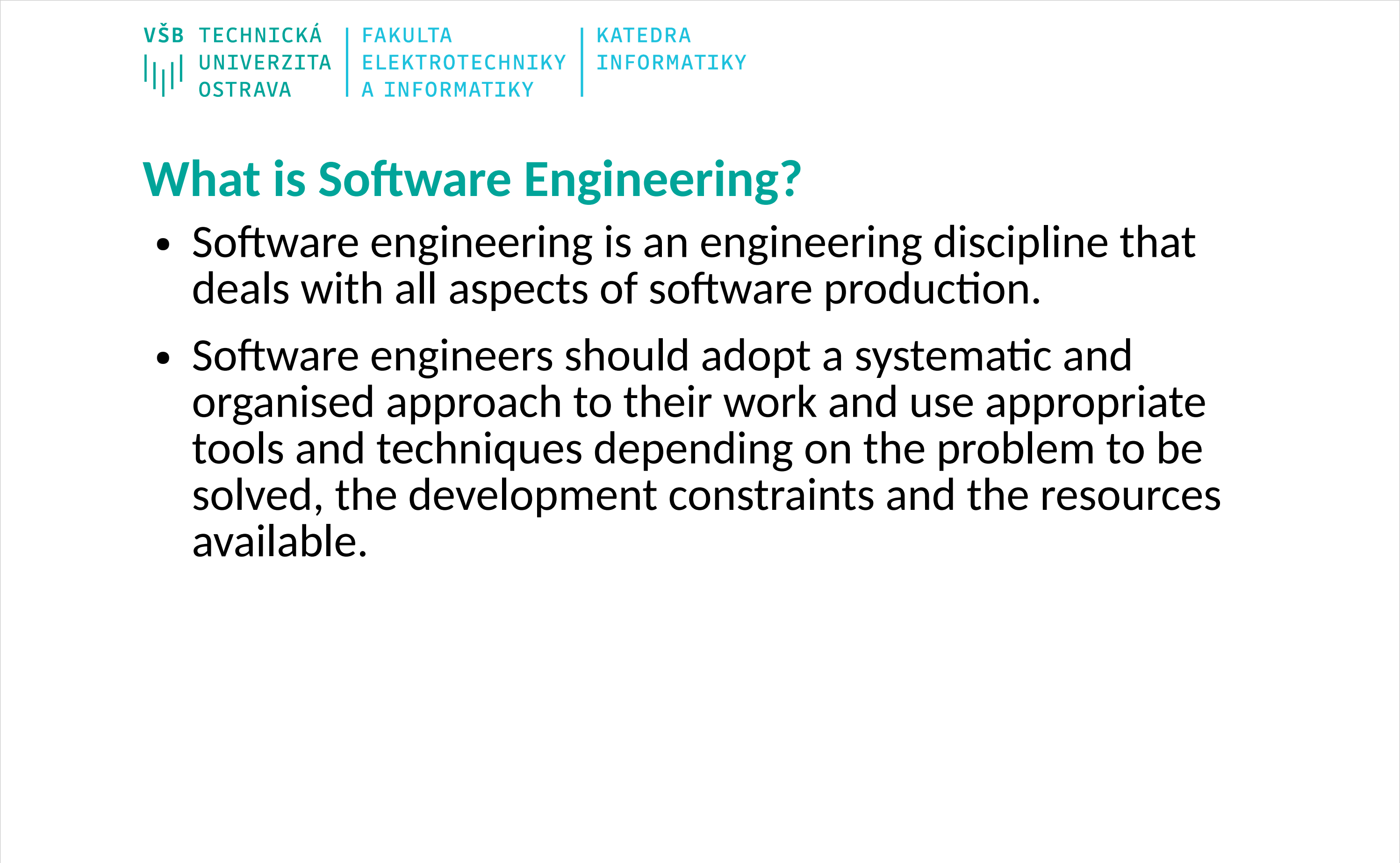
Software products may be developed for a particular customer or may be developed for a general market.

Software products may be

Generic - developed to be sold to a range of different customers e.g. PC software such as Excel or Word.

Custom-made (made to order) - developed for a single customer according to their specification.

New software can be created by developing new programs, configuring generic software systems or reusing existing software.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

What is Software Engineering?

- Software engineering is an engineering discipline that deals with all aspects of software production.
- Software engineers should adopt a systematic and organised approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available.

We have already thoroughly discussed the general answer to this question. That a software engineering is solving problems (with the help of) computers, but are there any specific tasks to do?

Software engineering is an engineering discipline that is concerned with all aspects of software production.

Software engineers should adopt a systematic and organised approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available.

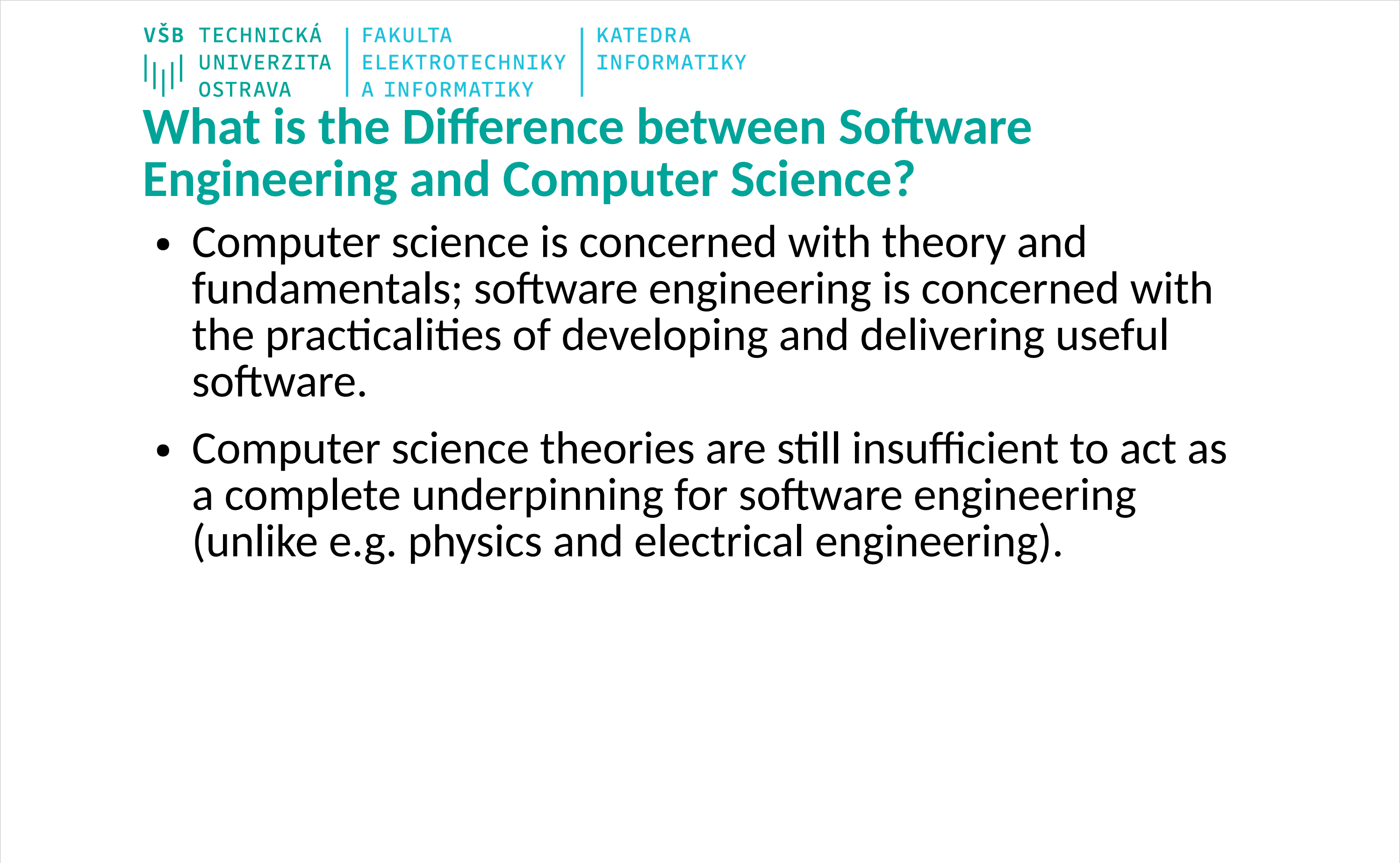
Analyze the problem, break it down into subproblems, create an architecture of the system to be built, compose the system from scratch. Deliver it to customer.

What is Software Engineering?

The IEEE Computer Society defines software engineering as:

(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

(2) The study of approaches as in (1).



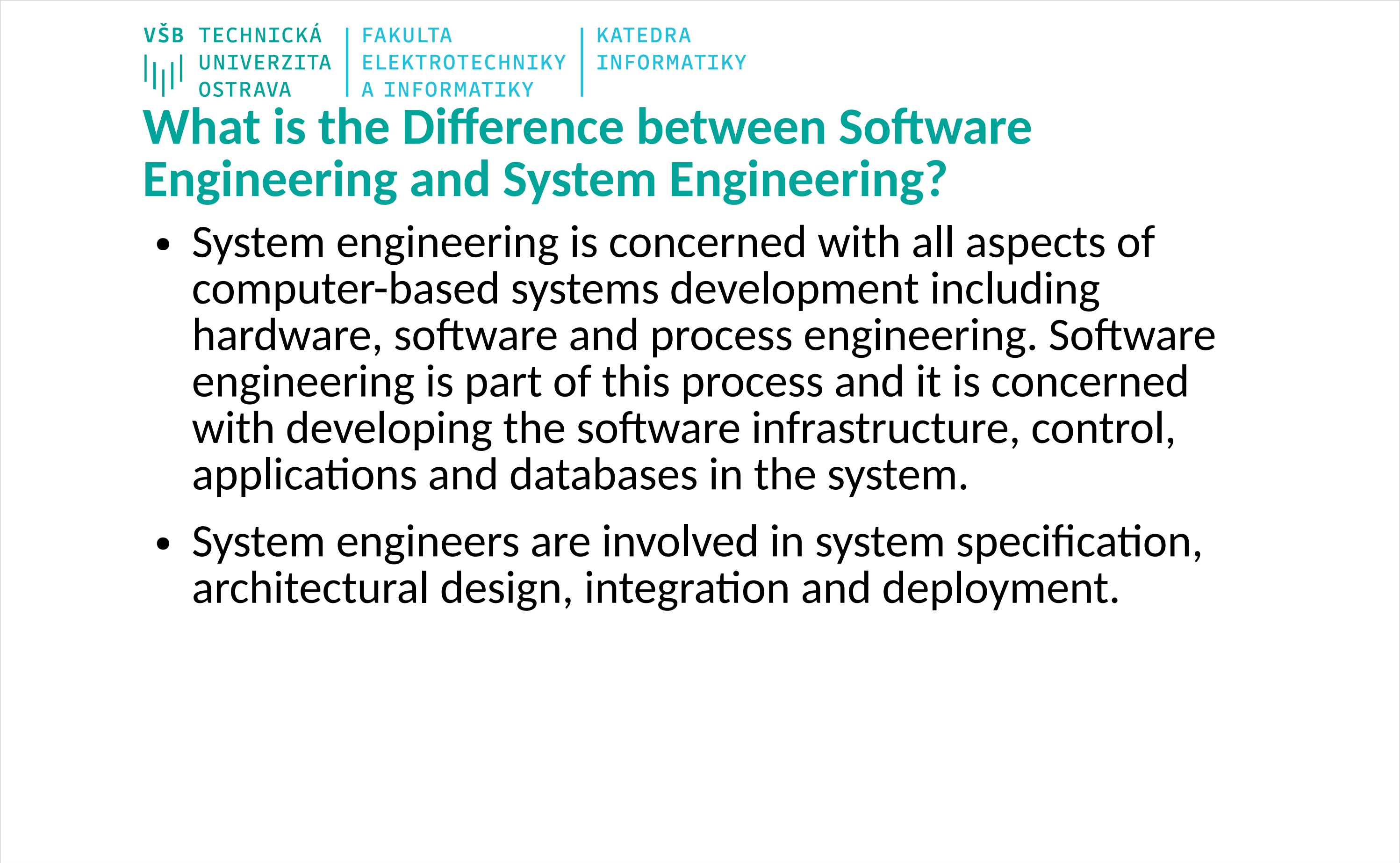
What is the Difference between Software Engineering and Computer Science?

- Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
- Computer science theories are still insufficient to act as a complete underpinning for software engineering (unlike e.g. physics and electrical engineering).

You should know the answer to this. Do you memorize it for long term? If not, use the opportunity as a second chance. You know what Software Engineering is – an engineering discipline that helps (to) solve problems of customers from different domains/areas.

Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.

Computer science theories are still insufficient to act as a complete underpinning for software engineering (unlike e.g. physics and electrical engineering).



What is the Difference between Software Engineering and System Engineering?

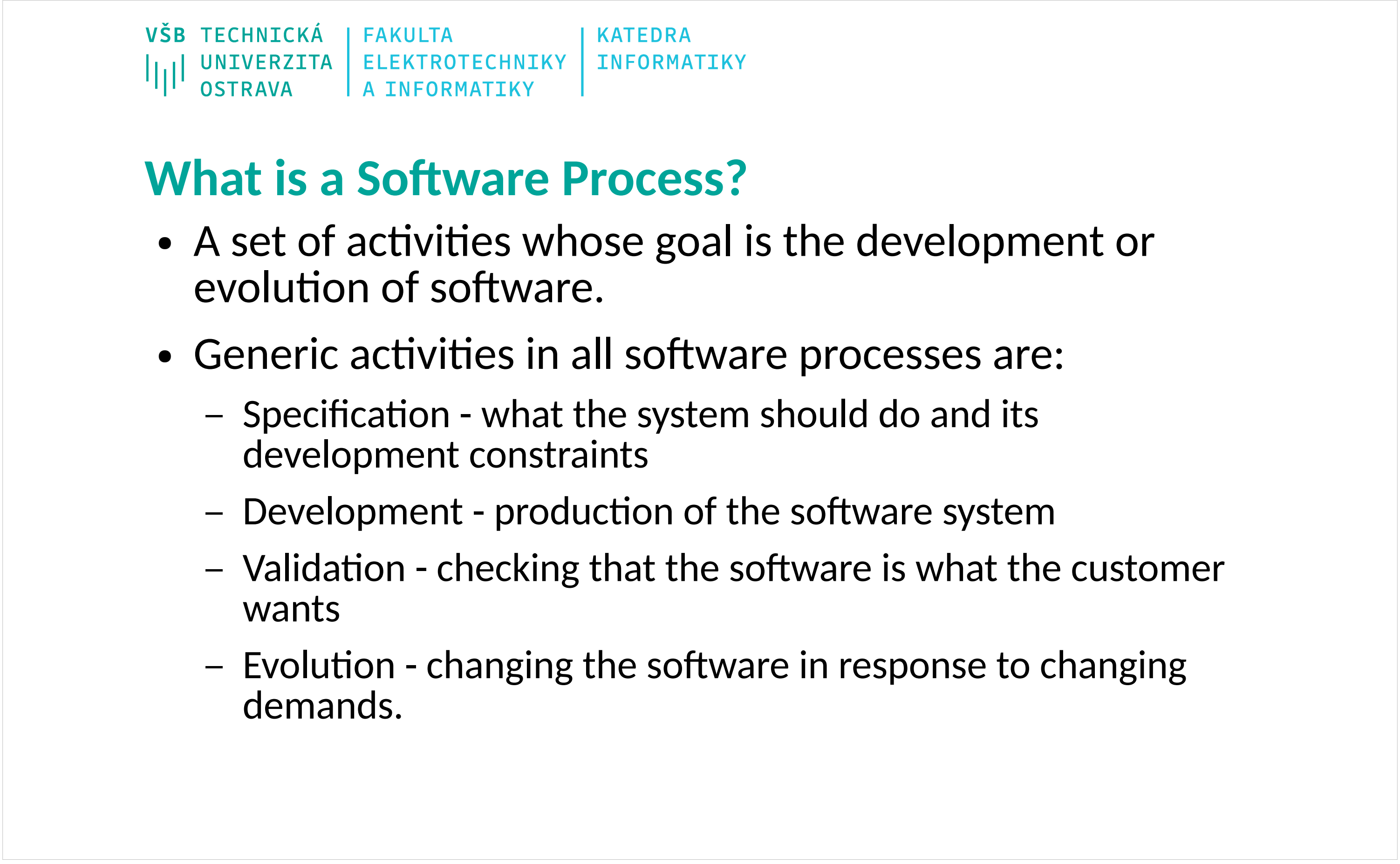
- System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this process and it is concerned with developing the software infrastructure, control, applications and databases in the system.
- System engineers are involved in system specification, architectural design, integration and deployment.

Have you ever heard of that term? System engineering? Yes or no? Ok there is the explanation of the difference:

System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this process concerned with developing the software infrastructure, control, applications and databases in the system.
System engineers are involved in system specification, architectural design, integration and deployment.

So, if I would like to let you solve my problem – develop a system for something like a ship. I would not ask you. I would ask a ship-building factory to do that. Anyway, they might then ask you to develop a software for the computers on that ship. Somebody has to deliver a control unit that definitely has some software in it etc.

Ok the boundary between the system and software engineering might not be completely strict. It always depends on the proposed building solution. Sometimes System Engineering might be only the small part of the complete delivery – like in the case of, say accounting software - where you have to deliver PCs and network staff as well, etc.



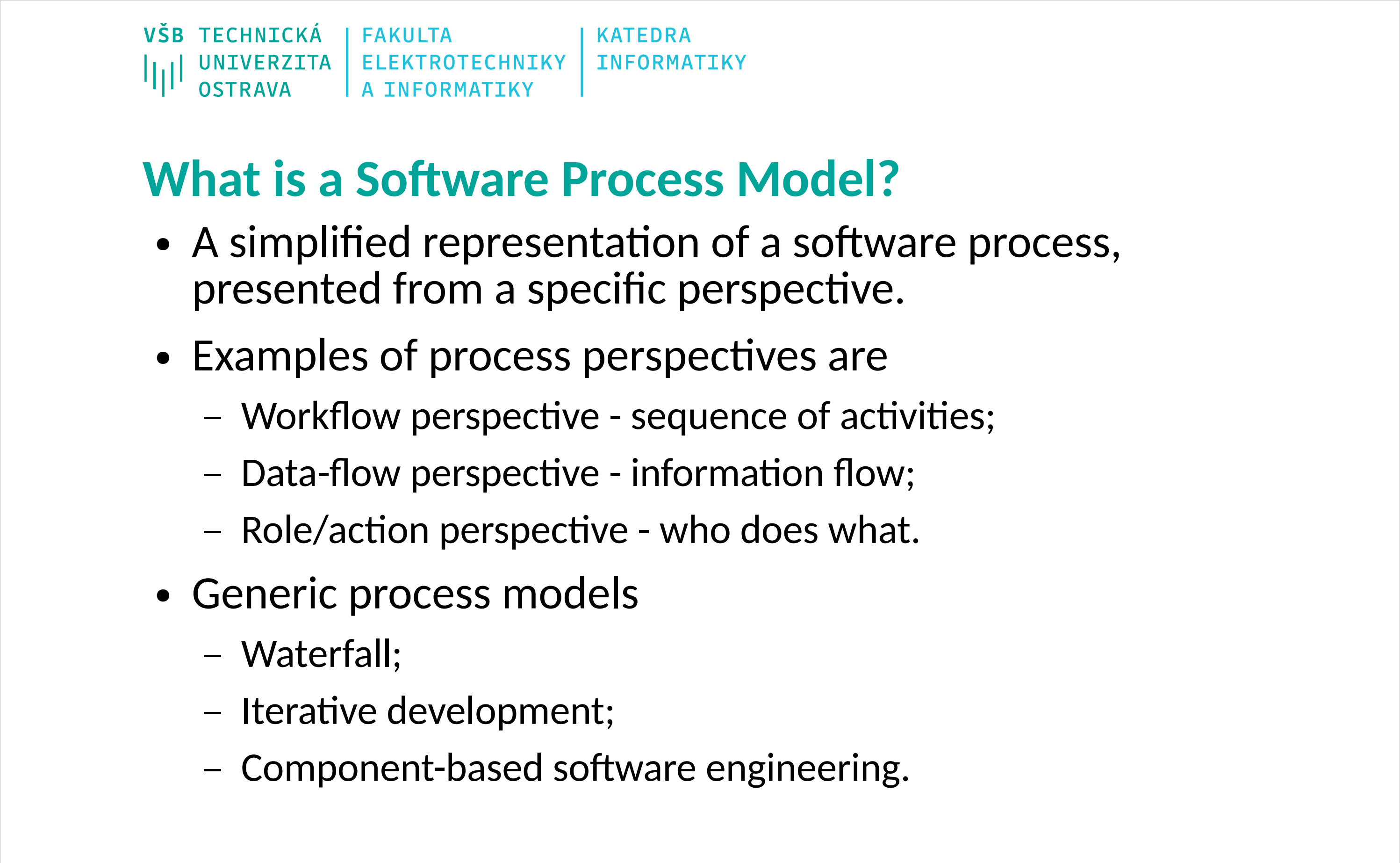
What is a Software Process?

- A set of activities whose goal is the development or evolution of software.
- Generic activities in all software processes are:
  - Specification - what the system should do and its development constraints
  - Development - production of the software system
  - Validation - checking that the software is what the customer wants
  - Evolution - changing the software in response to changing demands.

Do you remember that house building example? When we are building a house, do we have to start at some point and end at some point? Why? Because we have to build the house from the basement to the roof (from the ground up). It is obvious that there are tasks that have to be done only after some other tasks are finished. Like, for example, constructing a roof without walls is no use. Of course, sometimes you can do that, but these are rather exceptions.

The same thing works (goes) for the construction of software. It is proven that before you start to build software it is good to ask your customer questions about the software functionality. Then think about the architecture, build components, put them together and test them, deliver the system to the customer etc…

What is a software process?
A set of activities whose goal is the development or evolution of software.
Generic activities in all software processes are:
Specification - what the system should do and its development constraints
Development - production of the software system
Validation - checking that the software is what the customer wants
Evolution - changing the software in response to changing demands.



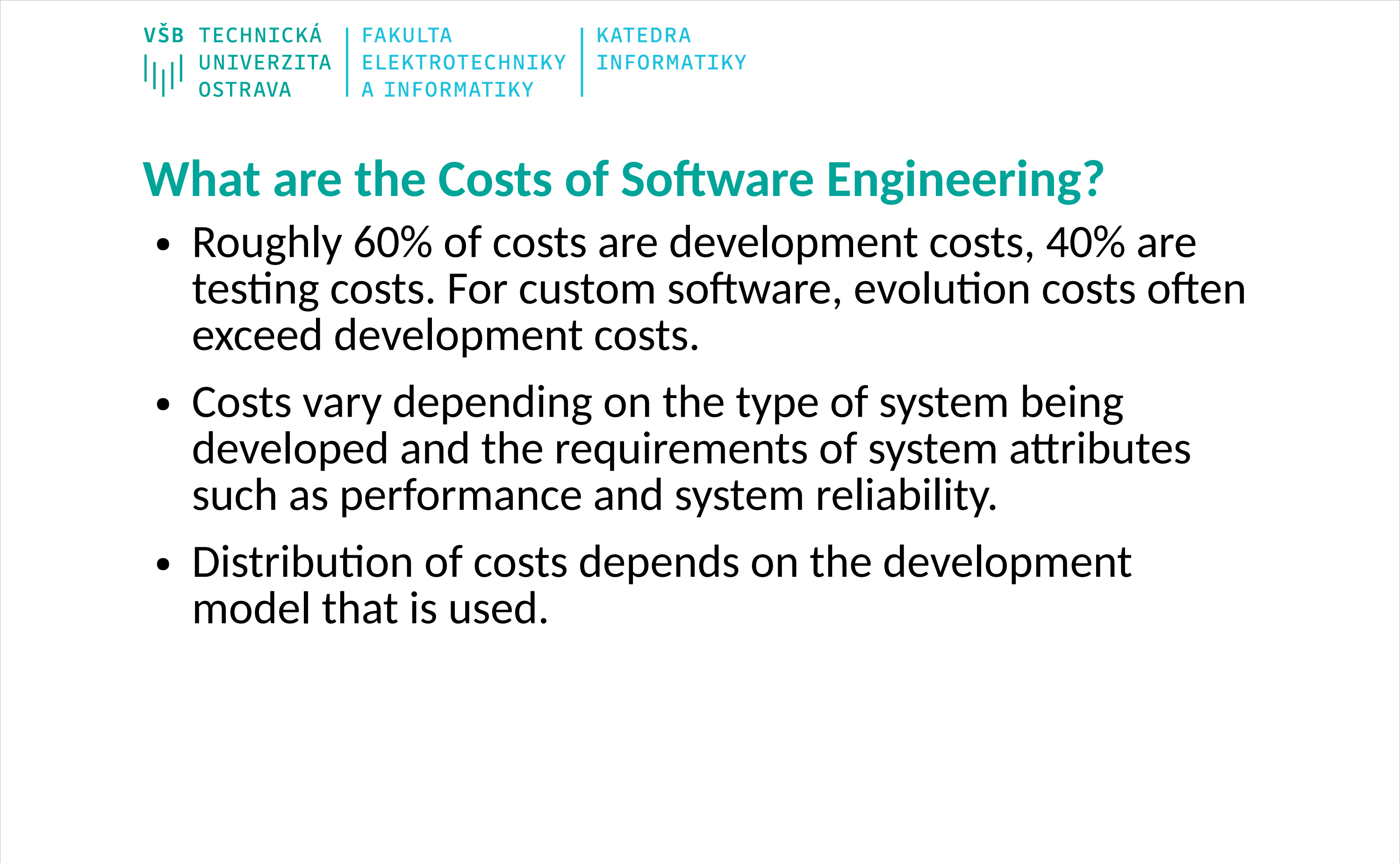
VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

What is a Software Process Model?

- A simplified representation of a software process, presented from a specific perspective.
- Examples of process perspectives are
  - Workflow perspective - sequence of activities;
  - Data-flow perspective - information flow;
  - Role/action perspective - who does what.
- Generic process models
  - Waterfall;
  - Iterative development;
  - Component-based software engineering.

Since the coordination of the work is complicated, sometimes it is useful to use blueprints to control the project. These blueprints model the process.

What is the software process model?
A simplified representation of a software process, presented from a specific perspective.
Examples of process perspectives are
Workflow perspective - sequence of activities;
Data-flow perspective - information flow;
Role/action perspective - who does what.
Generic process models
Waterfall;
Iterative development;
Component-based software engineering.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

What are the Costs of Software Engineering?

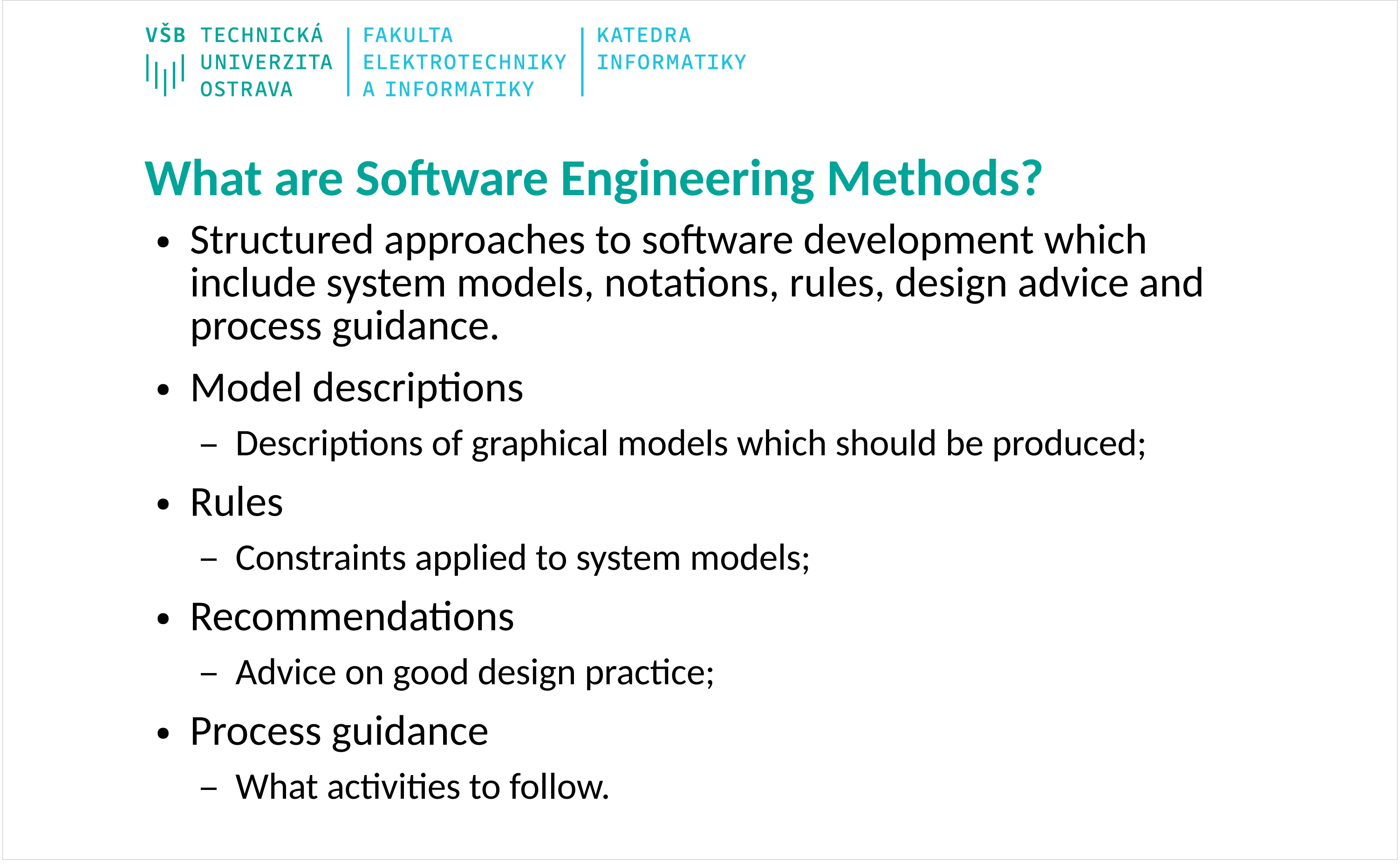
- Roughly 60% of costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
- Costs vary depending on the type of system being developed and the requirements of system attributes such as performance and system reliability.
- Distribution of costs depends on the development model that is used.

Money first, of course it is always about money. What is the cost?

Roughly 60% of costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.

Costs vary depending on the type of system being developed and the requirements of system attributes such as performance and system reliability.

Distribution of costs depends on the development model that is used.



What are Software Engineering Methods?

- Structured approaches to software development which include system models, notations, rules, design advice and process guidance.
- Model descriptions
  - Descriptions of graphical models which should be produced;
- Rules
  - Constraints applied to system models;
- Recommendations
  - Advice on good design practice;
- Process guidance
  - What activities to follow.

What are software engineering methods?
We have to do the task and activities somehow. How to do that? Use some method. What is a method?

Method is:

Structured approaches to software development which include system models, notations, rules, design advice and process guidance.

Model descriptions

Descriptions of graphical models which should be produced;

Rules

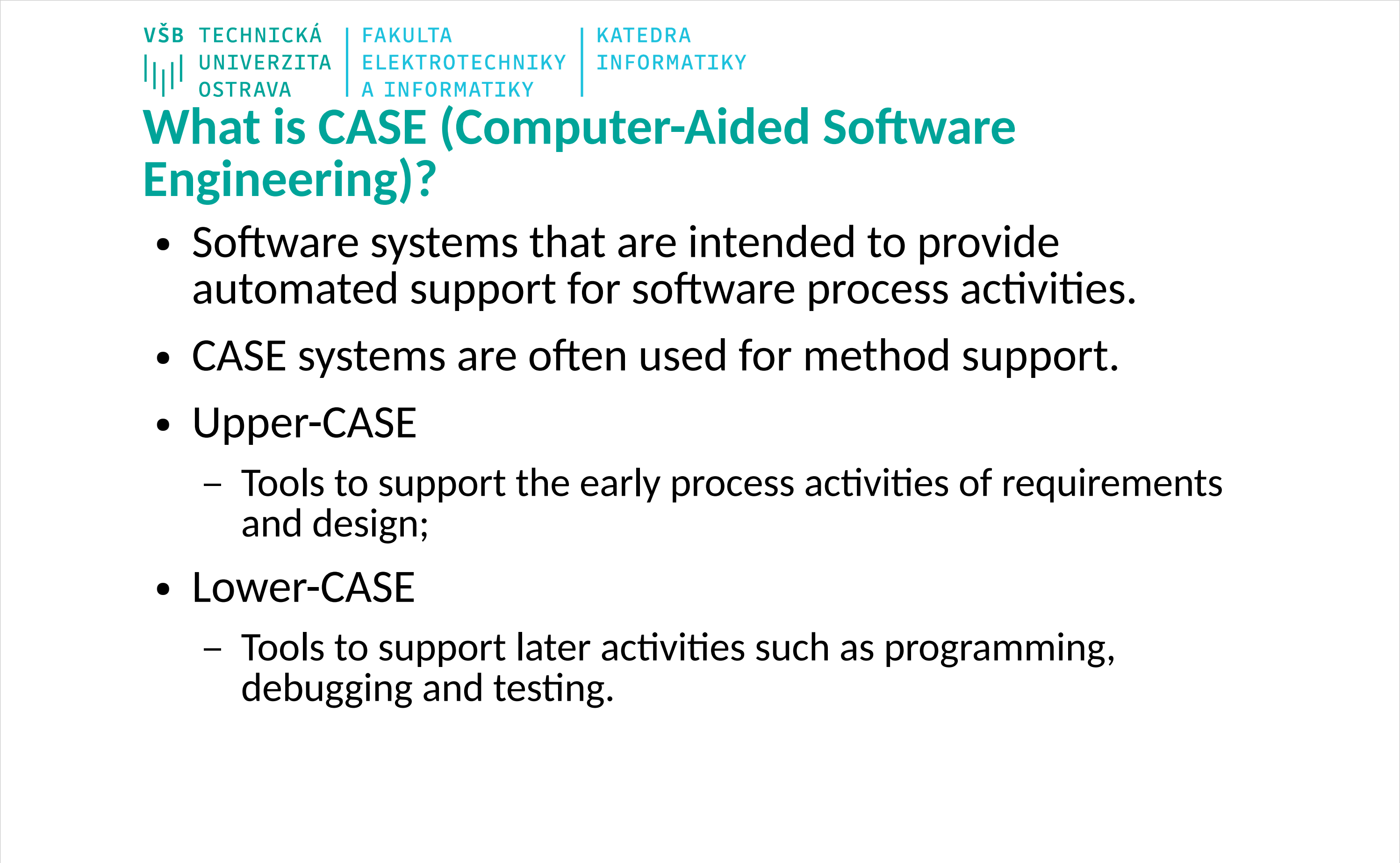
Constraints applied to system models;

Recommendations

Advice on good design practice;

Process guidance

What activities to follow.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

What is CASE (Computer-Aided Software Engineering)?

- Software systems that are intended to provide automated support for software process activities.
- CASE systems are often used for method support.
- Upper-CASE
  - Tools to support the early process activities of requirements and design;
- Lower-CASE
  - Tools to support later activities such as programming, debugging and testing.

What is CASE (Computer-Aided Software Engineering)?

You know that we have a computer. Is that right? And you know that software engineering is solving problems using computers. Is that right? Ok so why we should do everything in our engineering field manually. We should be the first who use computers to do the thinking for us. So

Software systems that are intended to provide automated support for software process activities.

CASE systems are often used for method support.

Upper-CASE

Tools to support the early process activities of requirements and design;

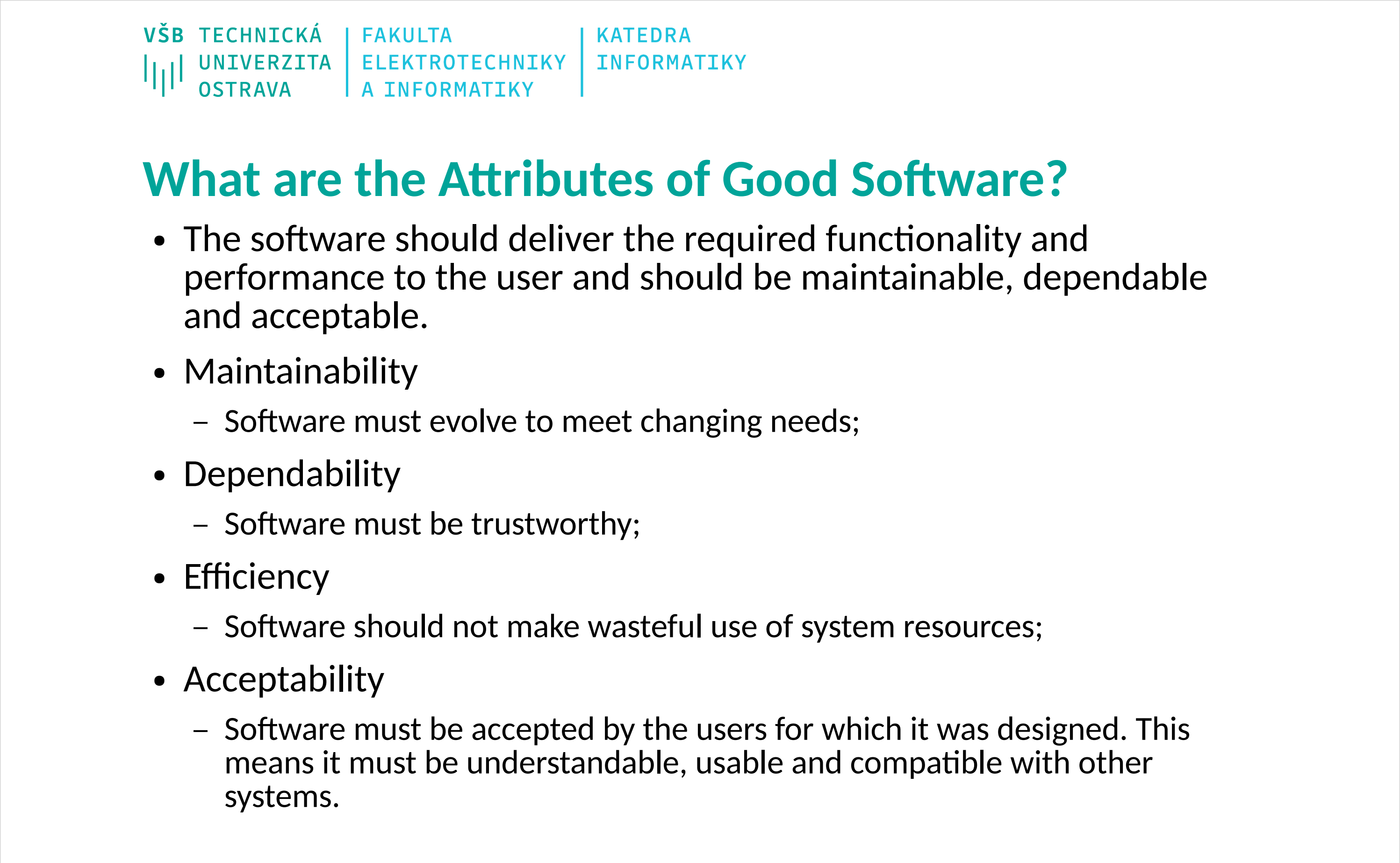
Like requirement engineering tools – requisite pro, MS Word, …

Design tools that support graphical view of the problem – UML based tools, …

Lower-CASE

Tools to support later activities such as programming, debugging and testing.

What about programming enviroment – MS Studio, Eclipse, NetBeans? Testing tools, …



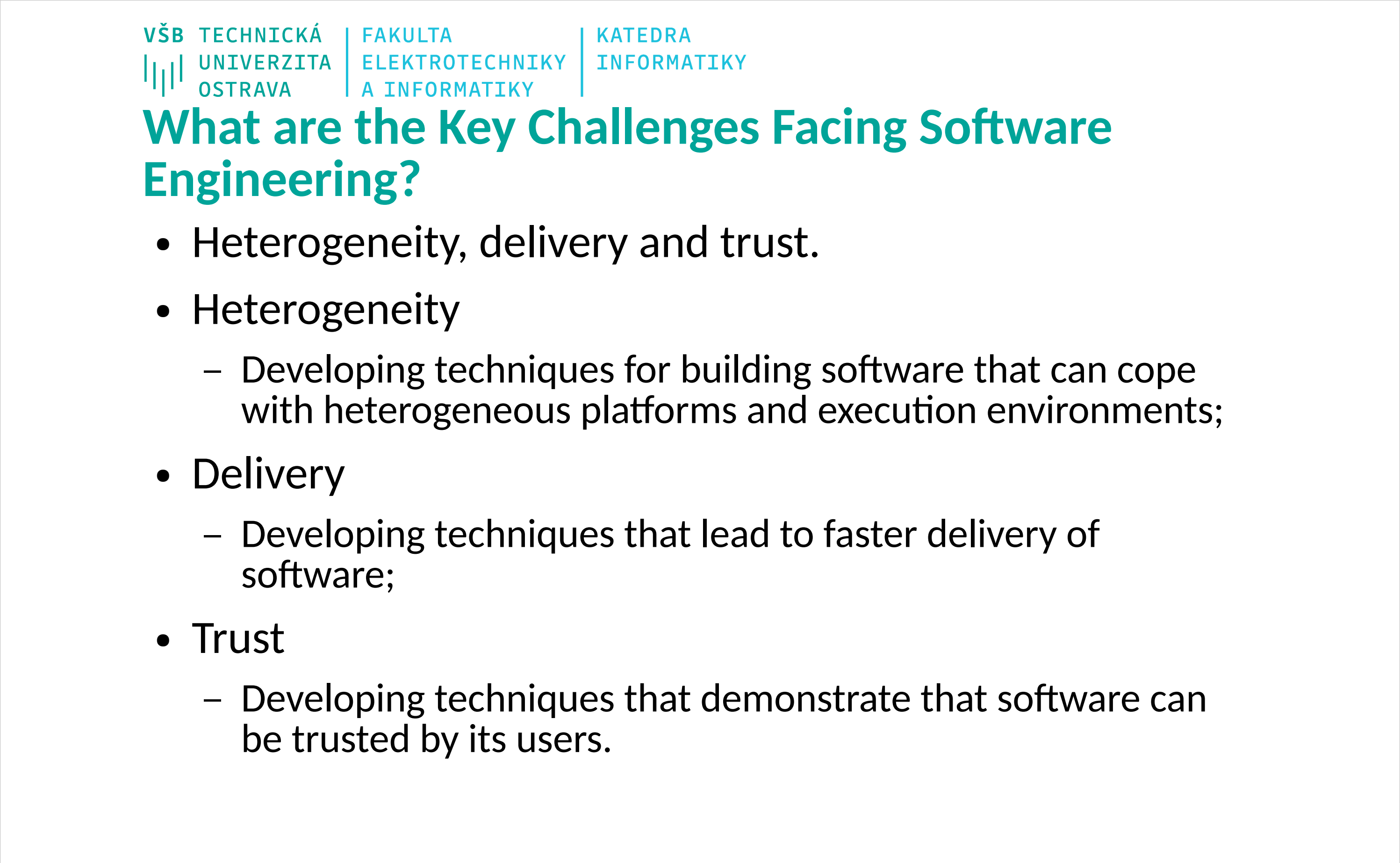
VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

What are the Attributes of Good Software?

- The software should deliver the required functionality and performance to the user and should be maintainable, dependable and acceptable.
- Maintainability
  - Software must evolve to meet changing needs;
- Dependability
  - Software must be trustworthy;
- Efficiency
  - Software should not make wasteful use of system resources;
- Acceptability
  - Software must be accepted by the users for which it was designed. This means it must be understandable, usable and compatible with other systems.

Do you want to build good or bad software?
Of course that you want to build good software..., but
What does it mean to build a good software, how can I recognize good software?
Let’s say that good software:

should deliver the required functionality and performance to the user and should be maintainable, dependable and acceptable.
Maintainability
Software must evolve to meet changing needs;
Dependability
Software must be trustworthy;
Efficiency
Software should not make wasteful use of system resources;
Acceptability
Software must accepted by the users for which it was designed. This means it must be understandable, usable and compatible with other systems.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

What are the Key Challenges Facing Software Engineering?

- Heterogeneity, delivery and trust.
- Heterogeneity
  - Developing techniques for building software that can cope with heterogeneous platforms and execution environments;
- Delivery
  - Developing techniques that lead to faster delivery of software;
- Trust
  - Developing techniques that demonstrate that software can be trusted by its users.

Now, let us think about the software engineering field.

What are the goals that we want to achieve? To build good software easily, quickly.

So the main challenges we face are for example:

Heterogeneity, delivery and trust.

Heterogeneity

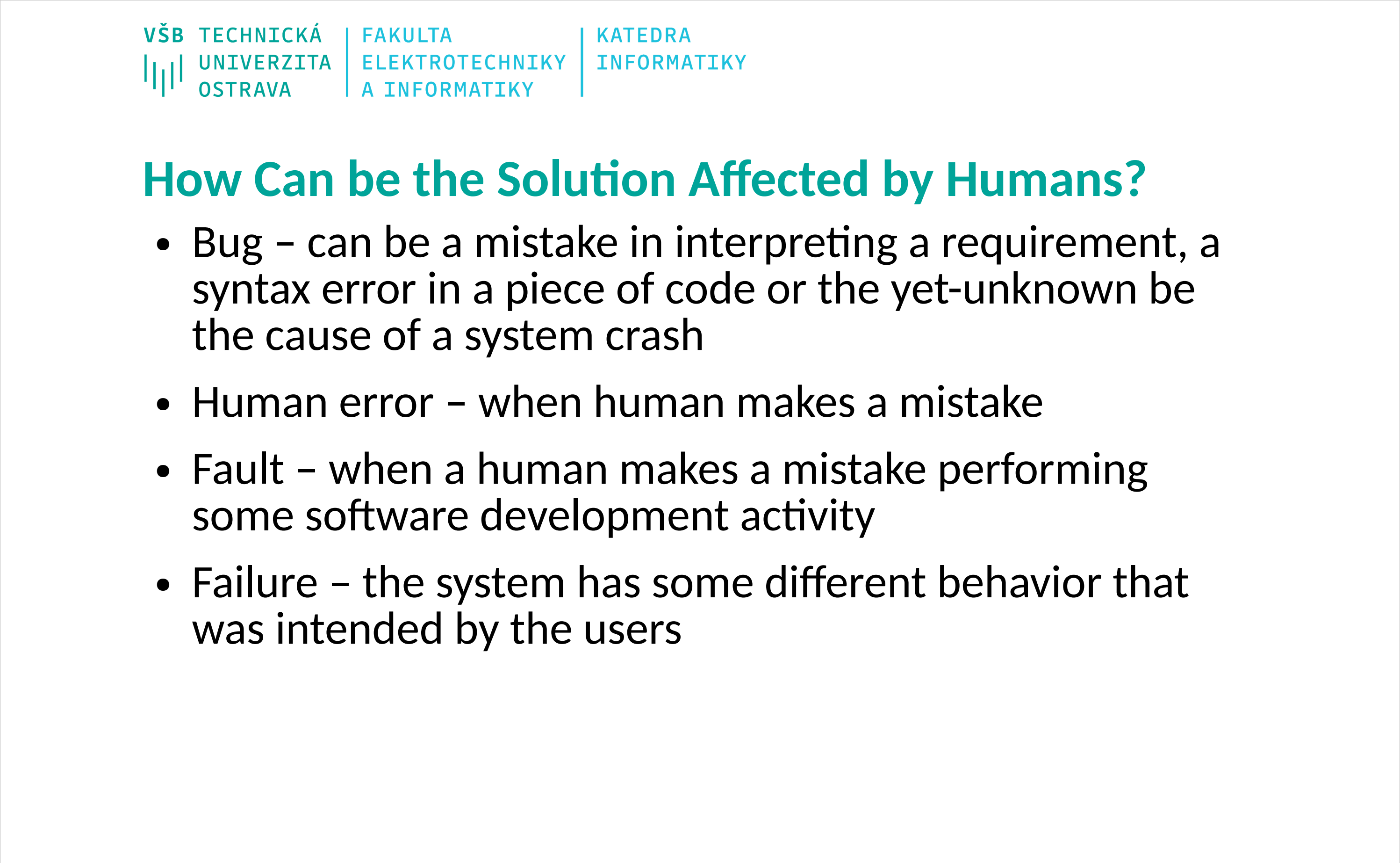
Developing techniques for building software that can cope with heterogeneous platforms and execution environments;

Delivery

Developing techniques that lead to faster delivery of software;

Trust

Developing techniques that demonstrate that software can be trusted by its users.



How Can be the Solution Affected by Humans?

- Bug – can be a mistake in interpreting a requirement, a syntax error in a piece of code or the yet-unknown be the cause of a system crash
- Human error – when human makes a mistake
- Fault – when a human makes a mistake performing some software development activity
- Failure – the system has some different behavior that was intended by the users

Thus, the fault is the inside view of the system – seen by the developers

Failure is an outside view of the system – seen by the users

SWE Knowledge Areas

- Software requirements
- Software design
- Software construction
- Software testing
- Software maintenance
- Software configuration management
- Software engineering management
- Software engineering process
- Software engineering tools and methods
- Software quality

What are the SWE Knowledge Areas?

To become a successful software engineer you should master or at least know most of ~~this~~ these areas.:

Software requirements – the discipline consist of definitions of the future software

Software design – the discipline that designs the future software

Software construction – the discipline that deals with the programming

Software testing – the discipline that covers the software testing

Software maintenance – the discipline that is the maintenance of the software that is already deployed

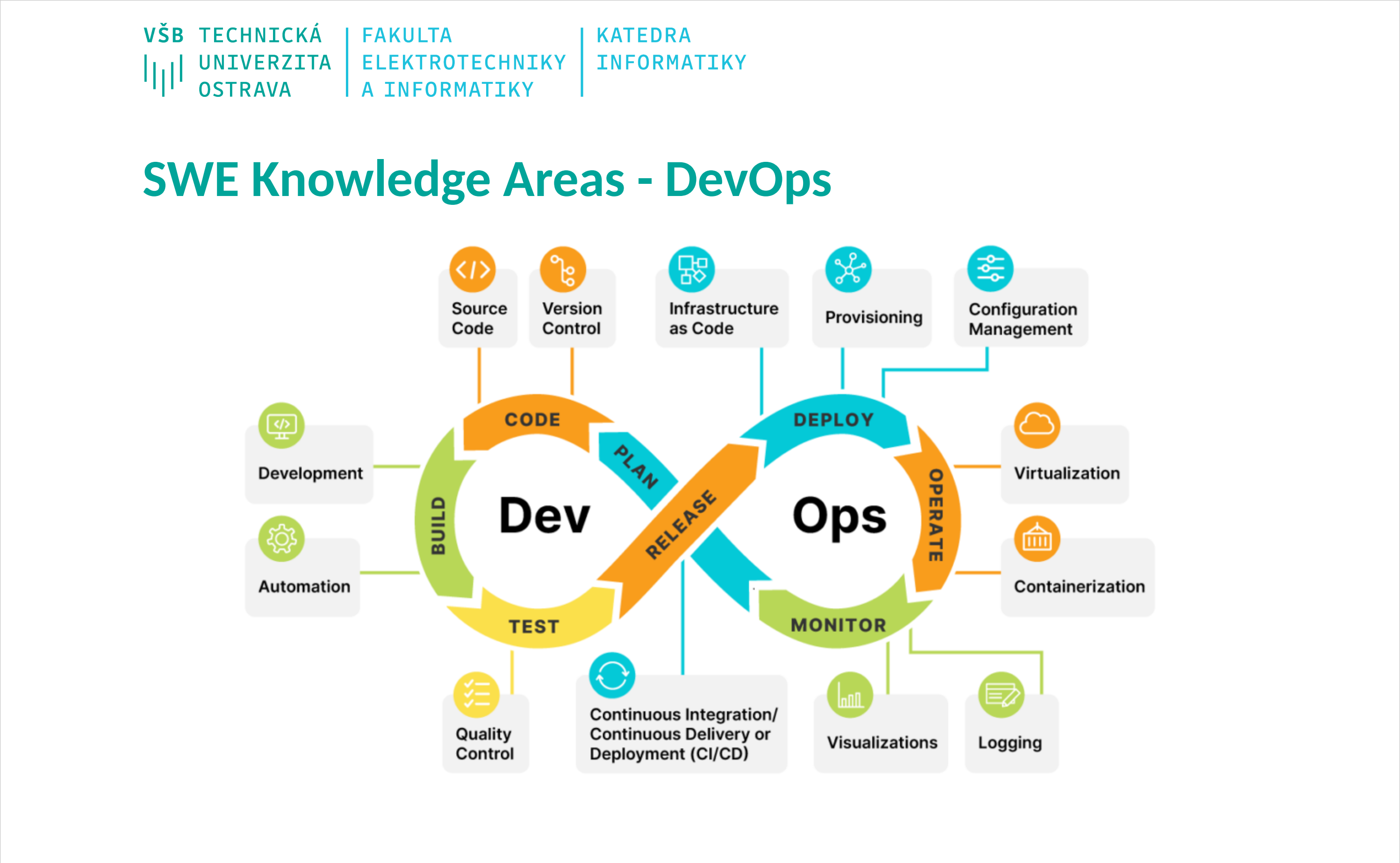
Software configuration management – what are the configurations that are running

Software engineering management – the discipline that covers the management of the development

Software engineering process – the discipline that covers processes, their creation, definition, description etc.

Software engineering tools and methods - the discipline that covers tools and development methods

Software quality – the discipline that deals with the quality of the development



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

SWE Knowledge Areas - DevOps

What are the SWE Knowledge Areas?

To become a successful software engineer you should master or at least know most of ~~this~~ these areas.:

Software requirements – the discipline consist of definitions of the future software

Software design – the discipline that designs the future software

Software construction – the discipline that deals with the programming

Software testing – the discipline that covers the software testing

Software maintenance – the discipline that is the maintenance of the software that is already deployed

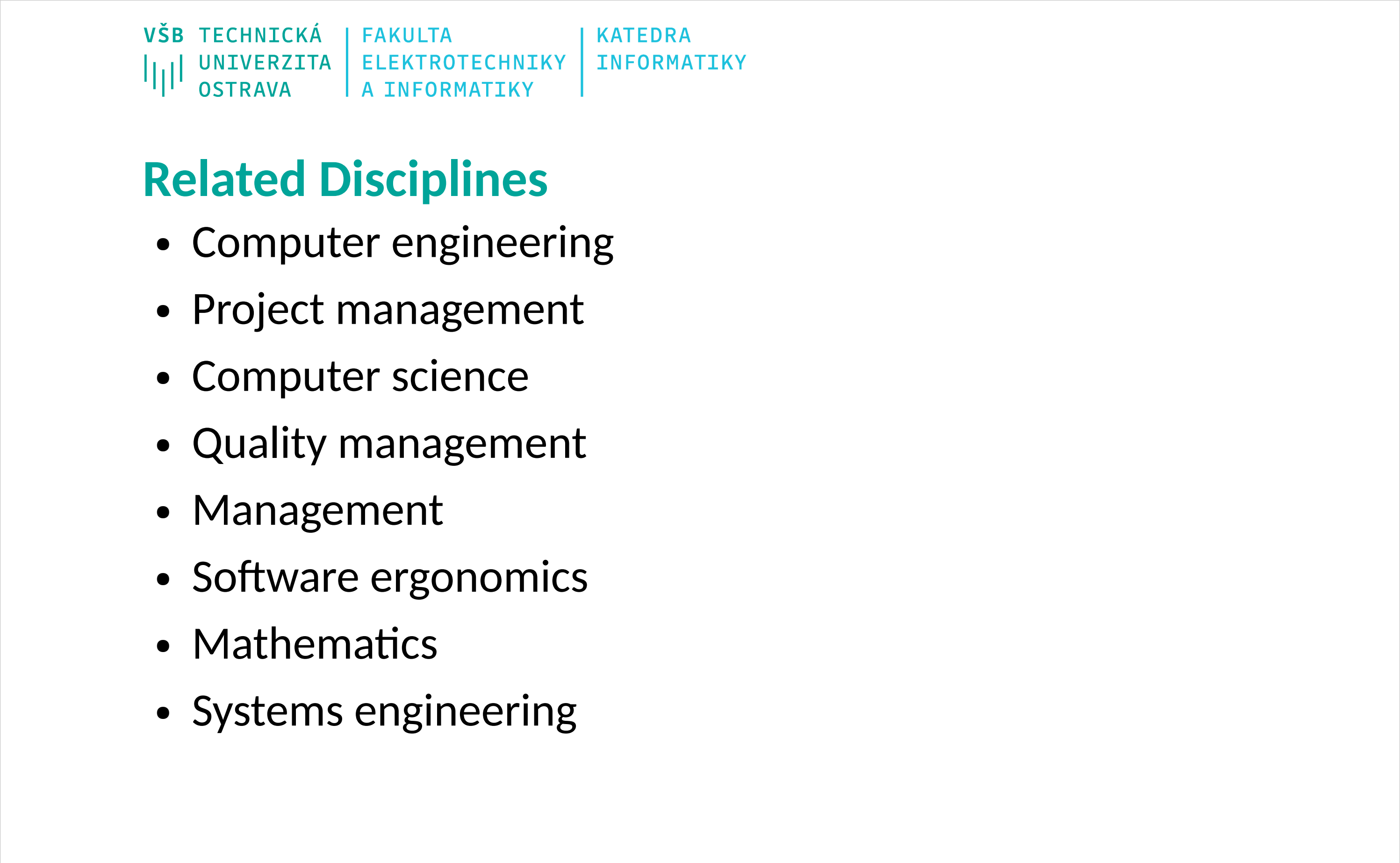
Software configuration management – what are the configurations that are running

Software engineering management – the discipline that covers the management of the development

Software engineering process – the discipline that covers processes, their creation, definition, description etc.

Software engineering tools and methods - the discipline that covers tools and development methods

Software quality – the discipline that deals with the quality of the development



Related Disciplines

- Computer engineering
- Project management
- Computer science
- Quality management
- Management
- Software ergonomics
- Mathematics
- Systems engineering

And of course there are many related disciplines that you should know about.

Like Computer engineering - the discipline that describes the construction of the computers

Project management – the discipline that covers the management of the project

Computer science – the discipline that covers the algorithms, methods and tools for the solution of the problems

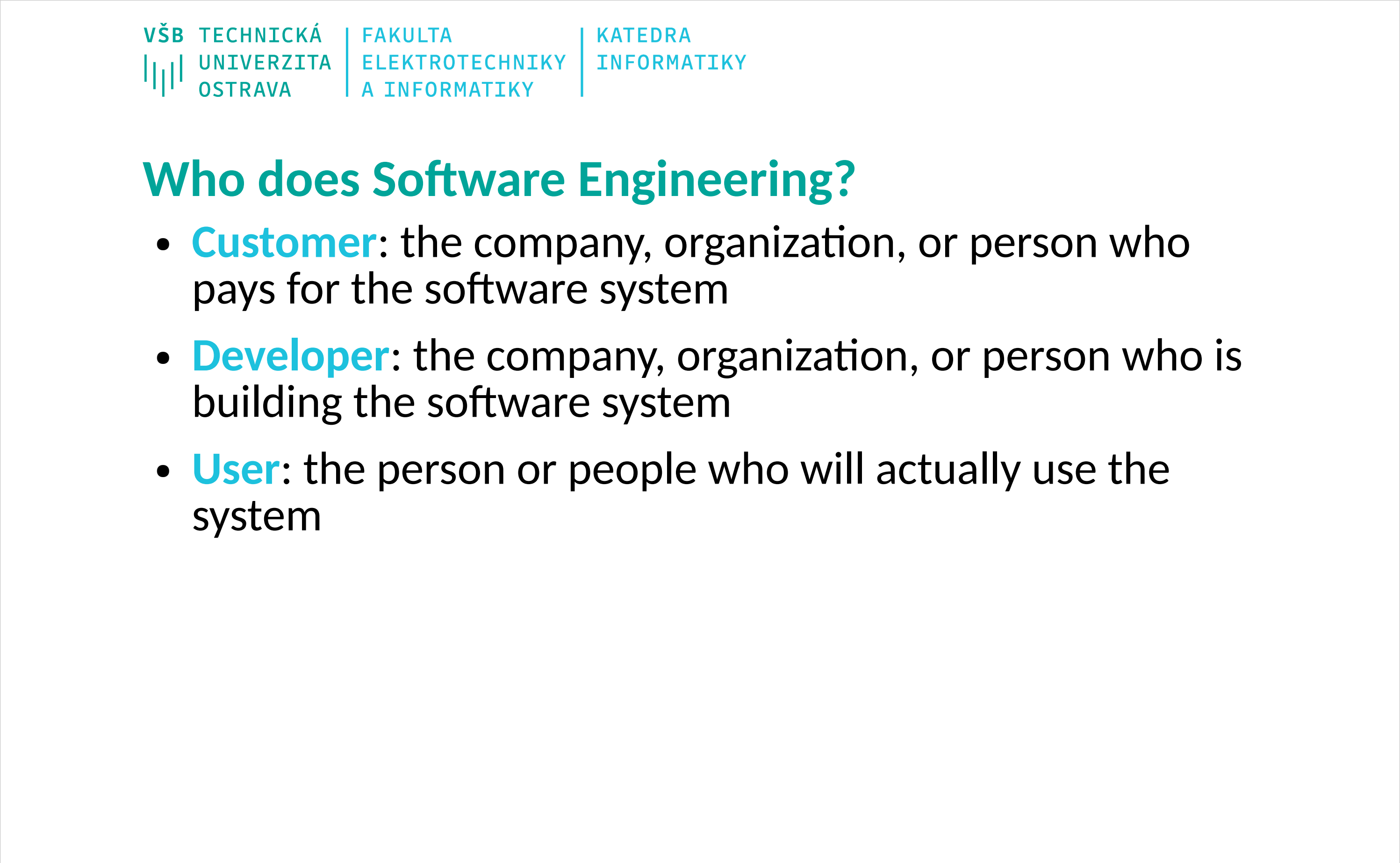
Quality management – discipline that is about managing the quality of the product and the process

Management – the discipline that describes some basic management stuff

Software ergonomics – the discipline that covers the ergonomic aspects of the software development

Mathematics – the discipline that covers mathematical and logical thinking

Systems engineering – the discipline that covers the knowledge about the whole system building not only the software part of it



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

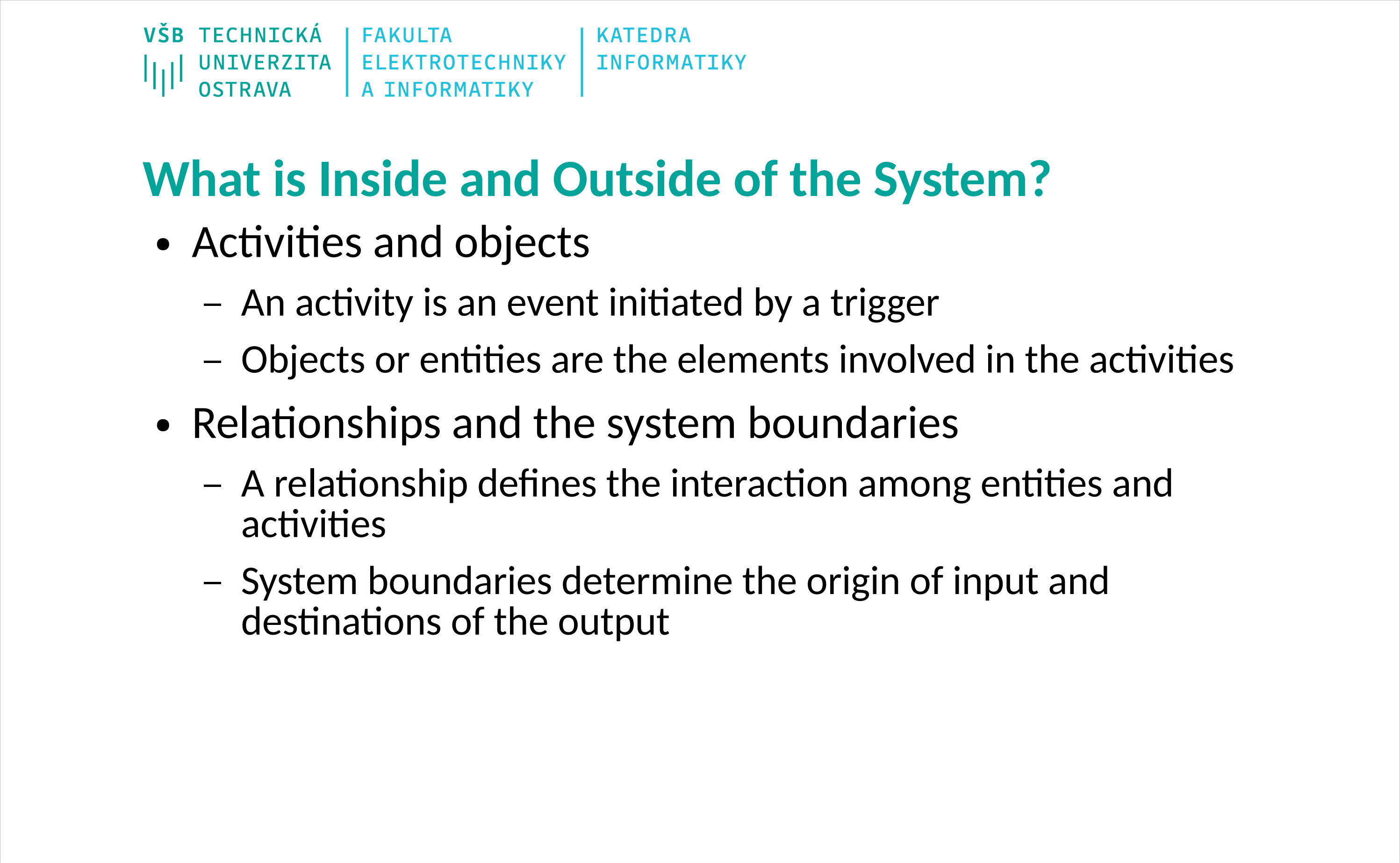
Who does Software Engineering?

- **Customer**: the company, organization, or person who pays for the software system
- **Developer**: the company, organization, or person who is building the software system
- **User**: the person or people who will actually use the system

Participants in a software development project are: customer, user and developer. Customer sponsors the system and has some needs. Developer builds the system for the customer based on the needs from the customer and from the users. Developer then delivers the system to (the customer’s) users.

What is a System?

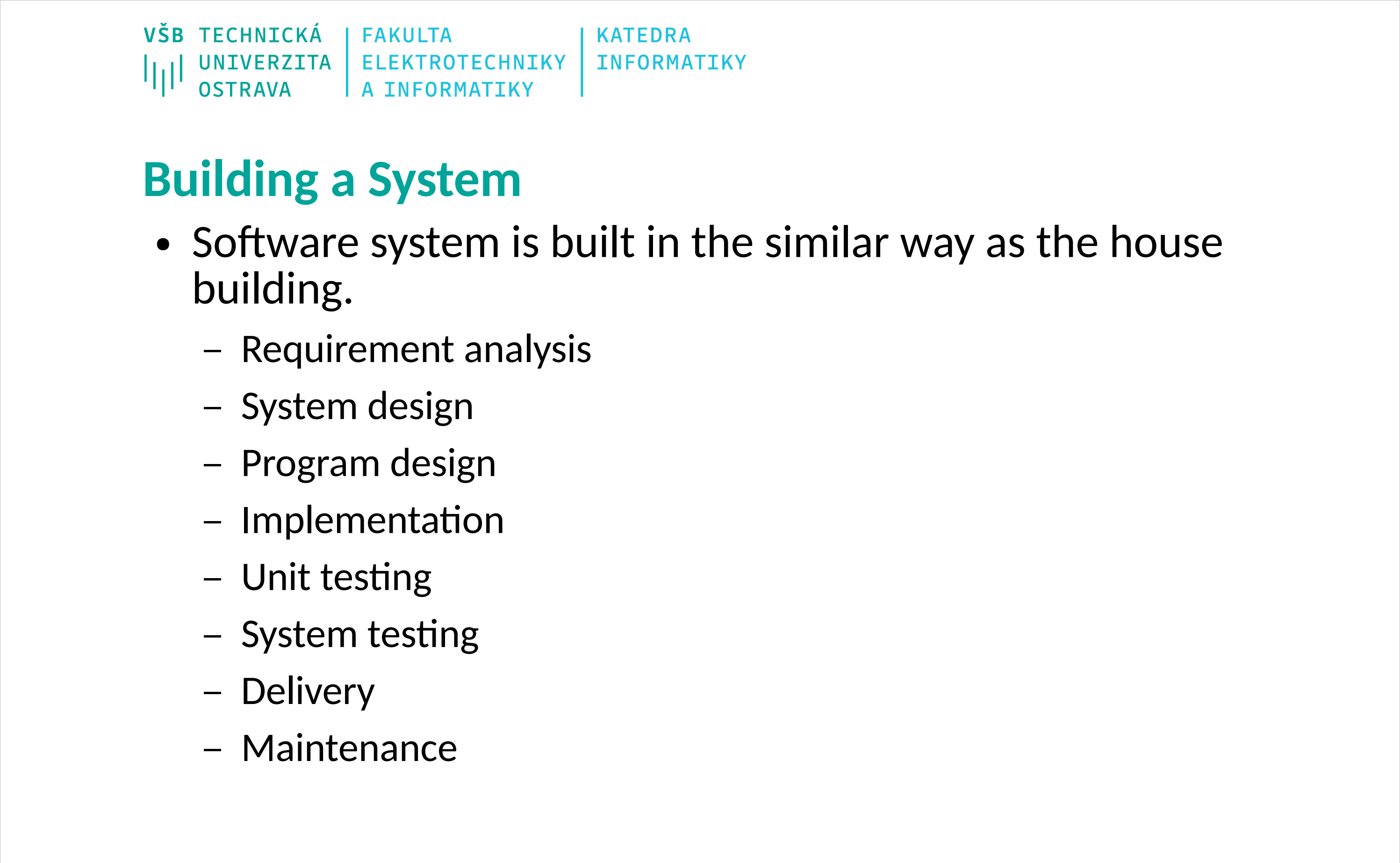
- A purposeful collection of inter-related components working together to achieve some common objective.
- A system may include software, mechanical, electrical and electronic hardware and be operated by people.
- System components are dependent on other system components
- The properties and behaviour of system components are inextricably inter-mingled



What is Inside and Outside of the System?

- Activities and objects
  - An activity is an event initiated by a trigger
  - Objects or entities are the elements involved in the activities
- Relationships and the system boundaries
  - A relationship defines the interaction among entities and activities
  - System boundaries determine the origin of input and destinations of the output

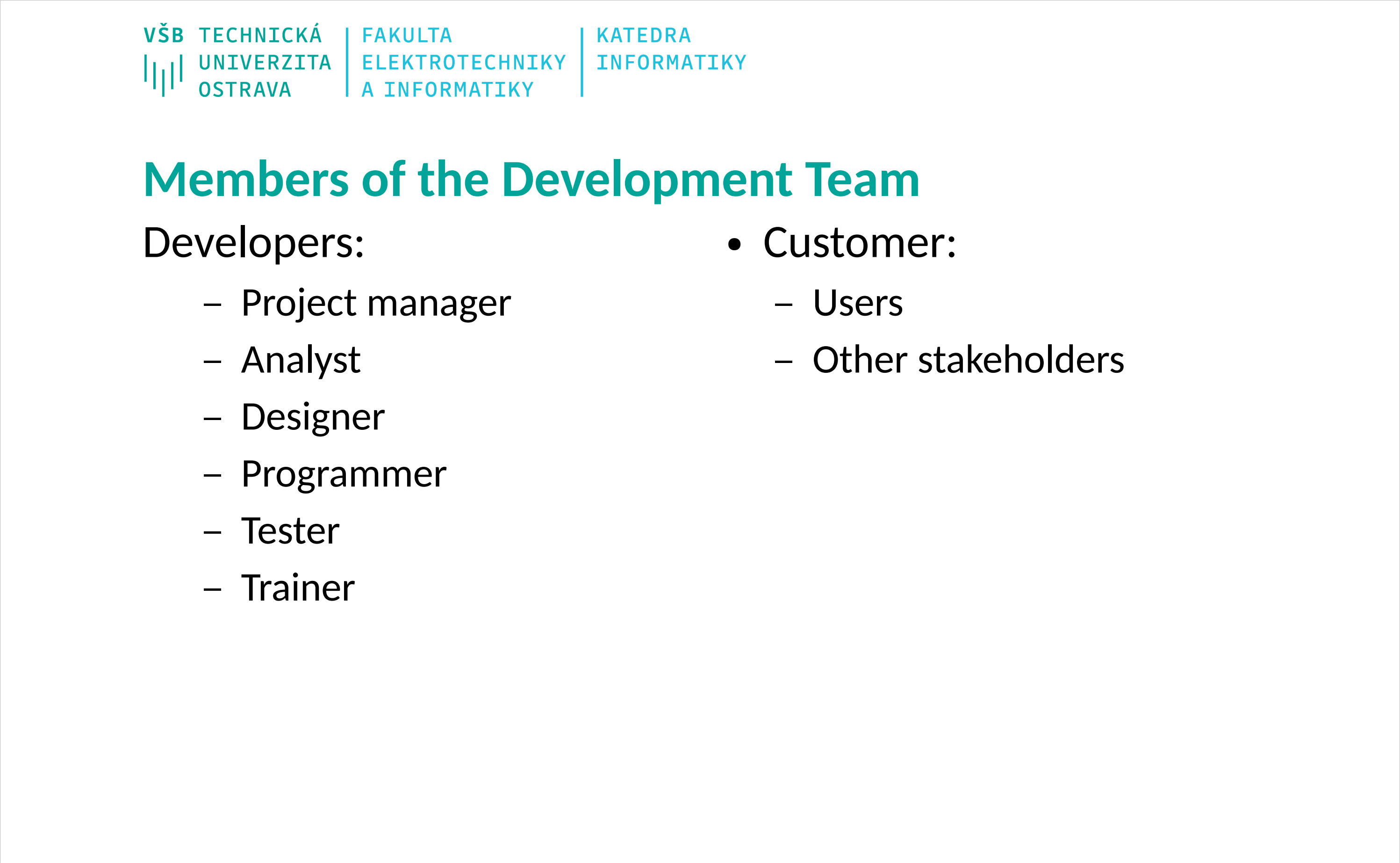
What about the system boundary? System does not exists in a vacuum, the hardware and software has to interact with the users or different systems. That interaction defines the boundary of the system.



Building a System

- Software system is built in the similar way as the house building.
  - Requirement analysis
  - System design
  - Program design
  - Implementation
  - Unit testing
  - System testing
  - Delivery
  - Maintenance

Software system is built in the similar way as a house. Customer defines the needs and maybe the customer is not a direct user, so the user defines needs as well. The need is considered as a requirement. Once requirements are defined, it is necessary to design the system to meet the specified requirements. The design is reviewed then accepted by the customer. Based on the design, the code is written. Once the program is written, it is tested etc. At the end, the final product is delivered to the customer. After the delivery it is possible to do the maintenance (After the delivery - maintenance is taken care of).



Members of the Development Team

Developers:

- Project manager
- Analyst
- Designer
- Programmer
- Tester
- Trainer

- Customer:
  - Users
  - Other stakeholders

The typical members of the development team are:

Project manager – responsible for the project management, coordinates the whole team

Analyst – responsible for the requirements gathering and analysis, cooperative to design

Designer – is responsible for the system design, cooperative to program design

Programmer – responsible for the program design and implementation, cooperative to the unit testing

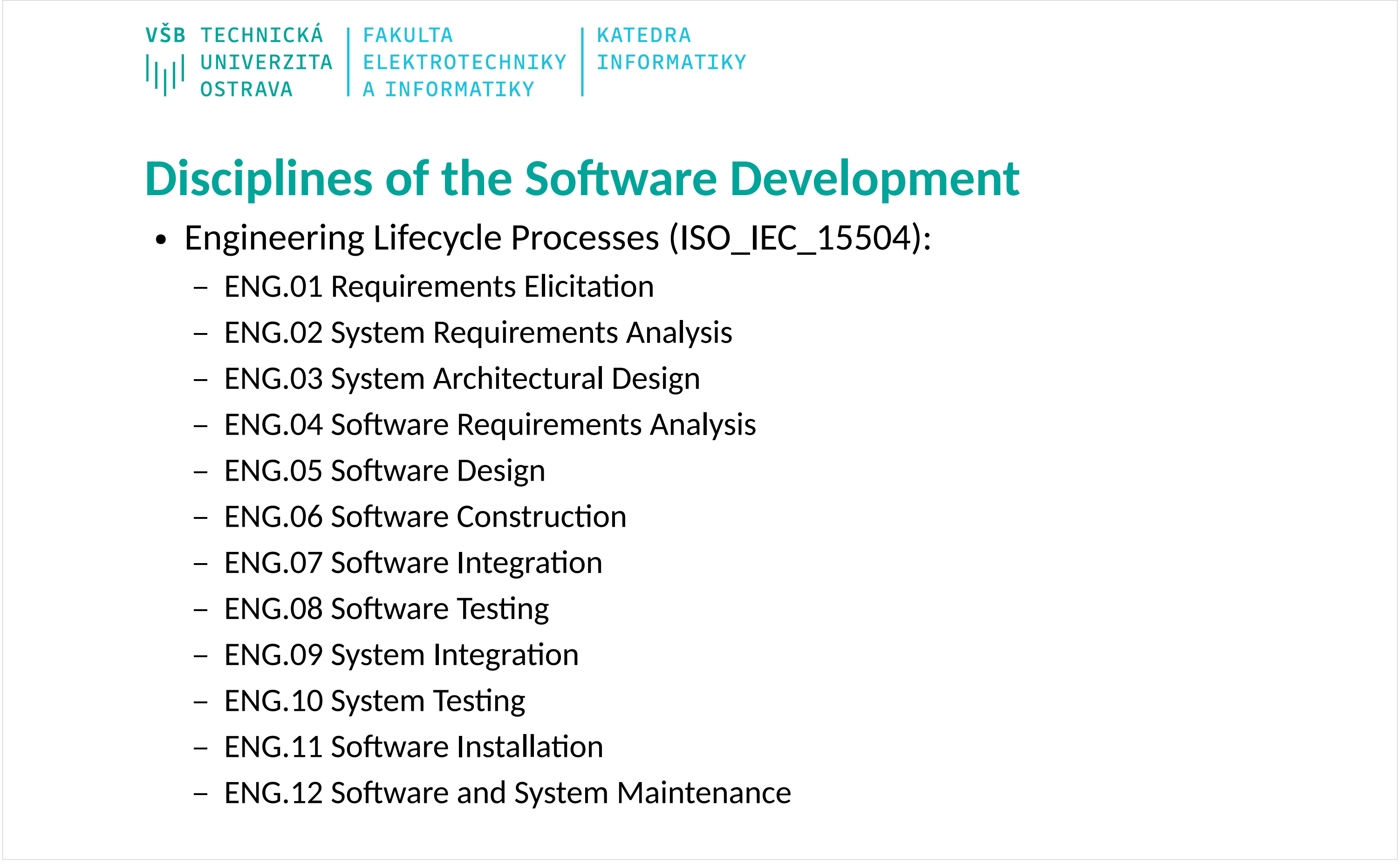
Tester – responsible for the unit testing and integration testing and system testing

Trainer – responsible for the system delivery and maintenance

Customer

User – uses the system, cooperation with developers

These are only the basic roles that are covering the main disciplines of the software development. There might be many other roles that might come. For example – architect – responsible for the architecture design, etc.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Disciplines of the Software Development

- Engineering Lifecycle Processes (ISO_IEC_15504):
  - ENG.01 Requirements Elicitation
  - ENG.02 System Requirements Analysis
  - ENG.03 System Architectural Design
  - ENG.04 Software Requirements Analysis
  - ENG.05 Software Design
  - ENG.06 Software Construction
  - ENG.07 Software Integration
  - ENG.08 Software Testing
  - ENG.09 System Integration
  - ENG.10 System Testing
  - ENG.11 Software Installation
  - ENG.12 Software and System Maintenance

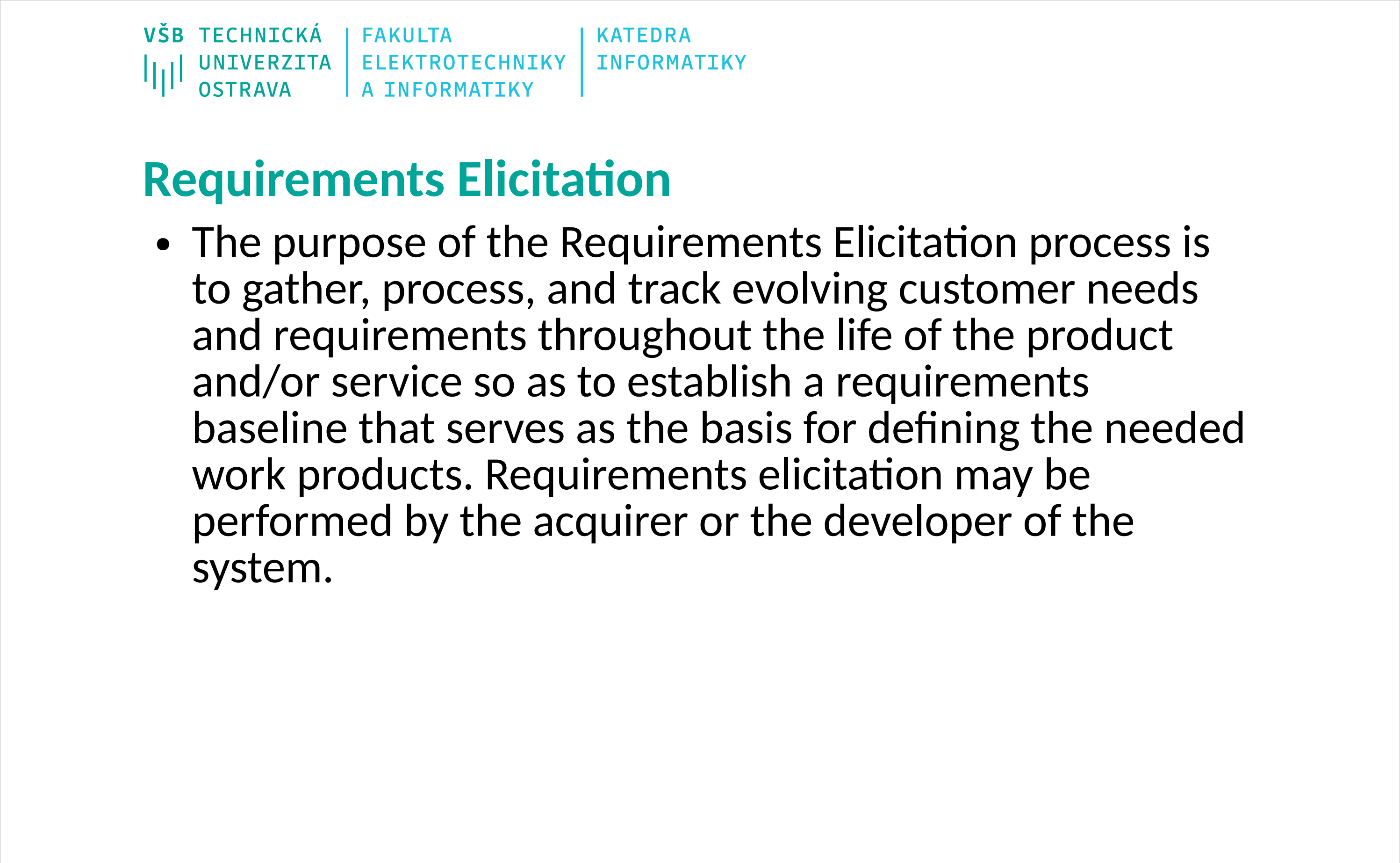
Disciplines of the Software Development

There are many ways to make software. One way of describing the software development is to divide it into parts – disciplines that do the specific tasks to develop software.

ISO_IEC_15504:

ENG.01 Requirements Elicitation
ENG.02 System Requirements Analysis
ENG.03 System Architectural Design
ENG.04 Software Requirements Analysis
ENG.05 Software Design
ENG.06 Software Construction
ENG.07 Software Integration
ENG.08 Software Testing
ENG.09 System Integration
ENG.10 System Testing
ENG.11 Software Installation
ENG.12 Software and System Maintenance

These disciplines describe the whole software development lifecycle. Of course, there are more disciplines like e.g. project management, but these twelve disciplines are directly related to the development itself. Other disciplines are important too, but are supportive to those basic ones.

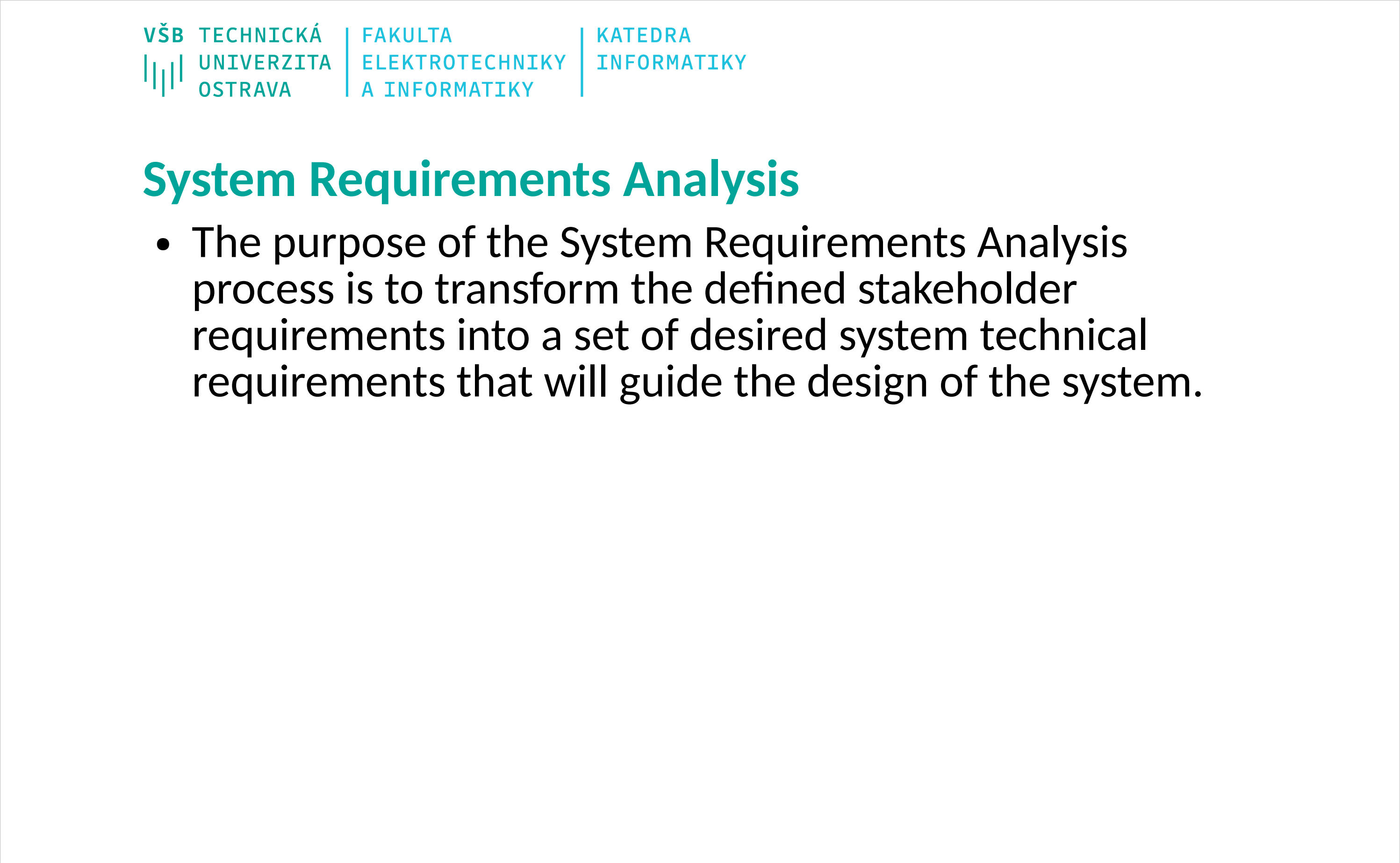


VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
KATEDRA INFORMATIKY

Requirements Elicitation

- The purpose of the Requirements Elicitation process is to gather, process, and track evolving customer needs and requirements throughout the life of the product and/or service so as to establish a requirements baseline that serves as the basis for defining the needed work products. Requirements elicitation may be performed by the acquirer or the developer of the system.

The purpose of the Requirements Elicitation process is to gather, process, and track evolving customer needs and requirements throughout the life of the product and/or service so as to establish a requirements baseline that serves as the basis for defining the needed work products. Requirements elicitation may be performed by the acquirer or the developer of the system.



System Requirements Analysis

- The purpose of the System Requirements Analysis process is to transform the defined stakeholder requirements into a set of desired system technical requirements that will guide the design of the system.

As a result of successful implementation of the process:

A defined set of system functional and non-functional requirements describing the problem to be solved are established.

The appropriate techniques are performed to optimize the preferred project solution.

System requirements are analyzed for correctness and testability.

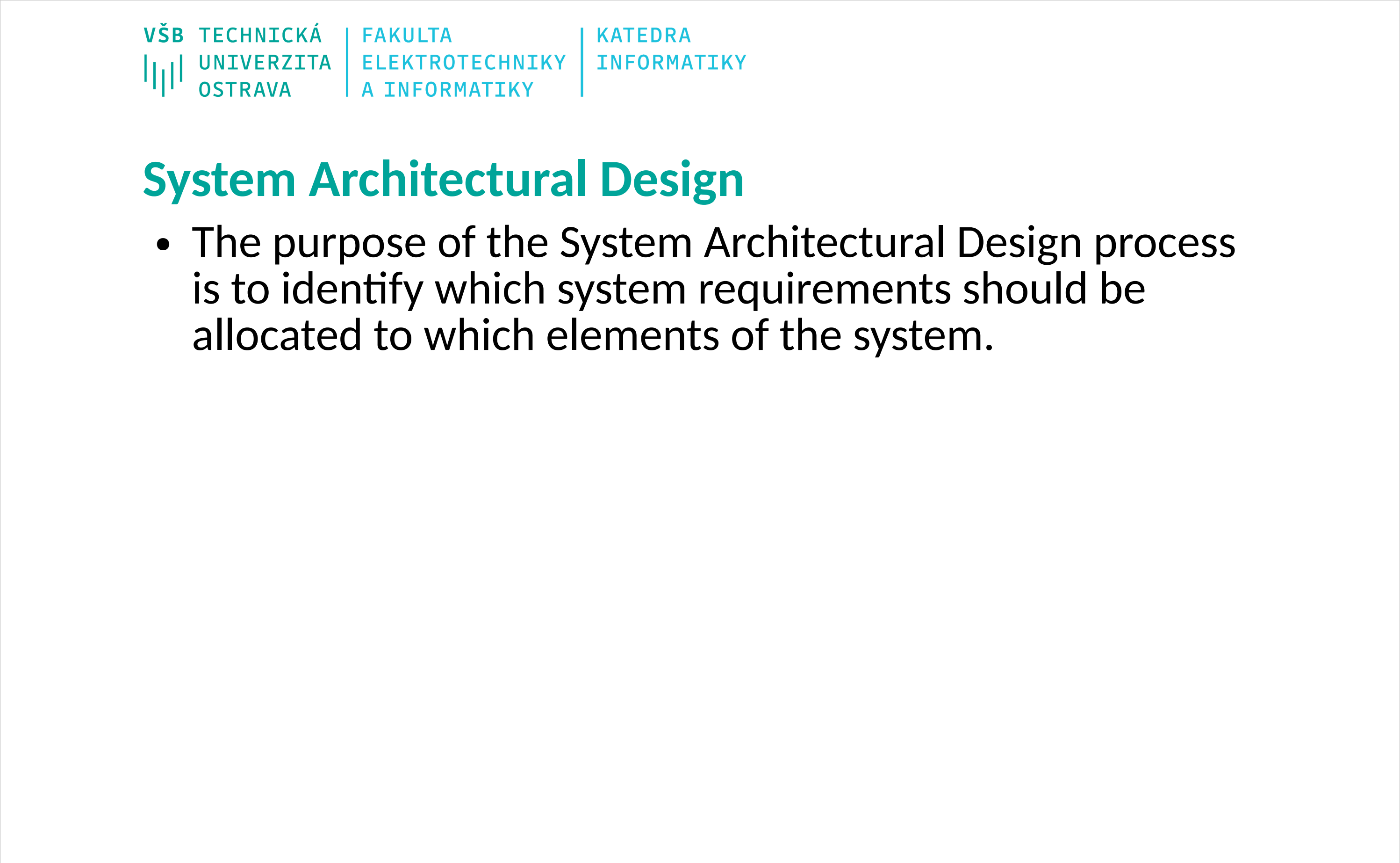
The impact of the system requirements on the operating environment are understood.

The requirements are prioritized, approved and updated as needed.

Consistency and traceability is established between the system requirements and the customer’s requirements baseline.

Changes to the baseline are evaluated for cost, schedule and technical impact.

The system requirements are communicated to all affected parties and baselined.



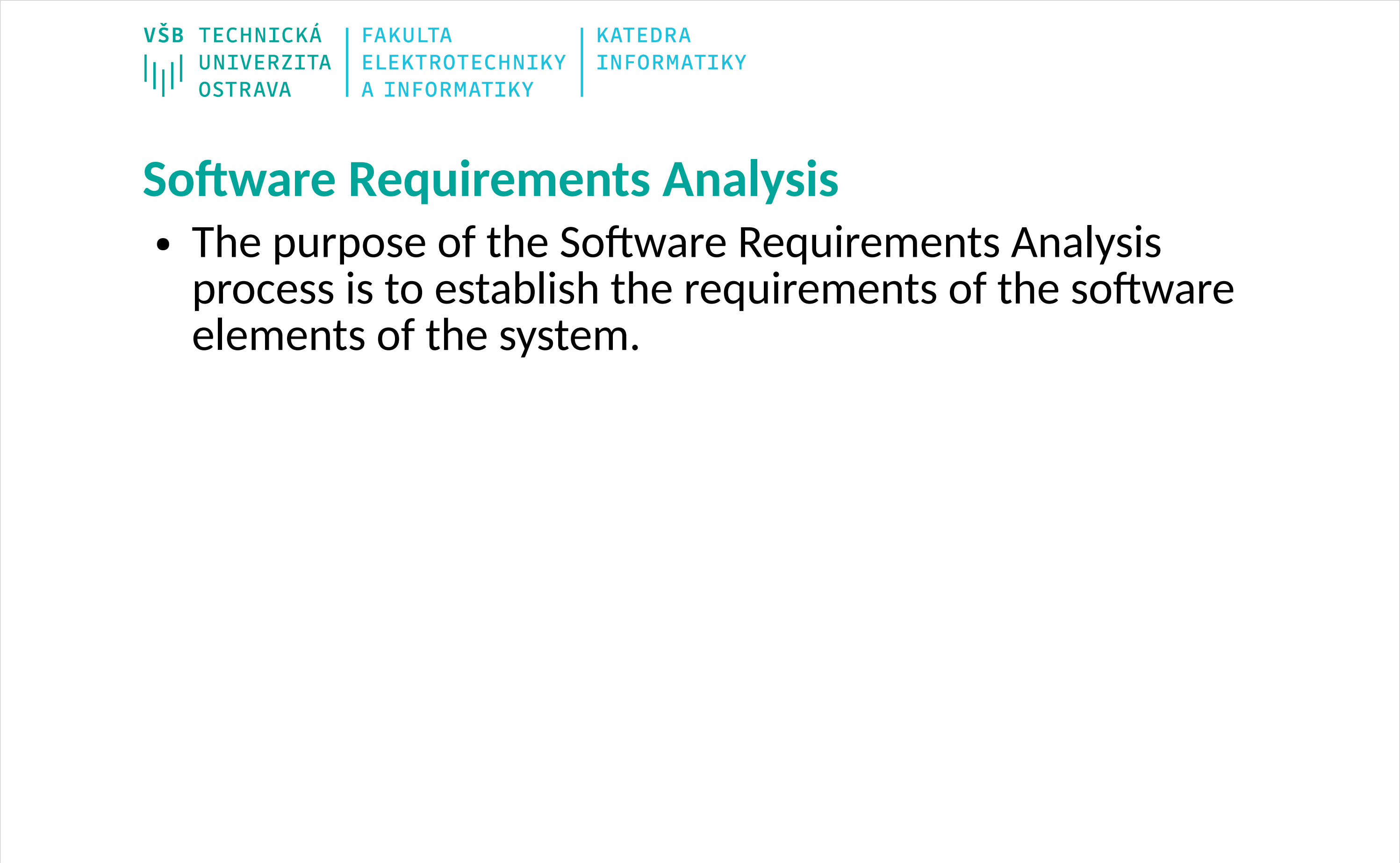
VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

System Architectural Design

- The purpose of the System Architectural Design process is to identify which system requirements should be allocated to which elements of the system.

As a result of a successful implementation of the process:

A system architecture design is a one that identifies the elements of the system and meets the defined requirements.
The system's functional and non-functional requirements are addressed.
The requirements are allocated to the elements of the system.
Internal and external interfaces of each system element are defined.
Verification between the system requirements and the system architecture is performed.
The requirements allocated to the system elements and their interfaces are traceable to the customer's requirements baseline.
Consistency and traceability between the system requirements and system architecture design is maintained.
The system requirements, the system architecture design, and their relationships are baselined and communicated to all affected parties.



Software Requirements Analysis

- The purpose of the Software Requirements Analysis process is to establish the requirements of the software elements of the system.

As a result of a successful implementation of Software requirements analysis process:

The requirements allocated to the software elements of the system and their interfaces are defined

Software requirements are analyzed for correctness and testability

The impact of software requirements on the operating environment are understood

Consistency and traceability are established between the software requirements and system requirements

Prioritization for implementing the software requirements is defined

The software requirements are approved and updated as needed

Changes to the software requirements are evaluated for cost, schedule and technical impact

The software requirements are baselined and communicated to all affected parties.

Software Design

- The purpose of the Software Design Process is to provide a design for the software **that implement** and can be verified against the requirements.

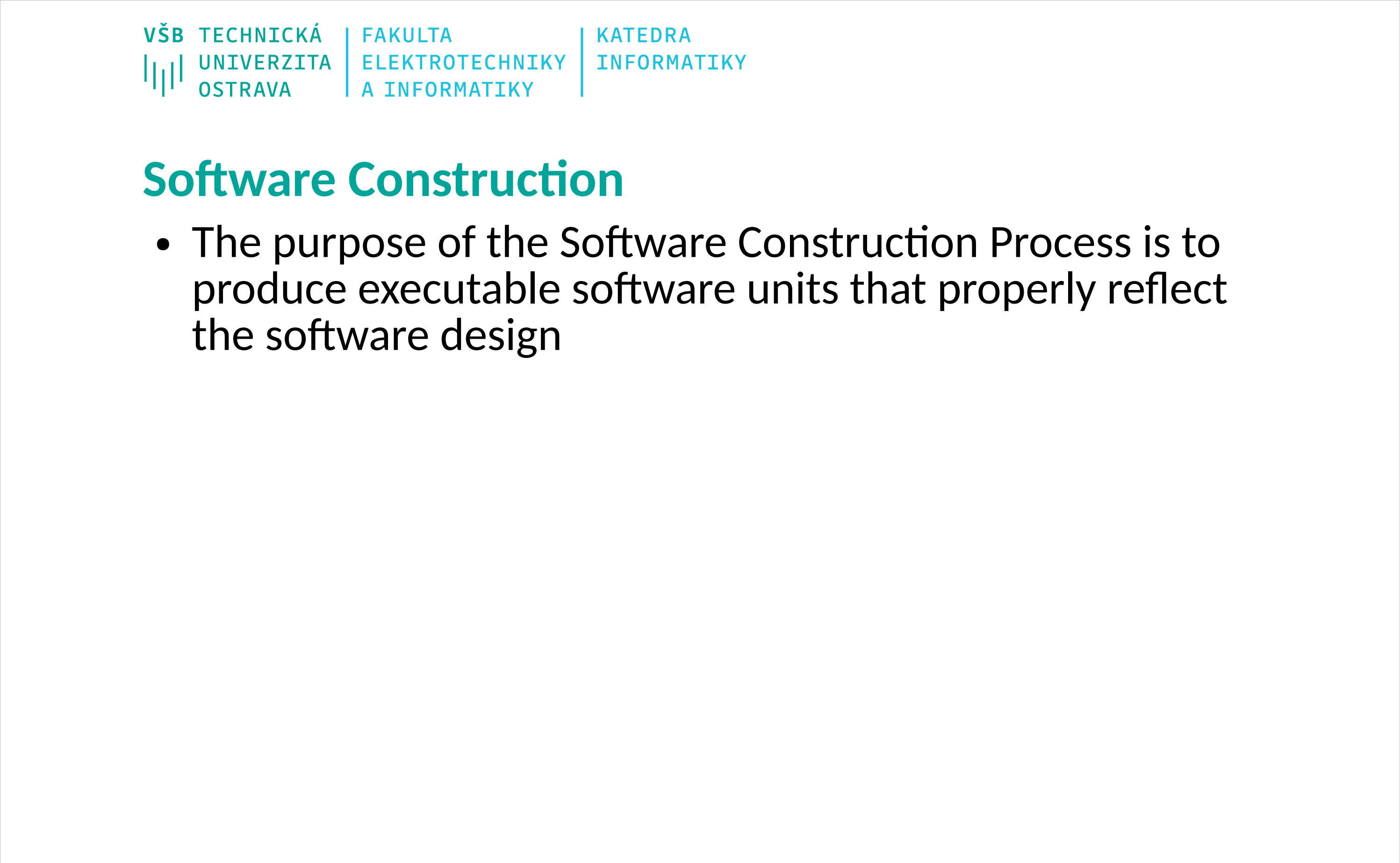
As a result of a successful implementation of the process:

A software architectural design is developed and baselined that describes the software elements that will implement the software requirements.

Internal and external interfaces of each software elements are defined.

A detailed design is developed that describes software units that can be built and tested.

Consistency and traceability are established between software requirements and software design.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Software Construction

- The purpose of the Software Construction Process is to produce executable software units that properly reflect the software design

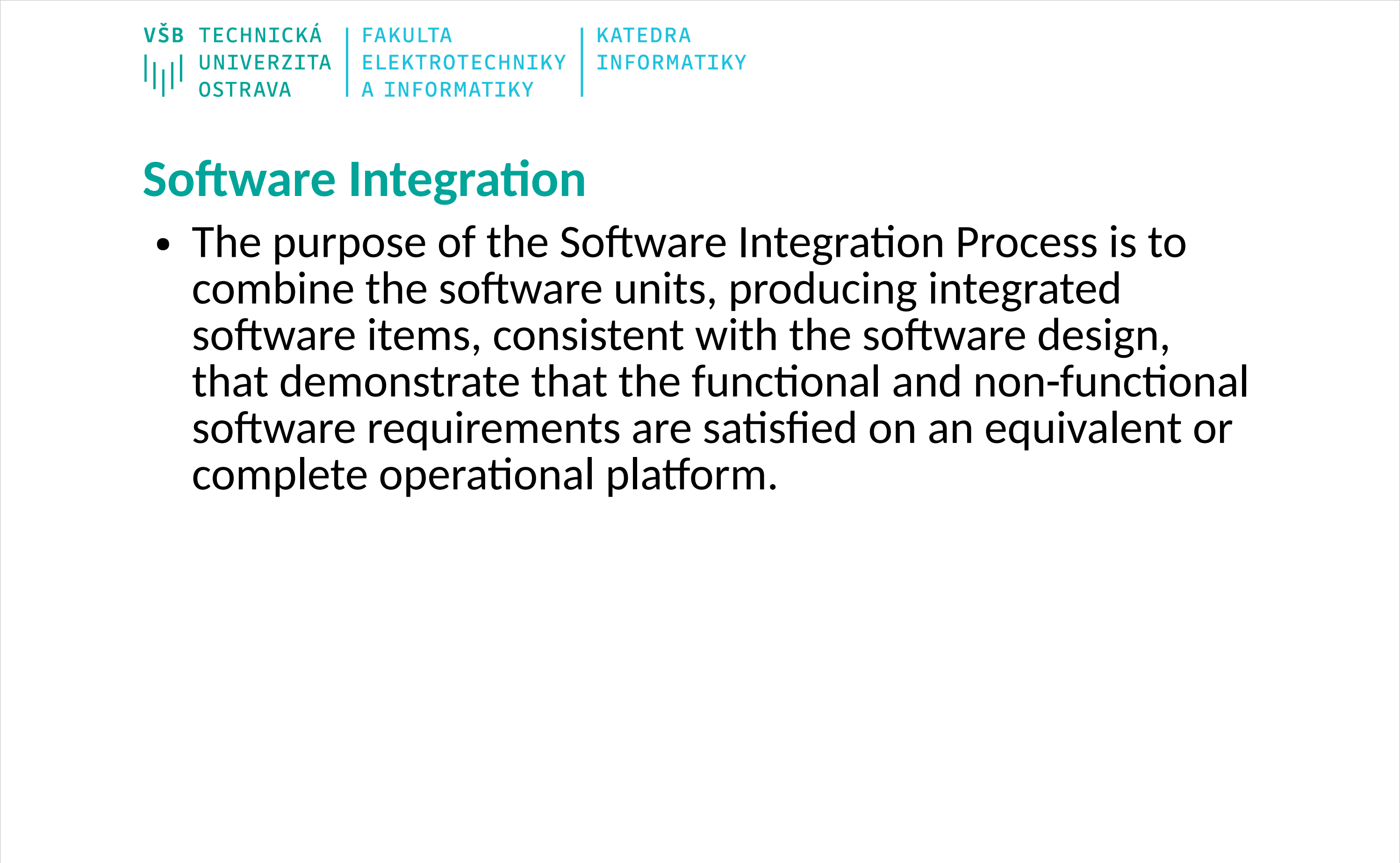
As a result of a successful implementation of the process:

Verification criteria are defined for all software units against their requirements.

Software units defined by the design are produced.

Consistency and traceability are established between software requirements and design and software units.

Verification of the software units against the requirements and the design is accomplished.



Software Integration

- The purpose of the Software Integration Process is to combine the software units, producing integrated software items, consistent with the software design, that demonstrate that the functional and non-functional software requirements are satisfied on an equivalent or complete operational platform.

As a result of a successful implementation of the process:

An integration strategy is developed for software units consistent with the software design and the prioritized software requirements.

Verification criteria for software items are developed that ensure compliance with the software requirements allocated to the items.

Software items are verified using the defined criteria.

Software items defined by the integration strategy are produced.

Results of integration testing are recorded.

Consistency and traceability are established between software design and software items.

A regression strategy is developed and applied for re-verifying software items when a change in software units (including associated requirements, design and code) occur.

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Software Testing

- The purpose of the Software Testing Process is to confirm that the integrated software product meets its defined requirements.

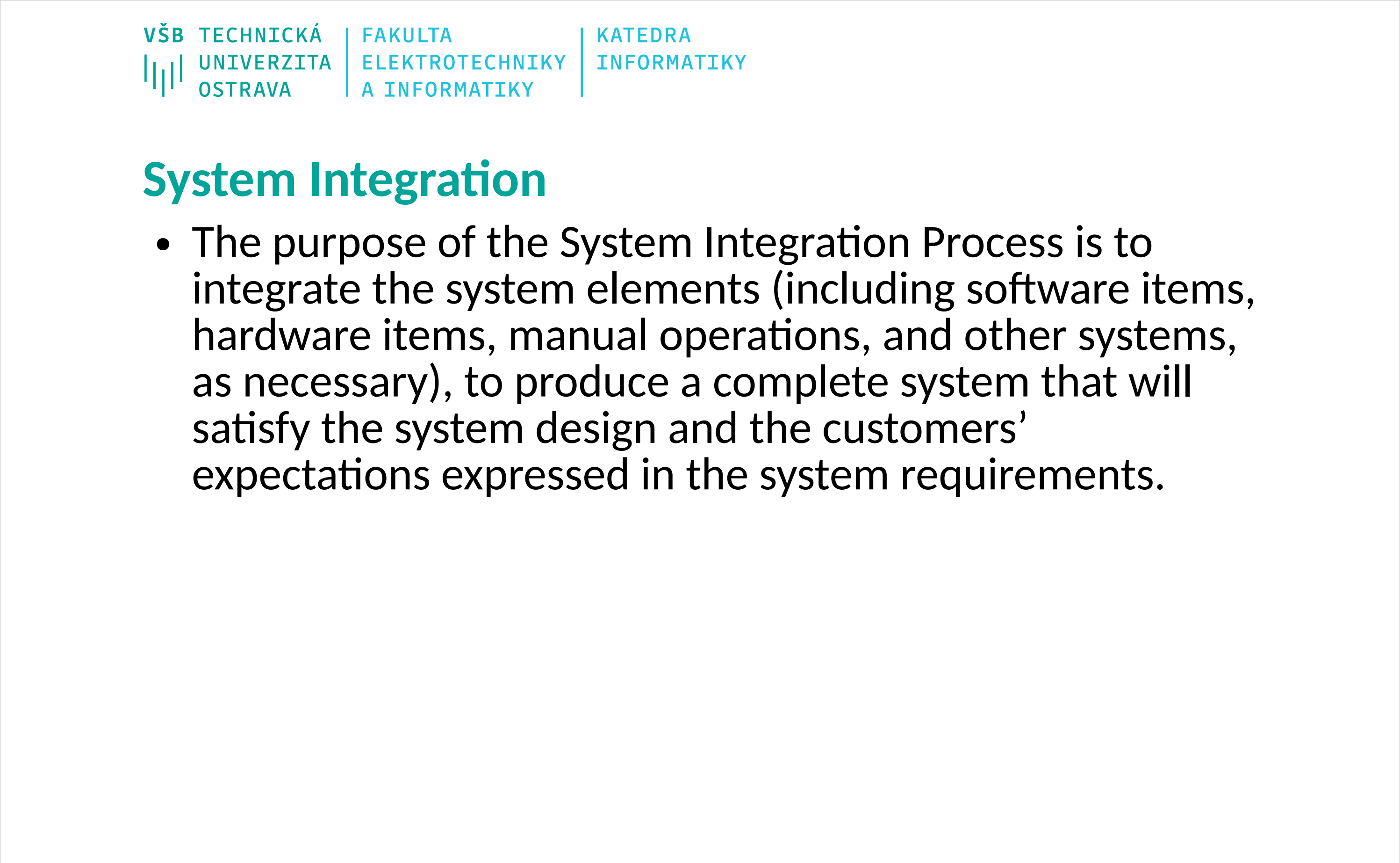
As a result of a successful implementation of the process:

Criteria for the integrated software are developed that demonstrate compliance with the software requirements.

Integrated software is verified using the defined criteria.

Test results are recorded.

A regression strategy is developed and applied for re-testing the integrated software when a change in software items is made.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

System Integration

- The purpose of the System Integration Process is to integrate the system elements (including software items, hardware items, manual operations, and other systems, as necessary), to produce a complete system that will satisfy the system design and the customers' expectations expressed in the system requirements.

As a result of a successful implementation of the process:

A strategy is developed to integrate the system according to the priorities of the system requirements.

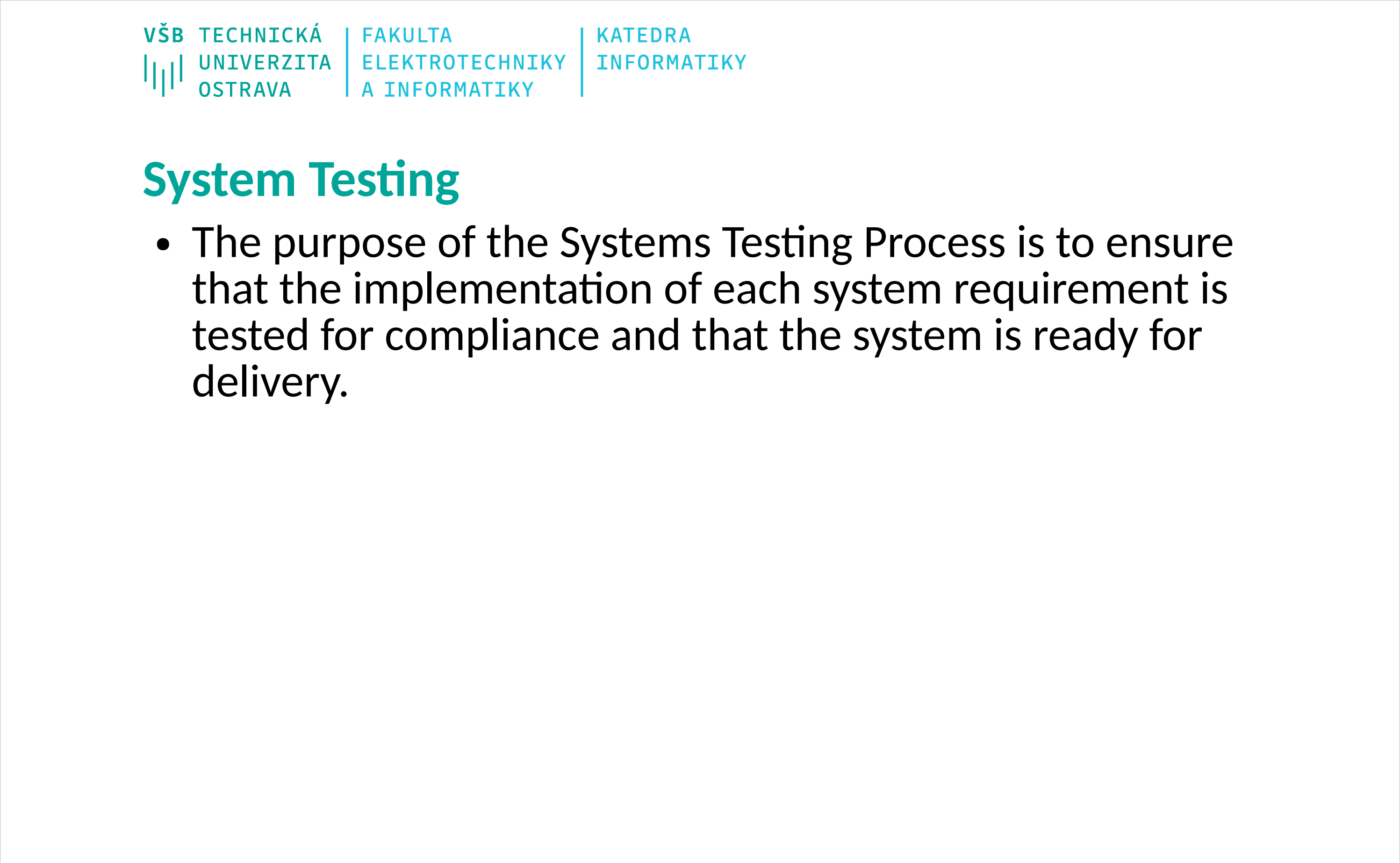
Criteria is developed to verify compliance with the system requirements allocated to the system elements, including the interfaces between system elements.

The system integration is verified using the defined criteria.

A regression strategy is developed and applied for re-testing the system [elements] when changes are made.

Consistency and traceability are established between the system design and the integrated system elements.

An integrated system, demonstrating compliance with the system design and validation that a complete set of useable deliverable system elements exists, is constructed.

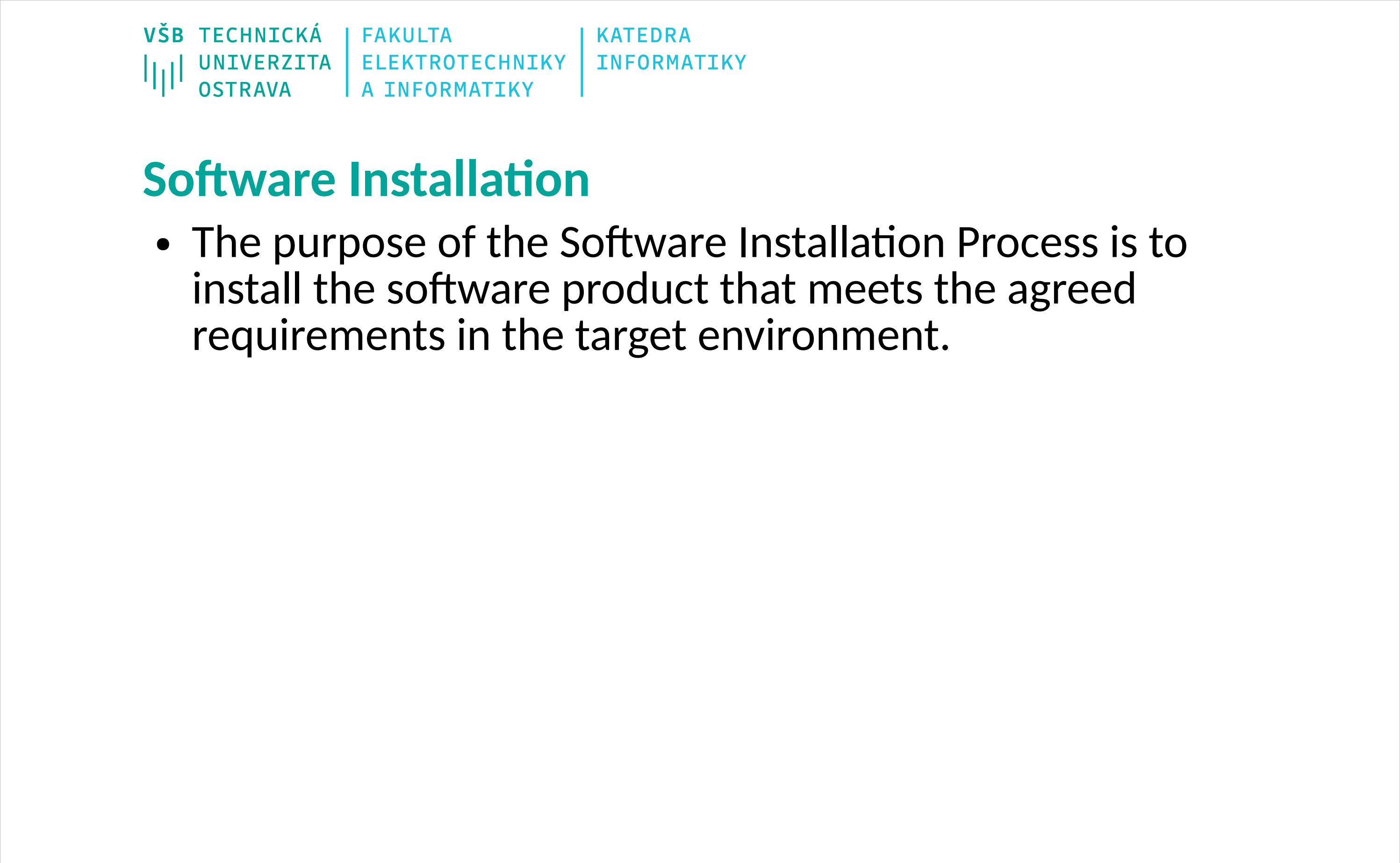


System Testing

- The purpose of the Systems Testing Process is to ensure that the implementation of each system requirement is tested for compliance and that the system is ready for delivery.

As a result of a successful implementation of the process:

Criteria for the integrated system are developed that demonstrate compliance with system requirements.
The integrated system is verified using the defined criteria.
Test results are recorded.
A regression strategy is developed and applied for re-testing the integrated system should a change be made to existing system elements.

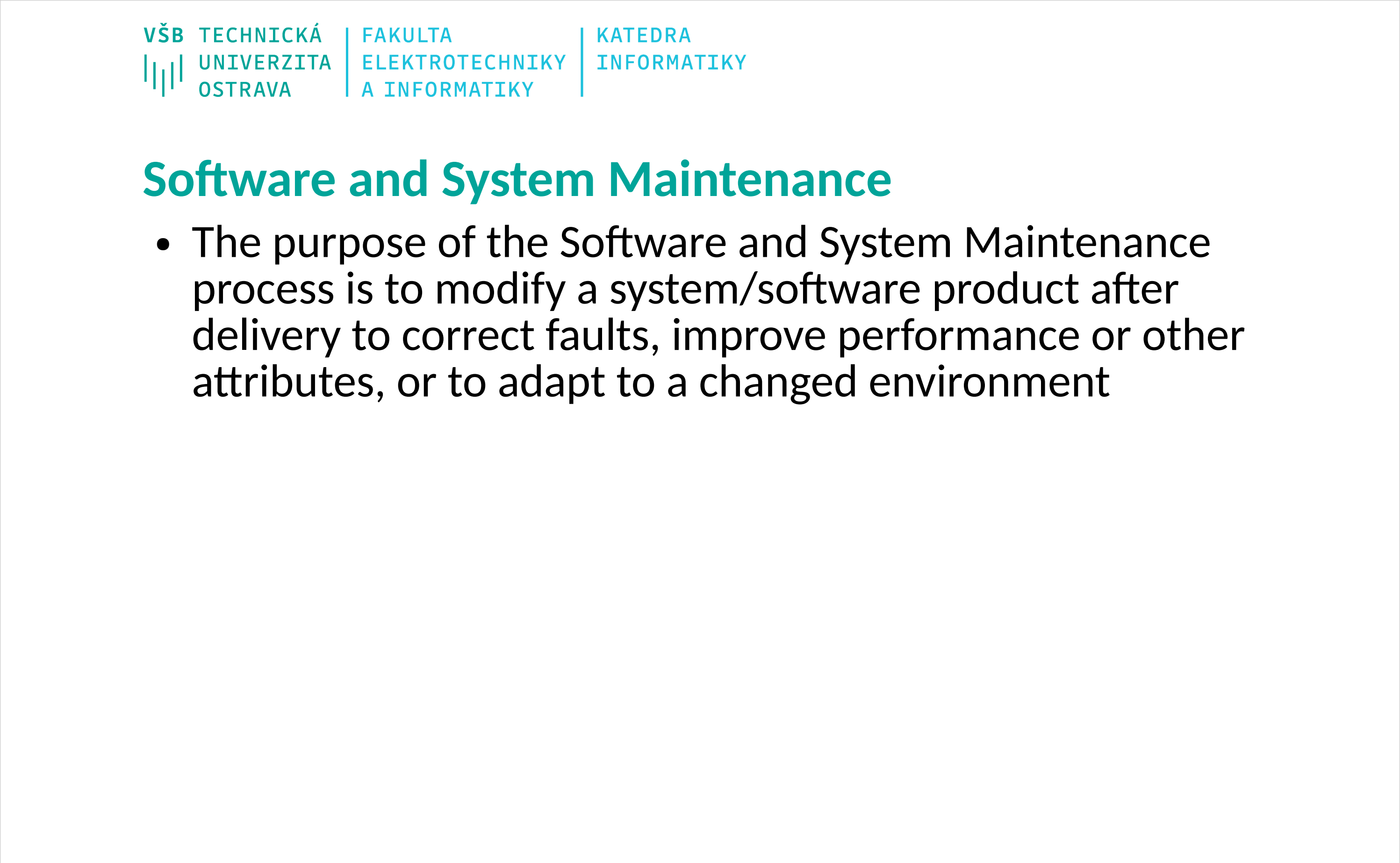


Software Installation

- The purpose of the Software Installation Process is to install the software product that meets the agreed requirements in the target environment.

As a result of a successful implementation of the process:

A software installation strategy is developed.
Criteria for software installation is developed that demonstrates compliance with the software installation requirements.
The software product is installed in the target environment.
Assure that the software product is ready for use in its intended environment.



Software and System Maintenance

- The purpose of the Software and System Maintenance process is to modify a system/software product after delivery to correct faults, improve performance or other attributes, or to adapt to a changed environment

As a result of a successful implementation of the process:

A maintenance strategy is developed to manage modification, migration and retirement of products according to the release strategy.

The impact of changes to the existing system on organization, operations or interfaces are identified.

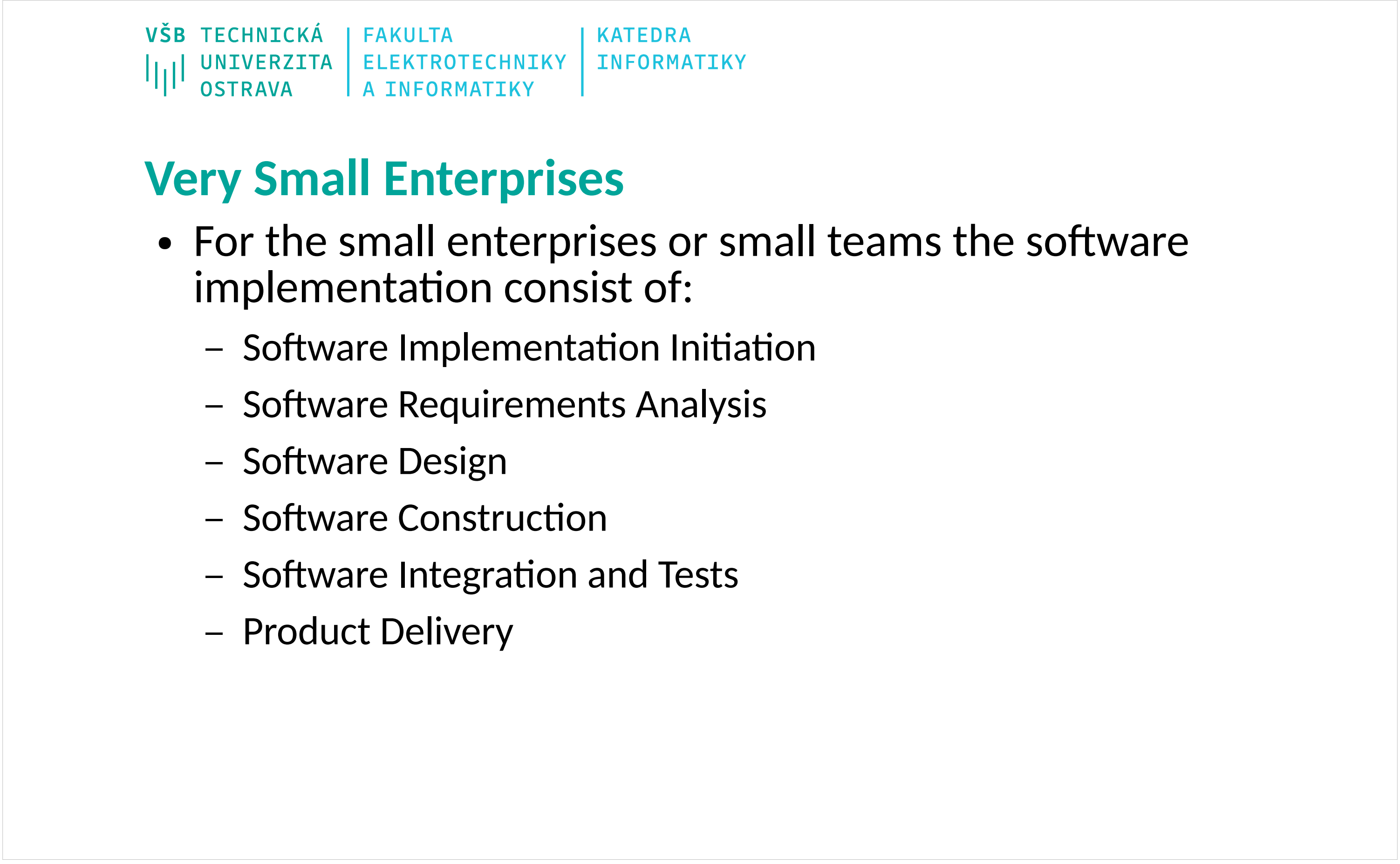
Affected system/software documentation is updated as needed.

Modified products are developed with associated tests that demonstrate that requirements are not compromised.

Product upgrades are migrated to the customer's environment.

On request, products are retired from use in a controlled manner that minimizes disturbance to the customers.

The system/software modification is communicated to all affected parties.



Very Small Enterprises

- For the small enterprises or small teams the software implementation consist of:
  - Software Implementation Initiation
  - Software Requirements Analysis
  - Software Design
  - Software Construction
  - Software Integration and Tests
  - Product Delivery

We can use a model VSE for the small enterprises or very small development teams. Of course, we are presenting only the development processes.

The software implementation initiation subprocess ensures that the project plan is committed to by the work team.

The software requirements analysis subprocess analyzes the agreed to customer’s requirements and establishes the validated project requirements.

The software architecture and detailed design subprocess transforms the software requirements to the system software architecture and software detailed design.

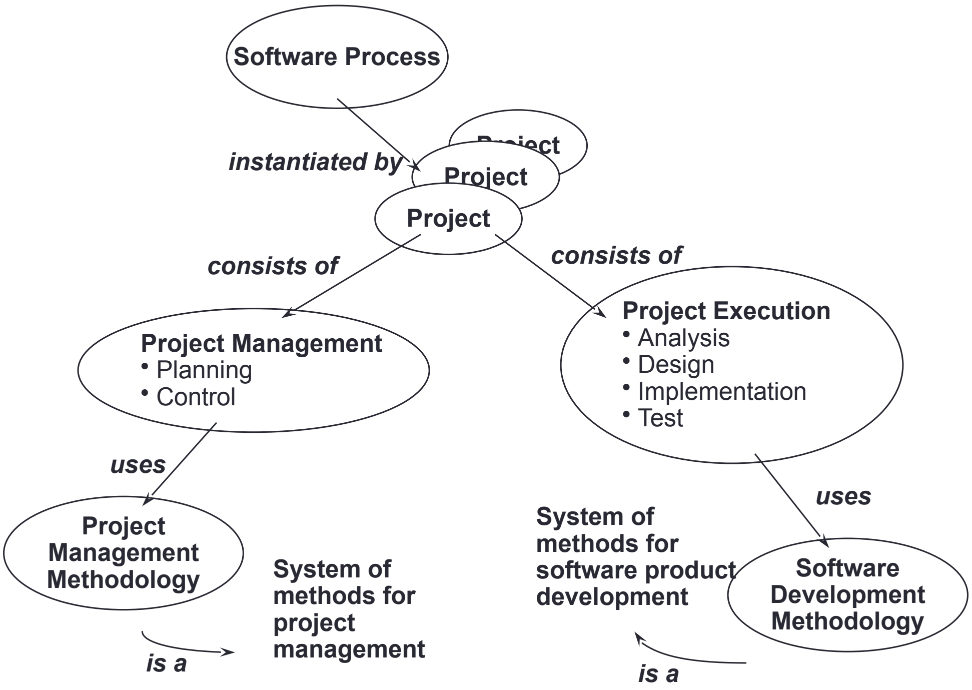
The software construction subprocess develops the software code and data from the software design.

The software integration and test subprocess ensures that the integrated software components satisfy the software requirements.

The product delivery subprocess provides the integrated software product to the customer.

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Software Production Layout



Software Process

instantiated by

Project

Project

Project

consists of

consists of

Project Management
- Planning
- Control

Project Execution
- Analysis
- Design
- Implementation
- Test

uses

uses

Project Management Methodology

is a

System of methods for project management

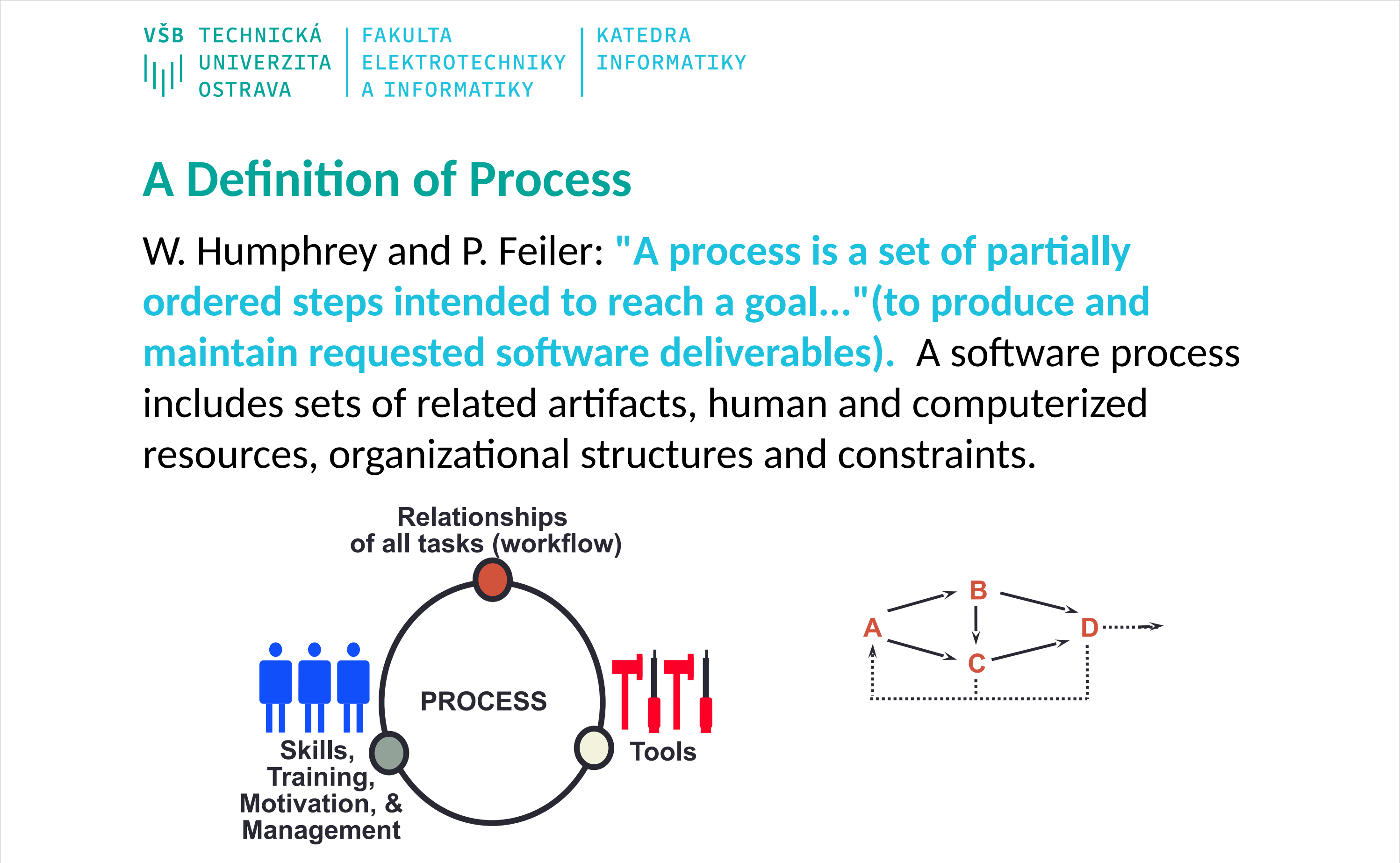
System of methods for software product development

is a

Software Development Methodology

Software Production Layout.

What is necessary to produce a software. We have already talked about the disciplines. Now, we have to talk about an ordering of the disciplines. The basic ordering is described by the process. Since we are developing software - software process. We can think of a process as a set of ordered tasks. It means series of steps involving activities, constraints, and resources that produce an intended output of some kind. In our case a software system.



A Definition of Process

W. Humphrey and P. Feiler: **"A process is a set of partially ordered steps intended to reach a goal..."(to produce and maintain requested software deliverables).** A software process includes sets of related artifacts, human and computerized resources, organizational structures and constraints.

A Definition of Process

A process usually involves a set of tools, techniques, relationships.

Any process has the following characteristics:

The process prescribes all the major process activities.

The process uses resources, subject to a set of constraints, and produce intermediate and final product.

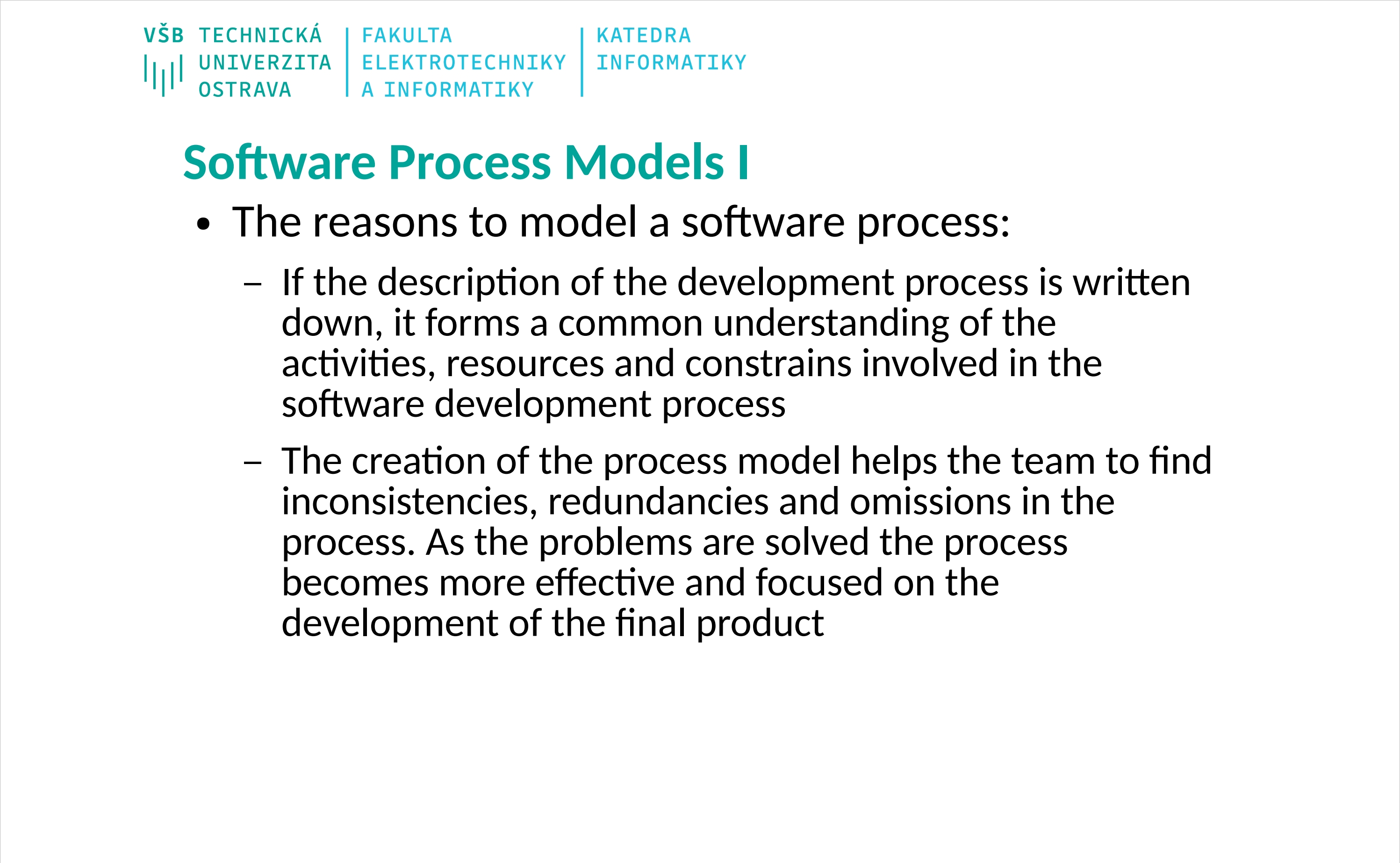
The process may be composed of subprocesses that are linked in some way. The process may be defined as a hierarchy of processes, organized so that each subprocess has its own model.

Each process activity has entry and exit criteria, so we know when the activity starts and ends.

The activities are organized in a sequence, so that it is clear when one activity is performed relative to the other activities.

Every process has a set of guiding principles that explain a goal of each activity.

Constrains and controls may apply to an activity, resource or product.



Software Process Models I

- The reasons to model a software process:
  - If the description of the development process is written down, it forms a common understanding of the activities, resources and constrains involved in the software development process
  - The creation of the process model helps the team to find inconsistencies, redundancies and omissions in the process. As the problems are solved the process becomes more effective and focused on the development of the final product

Why to model a software development process?

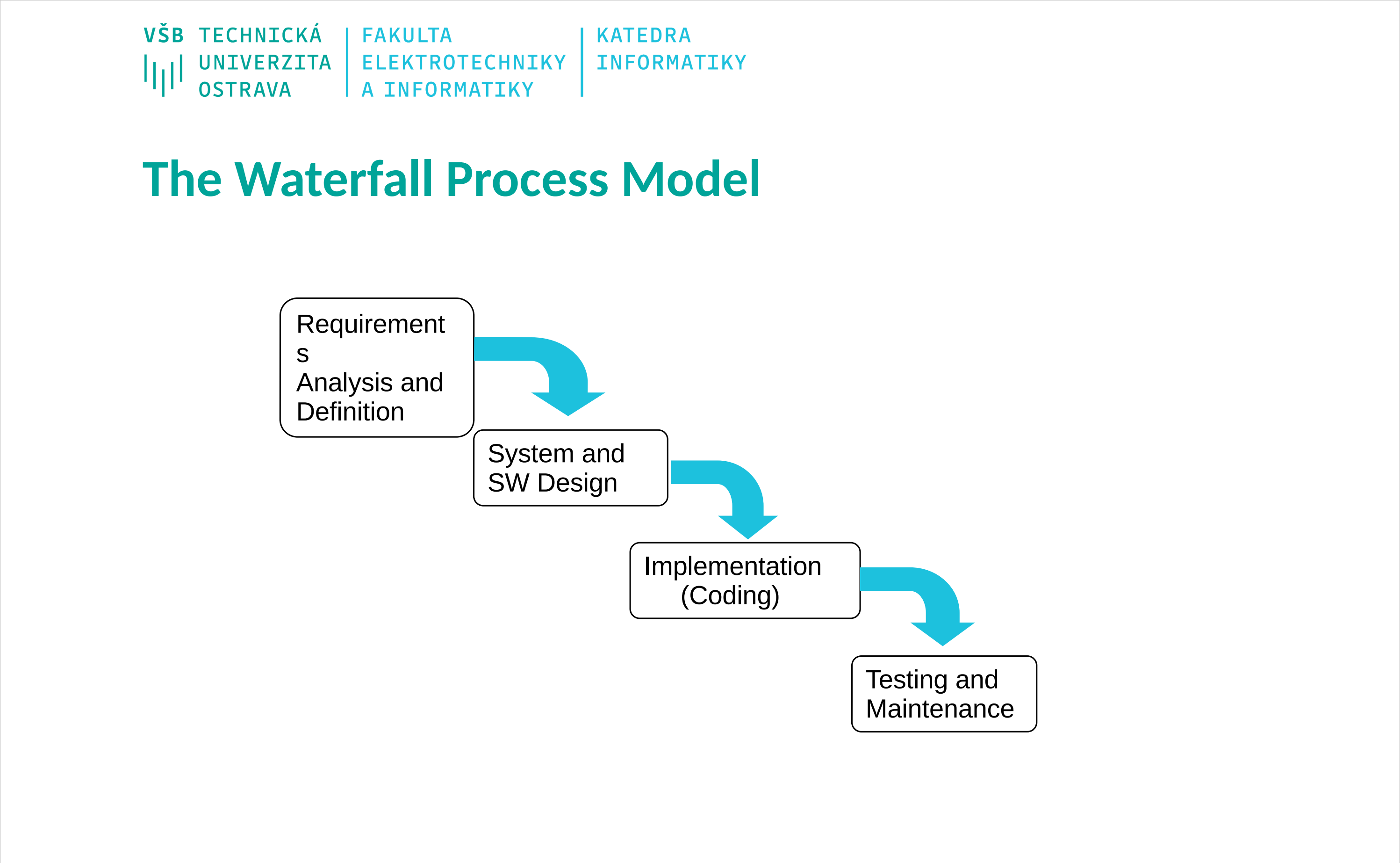
The reasons to model a software process:

If the description of the development process is written down, it forms a common understanding of the activities, resources and constrains involved in the software development process

The creation of the process model helps the team to find inconsistencies, redundancies and omissions in the process. As the problems are solved the process becomes more effective and focused on the development of the final product

Software Process Models II

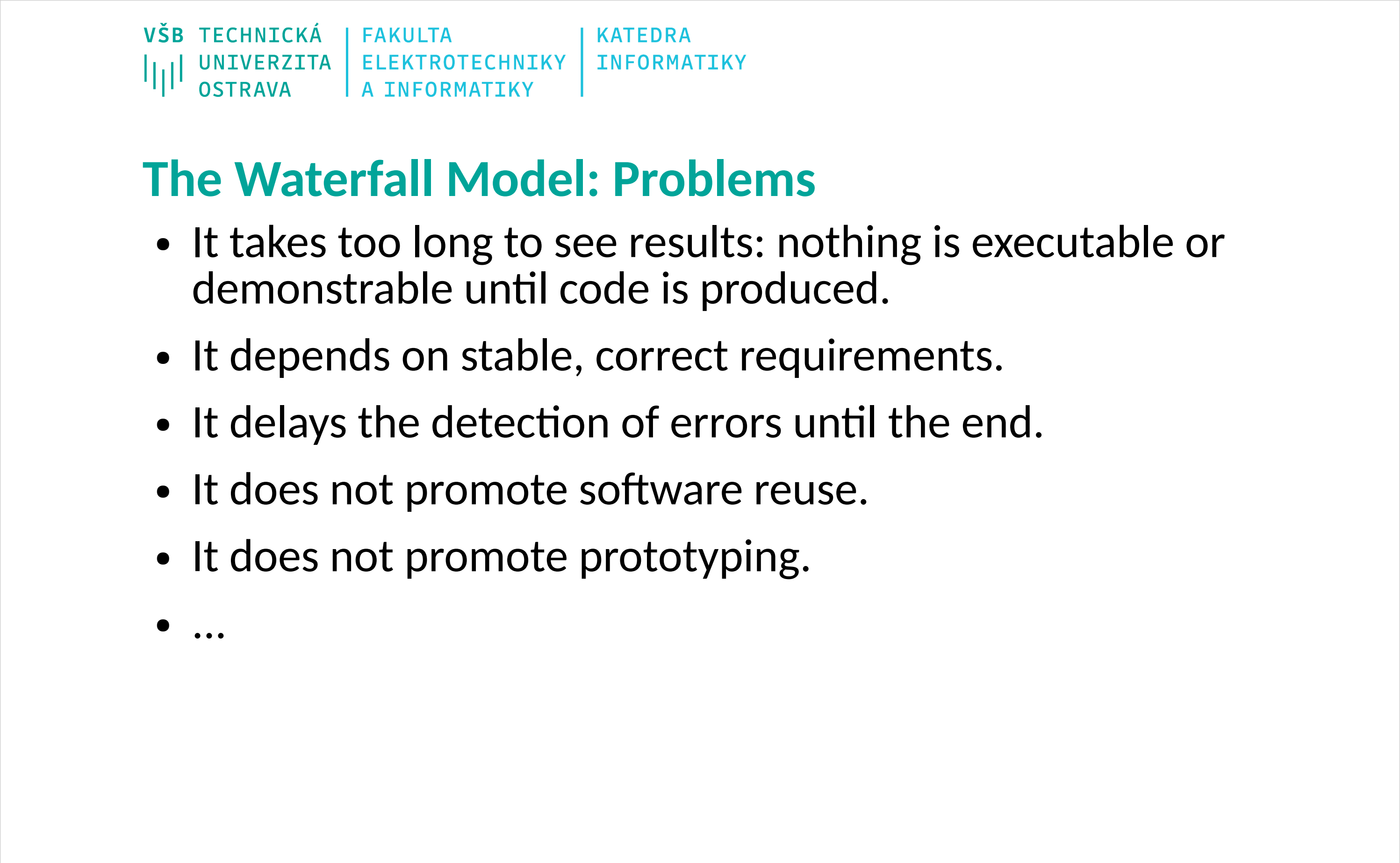
- The reasons to model a software process:
  - The model should reflect the goals of the development – building a high quality software system, finding faults in early stages of the development, meet the required budget, follow the prescribed schedule. When the model is built, the team may go through it and check whether all necessary components are there, or support some.
  - Each project has its process tailored. Every project is special. Software Process Development Model helps the team to understand where that tailoring is to occur.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

The Waterfall Process Model

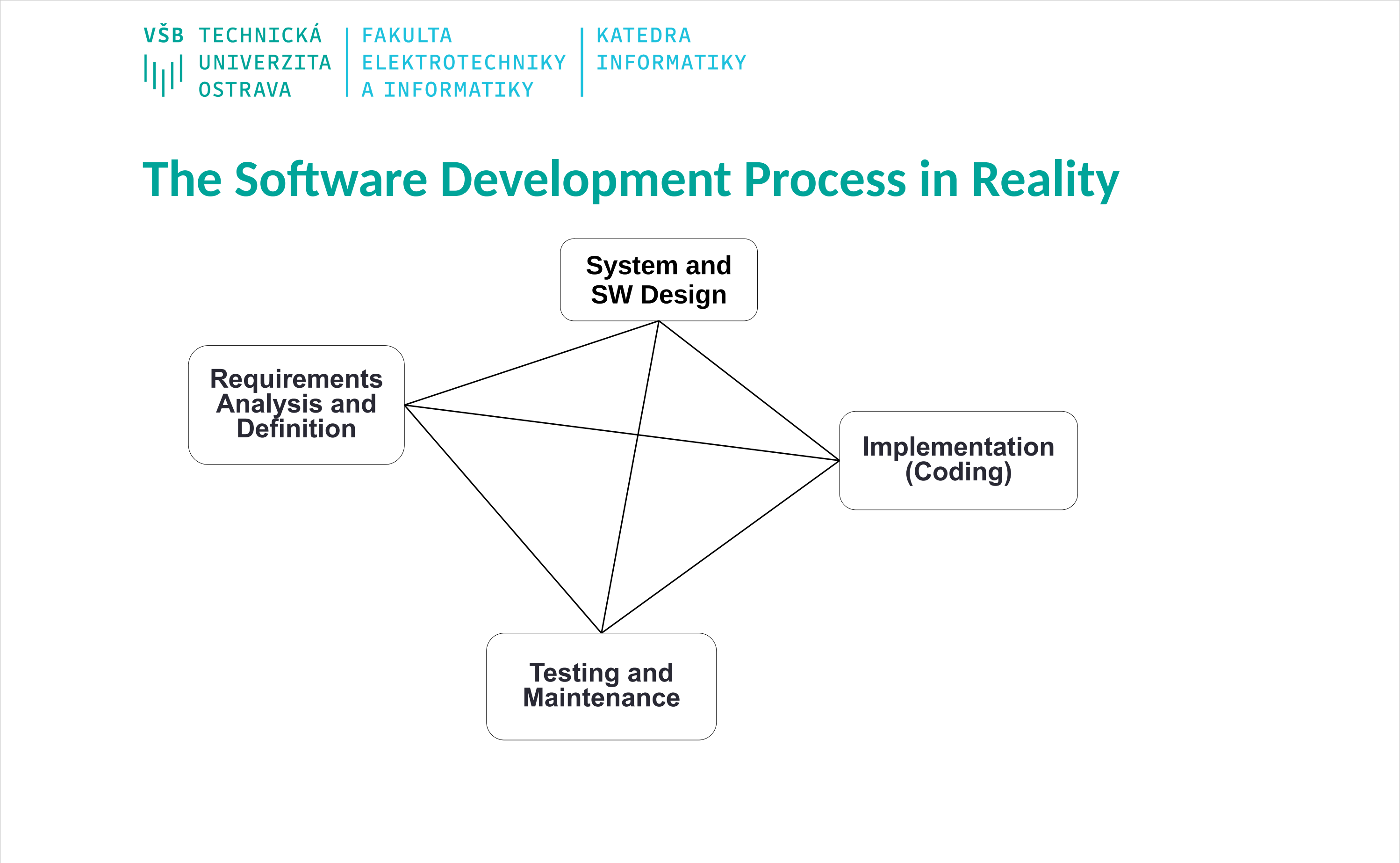
One of the first proposed models was the Waterfall Model. As the figure implies, the previous development stage should be completed before the next phase begins. Requirements are elicited from the customer and analyzed. System architecture is developed and the modules are designed. Then the system is implemented. After that the system goes to testing, it is installed and maintained. The model is easy to understand and describes all necessary prescriptions for the software development process.



The Waterfall Model: Problems

- It takes too long to see results: nothing is executable or demonstrable until code is produced.
- It depends on stable, correct requirements.
- It delays the detection of errors until the end.
- It does not promote software reuse.
- It does not promote prototyping.
- …

The Waterfall Model: Problems
It takes too long to see results: nothing is executable or demonstrable until code is produced.
It depends on stable, correct requirements.
It delays the detection of errors until the end.
It does not promote software reuse.
It does not promote prototyping.
The biggest problem is that the model does not reflect the way in which the code has been developed. Unless the problem is very well understood, the software is usually developed in iterations.
That is (also) the main issue that the model cannot solve. Of course, there is a simple solution to that but it is provided by other models.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
KATEDRA INFORMATIKY

The Software Development Process in Reality

System and SW Design

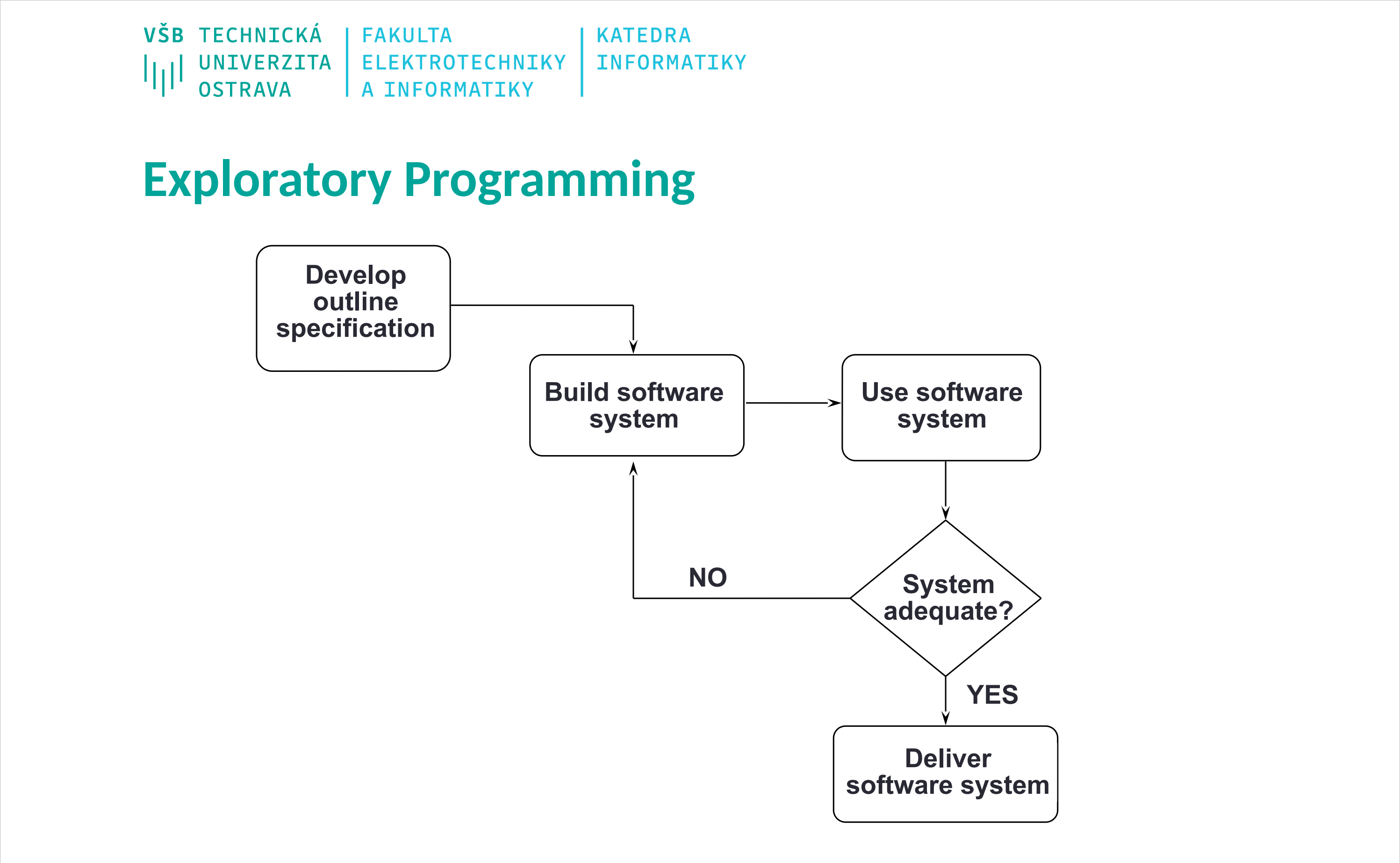
Requirements Analysis and Definition

Implementation (Coding)

Testing and Maintenance

The Software Development Process in Reality

If the developers use a simple waterfall model or do not use a model at all, the reality might look messy. In reality the software development process is a mess, the developer switches from one discipline to another. The process is uncontrolled and the developers try to do what they think is (the) best for the progress. The result is a complete mess (chaos). They try to gather some requirements, once they get them they attempt to design, they realize that the design need some repair of previously written code, that code must be redesigned etc.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Exploratory Programming

Develop outline specification

Build software system

Use software system

System adequate?

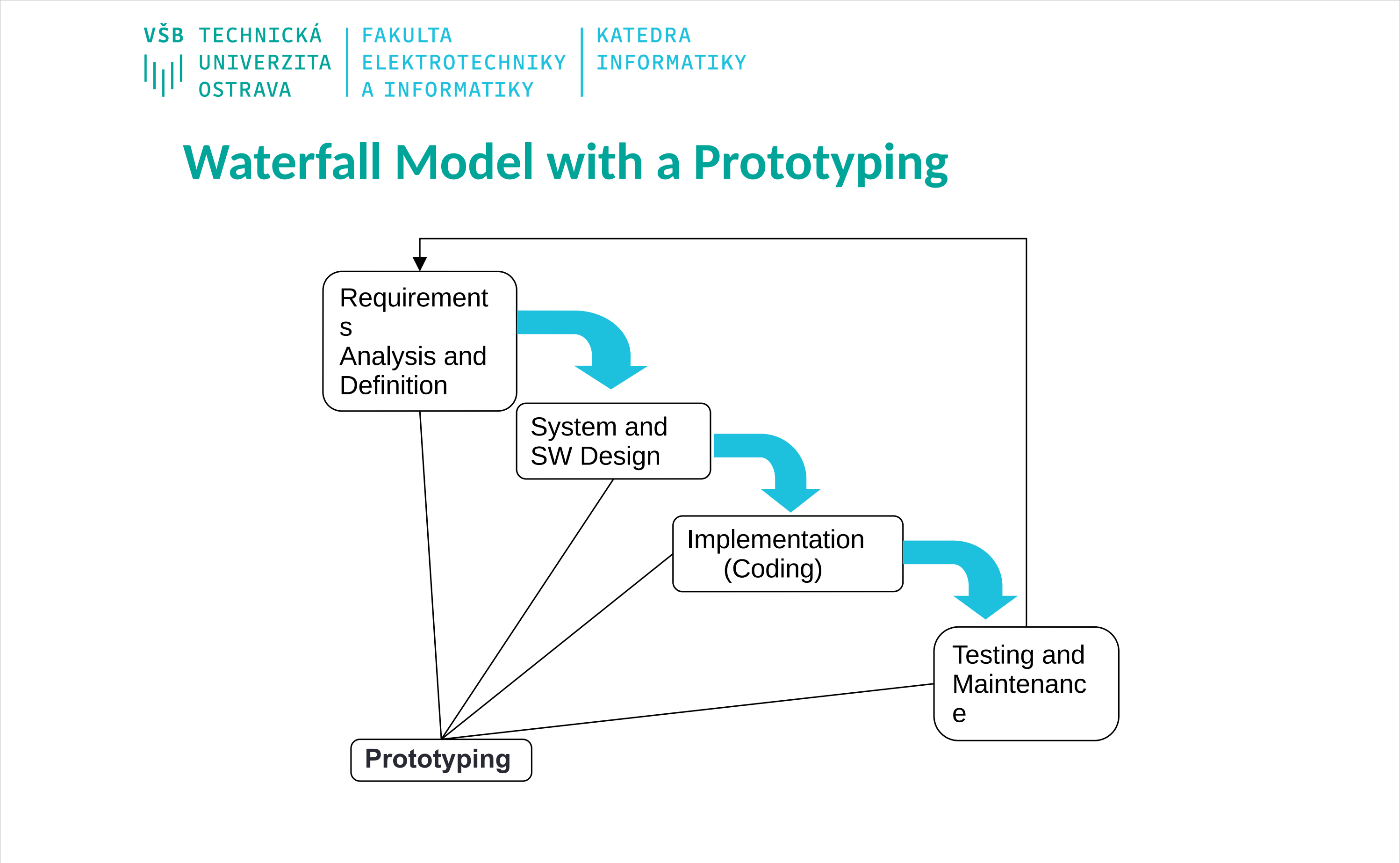
NO

YES

Deliver software system

Exploratory Programming

Exploratory programming might be one of the solutions. Of course, this approach is a very abstract and describes only the very basics. The idea is that the specification is developed and after that a system is built. When the system is built somebody uses or tests it and answers the question whether it does what it should (what it is supposed to). If not, the system is rebuilt until it does what is expected by the user or tester. This approach is something like a predecessor of the prototyping approach. Prototyping will be explained later.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Waterfall Model with a Prototyping

Requirement
s
Analysis and
Definition

System and
SW Design

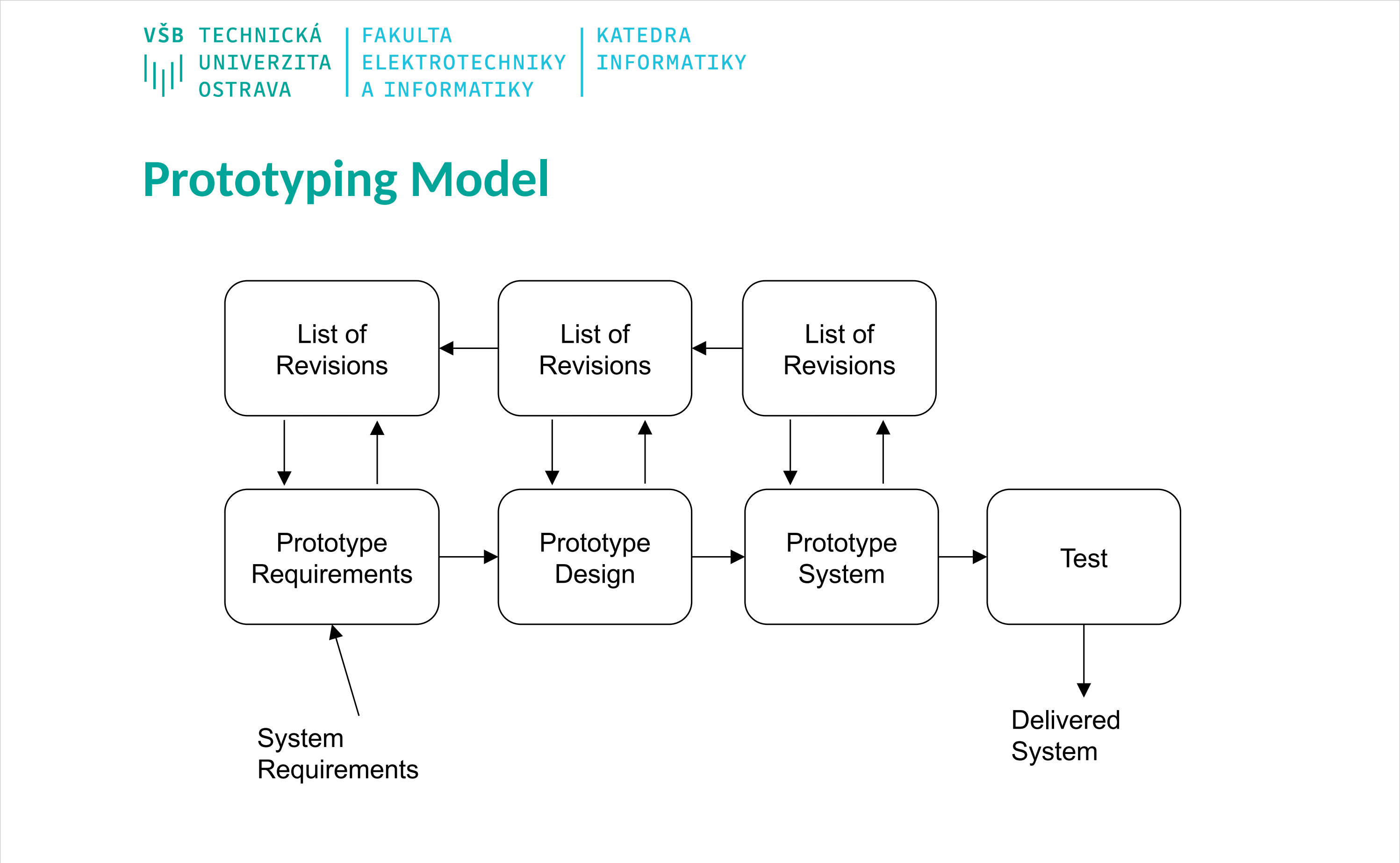
Implementation
(Coding)

Testing and
Maintenanc
e

Prototyping

Waterfall Model with a Prototyping

One of the easiest solutions to the “reality” problem is to introduce a prototyping to the waterfall development process. The software is made in small steps. The prototype is made in every stage – discipline, the prototype is evaluated and then if the prototype does what was expected from that part of the system, the next one is added. If not, the prototype is being rebuilt until appropriate solution is found. Then the other functionality is added.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Prototyping Model

List of Revisions

List of Revisions

List of Revisions

Prototype Requirements

Prototype Design

Prototype System

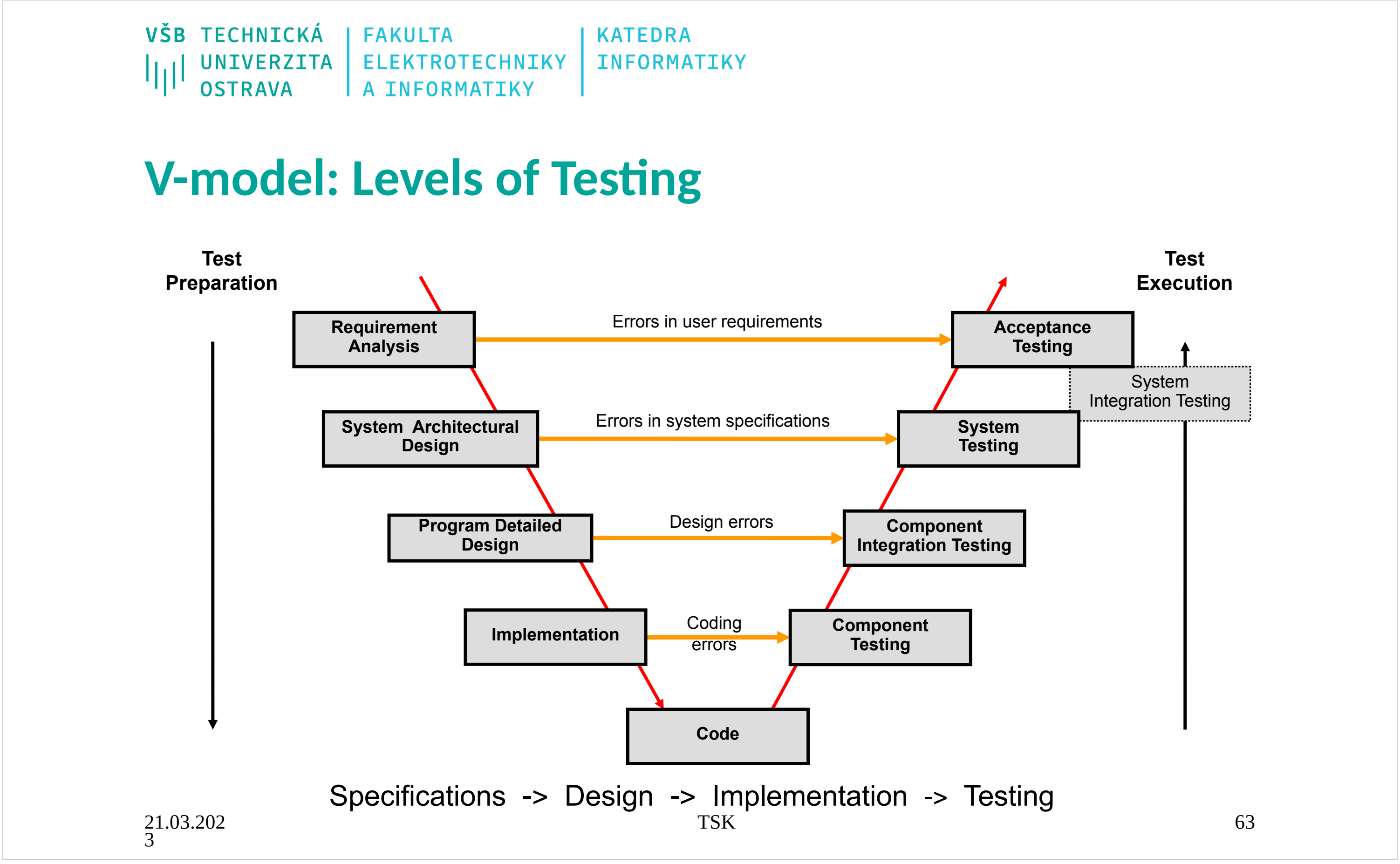
Test

System Requirements

Delivered System

Prototyping model.

We have presented the way the waterfall model can be connected to the prototyping model in the previous slide. On the other hand the prototyping model can itself be a base for a good process model. The model is based on the fact that the requirements are implemented as soon as possible and the result is seen very quickly. Requirements are elicited, then the design of the prototype is made and the system is implemented. Every discipline is revised until the result is acceptable to all customers, users and developers.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

V-model: Levels of Testing

Test Preparation

Test Execution

Requirement Analysis

System Architectural Design

Program Detailed Design

Implementation

Code

Component Testing

Component Integration Testing

System Testing

Acceptance Testing

System Integration Testing

Errors in user requirements

Errors in system specifications

Design errors

Coding errors

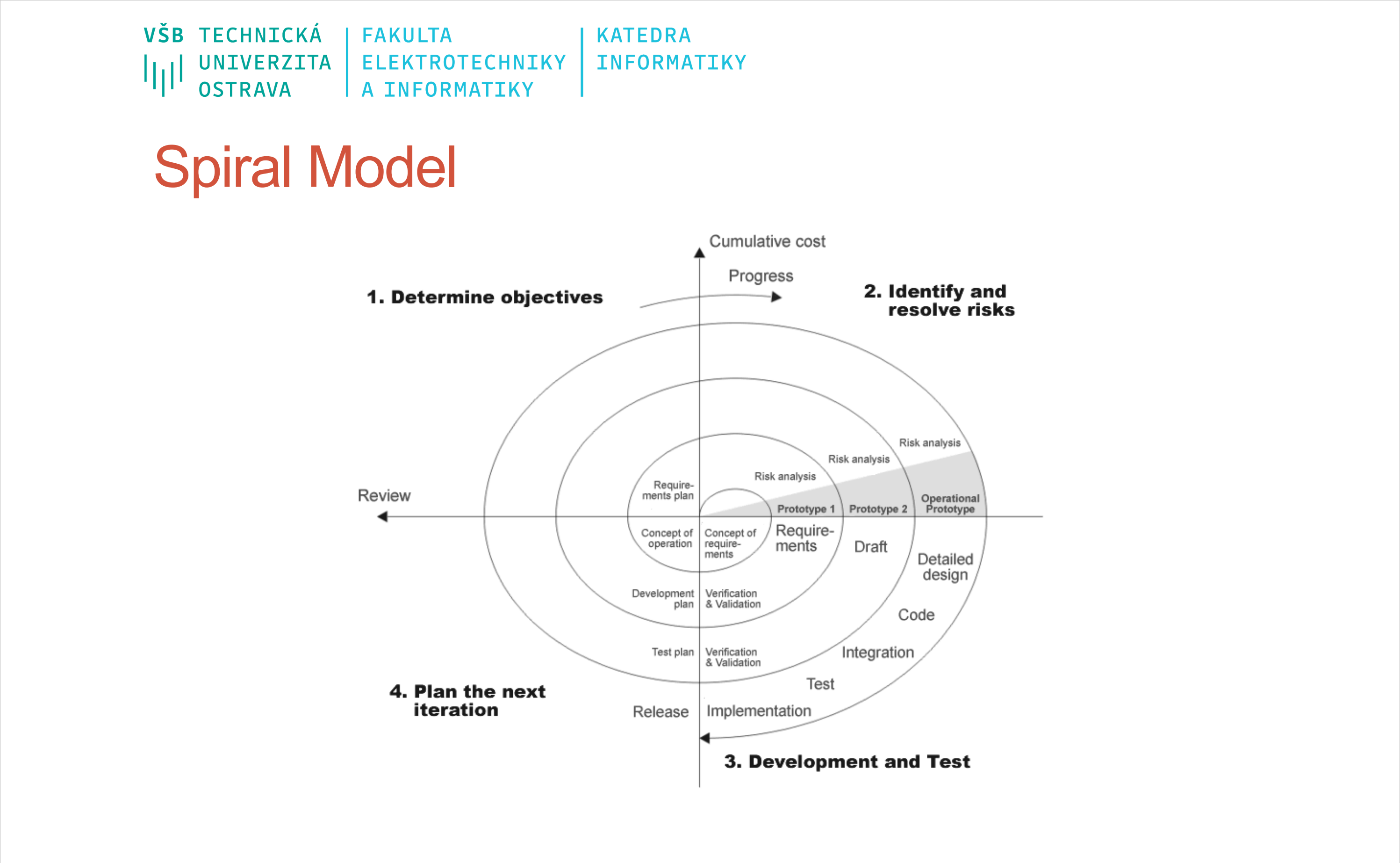
Specifications -> Design -> Implementation -> Testing

21.03.2023 TSK 63

For each stage in the model there are deliverables to the next stage, both development and testing. Such a delivery is an example of a baseline.

For example, when the user requirements are ready, they are delivered both to the next development stage and to the corresponding test level, i.e. acceptance testing. The user requirements will be used as input to the system specification (where the system requirements will be the deliverable to the next stage) and the acceptance test design.

Note that this is a simplified model. In reality, the arrows should point in both directions since each stage naturally will find faults and give feedback to the previous stages.

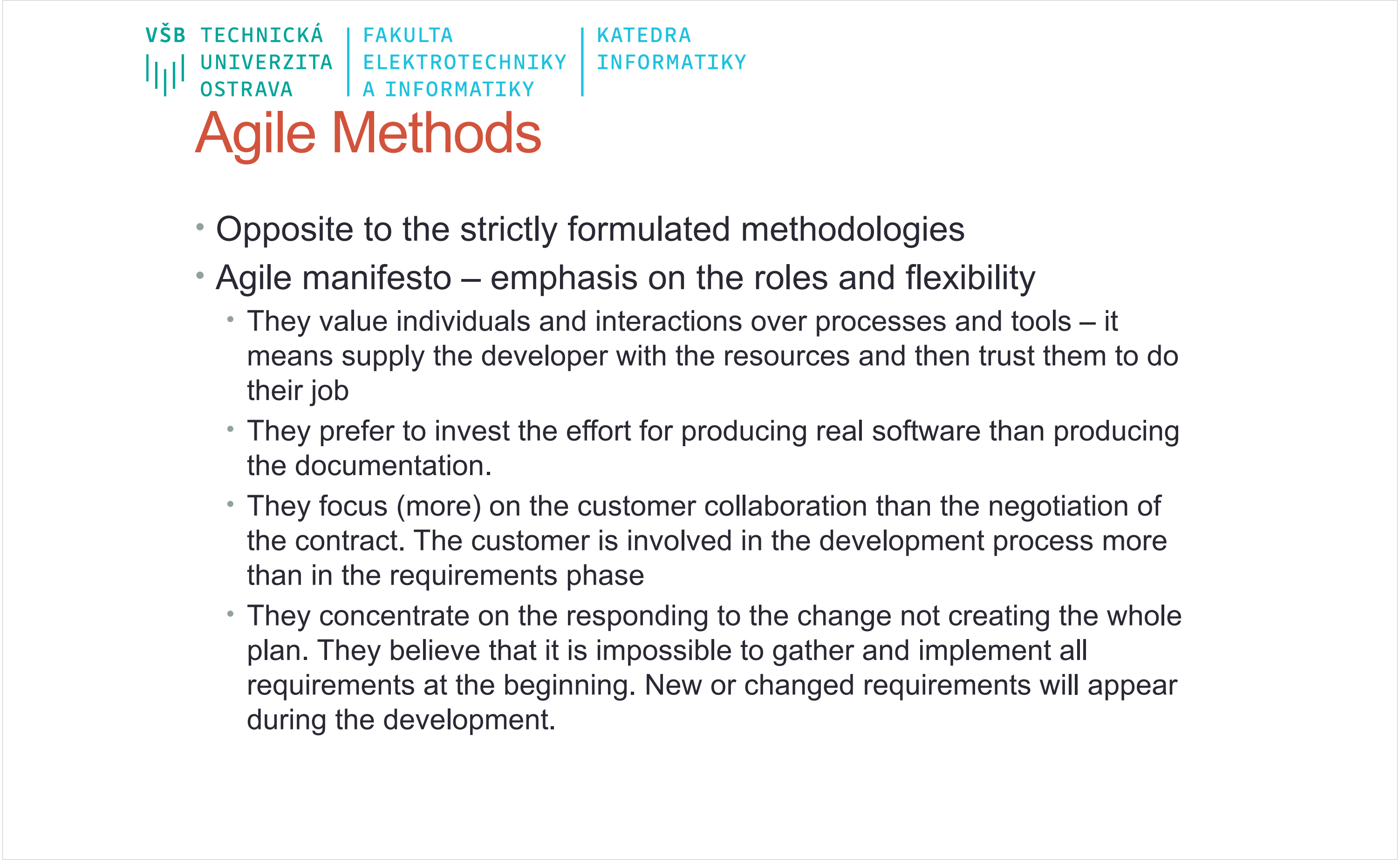


VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Spiral Model

Spiral Model.

Spiral model is based on the iterative development. The spiral starts with the requirements and the plan of the project and iteration and requirements plan. The concept is evaluated. The principle works for all types of disciplines. The first result and prototype is a concept, the second prototype are requirements, the third prototype is a software design and the next prototype is a detailed design. The concept is also based on the risks evaluation. For example, if the designer is not sure whether the user will like the type of GUI, they might prototype it and run a test of what is preferred.



Agile Methods

- Opposite to the strictly formulated methodologies
- Agile manifesto – emphasis on the roles and flexibility
  - They value individuals and interactions over processes and tools – it means supply the developer with the resources and then trust them to do their job
  - They prefer to invest the effort for producing real software than producing the documentation.
  - They focus (more) on the customer collaboration than the negotiation of the contract. The customer is involved in the development process more than in the requirements phase
  - They concentrate on the responding to the change not creating the whole plan. They believe that it is impossible to gather and implement all requirements at the beginning. New or changed requirements will appear during the development.

Extreme Programming (XP)

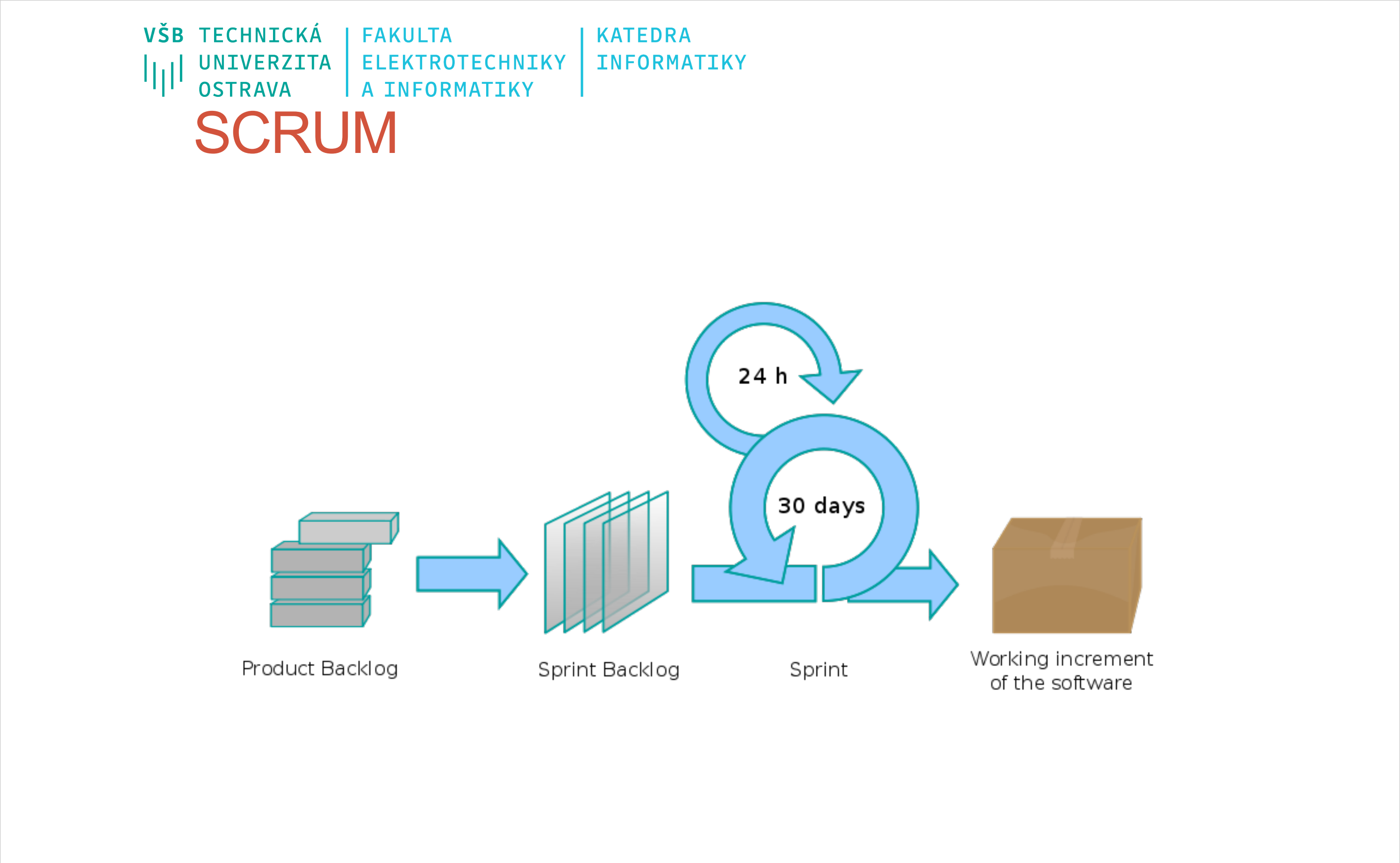
- Set of techniques to emphasize the creativity of developers and minimize the administration
- Extreme Programming is based on **12 principles**:
  - **The Planning Process** -- Quickly determine the scope of the next release by combining business priorities and technical estimates. As reality overtakes the plan, update the plan.
  - **Small Releases** -- The software is developed in small stages that are updated frequently, typically **every two weeks**.
  - **Metaphor** -- Guide all development with a simple shared story of how the whole system works.
  - **Simple Design** -- The software should include only the code that is necessary to achieve the desired results communicated by the customer at each stage in the process. Extra complexity is removed as soon as it is discovered.
  - **Testing** -- Testing is done consistently throughout the process. Programmers **design the tests first** and then write the software to fulfill the requirements of the test. The customer also provides acceptance tests at each stage to ensure the desired results are achieved.
  - **Refactoring** -- XP programmers improve the design of the software through every stage of development instead of waiting until the end of the development and going back to correct flaws.
  - **Pair Programming** -- All code is written by a pair of programmers working at the same machine.
  - **Collective Ownership** -- Anyone can change any code anywhere in the system at any time.
  - **Continuous Integration** -- The XP team integrates and builds the software system multiple times per day to keep all the programmers at the same stage of the development process at once.
  - **40-Hour Week** -- The XP team does not work excessive overtime to ensure that the team remains well-rested, alert and effective.
  - **On-Site Customer** -- The XP project is directed by the customer who is available all the time to answer questions, set priorities and determine requirements of the project.
  - **Coding Standard** -- The programmers all write code in the same way. This allows them to work in pairs and to share ownership of the code.

Crystal

- Is based on the fact that every project needs a different set of polices, conventions and methodologies. Crystal is based on people’s trust
- People largely influence the quality of the software. The quality of the projects and processes improve as the quality of the people involved increases.
- Productivity increases through better communication and frequent delivery. There is less need for intermediate products.

Adaptive Software Development

- Has six basic principles
- Mission that acts as a guideline – setting up a goal but does not prescribe how to get there
- Customer value – futures are viewed through the customer value
- The project is organized around building a components around the features
- Iteration is important – change is not viewed as a correction but as an adjustment to the reality of the software development.
- Fixed delivery times – force the developers to plan the scope of an iteration
- Risk is embraced – the developers tackle the hardest problems first



SCRUM

SCRUM

SCRUM is iterative software development process that is considered as an agile approach.

Scrum is based on the principles of extreme programming and extends them. SCRUM can be used for the development of the process, maintenance of the process or software management.

The main principles of scrum are:

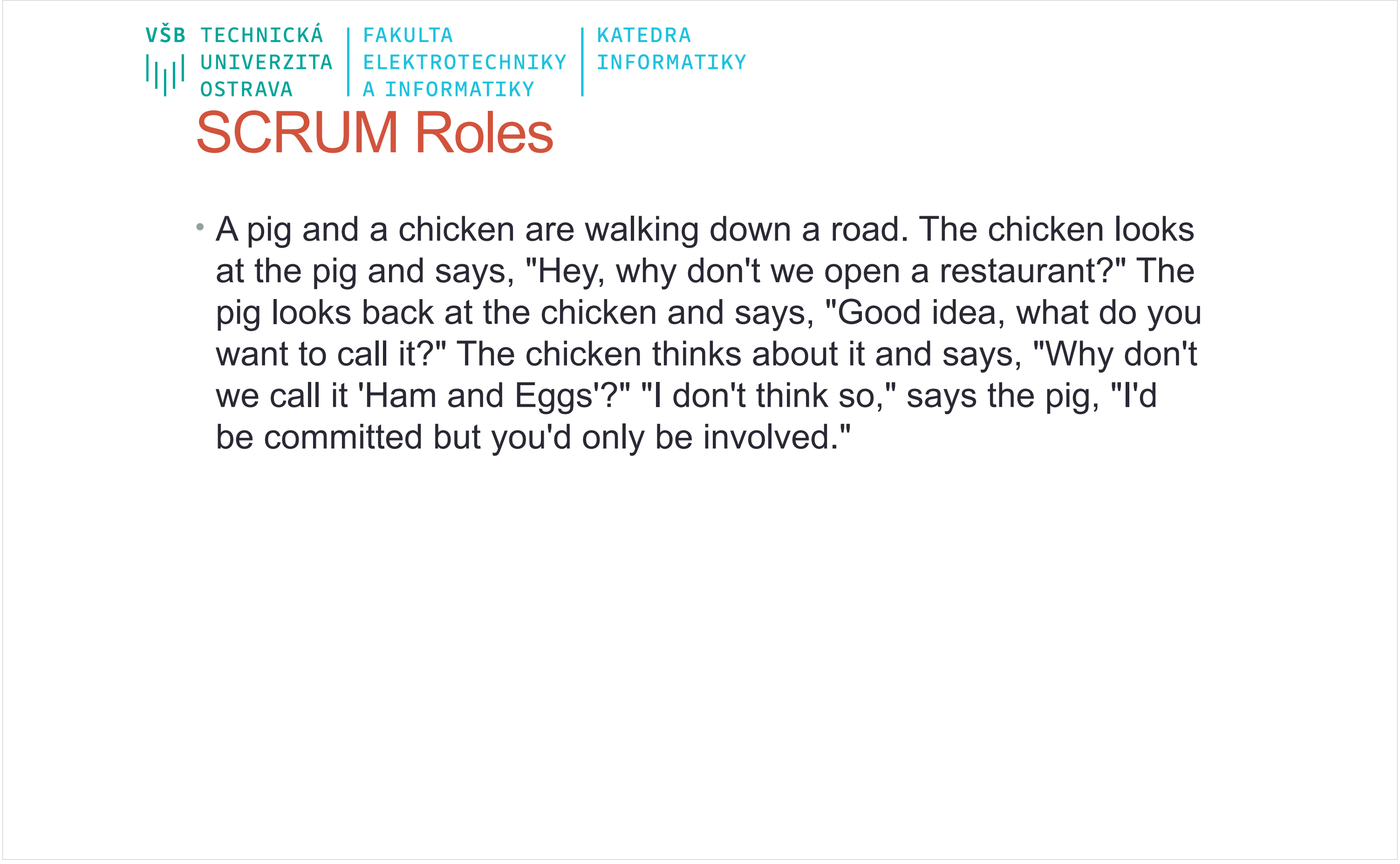
There is no manager in the usual sense.

Team activities are controlled by the SCRUM Master.

Team is involved in the planning.

Tasks are not assigned to the team members. Team members take them themselves.

15 minutes scrum meeting every day.

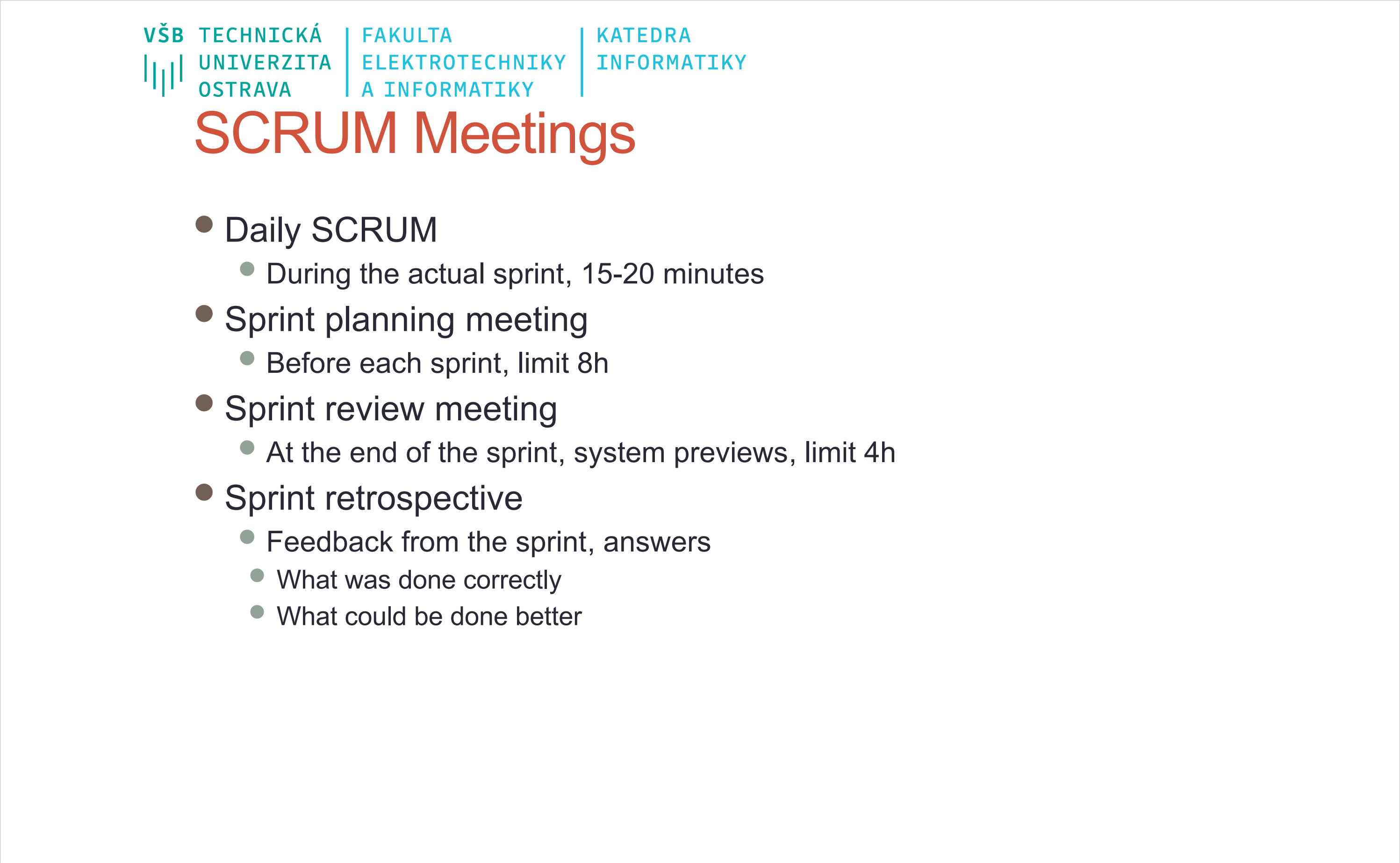


SCRUM Roles

- A pig and a chicken are walking down a road. The chicken looks at the pig and says, "Hey, why don't we open a restaurant?" The pig looks back at the chicken and says, "Good idea, what do you want to call it?" The chicken thinks about it and says, "Why don't we call it 'Ham and Eggs'?" "I don't think so," says the pig, "I'd be committed but you'd only be involved."

This joke is describe the basic idea of the scrum - there are two types of role groups – those who are committed and those that are only involved.

Pigs – scrum master, project owner, team. Chickens – users, stakeholders, managers



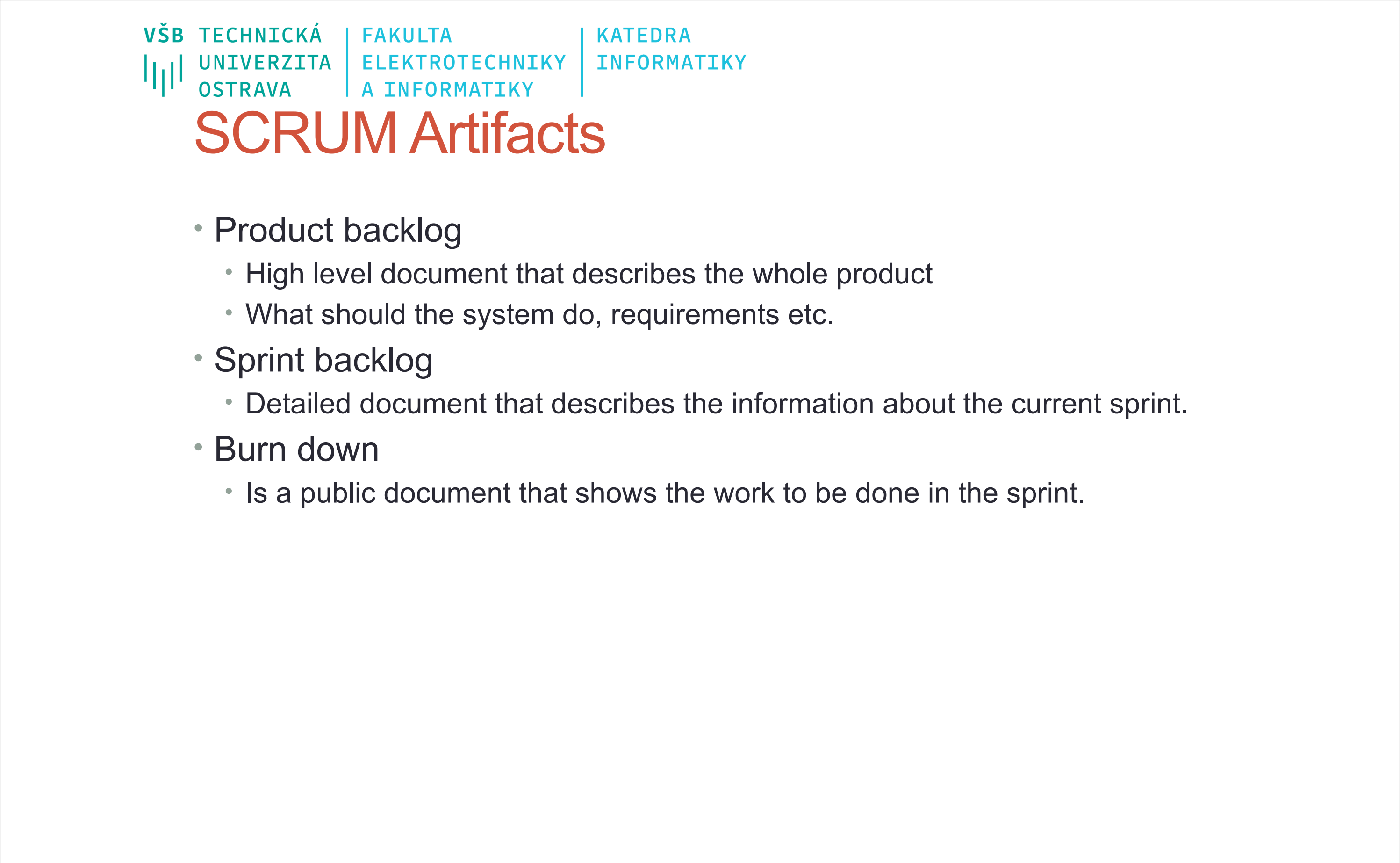
VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

SCRUM Meetings

- Daily SCRUM
  - During the actual sprint, 15-20 minutes
- Sprint planning meeting
  - Before each sprint, limit 8h
- Sprint review meeting
  - At the end of the sprint, system previews, limit 4h
- Sprint retrospective
  - Feedback from the sprint, answers
    - What was done correctly
    - What could be done better

There are several types of scrum meetings. Those meetings are held at the specific times and for a limited amount of time. For example the best daytime to do the daily scrum is a morning, when everybody starts his work or is at the end of previous work or in the middle of the problem. Everyone shares the information with (the) others. Sometimes the team members need some help from other team members.

- Daily SCRUM
  - During the actual sprint, 15-20 minutes
- Sprint planning meeting
  - Before each sprint, limit 8h
- Sprint review meeting
  - At the end of the sprint, system previews, limit 4h
- Sprint retrospective
  - Feedback from the sprint, answers
  - What was done correctly
  - What could be done better



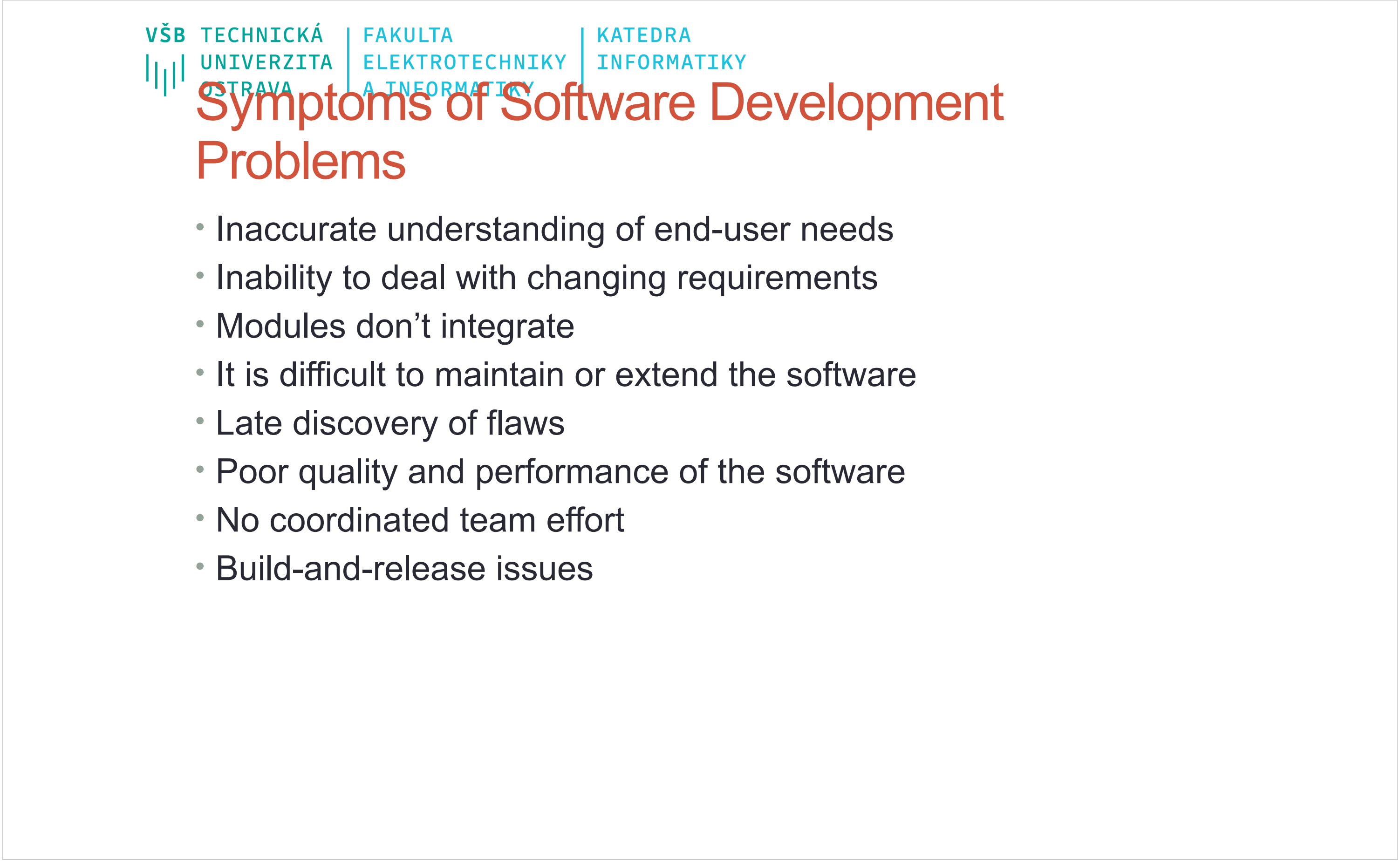
VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

SCRUM Artifacts

- Product backlog
  - High level document that describes the whole product
  - What should the system do, requirements etc.
- Sprint backlog
  - Detailed document that describes the information about the current sprint.
- Burn down
  - Is a public document that shows the work to be done in the sprint.

Scrum Artifacts

There are three basic scrum artifacts. The basic one is the product backlog that contains the requirements and the description what the software should do. Team selects the subset of the requirements set from this document and describes it in more detail in the sprint backlog. The tasks are described in the burn down document.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Symptoms of Software Development Problems

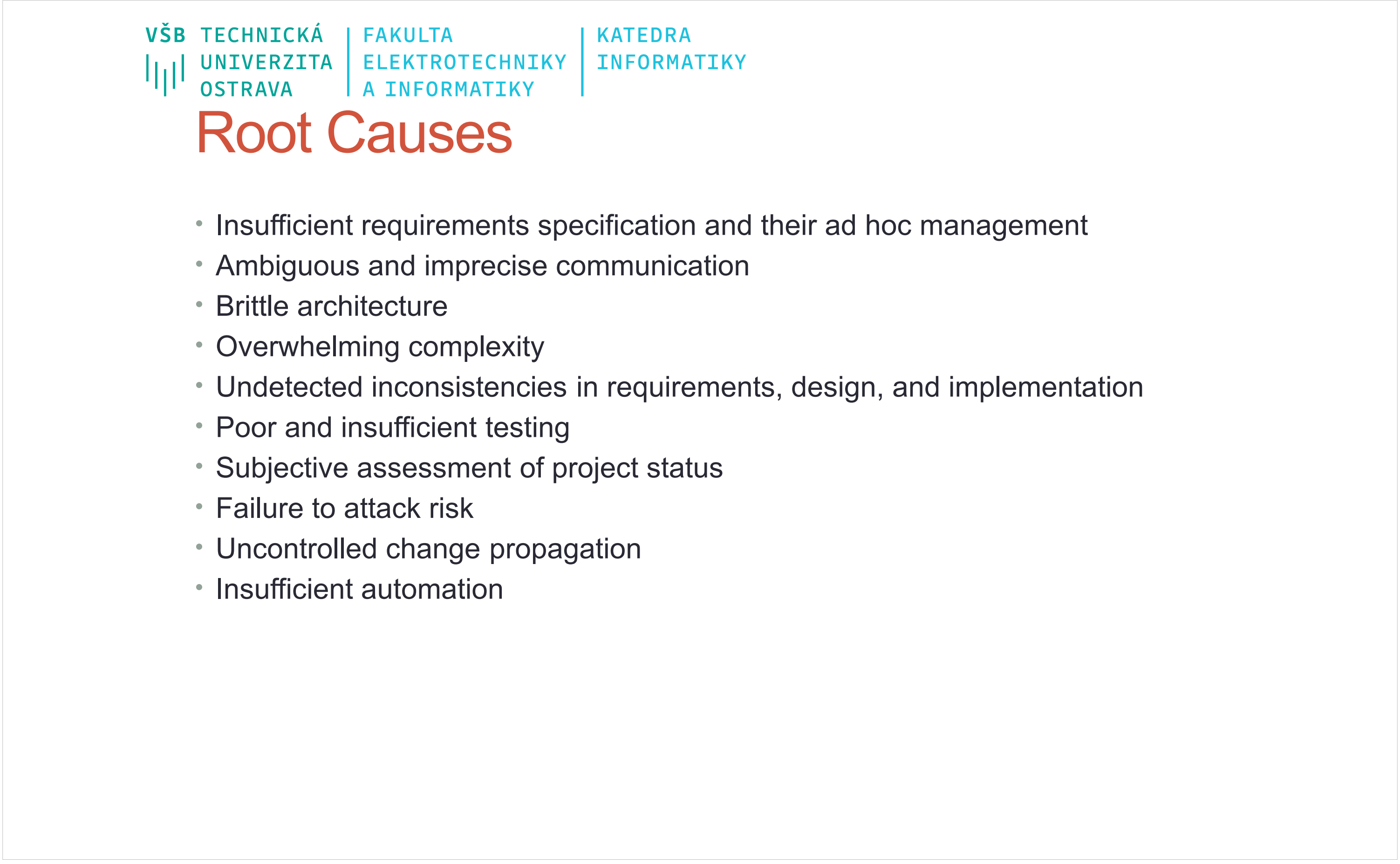
- Inaccurate understanding of end-user needs
- Inability to deal with changing requirements
- Modules don’t integrate
- It is difficult to maintain or extend the software
- Late discovery of flaws
- Poor quality and performance of the software
- No coordinated team effort
- Build-and-release issues

Let’s see the other development processes the so called rigid processes, why they started.

When the software system was developed, many project had some of these symptoms. When the symptom appeared the solution could be found but the roots of such a symptom must be treated (addressed) not the result.

Symptoms of Software Development Problems

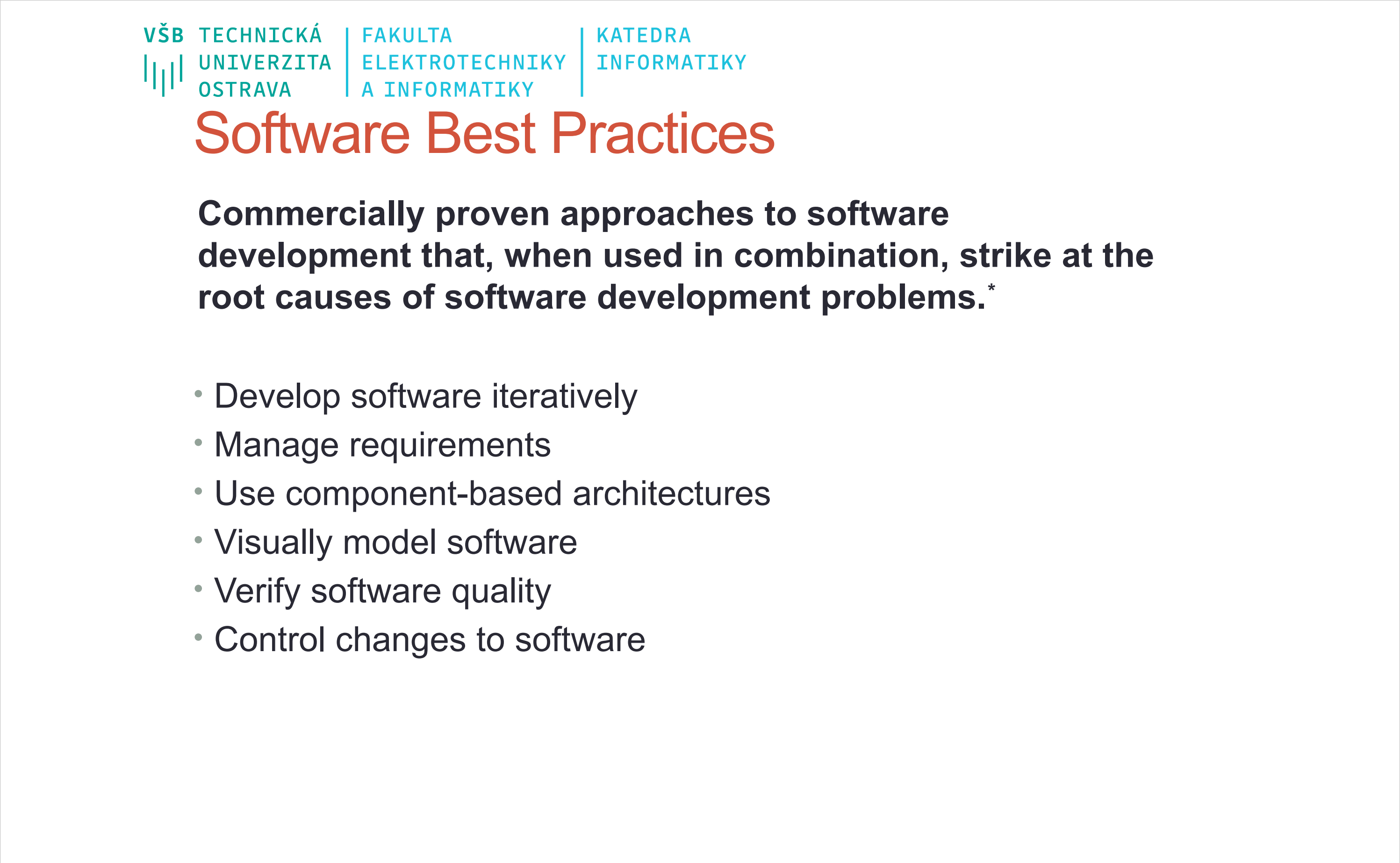
Inaccurate understanding of end-user needs
Inability to deal with changing requirements
Modules don’t integrate
It is difficult to maintain or extend the software
Late discovery of flaws
Poor quality and performance of the software
No coordinated team effort
Build-and-release issues



Root Causes

- Insufficient requirements specification and their ad hoc management
- Ambiguous and imprecise communication
- Brittle architecture
- Overwhelming complexity
- Undetected inconsistencies in requirements, design, and implementation
- Poor and insufficient testing
- Subjective assessment of project status
- Failure to attack risk
- Uncontrolled change propagation
- Insufficient automation

To treat these root causes eliminates the symptoms and enables to develop and maintain software in a repeatable and predictable way.



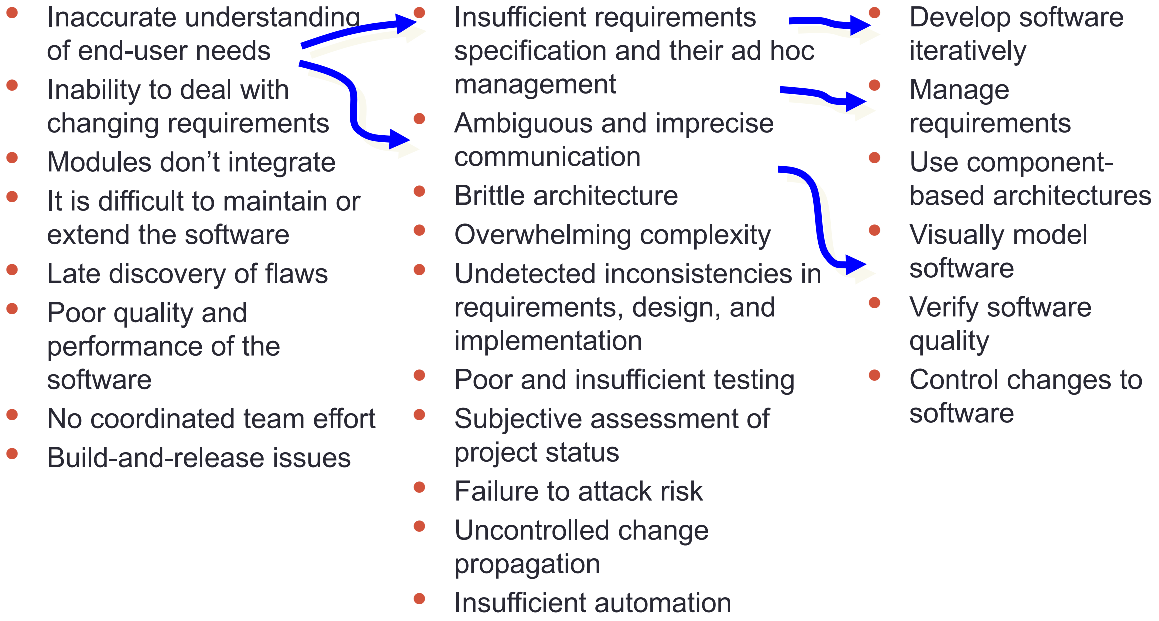
Software Best Practices

Commercially proven approaches to software development that, when used in combination, strike at the root causes of software development problems.*

- Develop software iteratively
- Manage requirements
- Use component-based architectures
- Visually model software
- Verify software quality
- Control changes to software

Techniques exist that are proved to be commercially successful in eliminating the problems arising from root causes. When these best practices are followed the root causes are solved and no symptoms should occur.

Tracing Symptoms to Root Causes and Best Practices



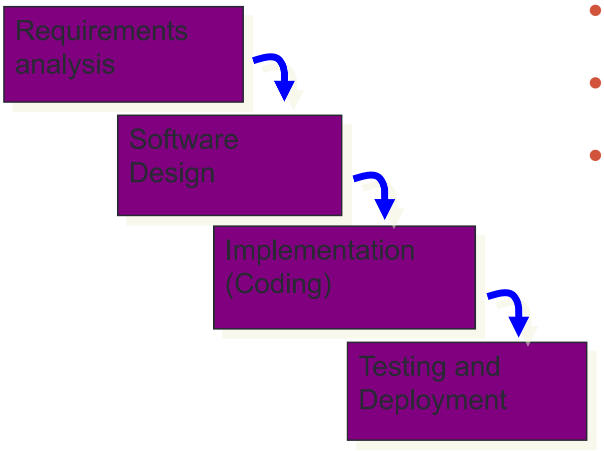
- Inaccurate understanding of end-user needs
- Inability to deal with changing requirements
- Modules don’t integrate
- It is difficult to maintain or extend the software
- Late discovery of flaws
- Poor quality and performance of the software
- No coordinated team effort
- Build-and-release issues

- Insufficient requirements specification and their ad hoc management
- Ambiguous and imprecise communication
- Brittle architecture
- Overwhelming complexity
- Undetected inconsistencies in requirements, design, and implementation
- Poor and insufficient testing
- Subjective assessment of project status
- Failure to attack risk
- Uncontrolled change propagation
- Insufficient automation

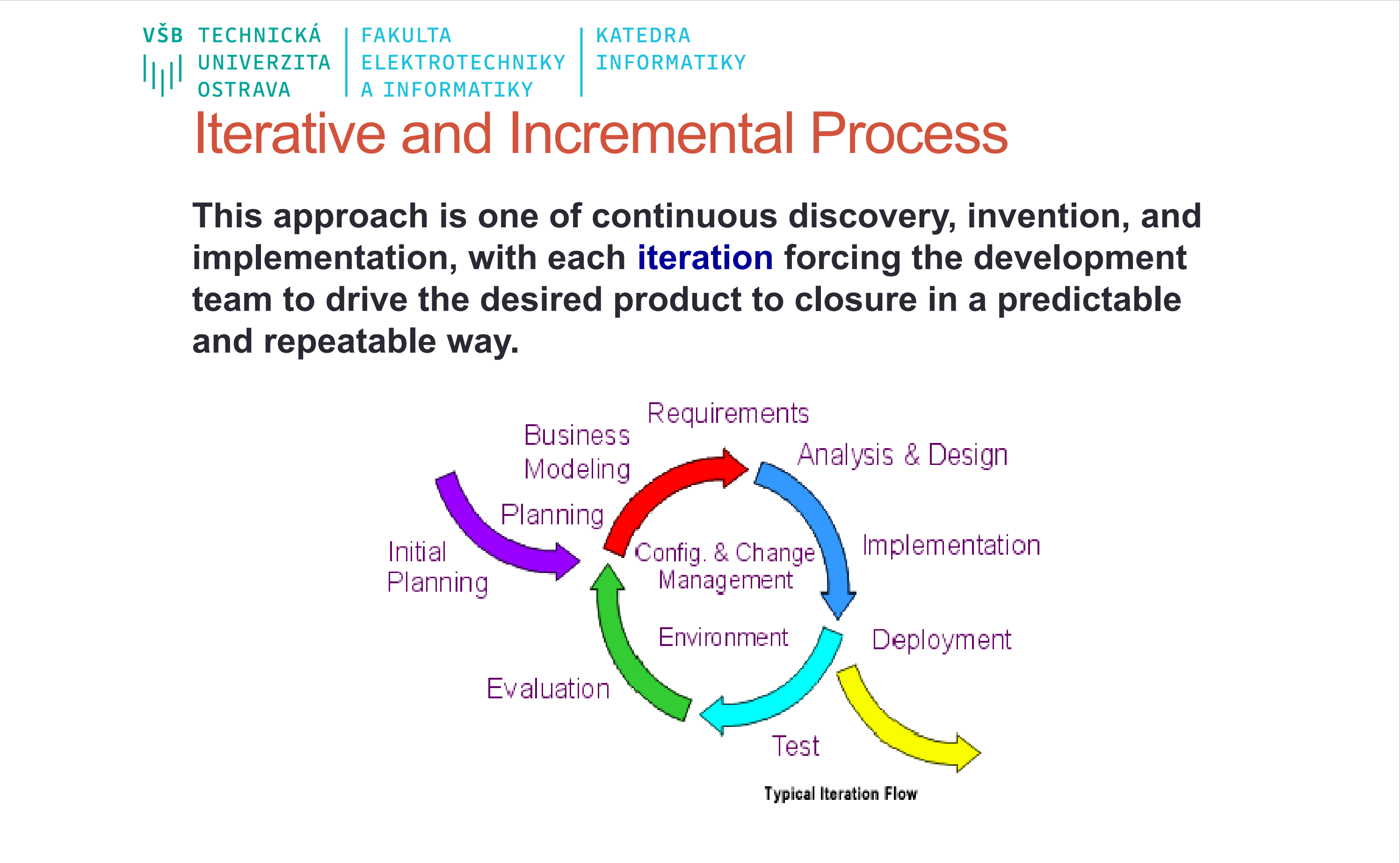
- Develop software iteratively
- Manage requirements
- Use component-based architectures
- Visually model software
- Verify software quality
- Control changes to software

Develop Software Iteratively

Classic software development processes follow the waterfall lifecycle. Development proceeds linearly from requirements analysis, through design, implementation, and testing.



- It takes too long to see results.
- It depends on stable, correct requirements.
- It delays the detection of errors until the end.
- It does not promote software reuse and prototyping.



Iterative and Incremental Process

This approach is one of continuous discovery, invention, and implementation, with each iteration forcing the development team to drive the desired product to closure in a predictable and repeatable way.

Typical Iteration Flow

An iteration is a complete development loop resulting in a release (internal or external) of an executable product, a subset of the final product under development, which grows incrementally from iteration to iteration to become the final system.

Solutions to Root Causes

- Serious misunderstandings are made visible early
- This approach enables user feedback
- The development team is forced to focus on most critical issues
- Continuous testing enables an objective assessment of the project status
- Inconsistencies among requirements, design, and implementation are detected early
- The workload of the team is spread more evenly during project lifecycle
- The team can leverage lessons learned and improve the process
- Stakeholders can be given concrete evidence of the project status

Manage Requirements

A requirement is a condition or capability a system must have.

- It is a real problem to capture all requirements before the start of development. Requirements change during project lifecycle. Understanding and identifying of requirements is a continuous process.
- The active management of requirements is about following three activities: eliciting, organizing, and documenting the system required functionality and constraints.

Solutions to Root Causes

- A disciplined approach is built into requirements management
- Communication is based on defined requirements
- Requirements have to be prioritized, filtered, and traced
- An objective assessment of functionality is possible
- Inconsistencies are detected more easily
- With a tool support it is possible to provide a repository for system requirements

Use Component-Based Architectures

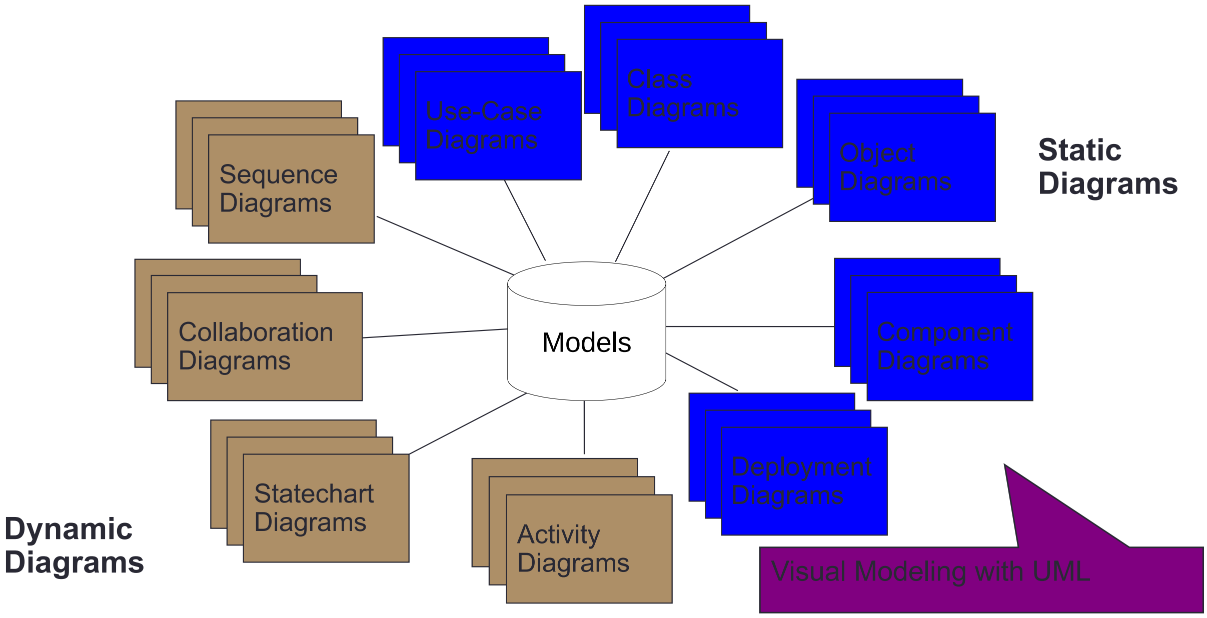
- Component-Based Development is an important approach allowing to build resilient software architecture because it enables the reuse of components from many available sources. Components make reuse possible on a larger scale, enabling systems to be composed from existing parts, off-the-shelf third-party parts, and a few new parts that address the specific domain and integrate the other parts together.
- Iterative approach involves the continuous evolution of the system architecture. Each iteration produces an executable architecture that can be measured, tested, and evaluated against the system requirements.

Solutions to Root Causes

- Components facilitate resilient architectures
- Modularity enables a clear separation of system elements that are subject to change
- Reuse is facilitated by leveraging standardized frameworks (COM, CORBA, EJB …) and commercially available components
- Components provide a natural basis for configuration management
- Visual modeling tools provide automation for component-based development

Visually Model Software

A model is a simplification of reality that completely describes a system from a particular perspective.



Models

Use-Case Diagrams

Class Diagrams

Object Diagrams

Component Diagrams

Deployment Diagrams

Static Diagrams

Sequence Diagrams

Collaboration Diagrams

Statechart Diagrams

Activity Diagrams

Dynamic Diagrams

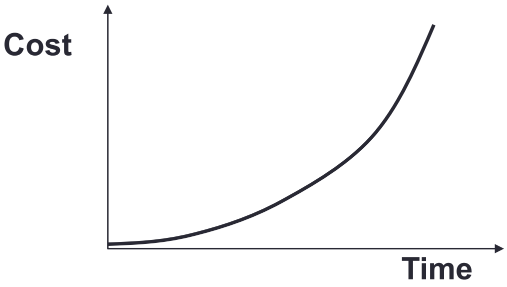
Visual Modeling with UML

Solutions to Root Causes

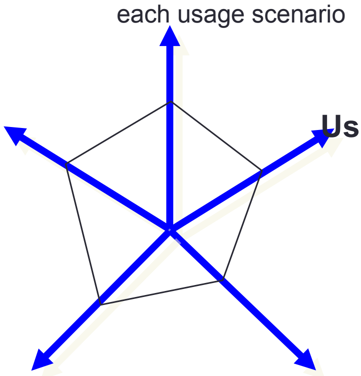
- Use cases and scenarios unambiguously specify behavior
- Software design is unambiguously captured by models
- Details can be hidden when needed
- Unambiguous design discovers inconsistencies more readily
- Application quality begins with good design
- Visual modeling tools provide support for UML modeling

Continuously Verify Software Quality

- Software problems are exponentially more expensive to find and repair after deployment than beforehand.
- Verifying system functionality involves creating test for each key scenario that represents some aspect of required behavior.
- Since the system is developed iteratively every iteration includes testing = continuous assessment of product quality.



Testing Dimensions of Quality



Functionality

Test the accurate workings of each usage scenario

Supportability

Test the ability to maintain and support application under production use

Usability

Test application from the perspective of convenience to end-user.

Performance

Test online response under average and peak loading

Reliability

Test that the application behaves consistently and predictably.

Solutions to Root Causes

- Project status assessment is made objective because test results are continuously evaluated
- This objective assessment exposes inconsistencies in requirements, design and implementation
- Testing and verification is focused on most important areas
- Defects are identified early and thus the costs of fixing them are reduced
- Automated testing tools provide testing for functionality, reliability, and performance

Control Changes to Software

- The ability to manage change - **making certain that each change is acceptable, and being able to track changes** - is essential in an environment in which change is inevitable.
- Maintaining traceability among elements of each release is essential for assessing and actively managing the impact of change.
- In the absence of disciplined control of changes, the development process degenerates rapidly into chaos.

Solutions to Root Causes

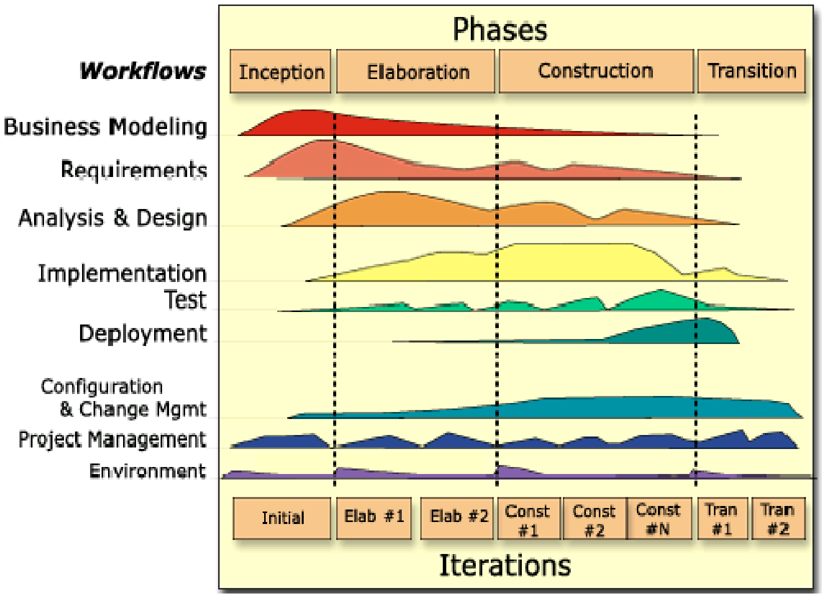
- The workflow of requirements changes is defined and repeatable
- Change requests facilitate clear communication
- Isolated workspaces reduce interferences among team members working in parallel
- Workspaces contain all artifacts, which facilitates consistency
- Change propagation is controlled
- Changes can be maintained in a robust system

The Rational Unified Process

The Rational Unified Process® (RUP) is a Software Engineering Process. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end-users, within a predictable schedule and budget.

- RUP is a process product. It is developed and maintained by Rational Software and integrated with its suite of software development tools available from IBM.
- RUP is a process framework that can be adapted and extended to suit the needs of an adopting organization.
- RUP captures many of best practices mentioned before (develop software iteratively, manage requirements, use component-based architectures, visually model software, continuously verify software quality, control changes to software).

Two Dimensions of the Process



RUP Overview Diagram

Two Dimensions of the Process

The rational unified process consist of phases – dynamic aspect of the process and workflows – static aspect of the process.

Static aspect of the process: how it is described in terms of activities, artifacts, workers and workflows – organization along content

Dynamic aspect of the process as it is enacted: it is expressed in terms of cycles, phases, iterations, and milestones – organization along time

Static structure of the process describes *who* is doing *what*, *how*, and *when*. The RUP is represented using following primary elements:

Roles: the *who*

Activities: the *how*

Artifact: the *what*

Workflow: the *when*

A discipline is the collection of above mentioned kinds of elements.

Cycles and Phases

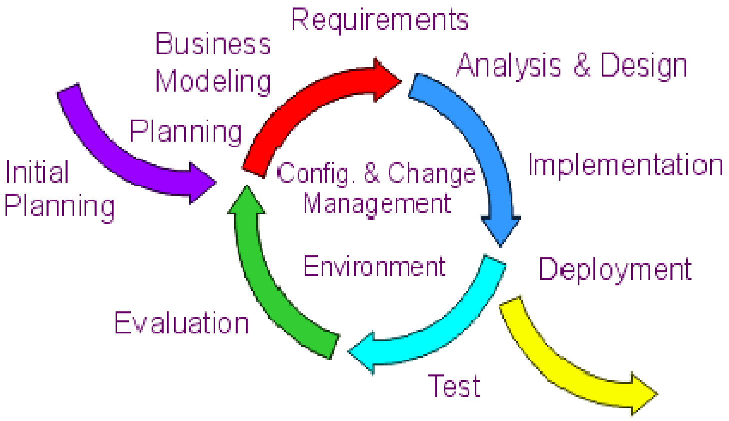
Each cycle results in a new release of the system, and each is a product ready for delivery. This product has to accommodate the specified needs.

The development cycle is divided in **four consecutive phases**

- **Inception:** a good idea is developed into a vision of the end product and the business case for the product is presented.
- **Elaboration:** most of the product requirements are specified and the system architecture is designed.
- **Construction:** the product is built – completed software is added to the skeleton (architecture)
- **Transition:** the product is moved to user community (beta testing, training …)

Iterations

Each phase can be further broken down into iterations. An iteration is a complete development loop resulting in a release (internal or external) of an executable product, a subset of the final product under development, which grows incrementally from iteration to iteration to become the final system.

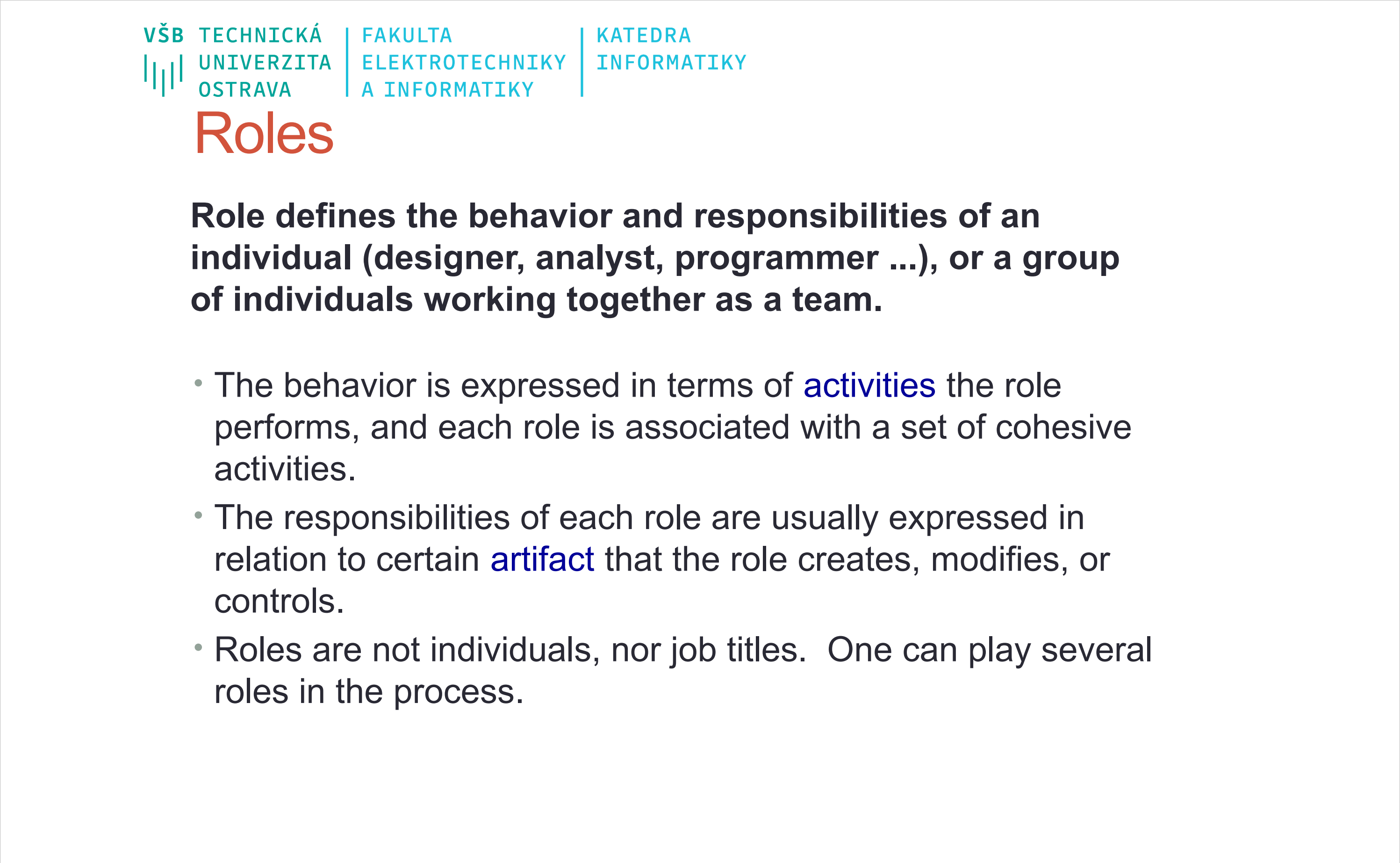


Typical Iteration Flow

Static Structure of the Process

A process describes **who** is doing **what**, **how**, and **when** using following modeling elements:

- **Workers** (Roles) define the behavior and responsibilities of an individual (designer, analyst, programmer ...), or a group of individuals working together as a team.
- **Artifacts** are things that are produced, modified, or used by a process (model, document, source code …).
- **Activities** are performed by workers to create or update some artifacts (review design, compile code, perform test …).
- **Workflows** are sequences of activities that produce results of observable value (business modeling, implementation …).



Roles

Role defines the behavior and responsibilities of an individual (designer, analyst, programmer ...), or a group of individuals working together as a team.

- The behavior is expressed in terms of activities the role performs, and each role is associated with a set of cohesive activities.
- The responsibilities of each role are usually expressed in relation to certain artifact that the role creates, modifies, or controls.
- Roles are not individuals, nor job titles. One can play several roles in the process.

Activities

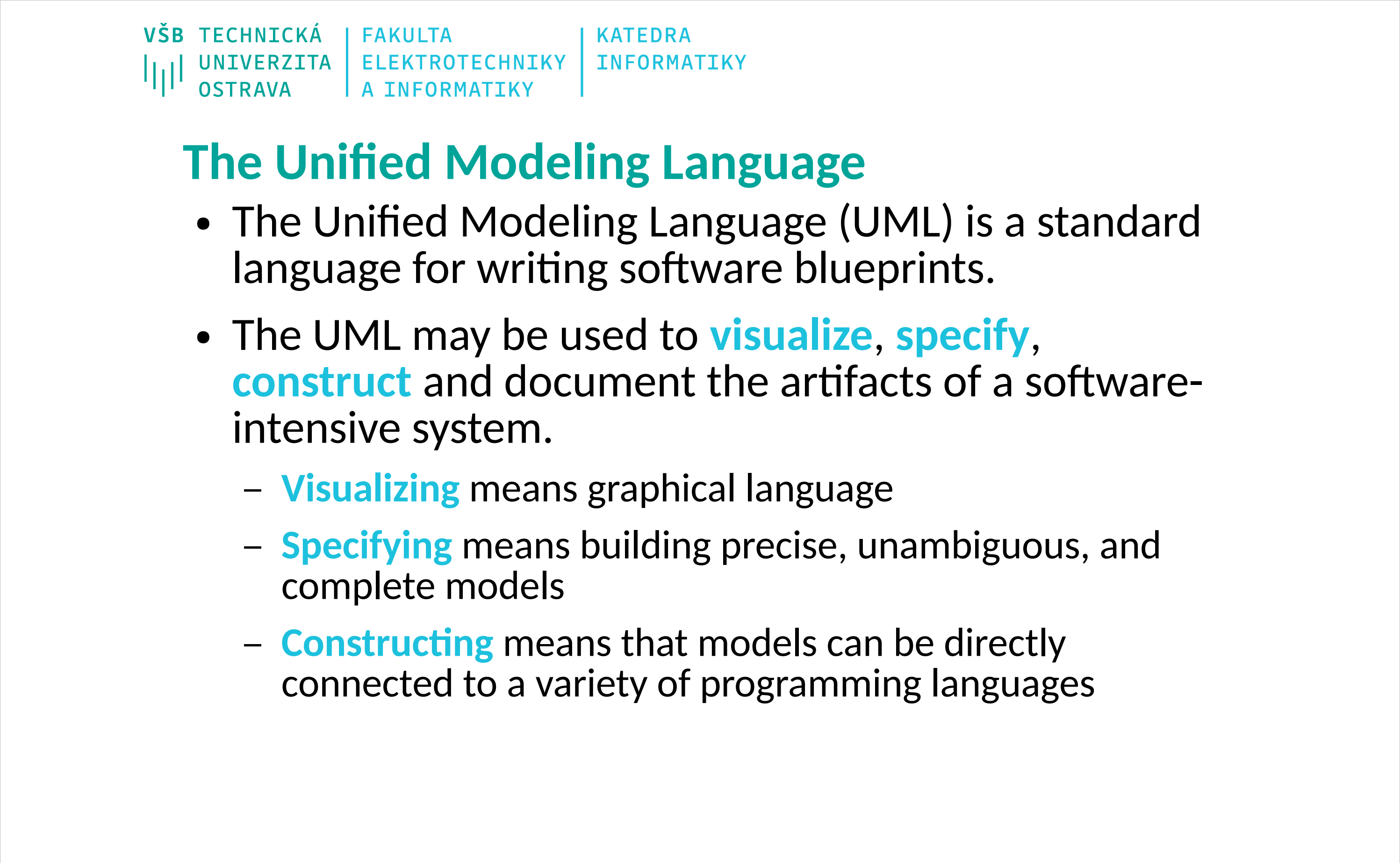
An activity is a unit of work that an individual in that role may be asked to perform and that produces a meaningful result in the context of the project.

- The granularity of an activity may vary from hours to days. It usually involves one person in the associated role and affects one or only small number of artifacts.
- Activities may be repeated several times on the same artifact, especially from one iteration to another.

Artifacts

Artifacts are things that are produced, modified, or used by a process (model, document, source code, executables …).

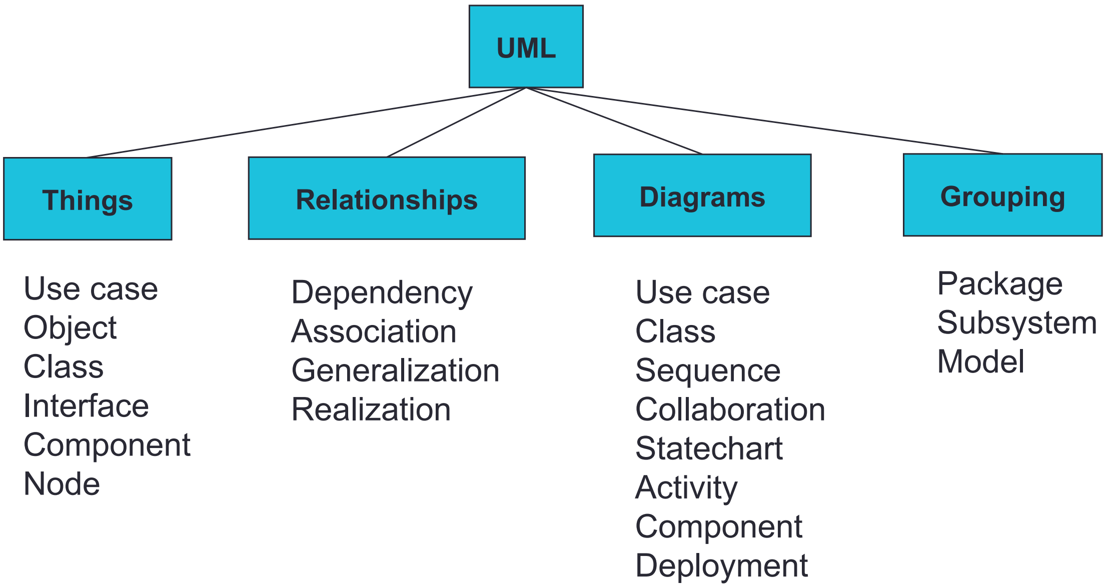
- Deliverables are only the subset of other artifacts.
- Artifacts are very likely to be subject to version control and configuration management.
- Sets of Artifacts:
  - **Management set** – planning and operational artifacts
  - **Requirements set** – the vision document and requirements in the form of stakeholders' needs
  - **Design set** – the design model and architecture description
  - **Implementation set** – the source code and executables, the associated data files
  - **Deployment set** – installation instructions, user documentation, and training material



The Unified Modeling Language

- The Unified Modeling Language (UML) is a standard language for writing software blueprints.
- The UML may be used to **visualize**, **specify**, **construct** and document the artifacts of a software-intensive system.
  - **Visualizing** means graphical language
  - **Specifying** means building precise, unambiguous, and complete models
  - **Constructing** means that models can be directly connected to a variety of programming languages

Building Blocks of the UML



UML

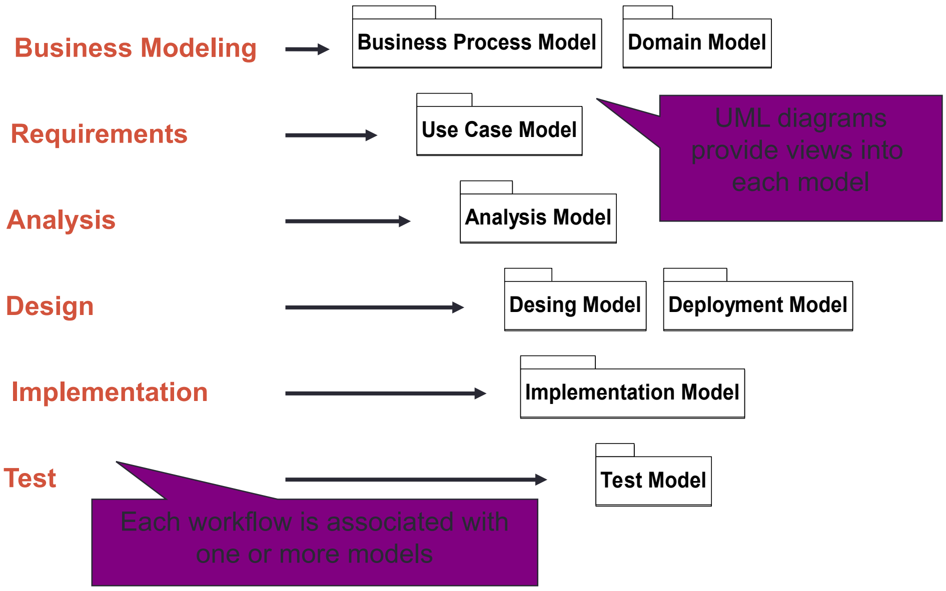
- **Things**
  - Use case
  - Object
  - Class
  - Interface
  - Component
  - Node
- **Relationships**
  - Dependency
  - Association
  - Generalization
  - Realization
- **Diagrams**
  - Use case
  - Class
  - Sequence
  - Collaboration
  - Statechart
  - Activity
  - Component
  - Deployment
- **Grouping**
  - Package
  - Subsystem
  - Model

http://www.uml.org

Core Engineering Workflows

- **Business Modeling** describes the structure and dynamics of the organization
- **Requirement** describe the use case-based method for eliciting requirements
- **Analysis and Design** describe the multiple architectural views
- **Implementation** takes into account sw development, unit test, and integration
- **Test** describes test cases and procedures
- **Deployment** covers the deliverable system configuration

Workflows and Models



Business Modeling

Business Process Model

Domain Model

Requirements

Use Case Model

Analysis

Analysis Model

Design

Desing Model

Deployment Model

Implementation

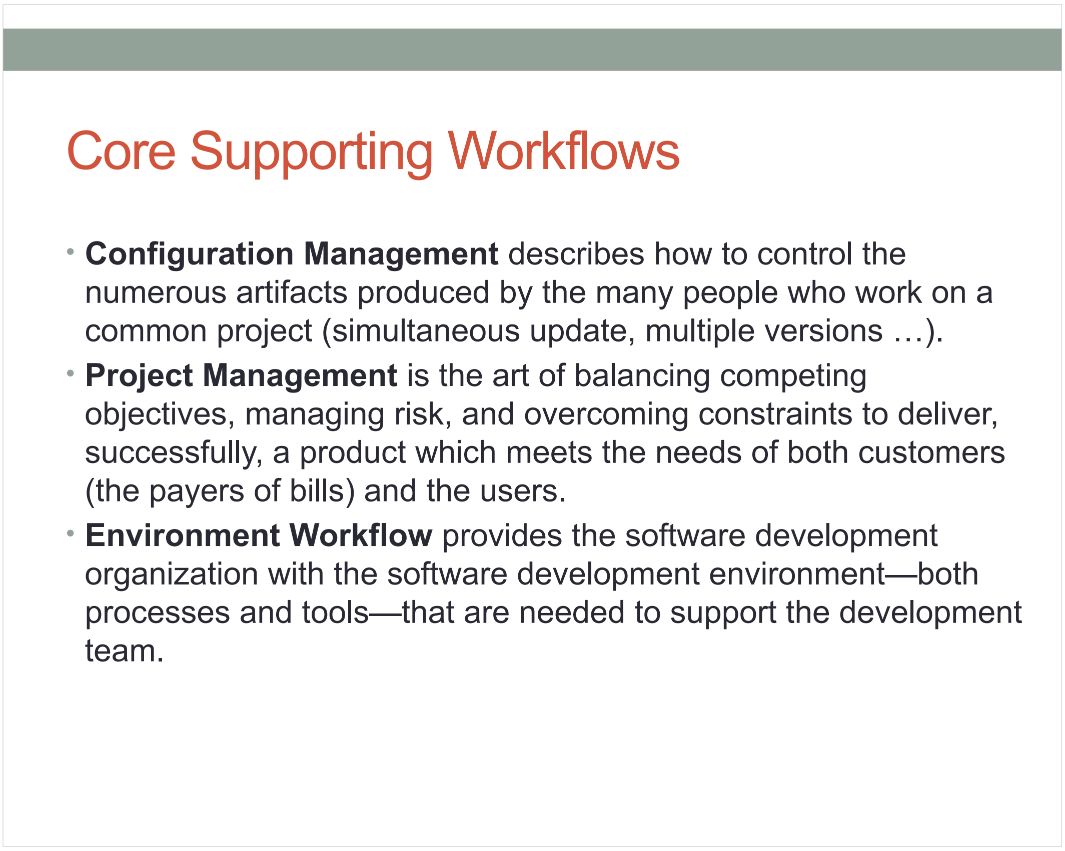
Implementation Model

Test

Test Model

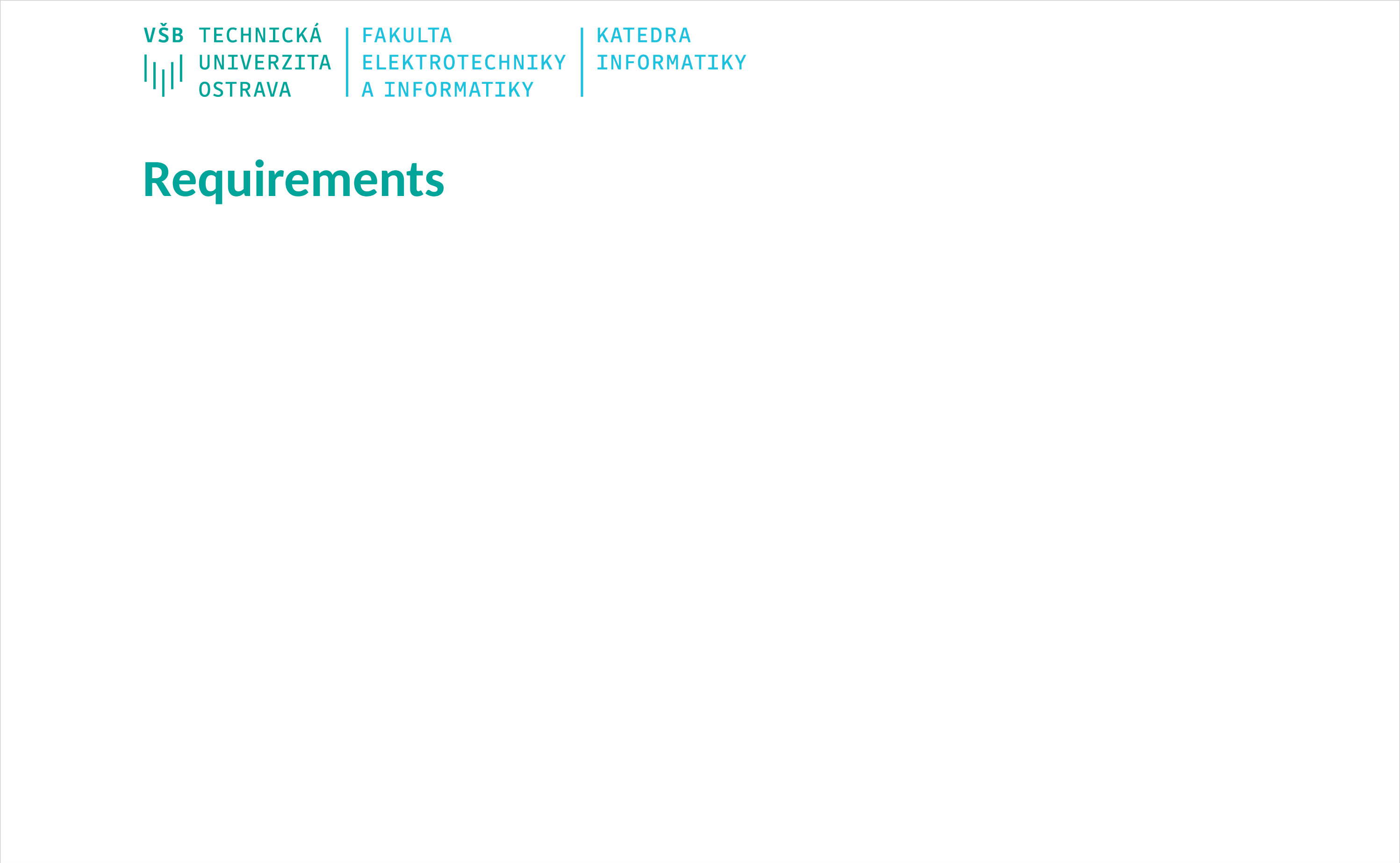
UML diagrams provide views into each model

Each workflow is associated with one or more models



Core Supporting Workflows

- **Configuration Management** describes how to control the numerous artifacts produced by the many people who work on a common project (simultaneous update, multiple versions …).
- **Project Management** is the art of balancing competing objectives, managing risk, and overcoming constraints to deliver, successfully, a product which meets the needs of both customers (the payers of bills) and the users.
- **Environment Workflow** provides the software development organization with the software development environment—both processes and tools—that are needed to support the development team.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Requirements

Requirements

In the next part, we are going to speak about requirements. The part will consist of requirements gathering, problems with the requirements gathering, models and tool to do the requirement analysis.

We have implemented all your needs

Yes, but it is not what I wanted

Many developers already experienced that the development of the project did not have to be only a complete success, but there might be many problems during the development of the software project.

These problems might even lead to a failure of the project or its rejection his reject by the customer.

The causes for this failure can make up a long list of possible problems.

It is well known that the successful handover of the software system is a big issue in the software engineering area.

Let’s have a look at four main symptoms that might lead to the problems.

Symptom no. 1.

Analyst or developer of the software project omits the important task that leads to the understanding of the customer needs. Analysts or developers think that they understand all the needs at once and want to implement them immediately.

Why you didn’t tell us that you want this requirement?
You did not ask me. I thought that you knew everything. It is your fault.

Symptom no. 2.

Software system requirements that are derived from the customer need were not described carefully, the stakeholders were not involved to the process of requirement definition. The developers or analysts were too selfish or thought that they knew what the customer needed. The customer does not know that the developer or analyst does not understand his/her needs. How could he understand? He probably does not know anything about the software development process. It is completely the fault of the developers (developers’ fault).

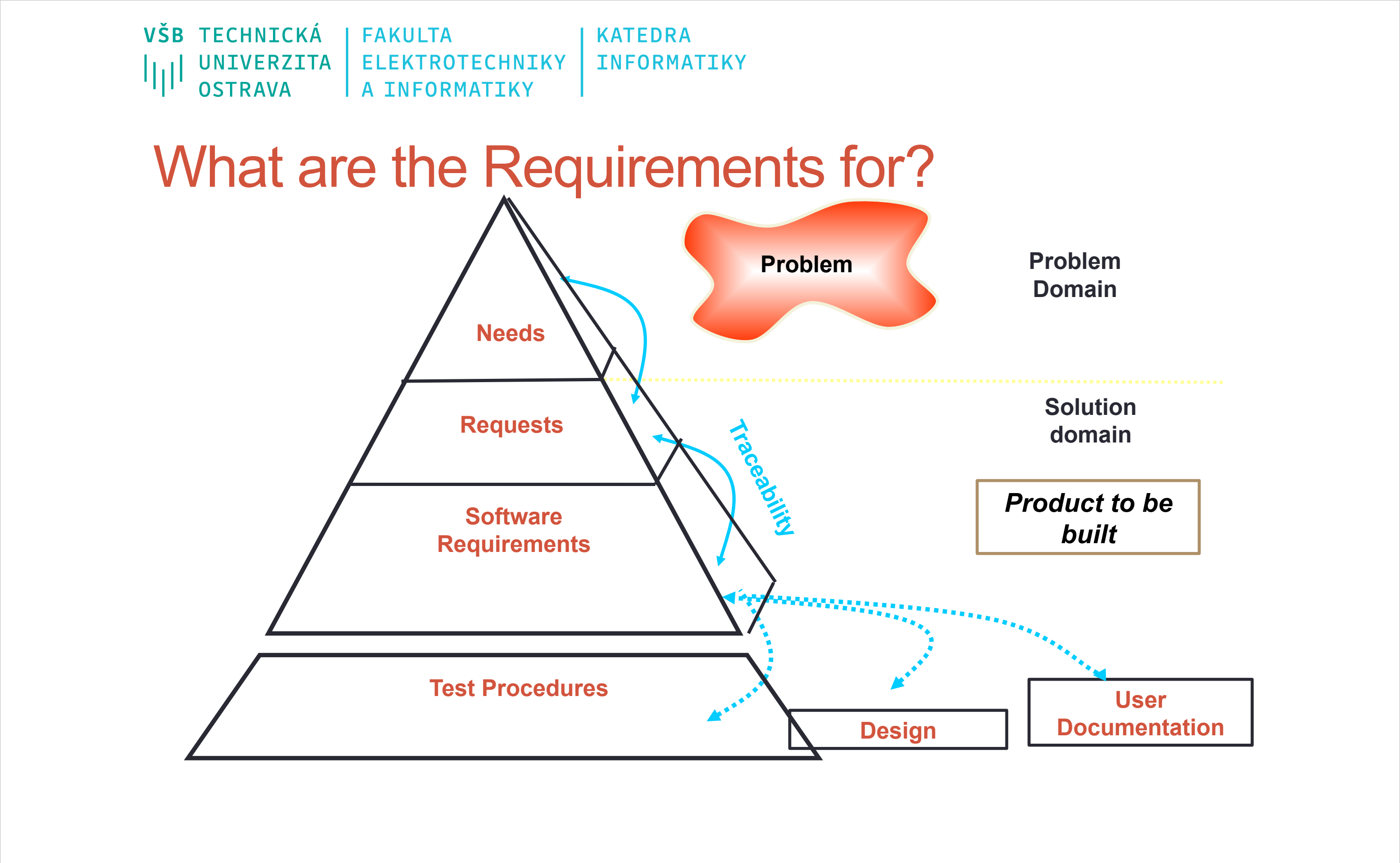
Symptom no. 3.

The developers think that they know everything. The requirements are not formally described and there is no change management. If they have to implement the requirement that is not completely clear they do it their way, they do not consult the problem with the analyst or customer. The problem is that the developers might not understand all customers intentions with this requirement and the connection to other requirements if it is not described.

I have a new great requirement. That requirement must be implemented now.
Sure, no problem…

Symptom no. 4.

The project is not managed correctly. Nobody knows or cares about the time management, nobody cares about the difficulty of the requirements. Customer changes requirements, brings new requirements and the developers do not know the difficulty. The problem is that the project might soon ravage the project budget or the time consuming requirements might take too much time to built the complete project on time.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

What are the Requirements for?

Problem

Problem Domain

Solution domain

Needs

Requests

Software Requirements

Test Procedures

Traceability

Product to be built

Design

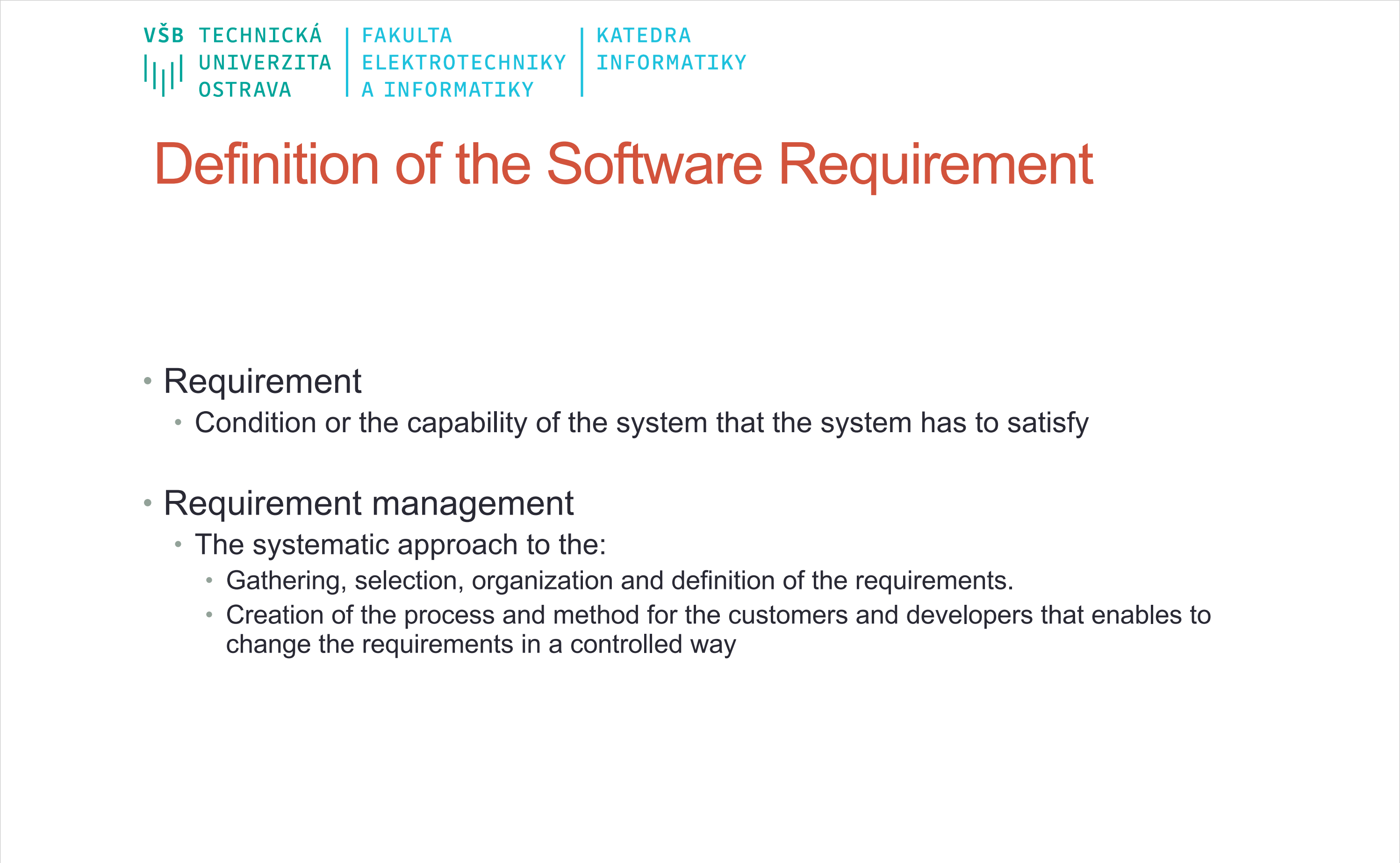
User Documentation

What are the requirements for?

Requirements management consists of the analysis of customers' needs. These needs are transformed to the software requirements. These software requirements are then divided into functional and non-functional requirements.

According to the requirements, the test procedures are made, design is made to satisfy the requirements, user documentation etc.

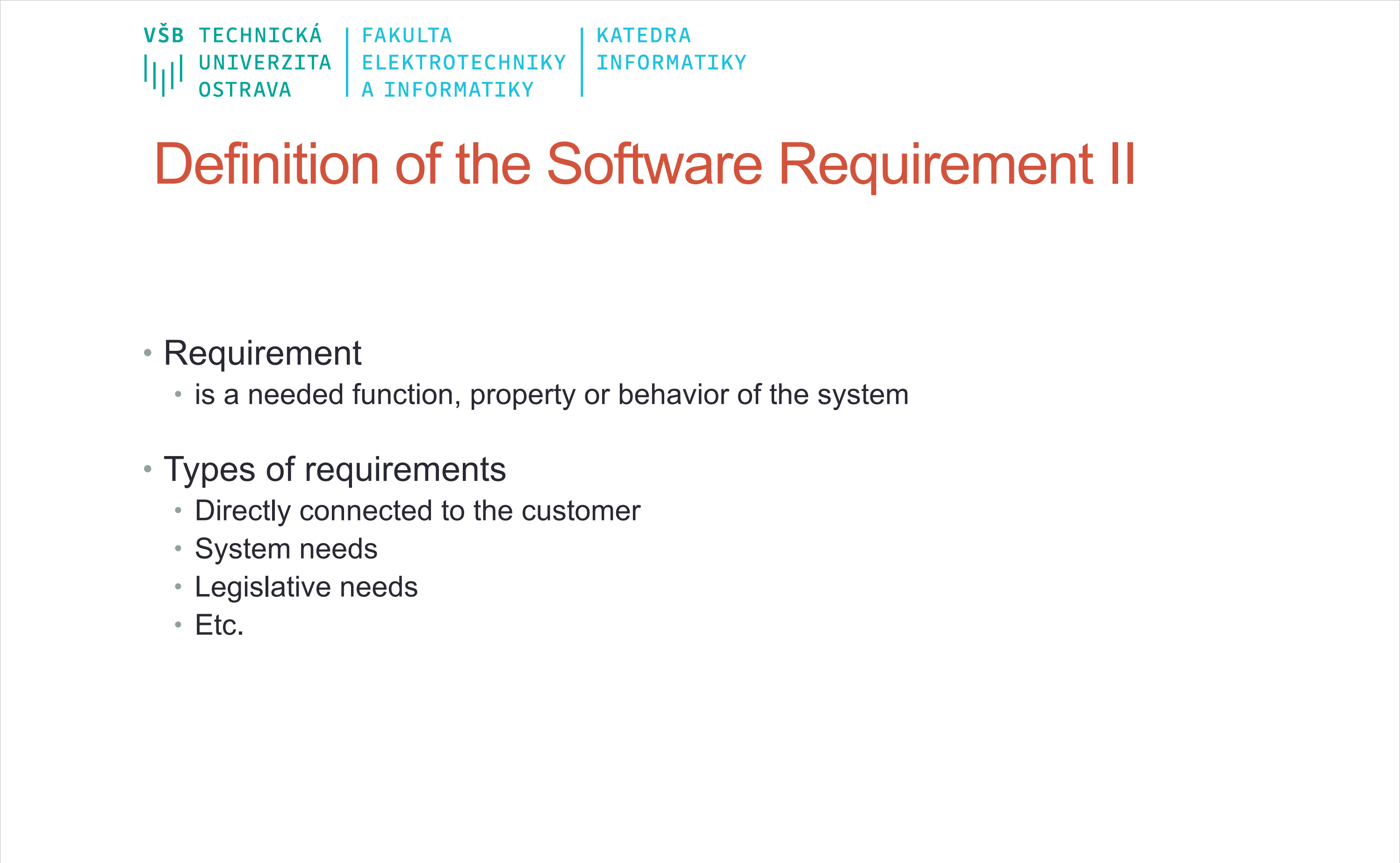
The pyramid shows the process that started in the problem domain, where the problem was defined, then the needs of the new software, requests that are derived from these needs, software requirements that are derived from the request and then the rest of the artifacts. The pyramid shows that the needs and requests are the core things that have to be solved, everything else relies on them.



Definition of the Software Requirement

- Requirement
  - Condition or the capability of the system that the system has to satisfy
- Requirement management
  - The systematic approach to the:
    - Gathering, selection, organization and definition of the requirements.
    - Creation of the process and method for the customers and developers that enables to change the requirements in a controlled way

Now, we have to ask a question, what is the requirement. Requirement is a capability or a condition that the system has to satisfy.



Definition of the Software Requirement II

- Requirement
  - is a needed function, property or behavior of the system
- Types of requirements
  - Directly connected to the customer
  - System needs
  - Legislative needs
  - Etc.

It is not important whether the requirements are gathered directly from the customers or they are defined in the various documents that have been signed with the customer, specification of the software requirement or other formally approved document.

According to the UML, the requirement is a needed function, property or behavior of the system.

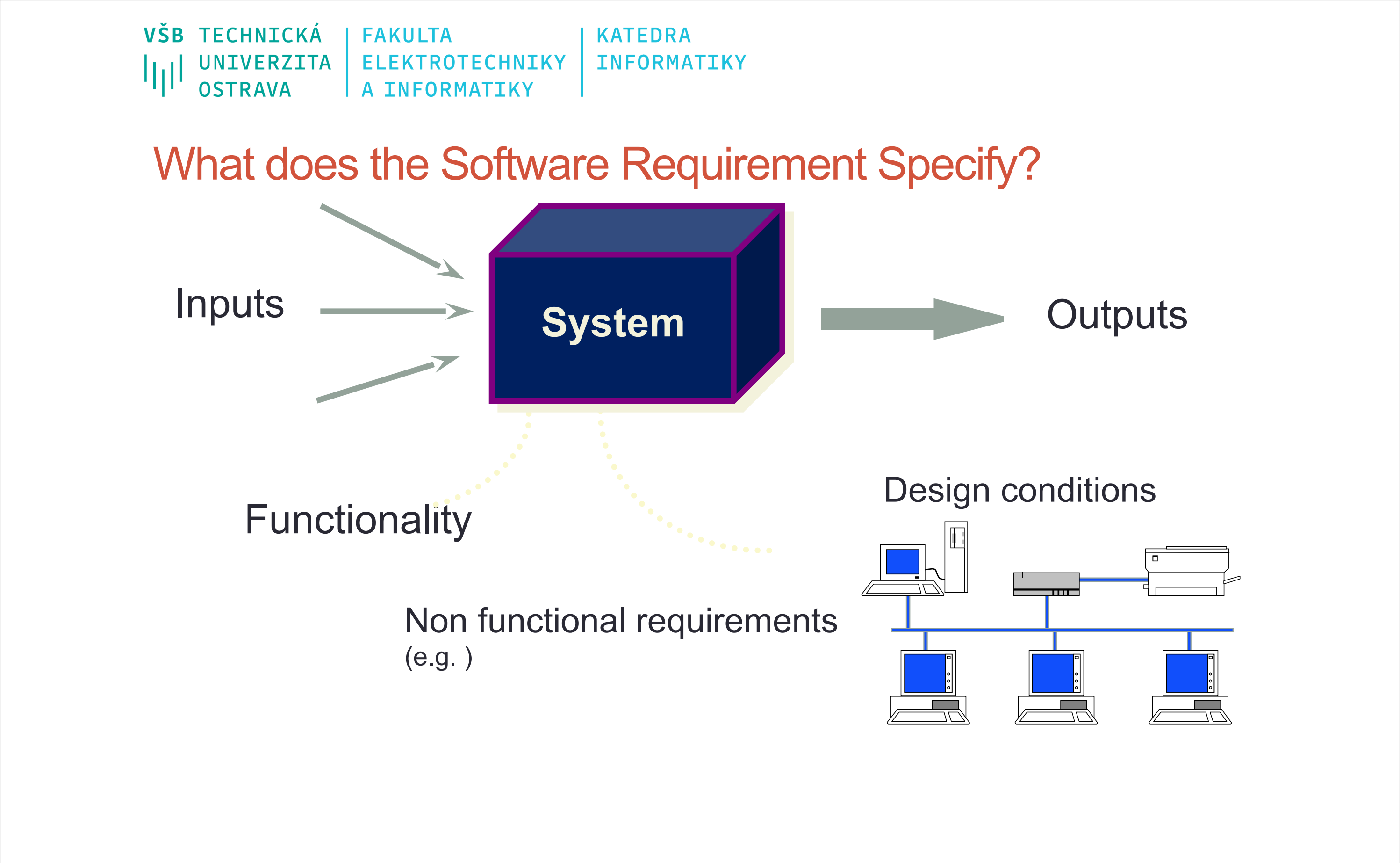
Anyway, a requirement specifies WHAT the system should do, but does not describe HOW to do that.

There are many types of requirements.

For example, requirements ruled by a function are directly connected to the customer, on the other hand, legal requirements are requirements that specify the legislative need that has to be followed by the system.

You have to know that it is not possible to define all requirements at once at the beginning of the project.

Requirement management is successful only when the requirements are briefly described at the beginning and then completely detailed.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

What does the Software Requirement Specify?

Inputs

System

Outputs

Functionality

Design conditions

Non functional requirements
(e.g.)

What does the software requirement specify?

As was already mentioned before, a requirement describes the condition or the capability that the system has to satisfy. Software requirements are in fact something like a description of the black box. Requirements define only the externally visible things WHAT the system should do.

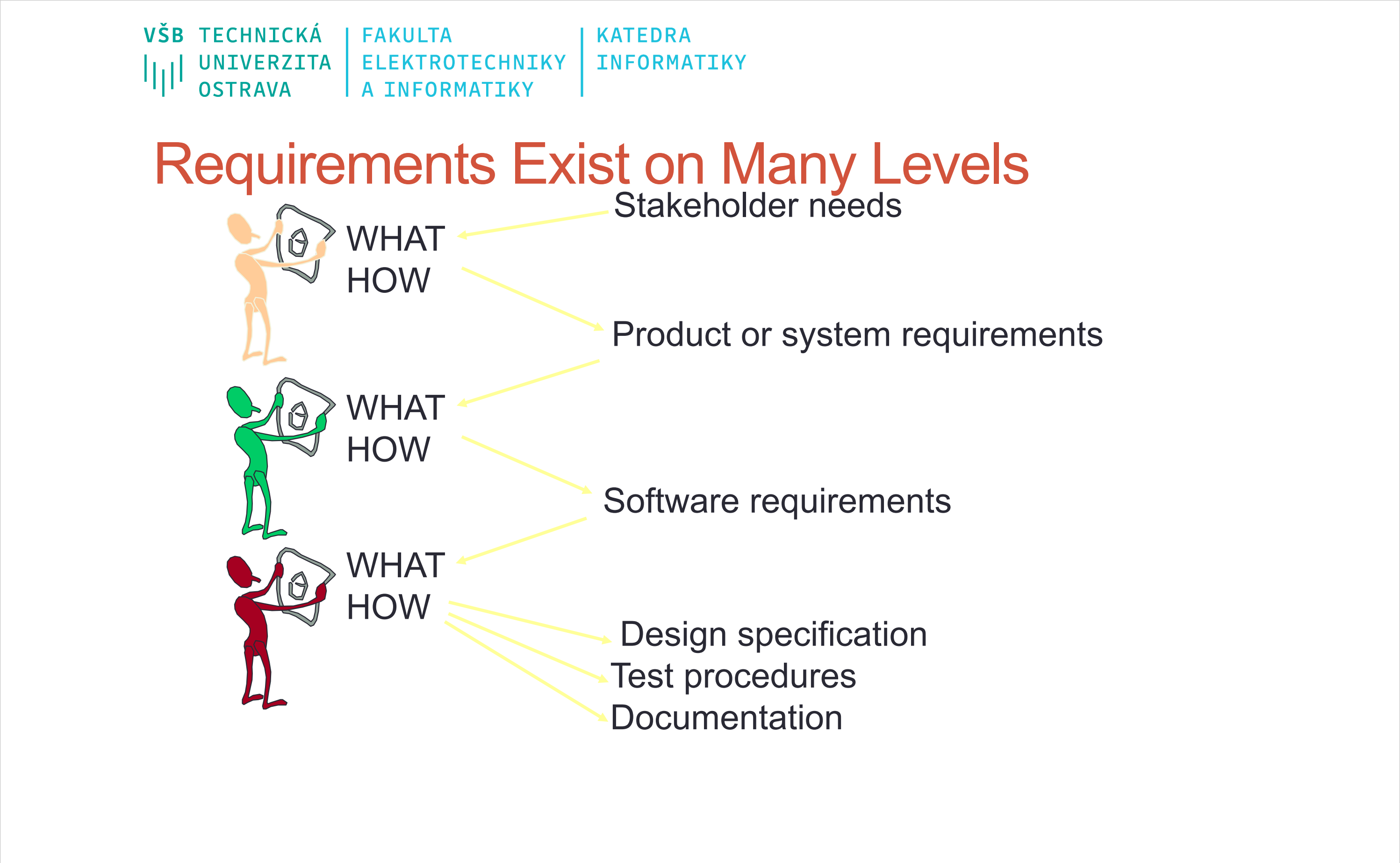
Sometimes there are necessary descriptions HOW the system should specifically do that. These descriptions are defined by the design conditions.

Definition of Terms

- User Request
  - Description of the customer needs without the connection to the specific solution
- Property
  - Externally visible service that ~~that~~ satisfies the stakeholder needs
- Software Requirement
  - Functional Requirement
    - Requirement that specifies how the solution to be implemented interacts with the surrounding world from the “black box” view
  - Non-Functional Requirement
    - Requirement that defines the quantitative attributes of the solution, again from the “black box” view
- Condition
  - Condition of the design or the process of the software development

Definition of Terms – Example – University System

- User request
  - Reduction of the administrative load
  - Teachers need immediate access to the students results
- Property
  - Information about students will be accessible from the semester view or from the group view.
- Software requirement
  - Functional requirement
    - When the student selects “subject registration” the system will show all available subjects
  - Non-functional requirement
    - The system will be accessible 99% of the time 24/7 (the system might not be available only 3,65 days per year)
- Condition
  - The system will run on the current mainframe university architecture.



Requirements Exist on Many Levels

Stakeholder needs

WHAT
HOW

Product or system requirements

WHAT
HOW

Software requirements

WHAT
HOW

Design specification
Test procedures
Documentation

Requirements exist on many levels

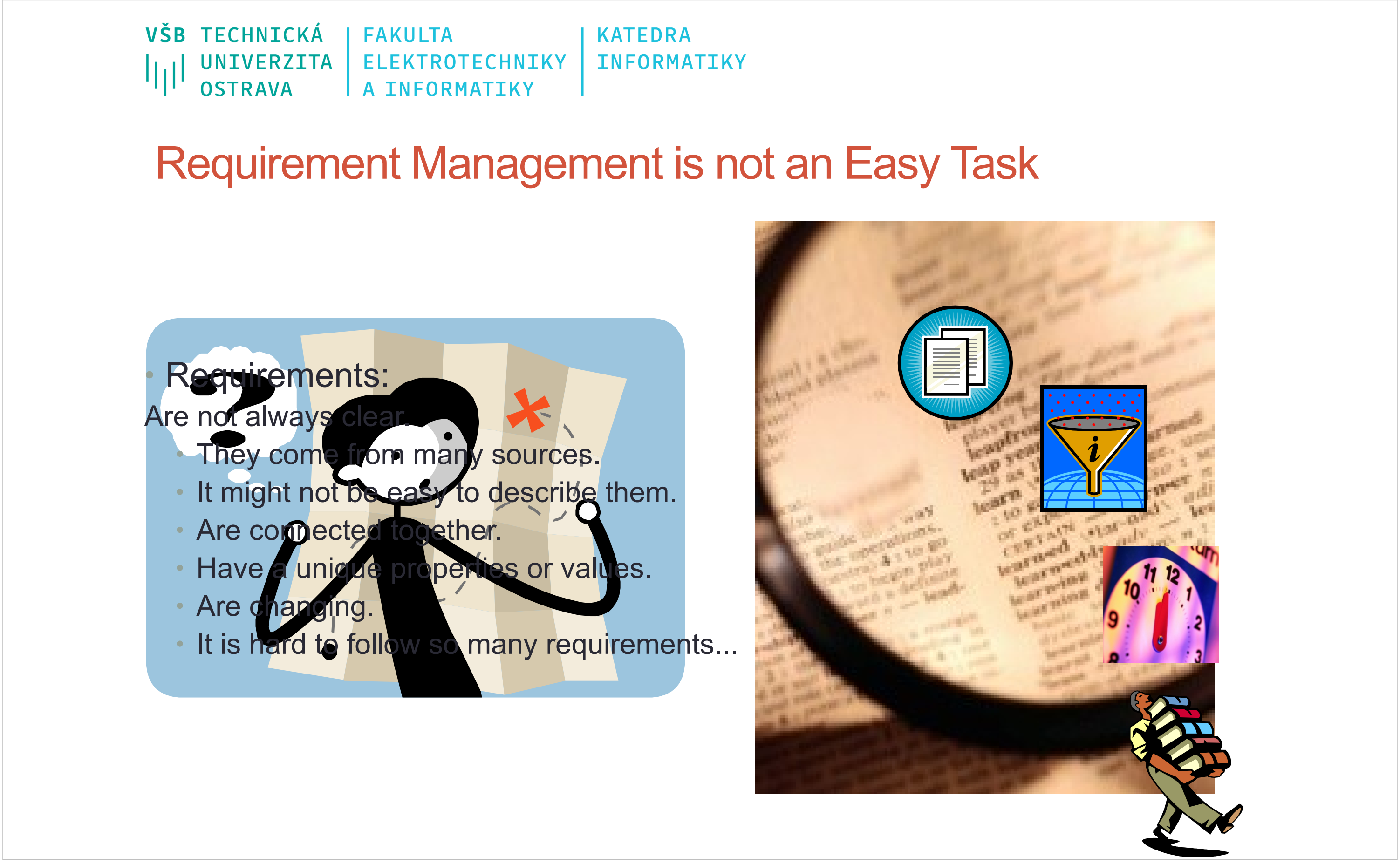
It depends on the perspective whether it is a requirement or a design condition.

For example, the stakeholder request is a requirement for the system analyst. Analyst creates a product or a system, requirements that are inputs to the software requirements.

Software requirements that are described by the use cases must specify what the system should do to satisfy the requested functionality.

The person responsible for the creation of the use cases creates a set of these use cases that are requirements for the system designers. The same principle continues.

When we want to develop a system that satisfies stakeholder needs, it is important to correctly understand all the needs. Therefore, the quality of requirements is very important during the whole development process.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Requirement Management is not an Easy Task

- Requirements:

Are not always clear.

- They come from many sources.
- It might not be easy to describe them.
- Are connected together.
- Have a unique properties or values.
- Are changing.
- It is hard to follow so many requirements...

Requirement Management is not an Easy Task

Requirements management might looks like an easy task, but it is not. The realization of the requirement management is hard task.

There are many reasons like:

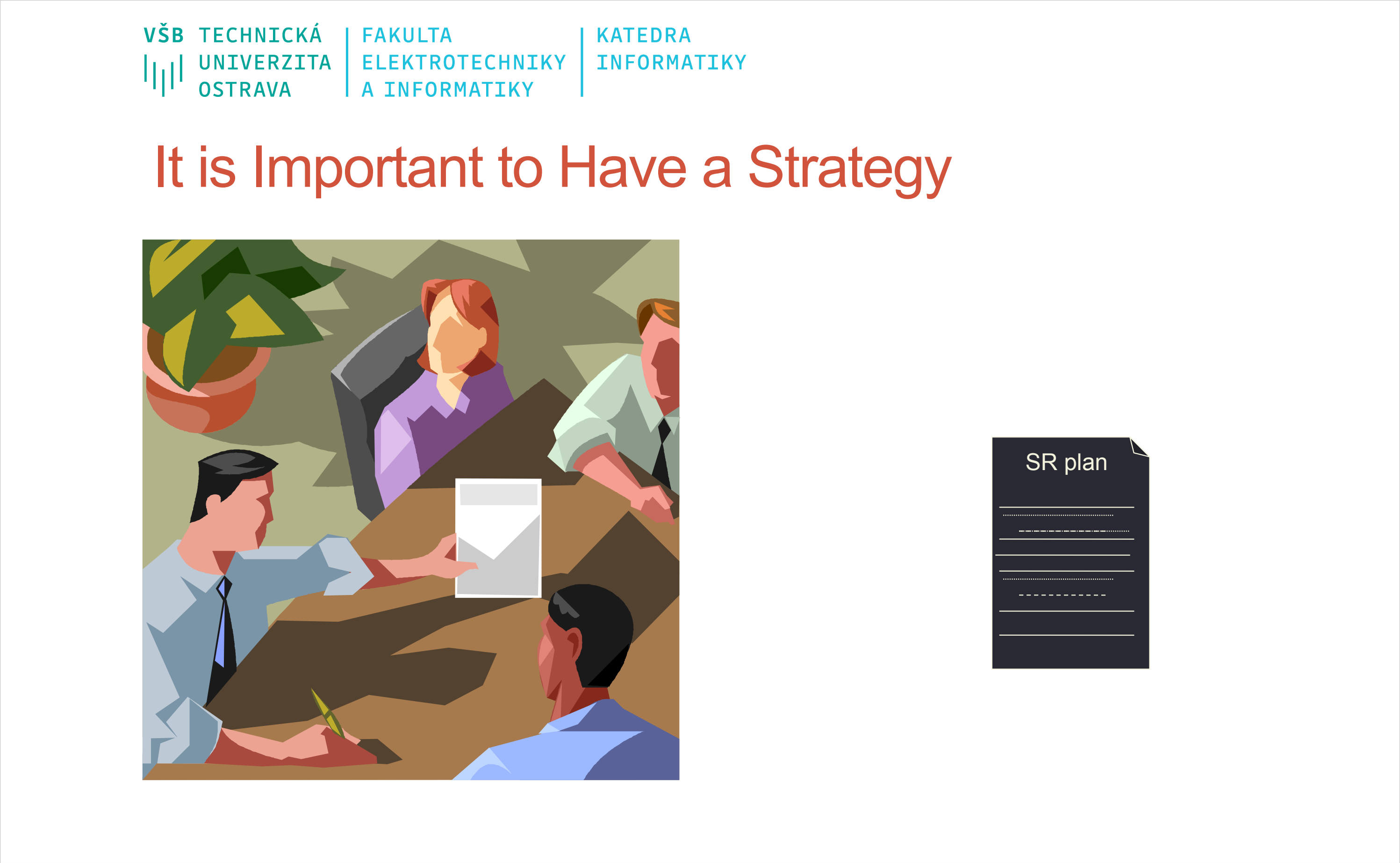
Customers not always know what they want

When the number of requirements grows it is not easy to follow them all and take them ~~in~~ into account.

It can be hard to recognize the dependency between the requirements

IT people have a problem to use a style of writing that is understandable to non-technical people.

All possible problems can be overcome by the right selection of the process.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
KATEDRA INFORMATIKY

It is Important to Have a Strategy

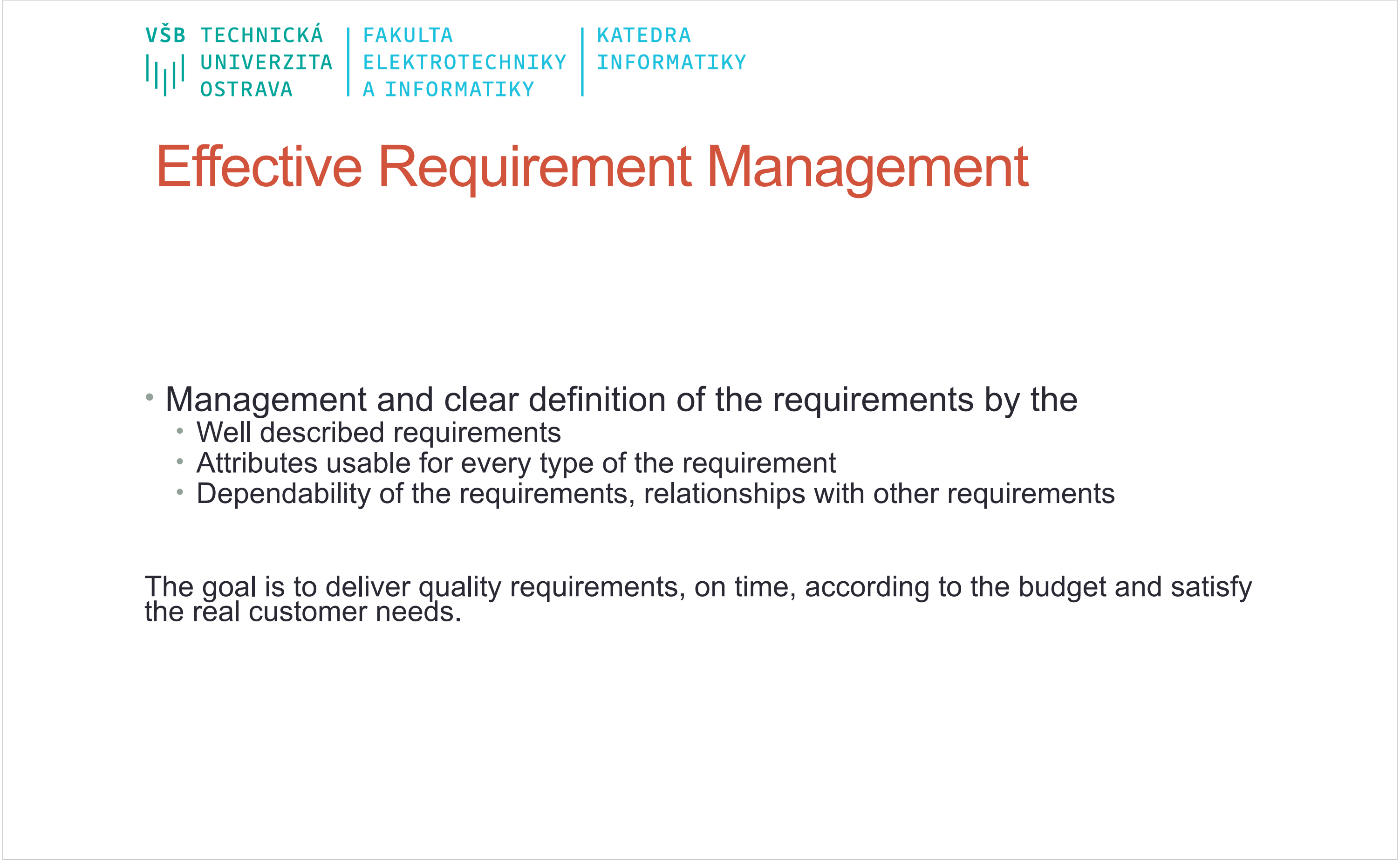
It is important to have a strategy.

It is important to have a requirement management plan. Requirement management plan specifies control mechanisms that are used for the gathering, description, measurement and management of the software requirements.

It is important to know how the requirements will be documented and organized. It is necessary to know how the requirements will be managed for the whole project lifecycle.

Requirement management plan describes then all important decisions that are concerning description of the requirements, methods a strategies for the gathering of the requirements, traceability of the requirements, etc.

Similarly to the glossary, that describes the terminology of the project, the requirements management plan is a life document. This document is created at the beginning of the project and then extended according to the timeline of the important decisions.



Effective Requirement Management

- Management and clear definition of the requirements by the
  - Well described requirements
  - Attributes usable for every type of the requirement
  - Dependability of the requirements, relationships with other requirements

The goal is to deliver quality requirements, on time, according to the budget and satisfy the real customer needs.

Effective Requirement Management

Management and clear definition of the requirements by the

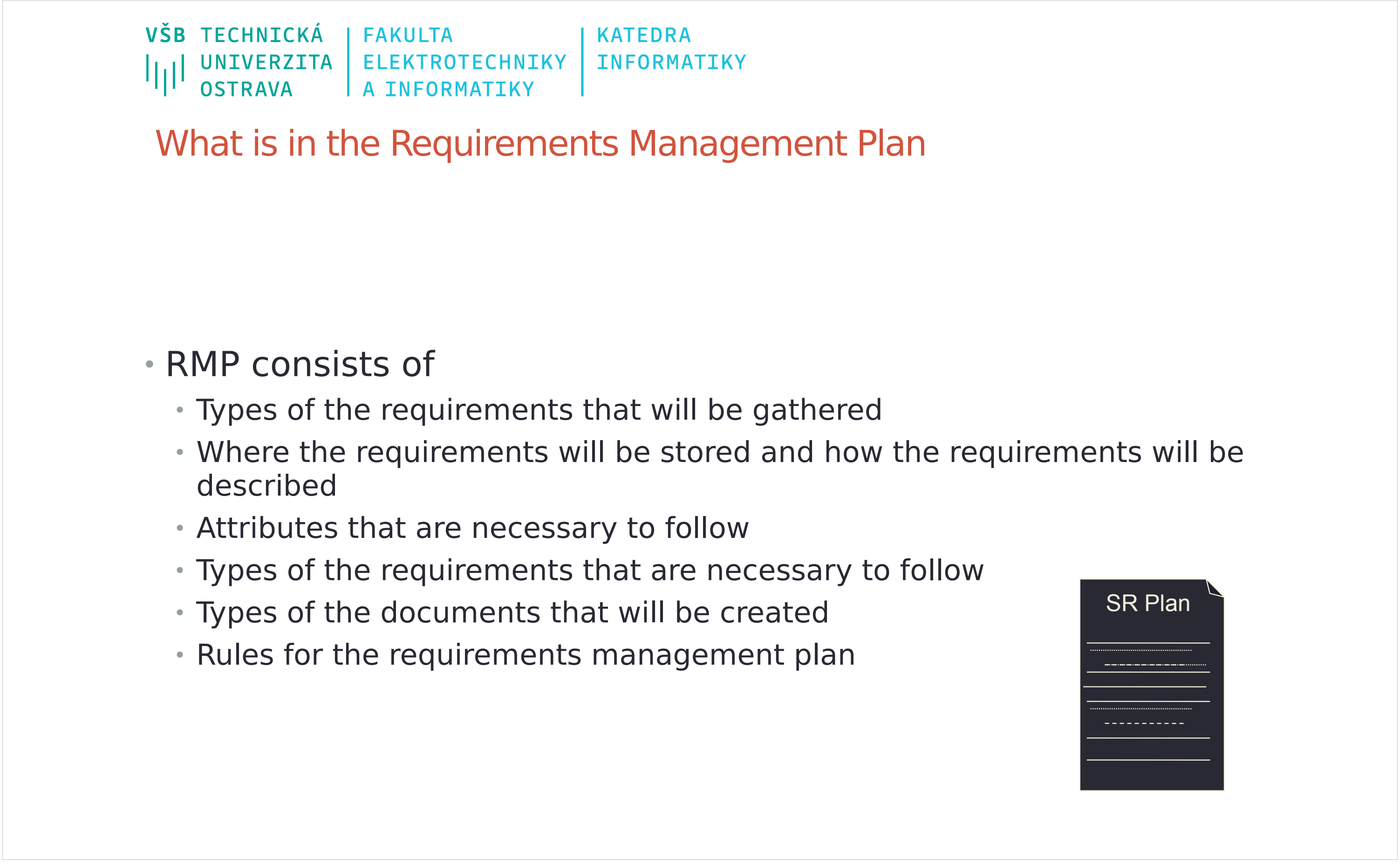
Well described requirements

Attributes usable for every type of the requirement

Dependability of the requirements, relationships with other requirements

The goal is to deliver quality requirements, on time, according to the budget and satisfy the real customer needs.

Management



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

What is in the Requirements Management Plan

- RMP consists of
  - Types of the requirements that will be gathered
  - Where the requirements will be stored and how the requirements will be described
  - Attributes that are necessary to follow
  - Types of the requirements that are necessary to follow
  - Types of the documents that will be created
  - Rules for the requirements management plan

SR Plan

Requirement management plan should be prepared as a template document in the company. Then, the document RMP is accustomed for every single project that needs that.

What is Quality Product?

- Quality is to:
  - Satisfy the requirements specification
  - All system tests were successful
  - Development according to the process

What is quality product?

In the past, the goal of the software development was only to deliver a system that satisfied all formal software specifications. The definition of the quality was: everything was delivered as-specified. But this concept did not ensure that the system was successful.

In this case, the project starts with the detailed plan and strictly follows prescribed processes. This quality approach is based on the idea that if the development process is strictly followed the product will be good software.

System tests were used to ensure that the system is correct, but nobody cared whether the system helped to solve user problems or not.

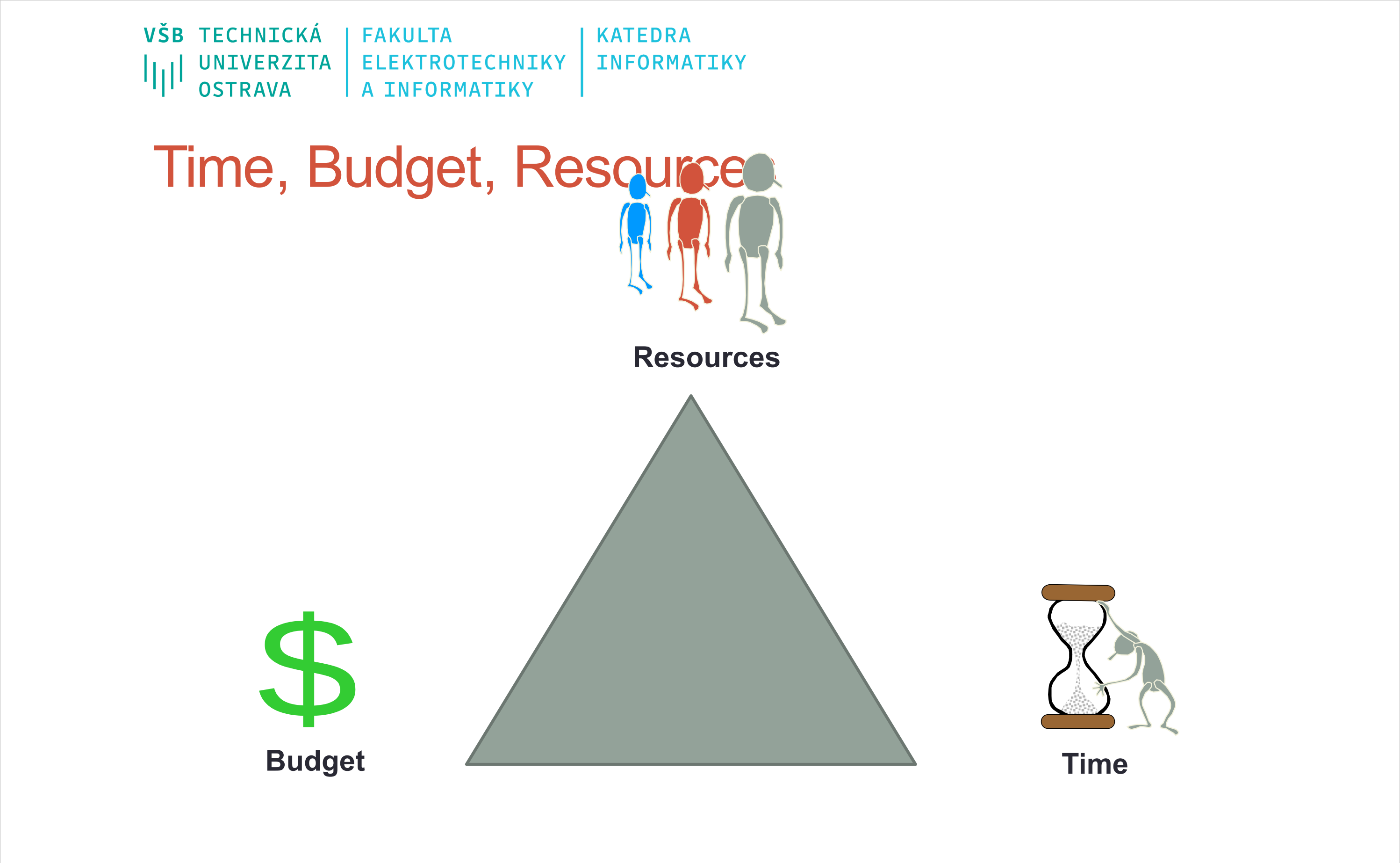
What is Quality Product?

- Quality is to:
  - Listen to and understand the customer needs
  - Continuous evaluation of all artifacts to ensure that all stakeholder needs are satisfied.

What is quality product?

It is clear that the development according to the development process is important, but it is also important to ensure the quality of the final product. The goal is the result not the process itself.

The other quality view is concentrated on the understanding what the customers and users expect from the system to solve their problems. Modern approaches have the methods and processes set in the way that ensures the requirements are understood correctly.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

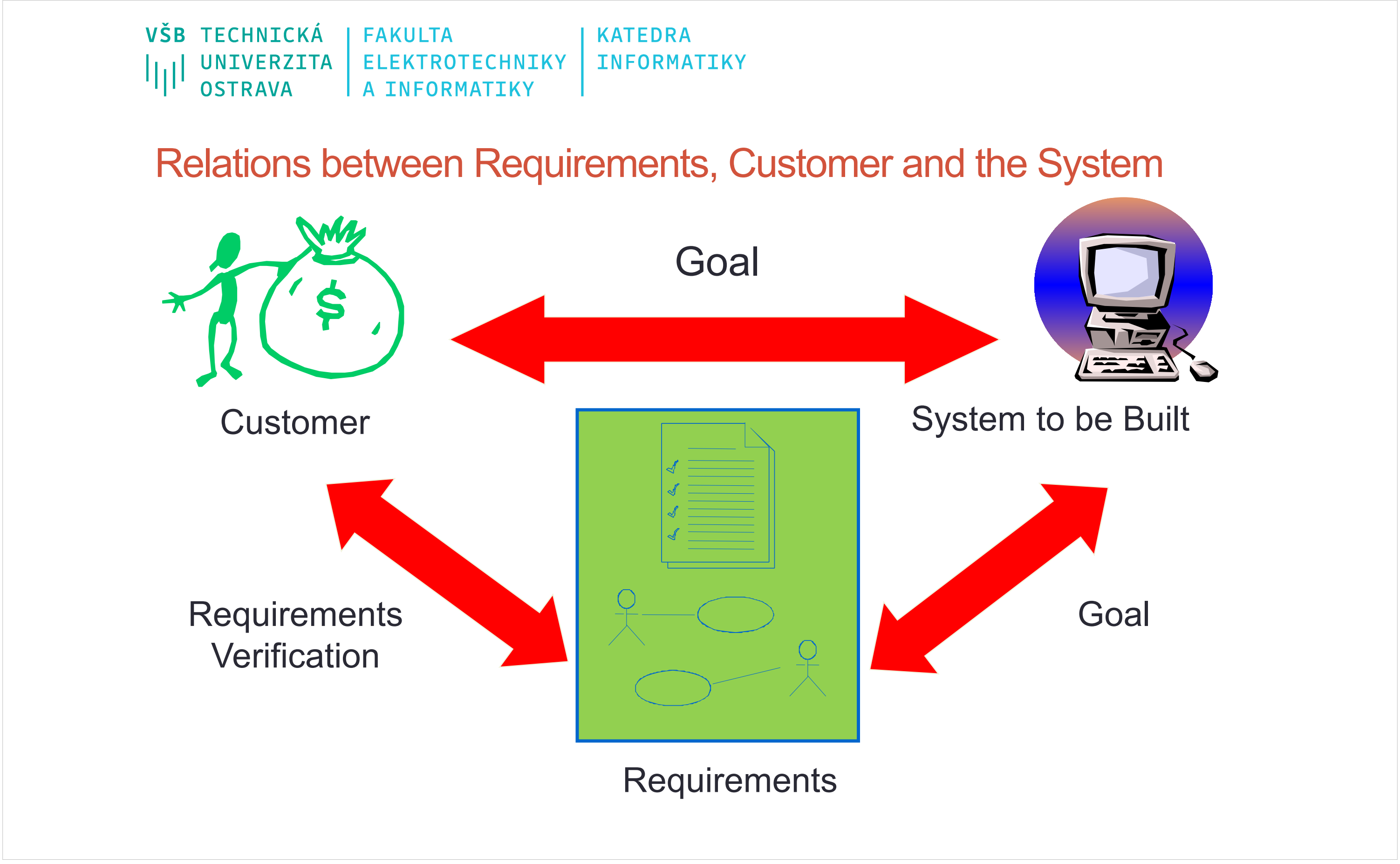
Time, Budget, Resources

Time, Budget, Resources

Since the time, budget and resources allocated to the project are limited, it is possible to deliver only limited amount of work.

It is important to think about how much work you are able to do in that limited time with limited budget and limited resources.

For example, if the amount of work is to be extended, one of the three factors has to be extended as well. If for example, the budget is decreased, the amount of work has to be decreased as well. This principle expects the quality to be the same.



Relations between Requirements, Customer and the System

Goal

Customer

System to be Built

Requirements Verification

Goal

Requirements

Relations between requirements, customer and the system

One of the goals of the requirement specification is to make a deal about the requirements. This deal is then implemented by the requirement specification.

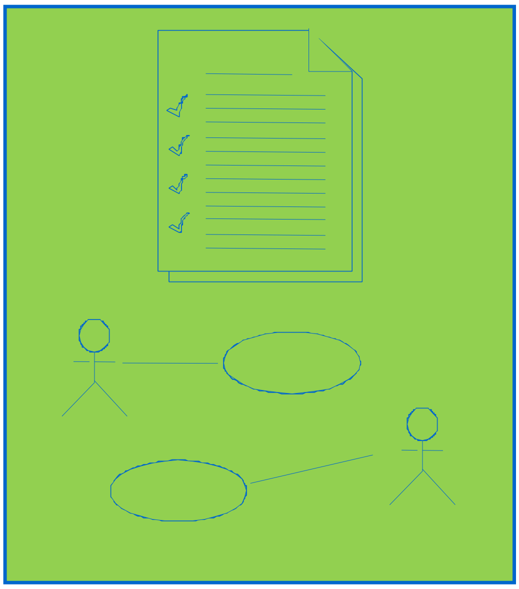
It is not always possible to contact the customer whenever we want and it is well know that customers change their mind very often, so it is essential to write down all requirements taken from the customers.

Requirements then substitute the customer, requirements should be so complete that the customer should not be necessary during the implementation phase.

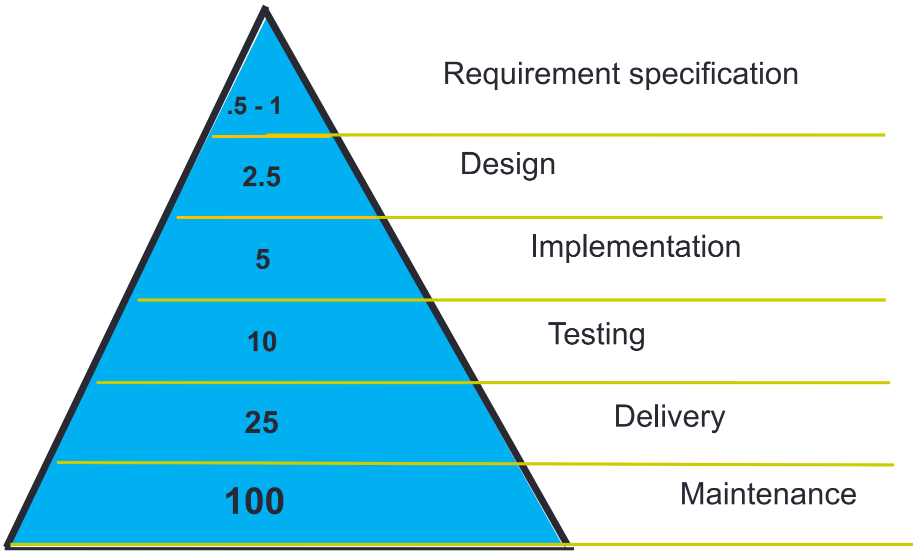
Requirements are representing the customer. It shows his wishes, what the system should do. Requirements should be written in a such form that is understandable to the customers and the development team as well.

Requirements are something like a secondary goal for the system development. On the other hand, they describe criteria for the acceptation and validation of the system being developed. This is why the requirements and their definitions are so important.

Requirements and their Role



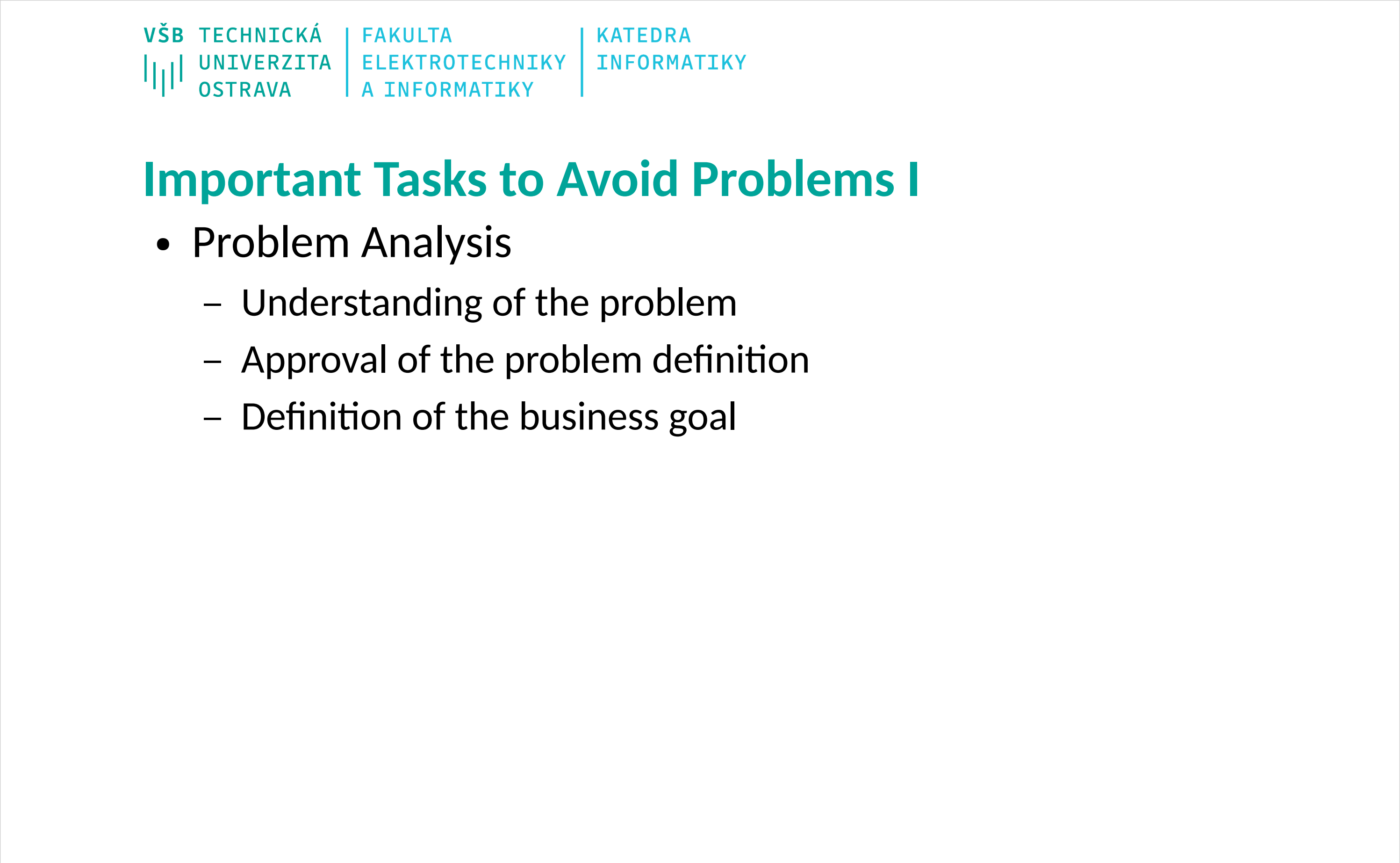
Cost of the Error Repair



.5 - 1	Requirement specification
2.5	Design
5	Implementation
10	Testing
25	Delivery
100	Maintenance

The cost of the error repair

The most expensive repair of the error is one that was made in the requirement specification and it is discovered during the maintenance by the customer. If the error is discovered during the requirement specification phase, the cost of such a repair is much cheaper. On this picture we can see relative cost of the errors that are made during the requirements specification phase and are repaired during next phases. The later the error is made, the so much more costly it becomes to repair.



Important Tasks to Avoid Problems I

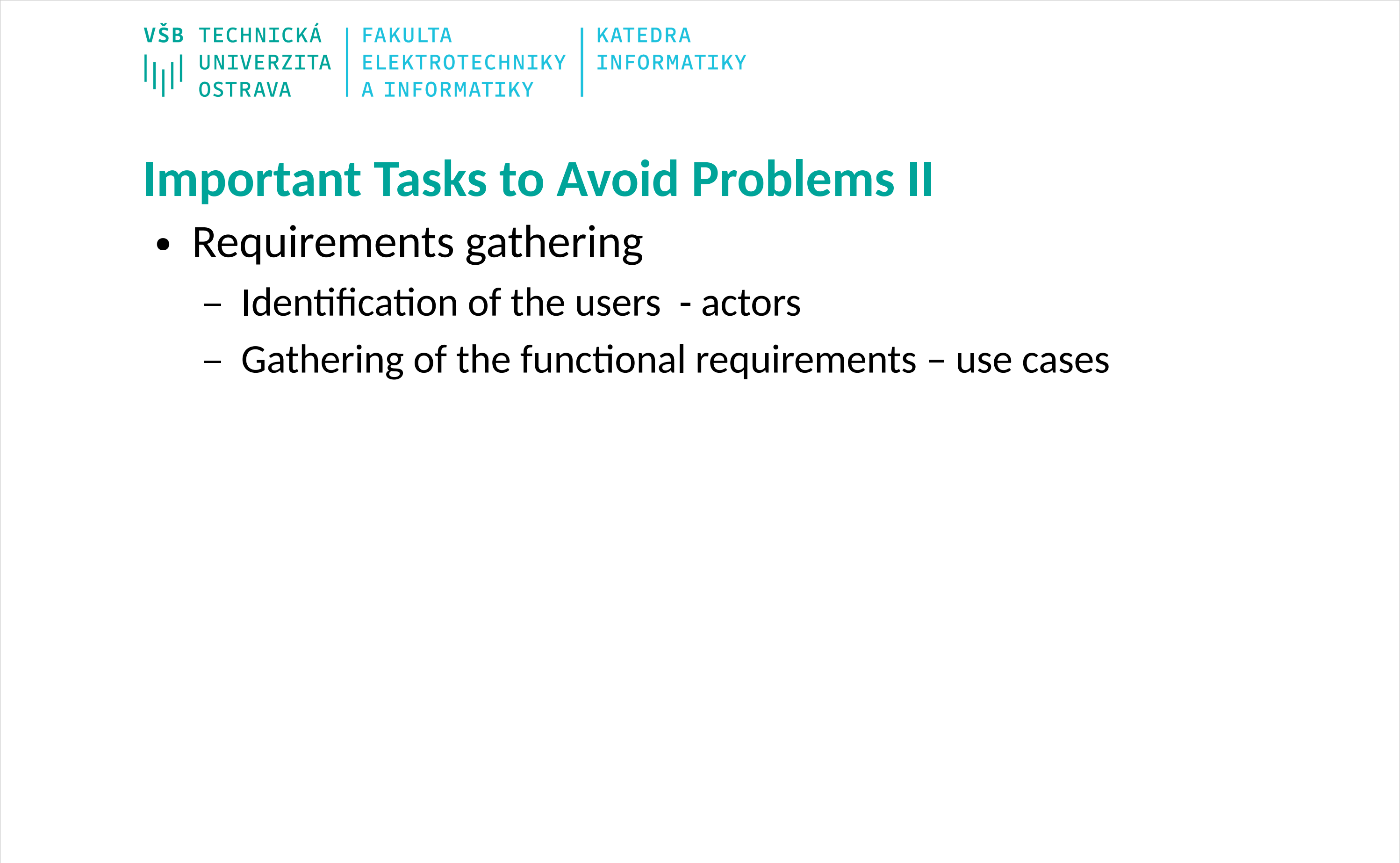
- Problem Analysis
  - Understanding of the problem
  - Approval of the problem definition
  - Definition of the business goal

Important Tasks to Avoid Problems

It is important to follow several important rules to be able to deliver or at least to be able to increase the probability of delivering the successful system.

First of all it is important to follow some basic rules.

The first and most necessary is the correct understanding of the problem. When the problem is defined then the customer and the development team must approve their views of the problem and define the business goal.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Important Tasks to Avoid Problems II

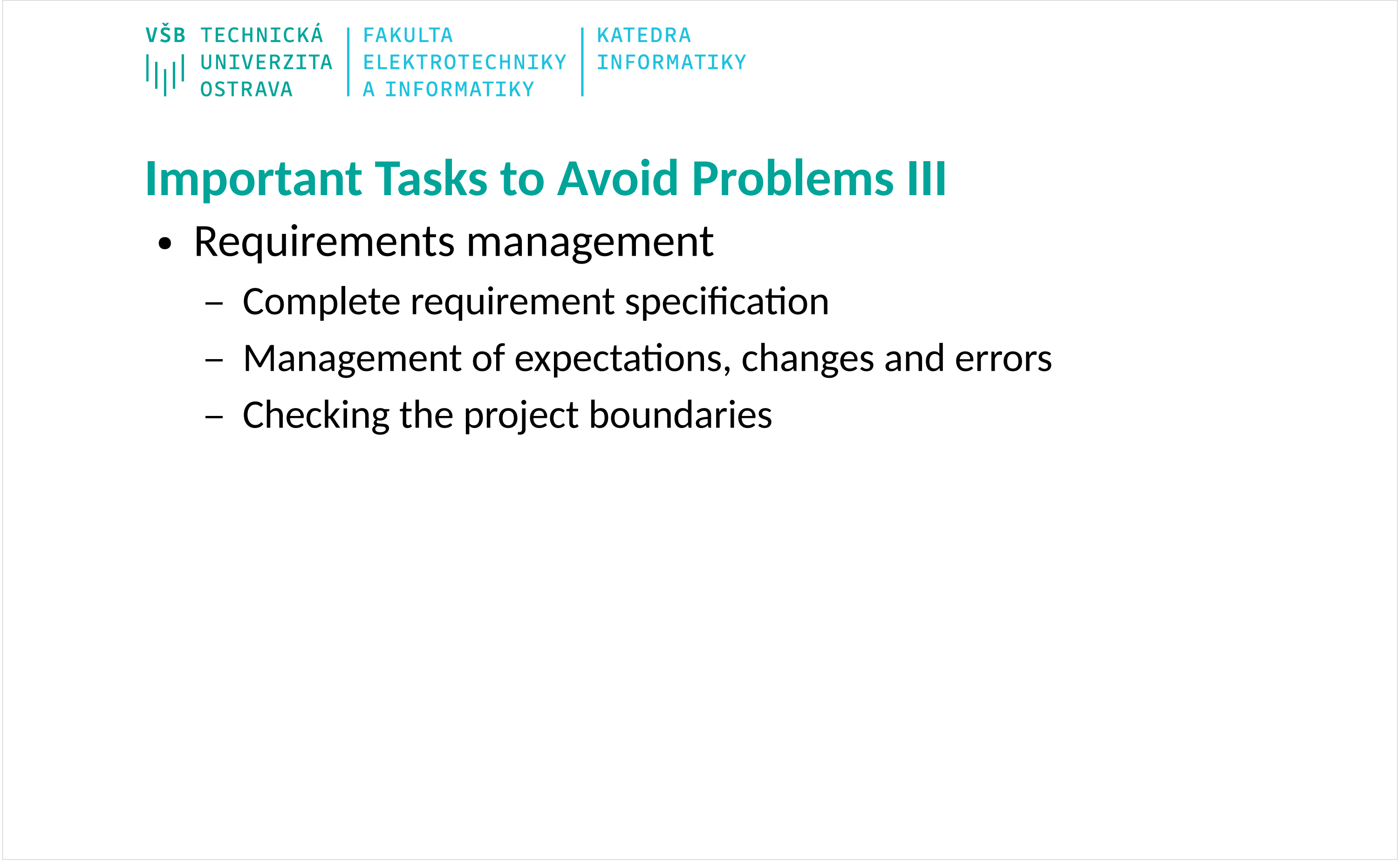
- Requirements gathering
  - Identification of the users - actors
  - Gathering of the functional requirements – use cases

Important Tasks to Avoid Problems II

One of the ways of describing a requirements is a use case model. Use cases enable us to organize requirements from the customer's viewpoint – actors.

All requirements that are necessary to complete one goal are concentrated into one use case. Use case model is then the collection of such use cases – goals.

Since the requirements described by the use cases are described from the customer viewpoint, the best way to gather them is to create and discuss them with the customer.



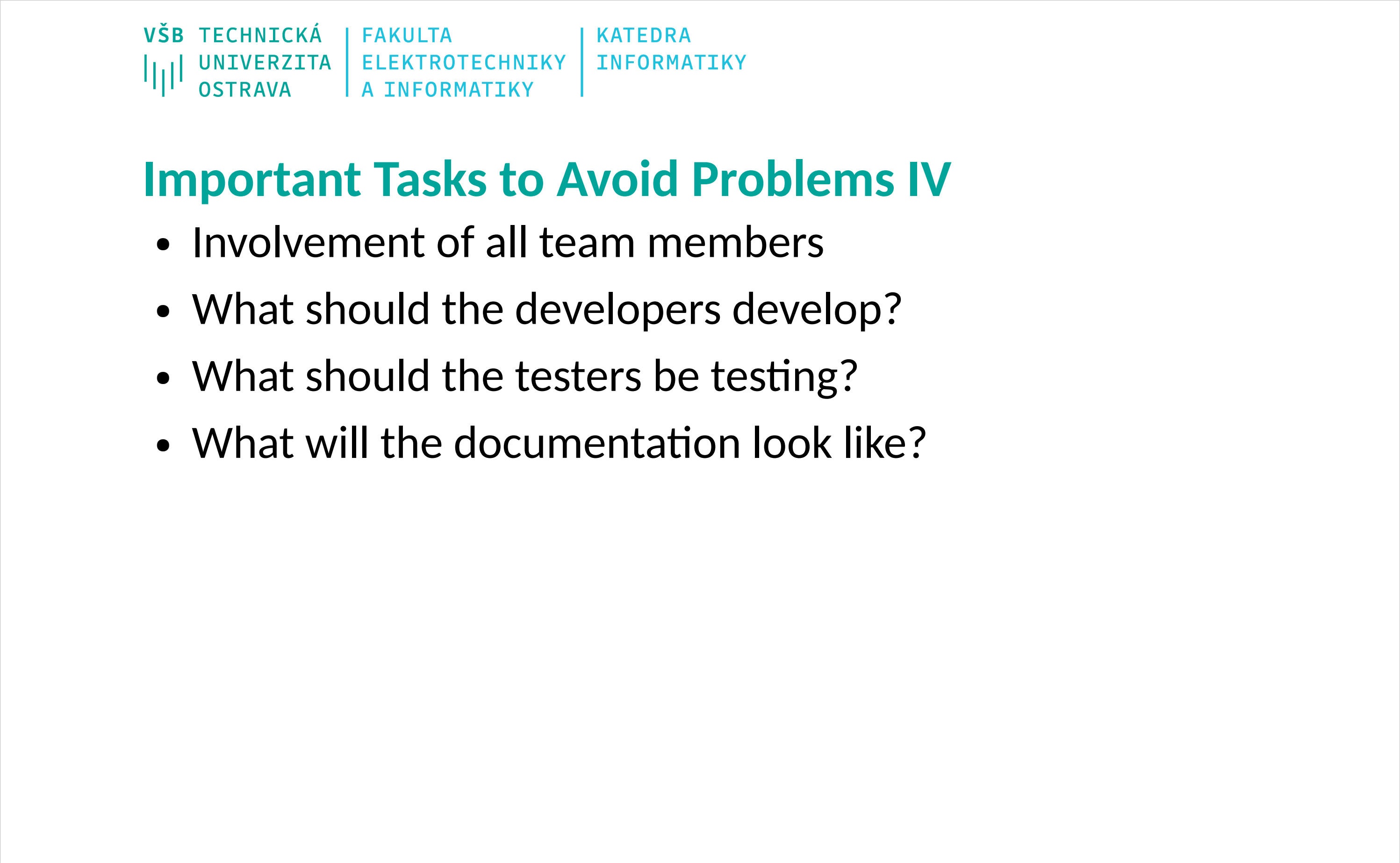
VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Important Tasks to Avoid Problems III

- Requirements management
  - Complete requirement specification
  - Management of expectations, changes and errors
  - Checking the project boundaries

Important Tasks to Avoid Problems III

How can a developer be sure that he is developing the right system? Ask users and customers, whether your use case model is what they expect from the system. Use case model is the key to the development of the whole system.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

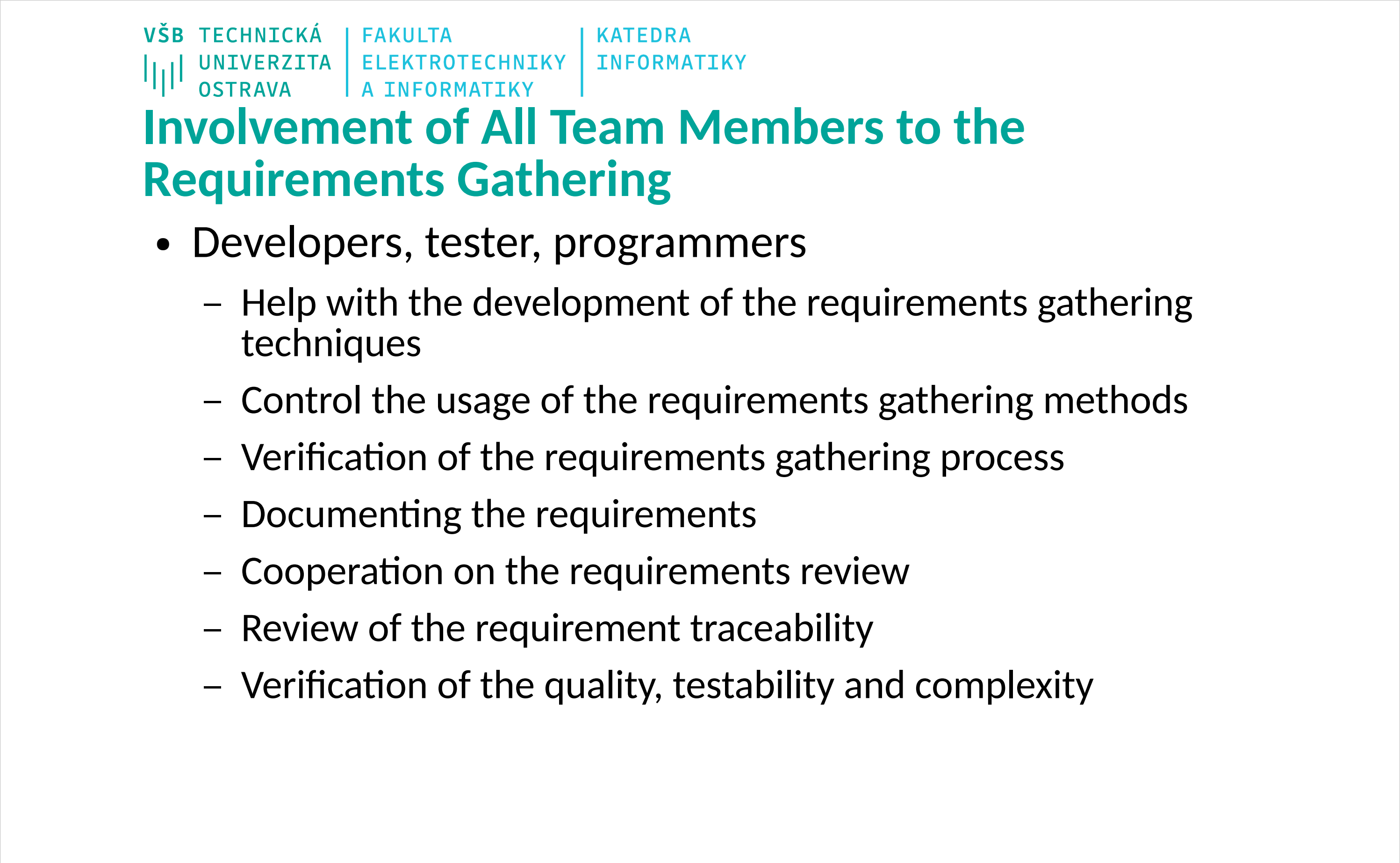
Important Tasks to Avoid Problems IV

- Involvement of all team members
- What should the developers develop?
- What should the testers be testing?
- What will the documentation look like?

What the developers develop? They develop a system that enables user to perform task that are specified in the use case model.

What the tester should be testing? They do the rest to ensure that the system is able to do all the tasks specified in the use case model correctly.
How the documentation will looks like? The documentation of the system will describe all tasks that are specified in the use case model.

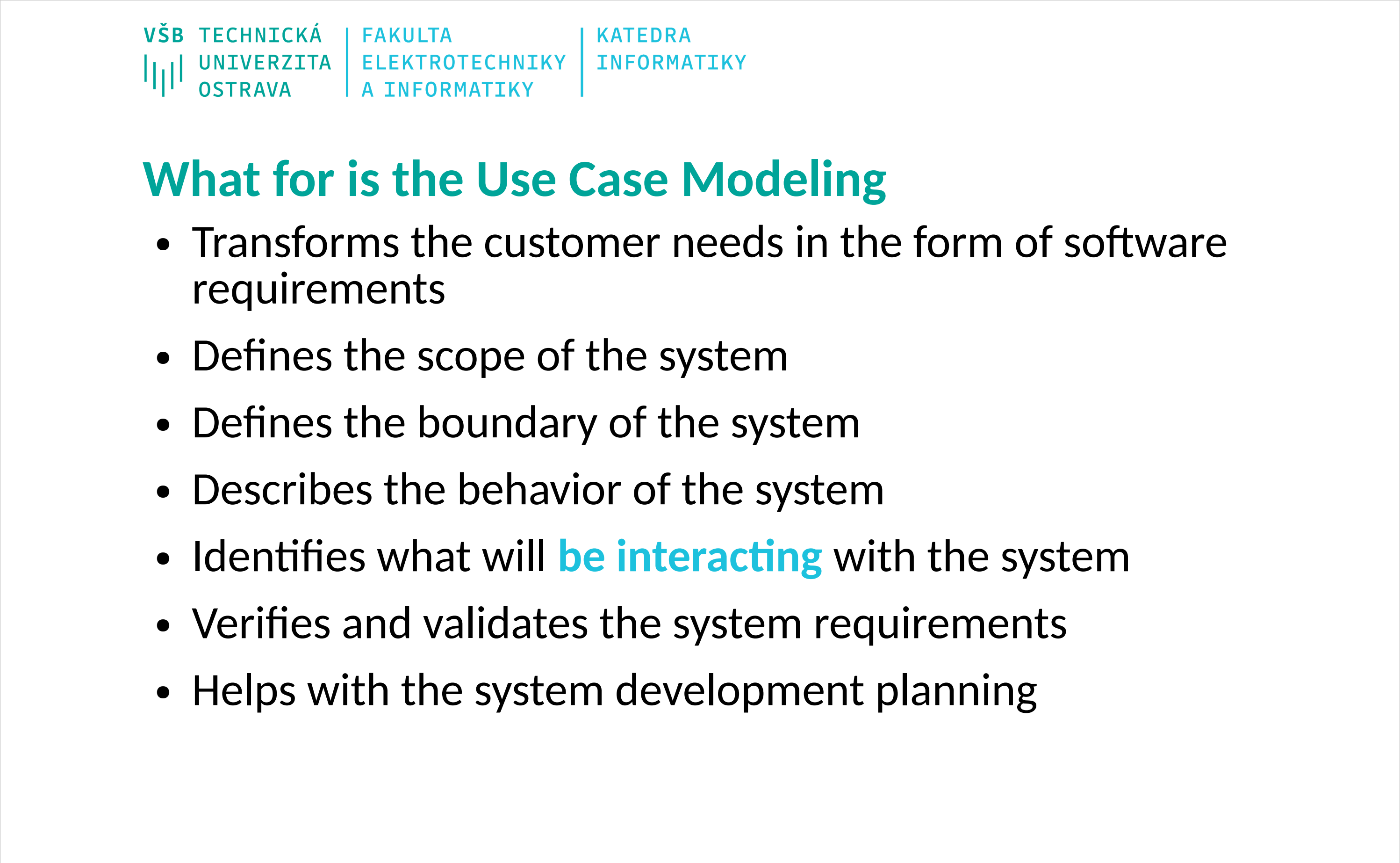
Do not forget that it is important to specify requirements correctly, control the expectation of the customers, changes and errors. Control the scope of the project and it boundaries and involve all team members to the system development.



Involvement of All Team Members to the Requirements Gathering

- Developers, tester, programmers
  - Help with the development of the requirements gathering techniques
  - Control the usage of the requirements gathering methods
  - Verification of the requirements gathering process
  - Documenting the requirements
  - Cooperation on the requirements review
  - Review of the requirement traceability
  - Verification of the quality, testability and complexity

What are the advantages of the involvement of all team members? The main advantage is that the whole team gets the clearer idea, what the requirements are and why they are important for the customer.



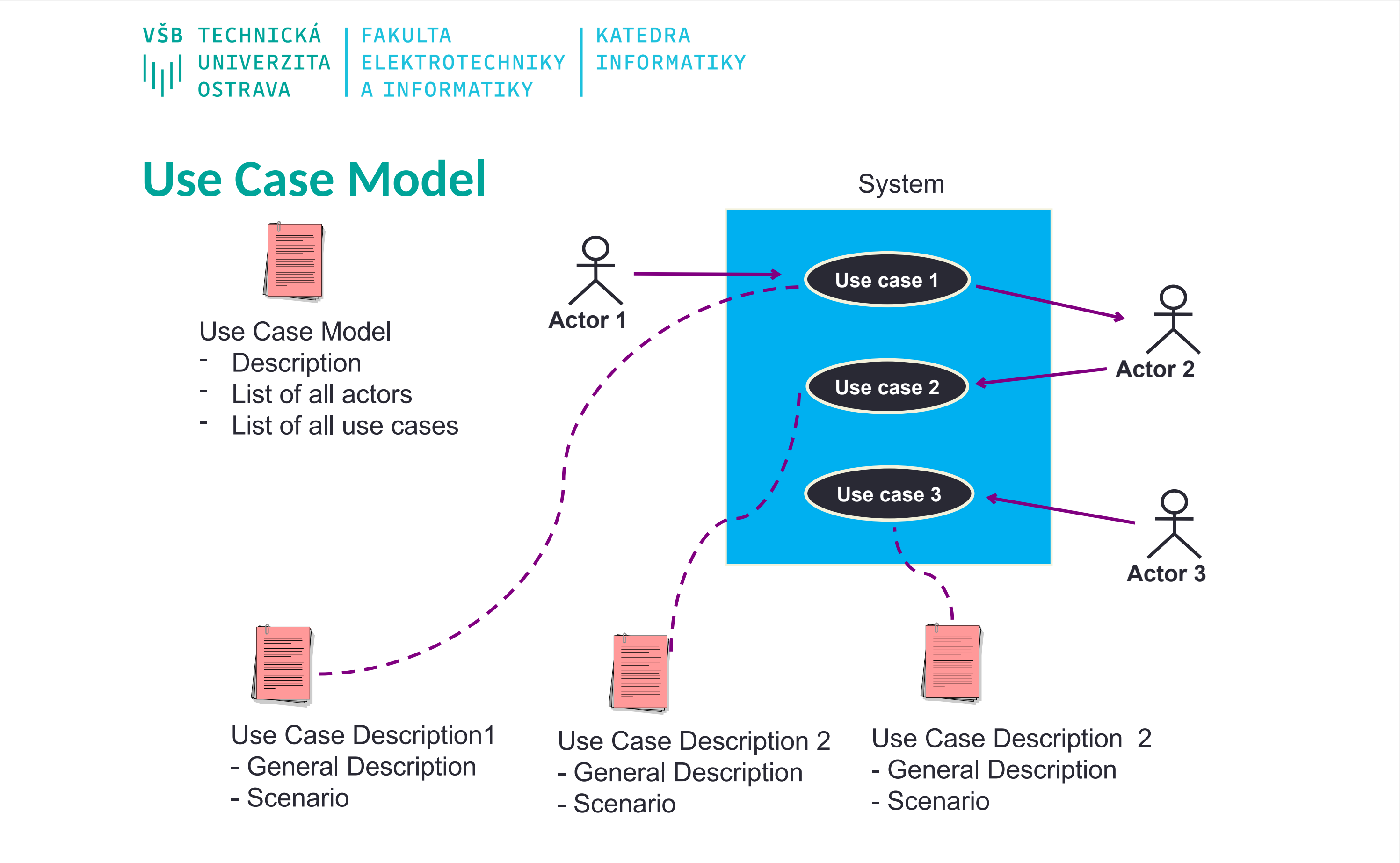
What for is the Use Case Modeling

- Transforms the customer needs in the form of software requirements
- Defines the scope of the system
- Defines the boundary of the system
- Describes the behavior of the system
- Identifies what will **be interacting** with the system
- Verifies and validates the system requirements
- Helps with the system development planning

Use case model is a model that describes the functionality of the system. The functionality is described by the use cases that are the user’s goals. Users are modeled as actors.

Let’s imagine the use case model as a menu in the restaurant. If you open the menu, you know what food is available and for what price. You know what type of cuisine ~~is~~ that restaurant is. The menu gives you a hint about what kind of behavior you can expect of the restaurant.

Since the use case model is a very effective tool to plan a development of the system, the use case model is used by all members of the development team in all phases and cycles of the development process.



Use Case Model

Use Case Model
- Description
- List of all actors
- List of all use cases

Use Case Description1
- General Description
- Scenario

Use Case Description 2
- General Description
- Scenario

Use Case Description 2
- General Description
- Scenario

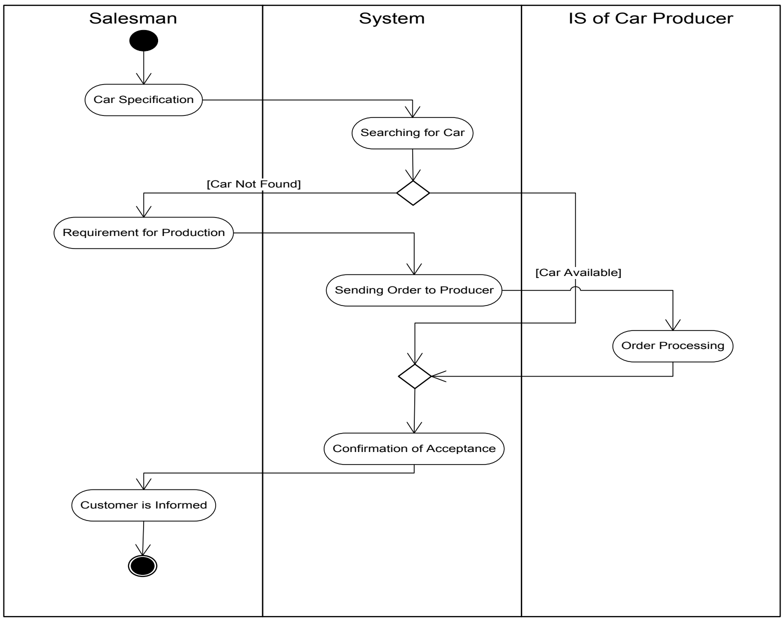
Use case model

Use case model consists of text descriptions and diagrams. Diagrams show the abstract view of the system functionality, the connection to the actors and view of the system boundary. Text describes the actors and each use case.

Each use case contains a detailed description of its own functionality.

Drawing of the diagrams is only small amount of work that needs to be done during the definition of the use case. More than 80 percent of all effort during the requirement specification is dedicated to the textual description of the use cases, non-functional requirements and rules.

Use Case Scenario – Activity Diagram

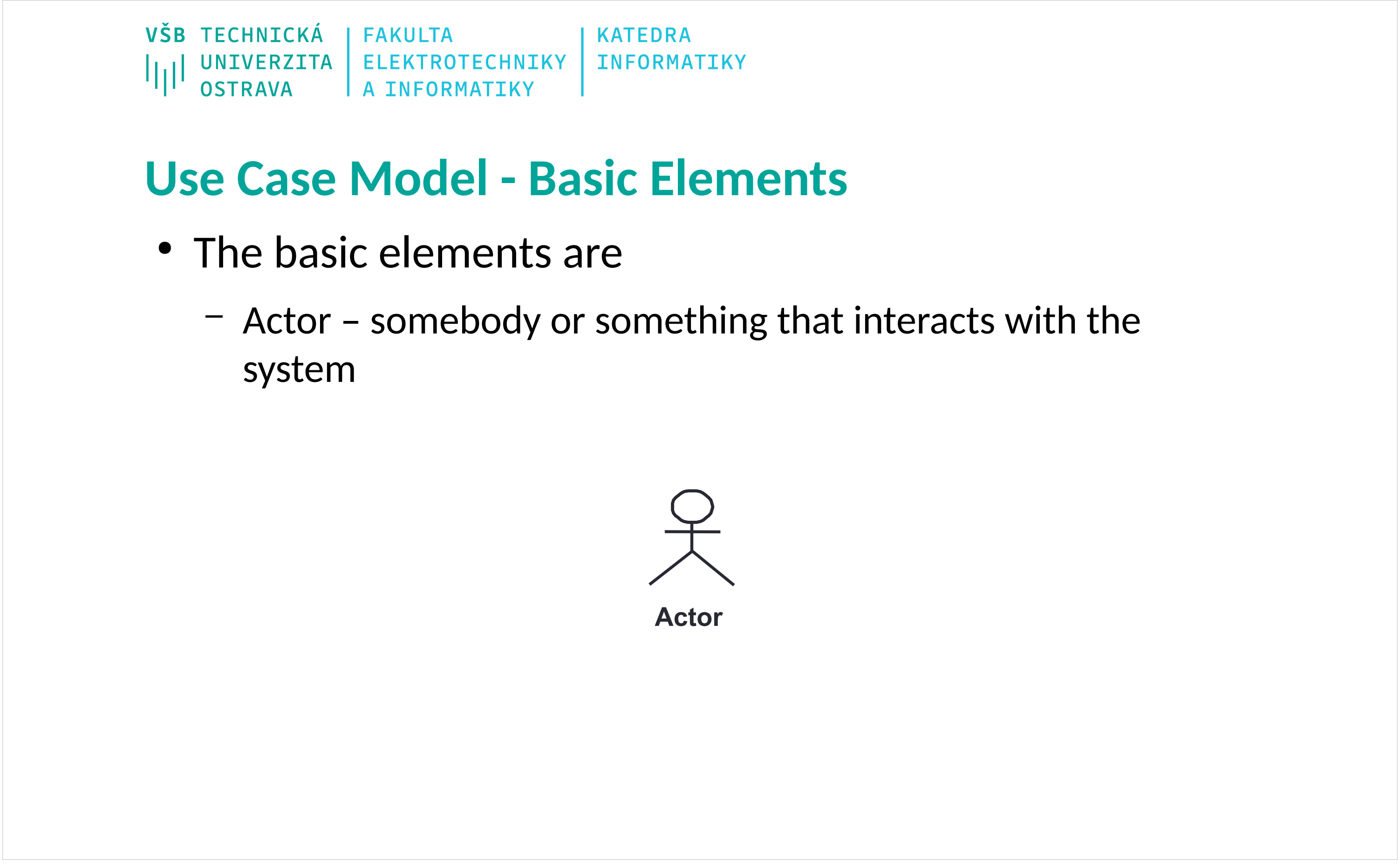


Use Case Scenario – Activity Diagram

The description of the flow of the communication between the customer and the system is called scenario.

Activity diagrams are another useful tool that can be used for the description of the scenario. It is common that more complex scenarios are described by the activity diagrams.

If there are activity diagrams used for the description of the scenarios then it is useful to use partitions in the diagrams. Partitions show the responsibility of each role to these activities.



Use Case Model - Basic Elements

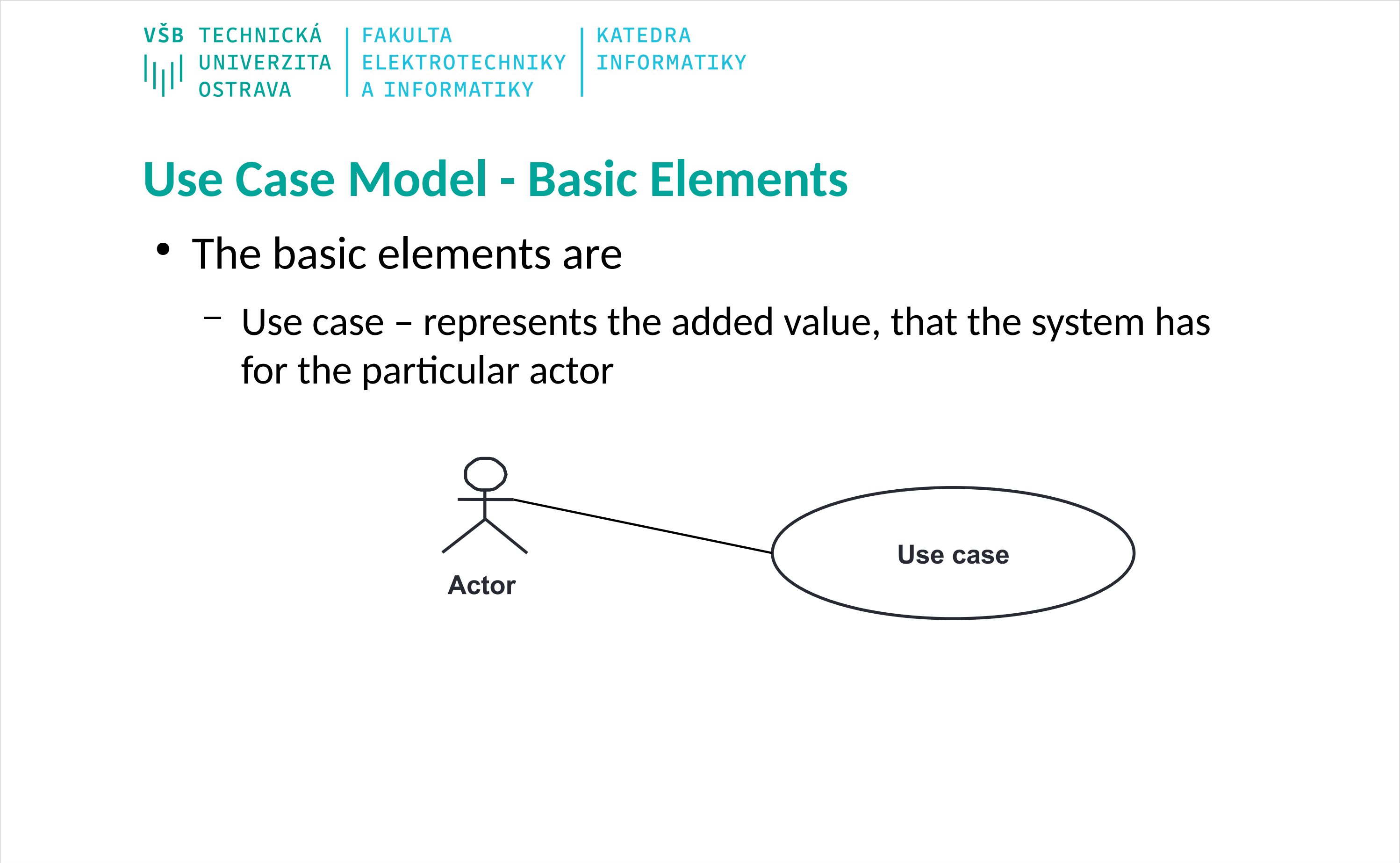
- The basic elements are
  - Actor – somebody or something that interacts with the system

The basic elements of the use case model

The basic elements are:

Actor - somebody or something that interacts with the system, is outside the system

Actor is a simple figure symbol in the diagram. The name of the actor is under the symbol.



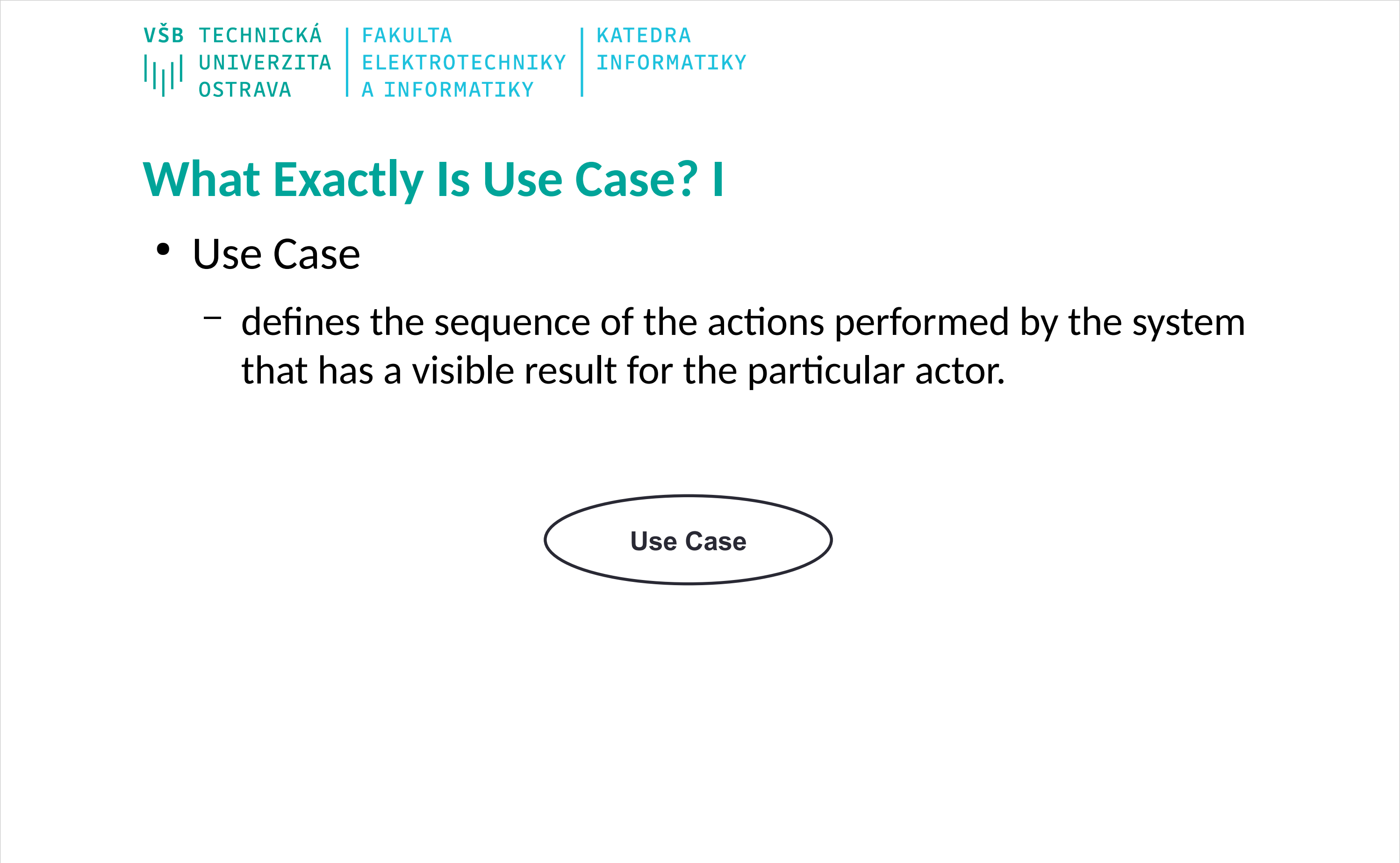
Use Case Model - Basic Elements

- The basic elements are
  - Use case – represents the added value, that the system has for the particular actor

Use case – represents the added value, that the system has for the particular actor
Use case describes the sequence of the activities that are performed by the actor and the system. Activities that are performed to reach the desired business goal that is supported by the system.

Use case describes:
System, its environment and the relations between them.
Surrounding systems or people that interact with the system.
Planned behavior of the system.

Use cases are something like containers that contain related requirements. They concentrate all requirements that are necessary to reach the desired business goal to one scenario that describes how to reach this goal.
Use case is modeled as an ellipse in the use case diagram. Name of the use case is in the middle of the ellipse or alternatively under the ellipse.



What Exactly Is Use Case? I

- Use Case
  - defines the sequence of the actions performed by the system that has a visible result for the particular actor.

What Exactly Is Use Case?

Use case describes possible communication with the system. It describes the complete walkthrough thorough the system that is necessary to reach the goal that demands the actor that is assigned to that use case.

Use case then:

Defines the sequence of the actions – that are atomic activities, decisions and requirements. Each action is performed completely or not performed at all.

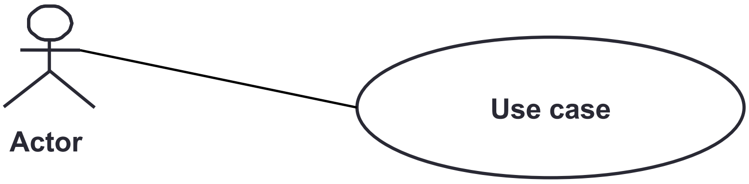
Actions performed by the system – actions that are performed by the system are functional requirements.

Actions that have visible results – it is important to remember that the use case has to have a visible result. Why should anybody use the system if there is no required result? If the use case is not helpful to anybody then it’s probably too small. It is necessary to combine it with other use case to reach the desired goal for the particular actor.

What Exactly Is Use Case? II

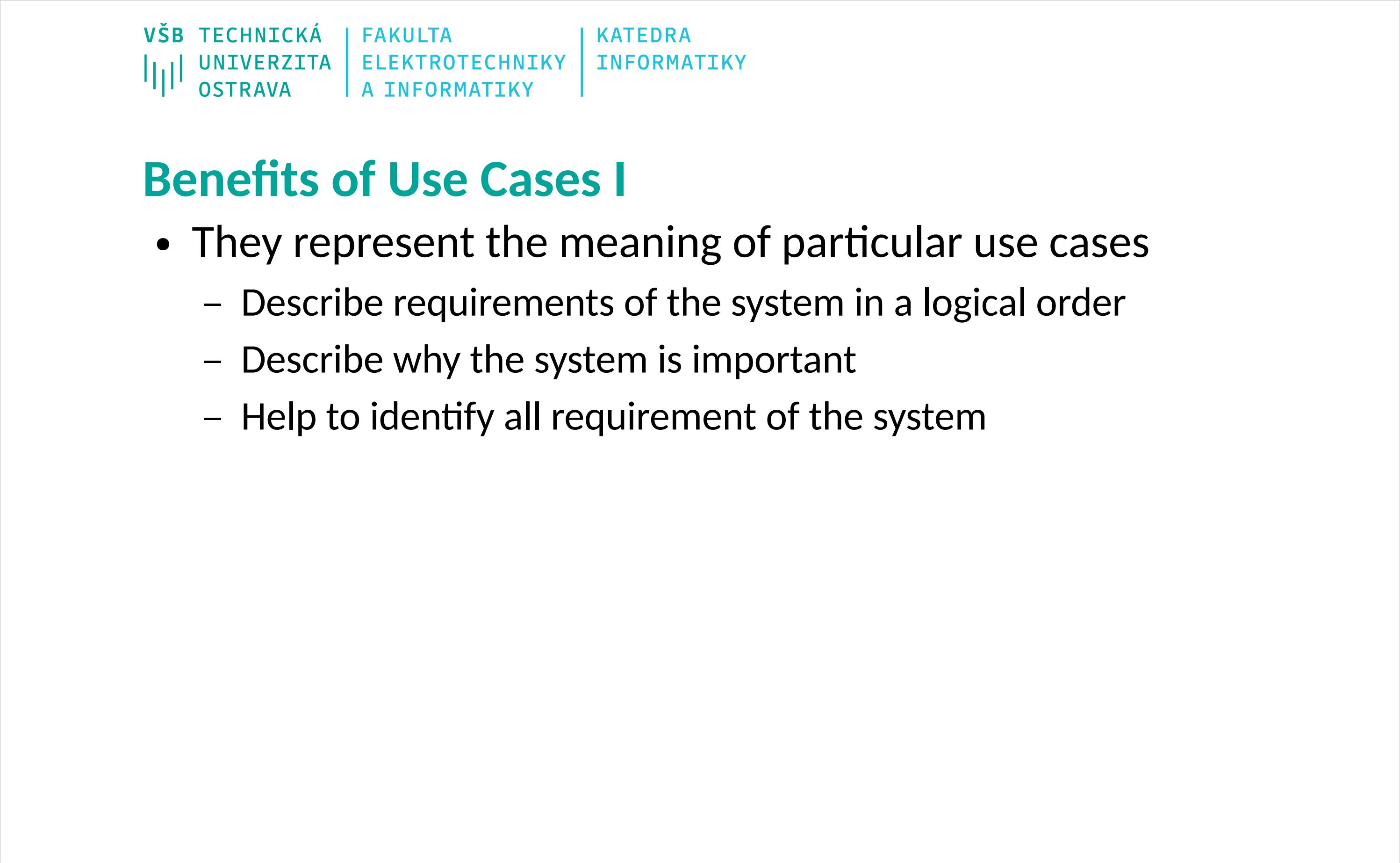
- Use Case

defines the sequence of the actions performed by the system that has a visible result for the particular actor.



For the particular actor – decision, which actor demands the result of the particular use case helps to eliminate too general use cases or too small use cases. If there are more actors that can reach different business goals by one use case then this use case is too general and tries to reach too many goals for different actors. If the goal is too small and it has no business value, than the use case is too small and should it be combined with other use cases to reach the business goal.

There are situations where there are more actors cooperating on one use case. Since only one actor reaches the desired business goal, this connection is allowed. The actor that reaches its goal is called primary actor. Primary actor is usually actor who initialized the use case. Other cooperating actors are called secondary actors.



Benefits of Use Cases I

- They represent the meaning of particular use cases
  - Describe requirements of the system in a logical order
  - Describe why the system is important
  - Help to identify all requirement of the system

Benefits of the Use Cases

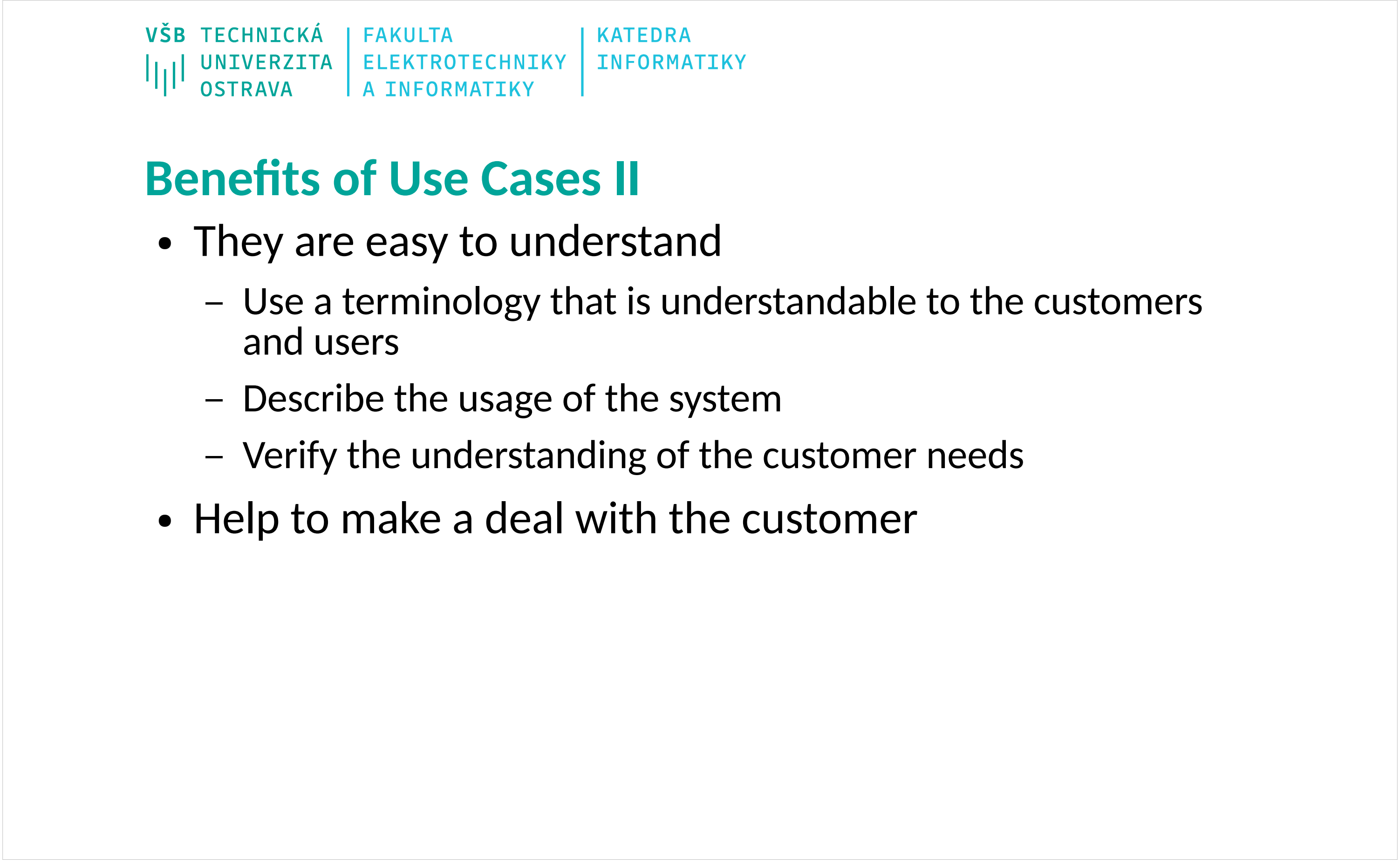
Use cases are a tool to organize requirements from the perspective of the user. All requirements that help (to) reach one particular goal for the user are collected to the use case that reaches the business goal. Use case model is then the collection of all particular use cases.

Benefits are:

They represent the meaning of particular use cases

Use cases show why the system is necessary and what goals can be reached by the using the system.

Use case then describes the requirements in a logical order, describes, why the system is important and helps to identify all requirements of the system.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Benefits of Use Cases II

- They are easy to understand
  - Use a terminology that is understandable to the customers and users
  - Describe the usage of the system
  - Verify the understanding of the customer needs
- Help to make a deal with the customer

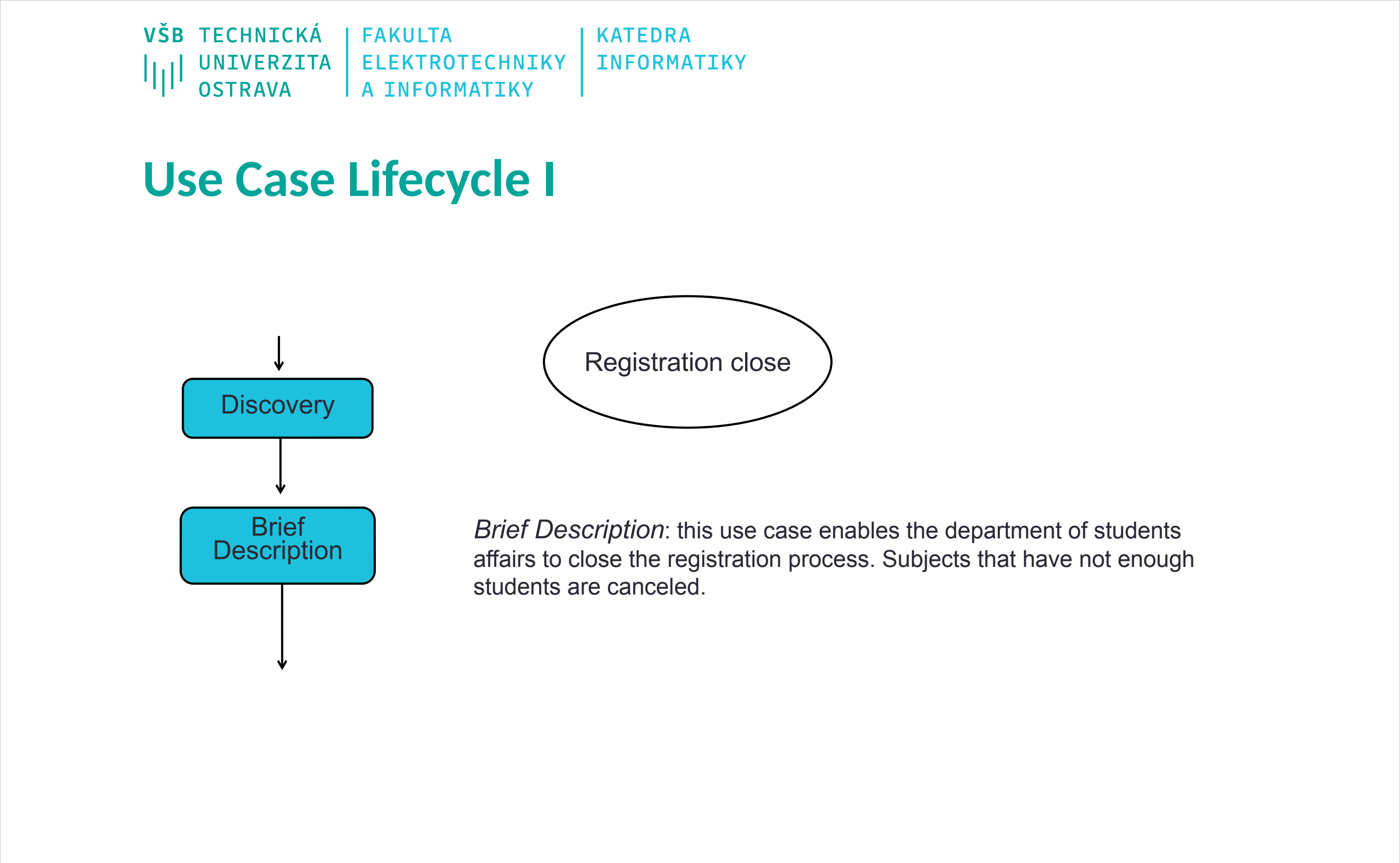
Benefits of the Use Cases II

Use cases are easy to understand

Use case model describes the requirements from the perspective of the user by normal language. Use case model shows what the user thinks that the system should do.

Helps to make a deal with the customer

Use case model is a mediator between the customer and the system developers.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Use Case Lifecycle I

Discovery

Brief Description

Registration close

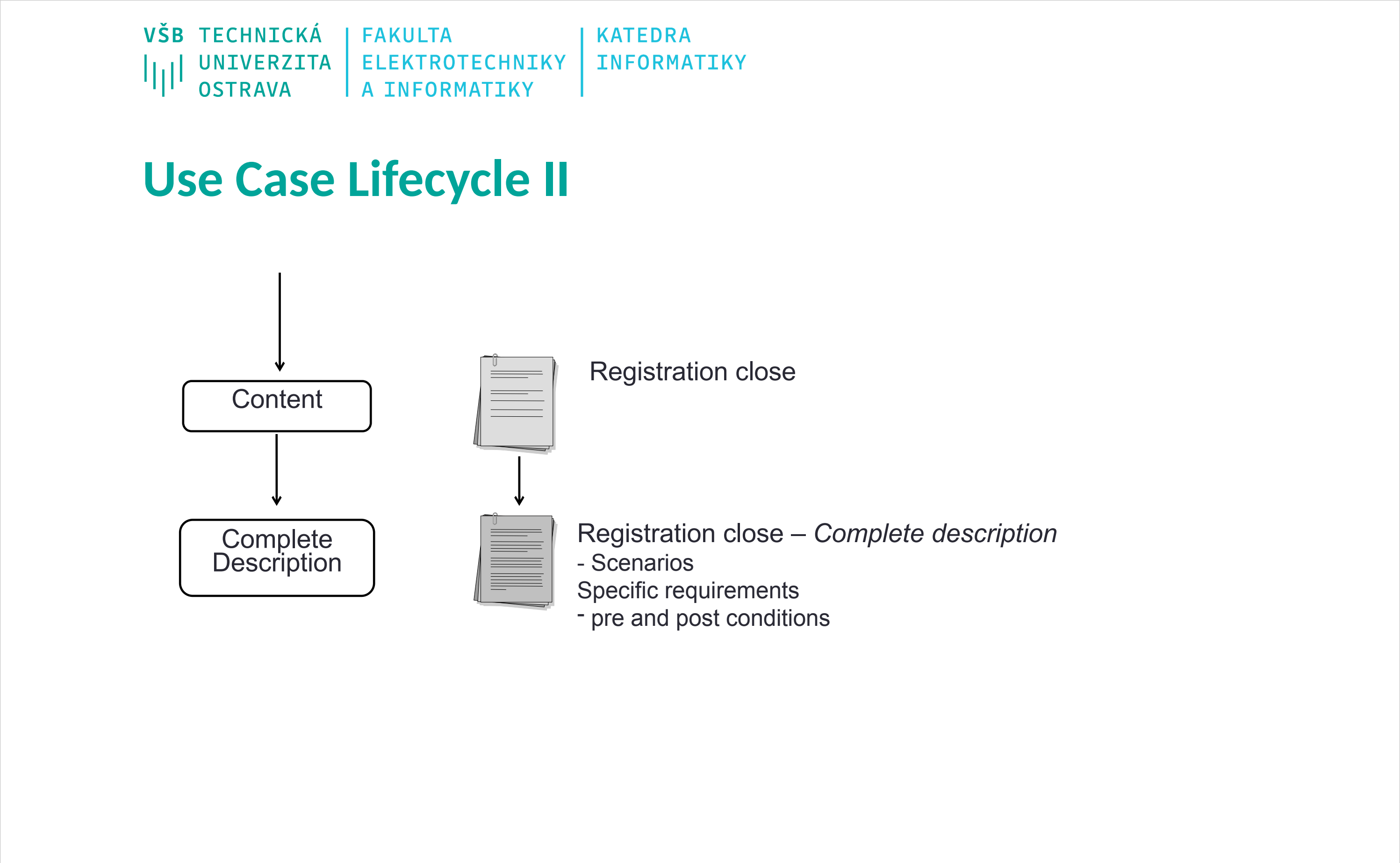
Brief Description: this use case enables the department of students affairs to close the registration process. Subjects that have not enough students are canceled.

Use case lifecycle

In most cases, it is not possible to describe use case completely. Similarly as any other iterative process, the description of the use case is continuous development activity that starts with the discovery of the use case and ends with its complete description.

The first step is a discovery of the use case. That is made by the identification of the use case name, goal and the name of primary actor of the use case. Use case is discovered if it is somehow named.

Immediately after the discovery of the use case there is another step. That step is a brief description of the use case. Brief description describes the goal of the use case, the description is made by several sentences. For example: we have identified the use case Registration close. Its brief description is: this use case enables the department of students affairs to close the registration process. Subjects that have not enough students are canceled.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Use Case Lifecycle II

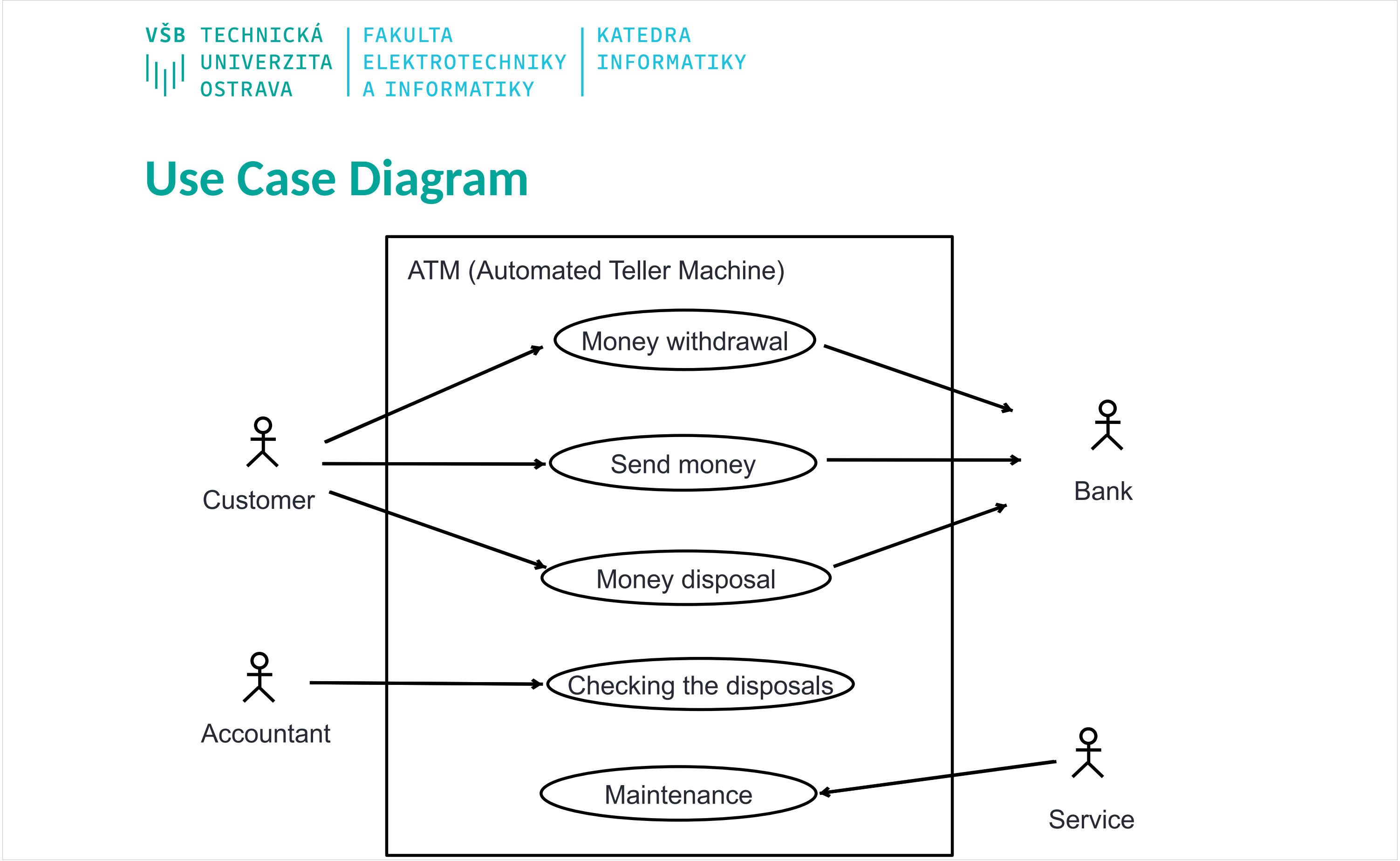
Registration close

Registration close – *Complete description*
- Scenarios
Specific requirements
- pre and post conditions

Use case lifecycle

Next step is a description of the use case content. Content means a simple list that describes the basic workflow of the use case and the identification of the alternative flows. Description of the content helps us to identify scenarios and helps us to understand how extensive the use case will be and how much effort will be necessary to implement it.

The last step is a detailed and complete description od the use case. Complete description is made iteratively, when there is a need to implement that particular scenario.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Use Case Diagram

Use Case diagram

Use case model describes what the system should do, its environment and relations between the actors and use cases.

The use case diagram is a graphical representation of the use case model.

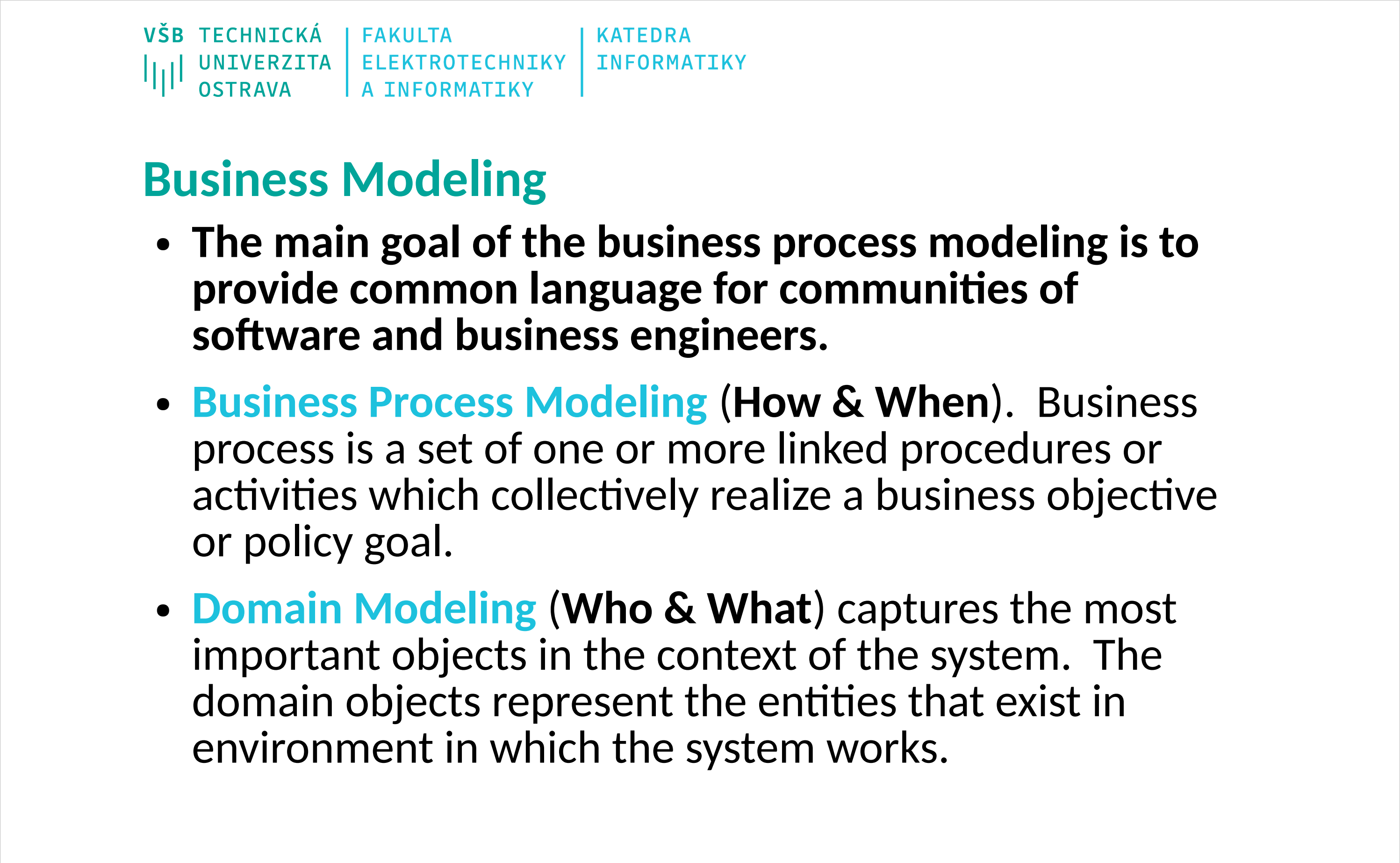
For example the use case diagram of the ATM machine.

When you see that diagram you can easily describe the basic function of the ATM machine system and its priorities.

The priority is the withdrawal of the money. The ATM machine needs that use case as a basic functionality. Other use cases, send money and money disposals disposal are complementary functions to the basic function money withdrawal, but they are necessary for the architecture of the system.

It is useful to distinguish between primary and secondary use cases. Primary use cases are those use cases that are directly connected to the business goals that the system should support. Secondary use cases are those use cases that are necessary because of the technical solution. For example – maintenance of the ATM, backup etc.

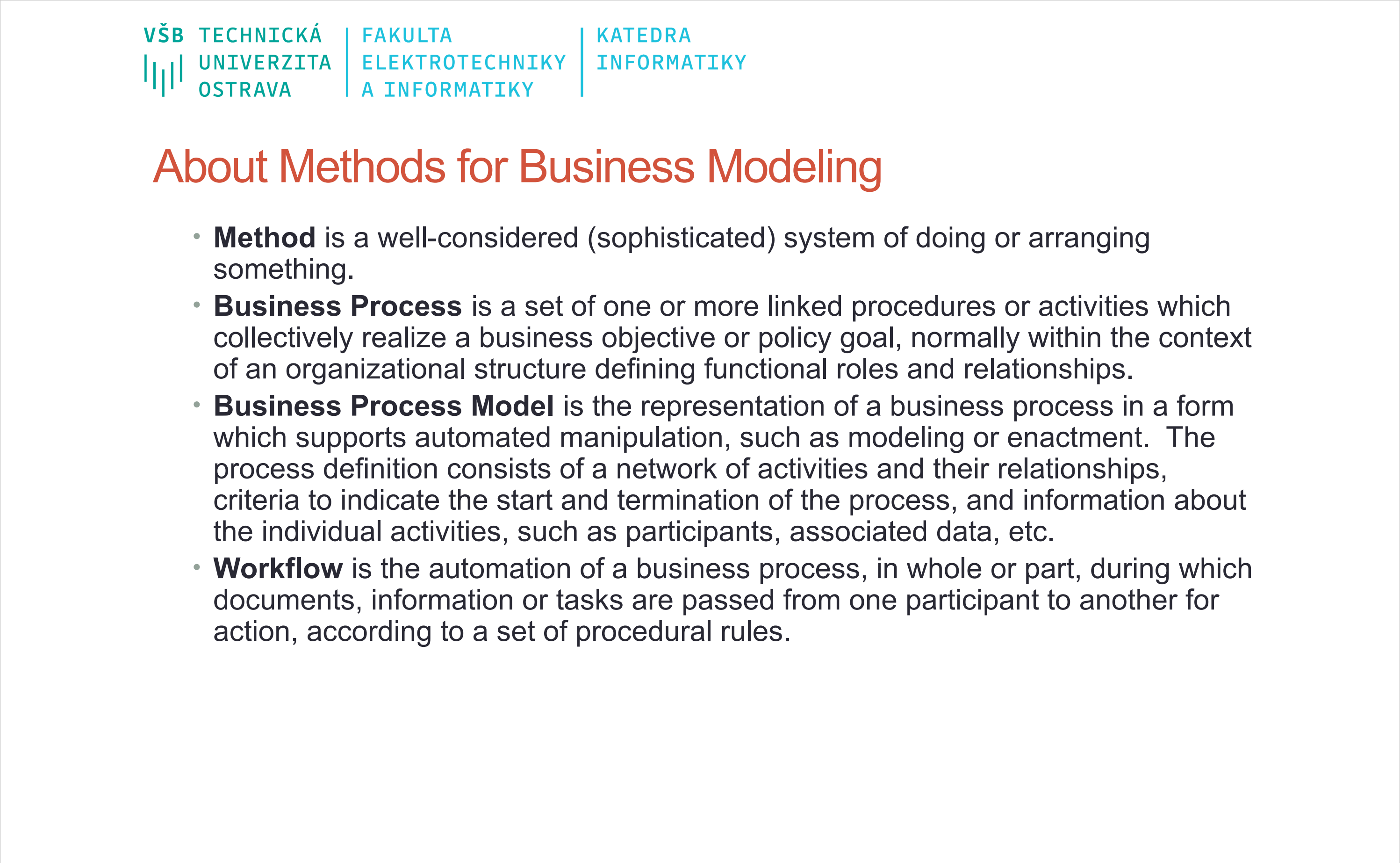
Use cases might be described in more diagrams than one for example to highlight the important use cases of each actor.



Business Modeling

- **The main goal of the business process modeling is to provide common language for communities of software and business engineers.**
- Business Process Modeling (**How & When**). Business process is a set of one or more linked procedures or activities which collectively realize a business objective or policy goal.
- Domain Modeling (**Who & What**) captures the most important objects in the context of the system. The domain objects represent the entities that exist in environment in which the system works.

Why to use business modeling? Business modeling can be used as a tool to understand the problem domain. If the domain is not known to the developer, the business modeling can be used. On the other hand business modeling helps to understand the goal of the software system itself and does not have to be used only for the understanding of the domain.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

About Methods for Business Modeling

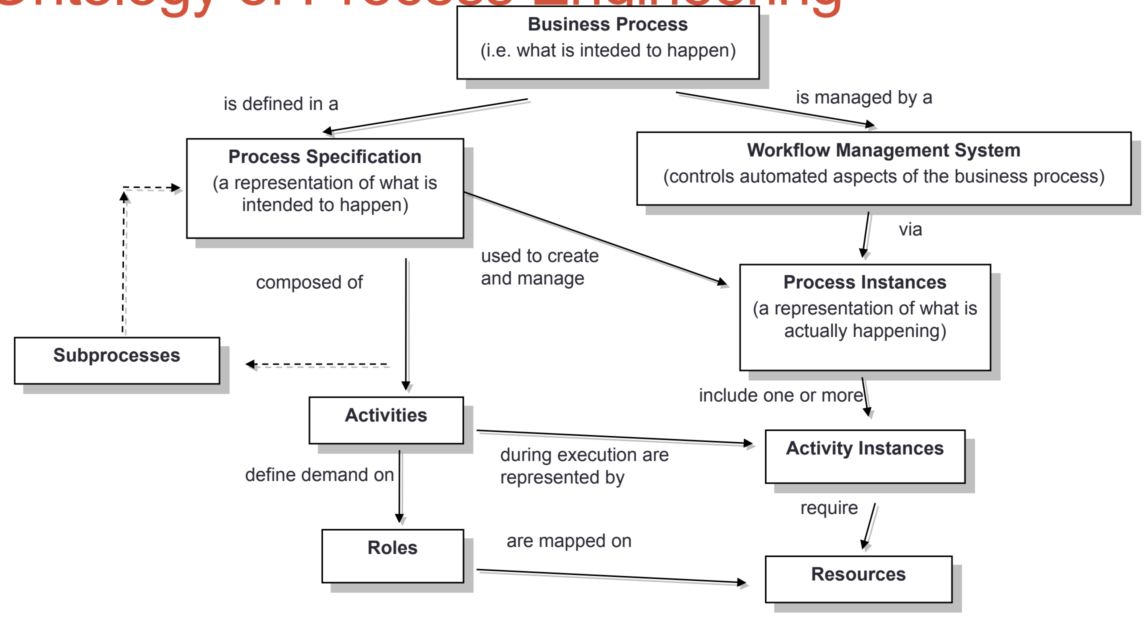
- **Method** is a well-considered (sophisticated) system of doing or arranging something.
- **Business Process** is a set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships.
- **Business Process Model** is the representation of a business process in a form which supports automated manipulation, such as modeling or enactment. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated data, etc.
- **Workflow** is the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.

Methods for business modeling represent a systematic way of specifying specify and analyzing business processes.

Purpose of Business Modeling

- **Business Process Re-engineering (BPR)** - methods that support activities by which an enterprise reexamines its goals and how it achieves them, followed by a disciplined approach of business process redesign.
- **Enterprise Resource Planning (ERP)** - an information system that integrates all manufacturing and related applications for an entire enterprise. Business modeling is the first step in the software process of the ERP implemenation.
- **Workflow Management (WFM)** – generic software systems used for definition, management, enactment and control of business processes.

Ontology of Process Engineering



Business Process
(i.e. what is inteded to happen)

is defined in a

is managed by a

Process Specification
(a representation of what is intended to happen)

Workflow Management System
(controls automated aspects of the business process)

via

composed of

used to create and manage

Process Instances
(a representation of what is actually happening)

Subprocesses

include one or more

Activities

Activity Instances

during execution are represented by

define demand on

require

Roles

are mapped on

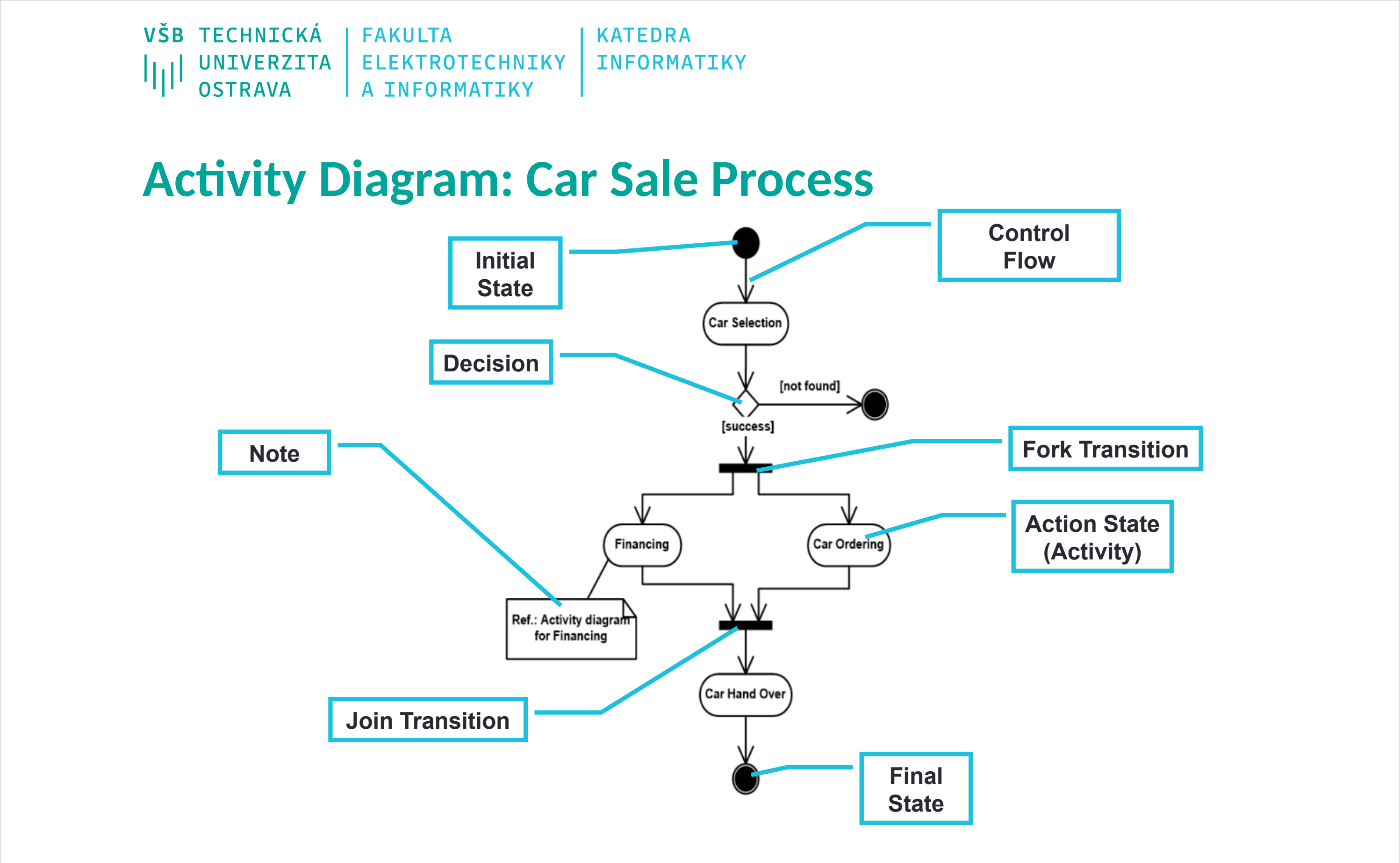
Resources

UML Diagrams for Business Modeling

- Activity Diagram is a variation of a state machine in which the states represent the performance of activities and the transitions are triggered by their **completion**.
  - The purpose of this diagram is to focus on flows driven by internal processing.
- Class Diagram is a graph of elements (in the scope of business modeling represented by **workers** and **entities**) connected by their various static relationships.
  - The purpose of this diagram is to capture static aspect of the business domain.

Motivating Example

- Develop an information system for a car dealer. The application should collect and provide information about customers, their orders, cars, payments etc. The possibility to communicate with the car manufacturer to obtain updated offer of available cars or to order a car required by a customer should be the part of the system. The goal is to make the customer happy!



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

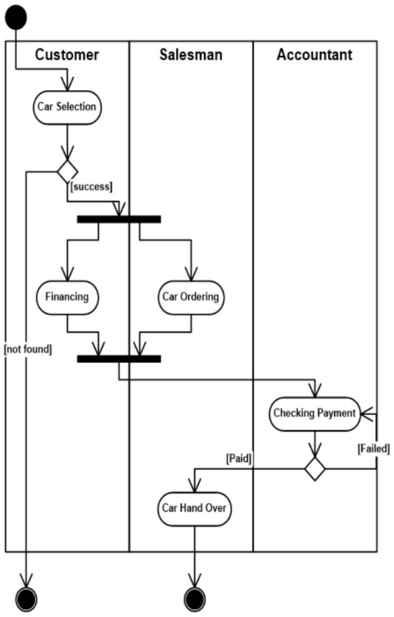
Activity Diagram: Car Sale Process

Activity Diagram: Car Sale Process

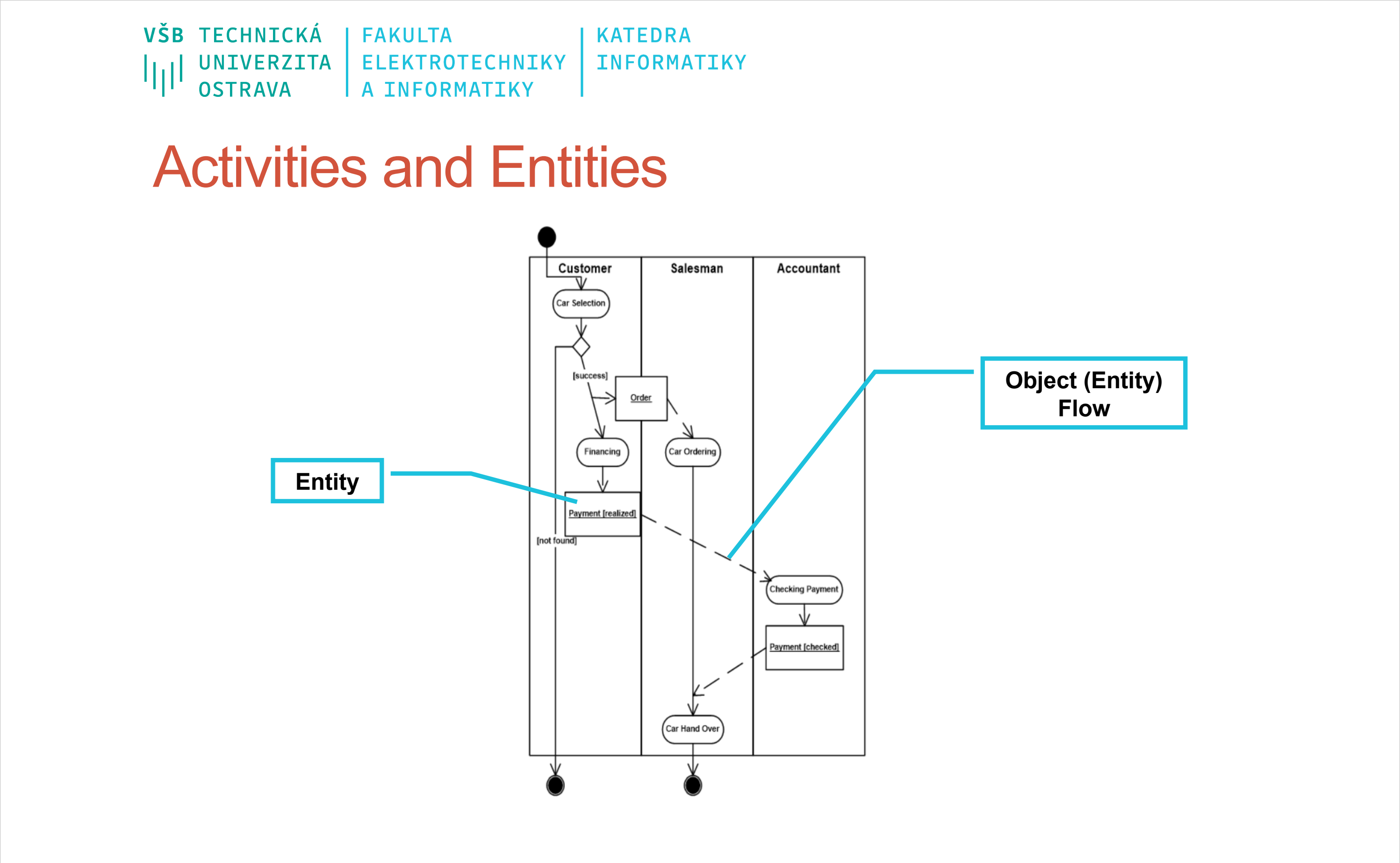
Let us explain the business modeling captured by the activity diagram on this picture. The process starts with the activity Car selection. Then there is a decision whether the car was found or not. When the car was not found, the process ends.

When the car was found, the parallel part begins, two activities can be performed concurrently – Financing and Car ordering. When both these activities are completed the synchronization part lets the control flow continue to the activity Car hand over. This is the simple description of the Car sale process in the UML activity diagram.

Swimlanes: Packages of Responsibilities



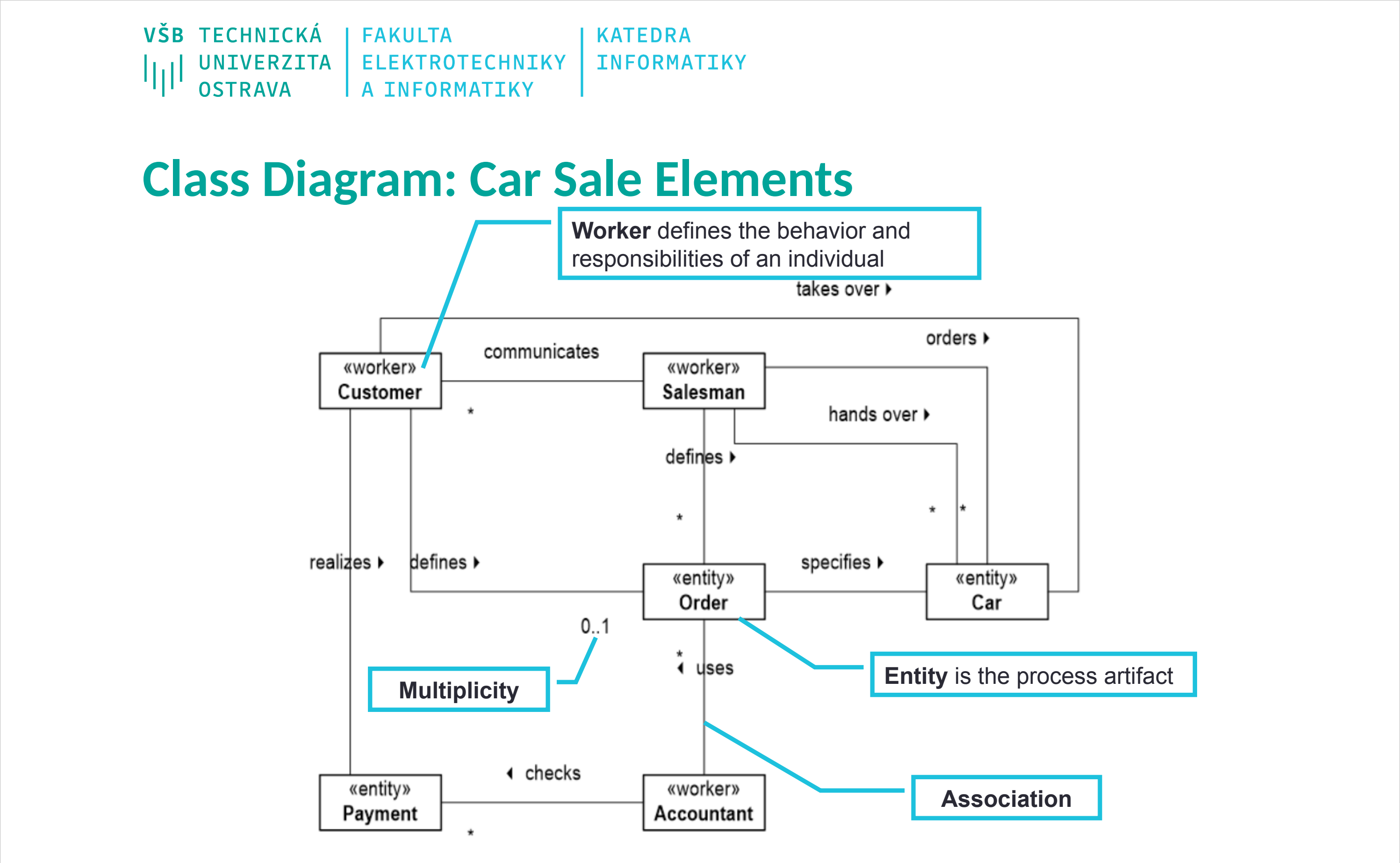
Swimlanes: Packages of Responsibilities
Actions may be organized into **swim lanes**. Swim lanes are a kind of package for organizing **responsibility for activities** provided by workers.
This diagram shows the previous process divided into the swim lanes. There are three swim lanes are Customer, Salesman, Accountant. We can see that the Customer is responsible for the Car Selection, Financing, Salesman is responsible for the Car Ordering, Accountant is responsible for Checking Payment.



Activities and Entities

Activities and Entities

Except of a activities and roles responsible for the particular activities, the diagram can show also the objects. Objects flow between the activities, objects are input and output data from and to activities. We can see, that the object Order, was created in the activity Car Selection. This object is then input to the activity Car Ordering. Object Payment in the state realized is the output of the activity Financing. Payment is then input object to the activity Checking Payment. The output of the checking payment activity is a object Payment again, but in the state checked. This payment is then the input activity to the activity Car Hand Over.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Class Diagram: Car Sale Elements

Worker defines the behavior and responsibilities of an individual

Entity is the process artifact

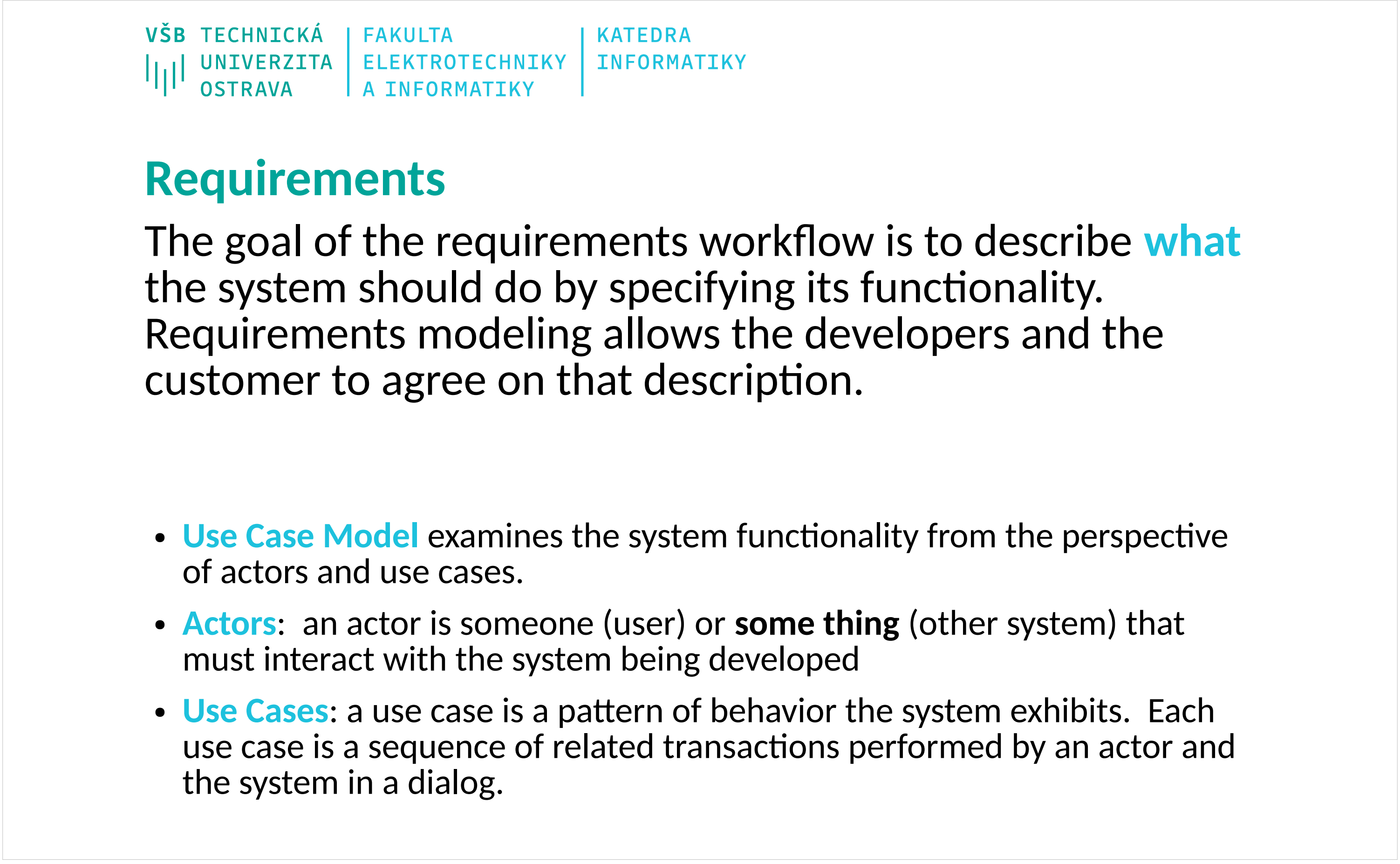
Multiplicity

Association

Class Diagram: Car Sale Elements

The activity diagram is not the only diagram that can be used for the documentation of the business processes. The class diagram is used there for the description of the static part of the process. The static part means objects and their relations. We can see, that the process car sale need some workers – and entities. Workers are individuals, humans, that do some tasks or are responsible for some tasks. Entities are artifacts that are created or used during the process.

There are entities> Payment, Order, Car, and workers Salesman, Customer and accountant.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Requirements

The goal of the requirements workflow is to describe **what** the system should do by specifying its functionality. Requirements modeling allows the developers and the customer to agree on that description.

- **Use Case Model** examines the system functionality from the perspective of actors and use cases.
- **Actors**: an actor is someone (user) or **some thing** (other system) that must interact with the system being developed
- **Use Cases**: a use case is a pattern of behavior the system exhibits. Each use case is a sequence of related transactions performed by an actor and the system in a dialog.

Requirements

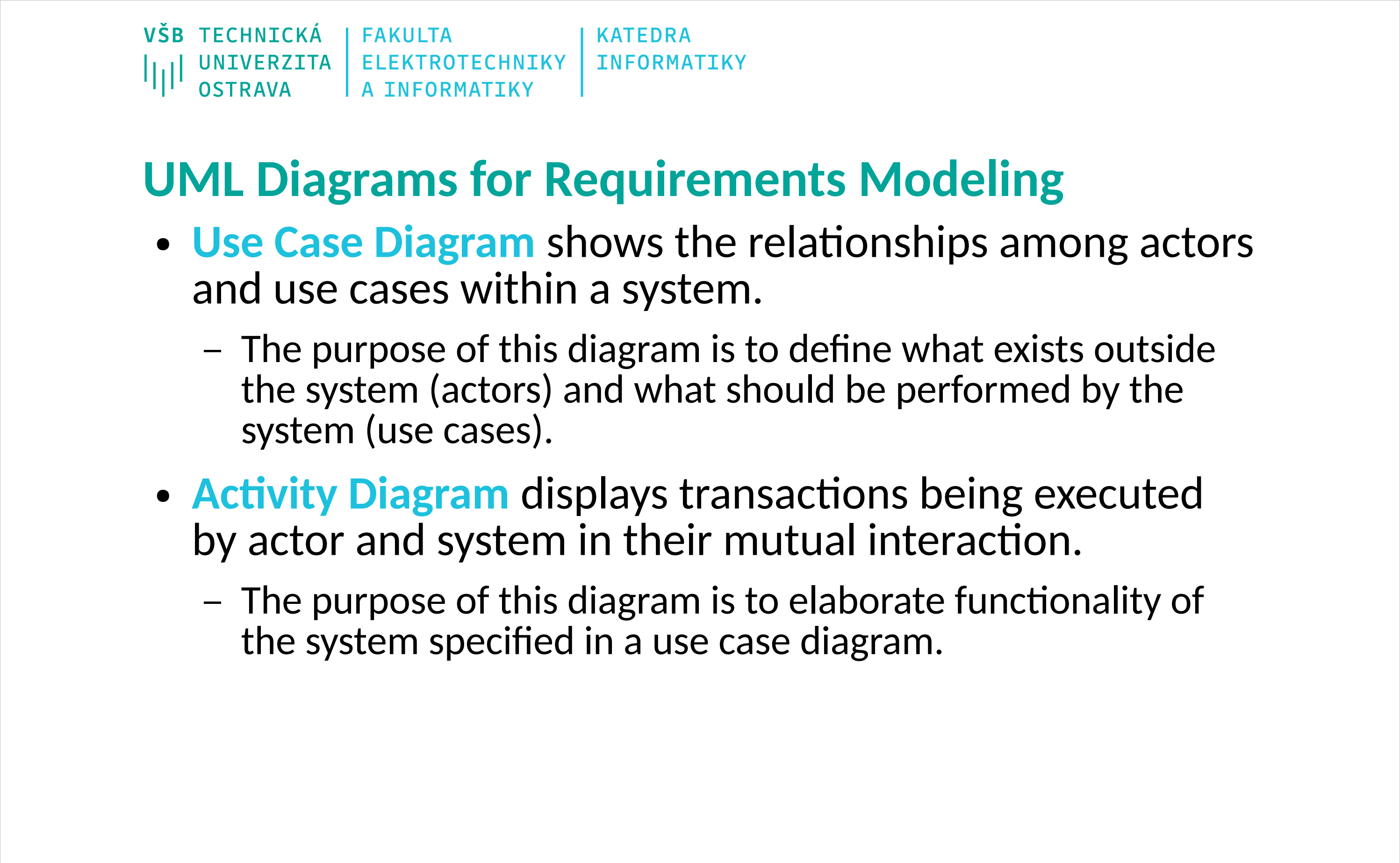
Let’s repeat what are the requirements and how to capture the requirements by the use cases.

The goal of the requirements workflow is to describe **what** the system should do by specifying its functionality. Requirements modeling allows the developers and the customer to agree on that description.

Use Case Model examines the system functionality from the perspective of actors and use cases.

Actors: an actor is someone (user) or some thing (other system) that must interact with the system being developed

Use Cases: ~~an~~ a use case is a pattern of behavior the system exhibits. Each use case is a sequence of related transactions performed by an actor and the system in a dialog.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

UML Diagrams for Requirements Modeling

- **Use Case Diagram** shows the relationships among actors and use cases within a system.
  - The purpose of this diagram is to define what exists outside the system (actors) and what should be performed by the system (use cases).
- **Activity Diagram** displays transactions being executed by actor and system in their mutual interaction.
  - The purpose of this diagram is to elaborate functionality of the system specified in a use case diagram.

UML Diagrams for Requirements Modeling

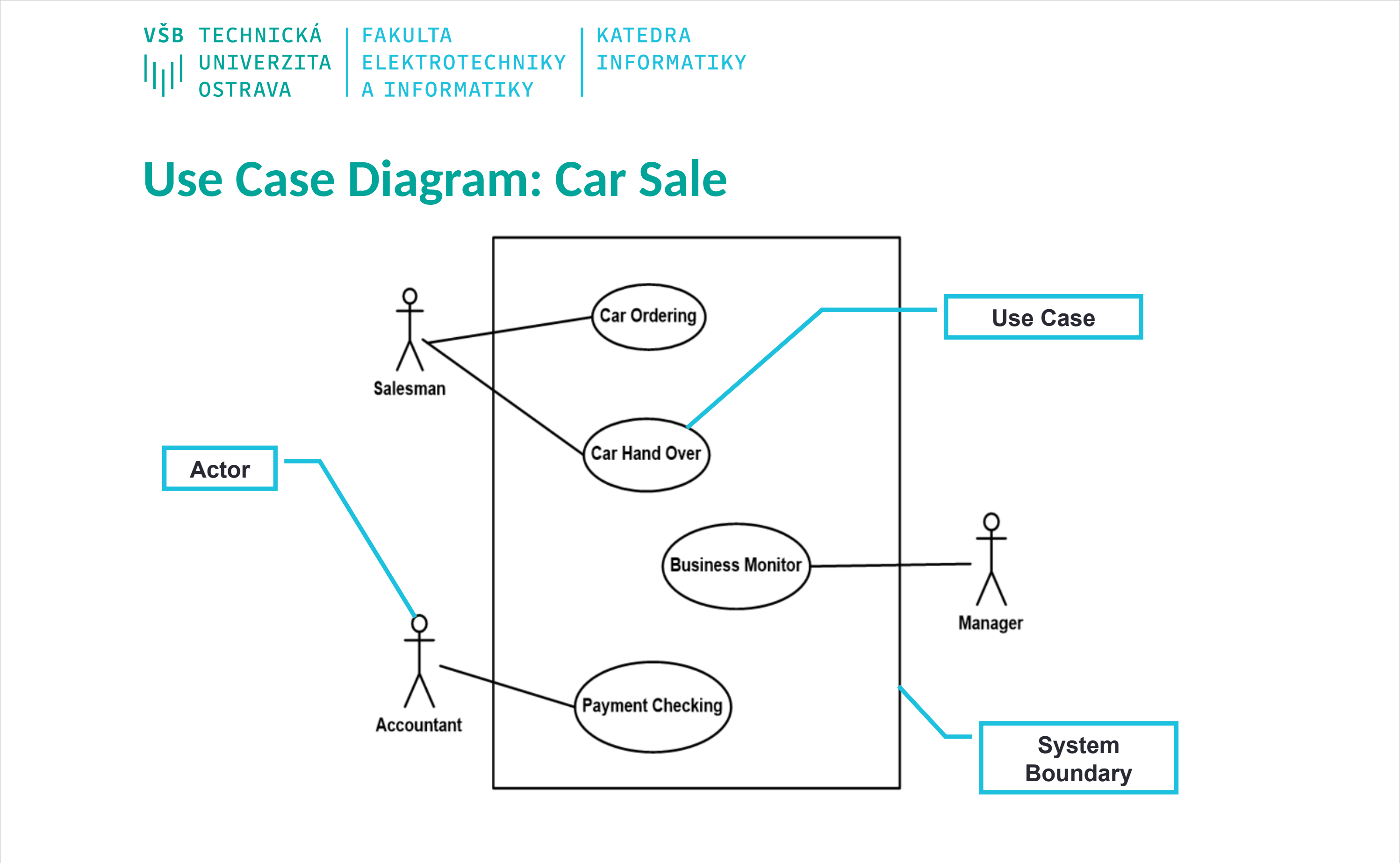
As we have already thoroughly discussed, sometime we need to show and visually model the requirements. We can use two types of the UML diagram for this purpose.

Use Case Diagram shows the relationships among actors and use cases within a system.

The purpose of this diagram is to define what exists outside the system (actors) and what should be performed by the system (use cases).

Activity Diagram displays transactions being executed by actor and system in their mutual interaction.

The purpose of this diagram is to elaborate functionality of the system specified in a use case diagram.



VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

Use Case Diagram: Car Sale

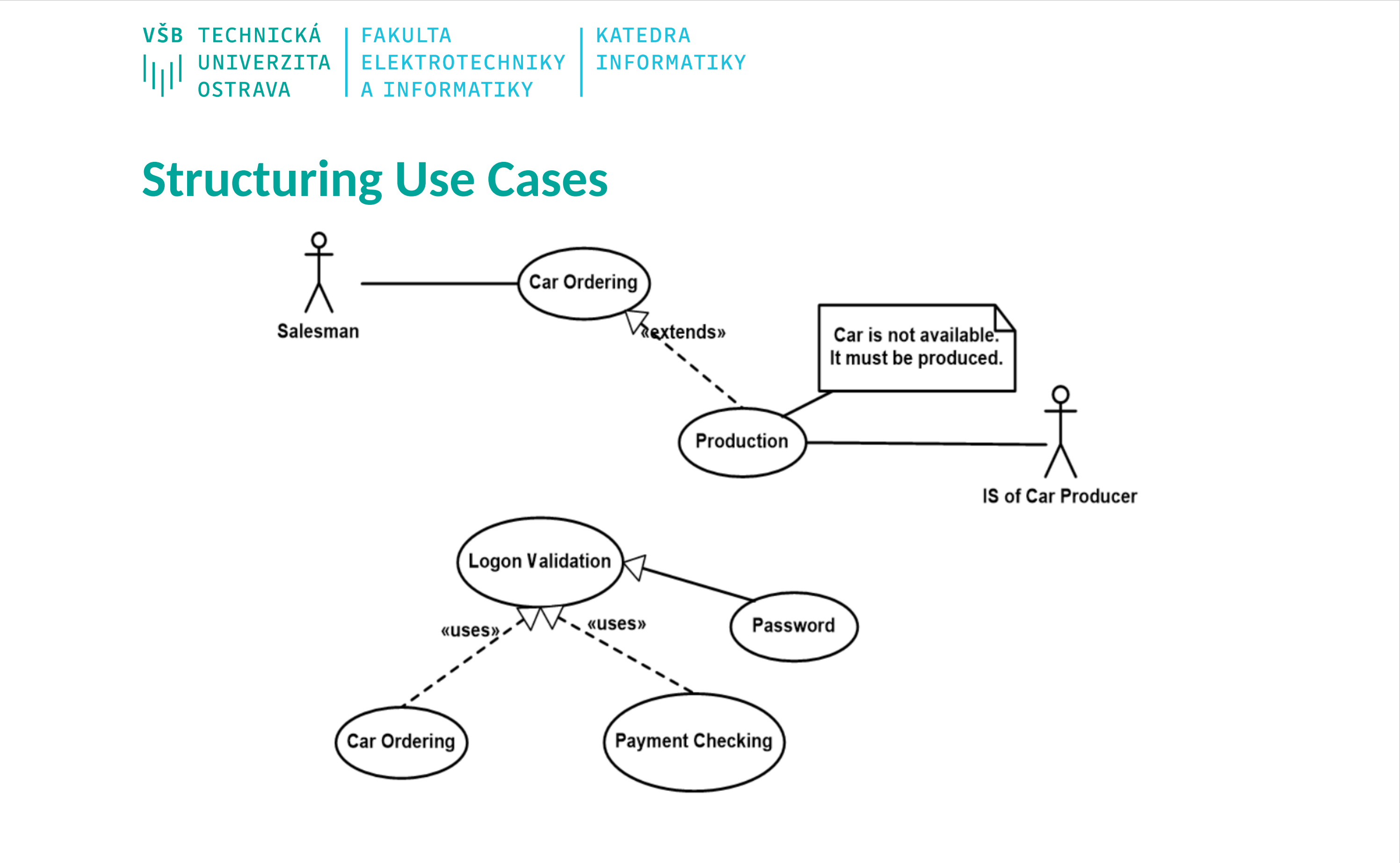
Use Case Diagram: Car Sale

We can now start to model the use case model for the car sale system. We use a use cases as functional requirements, and use a use case diagram as a tool to express and visually show the relations between the use cases and actors. And the boundary and scope of the system.

We can see that we have three actors – user of the system to be built – Salesman, Accountant and Manager.

Salesman starts and has a connection to the use cases Car Ordering, and Car Hand Over.

Manager has a connection to the Business Monitor. Accountant has a connection to the Payment Checking.



Structuring Use Cases

Structuring Use Cases

Since there are many use cases and the support use cases might be used by more than one use case it is possible to use relations.

The relations are:

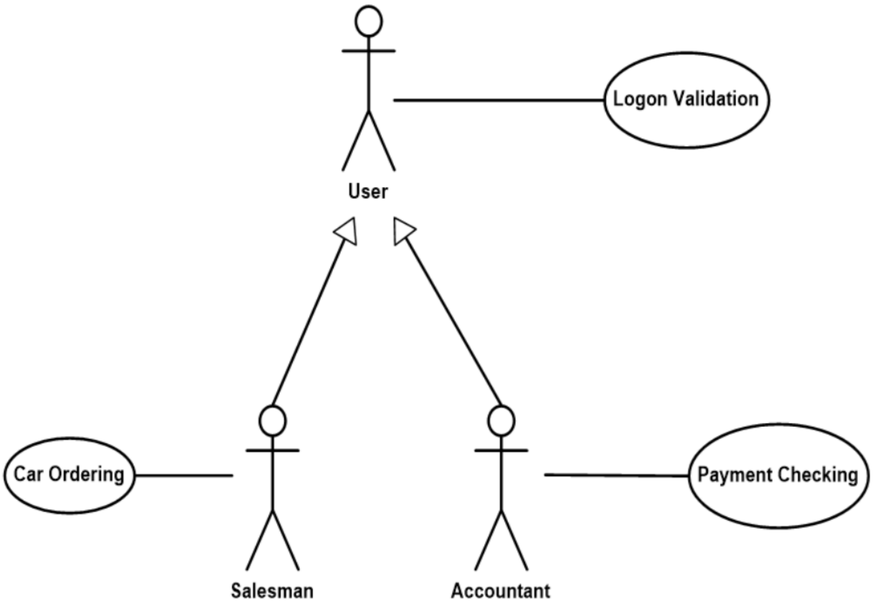
A **generalization** is the relationship between a more general use case (the parent) and a more specific use case (the child) that is fully consistent with first use case.

An **extends** relationship shows optional behavior

A **uses** relationship shows behavior that is common to one or more use cases

For example we can see, that the Car ordering is extended sometimes by the Car Production. Car Ordering uses Logon Validation, Payment Checking use a Logon Validation.

Structuring Actors



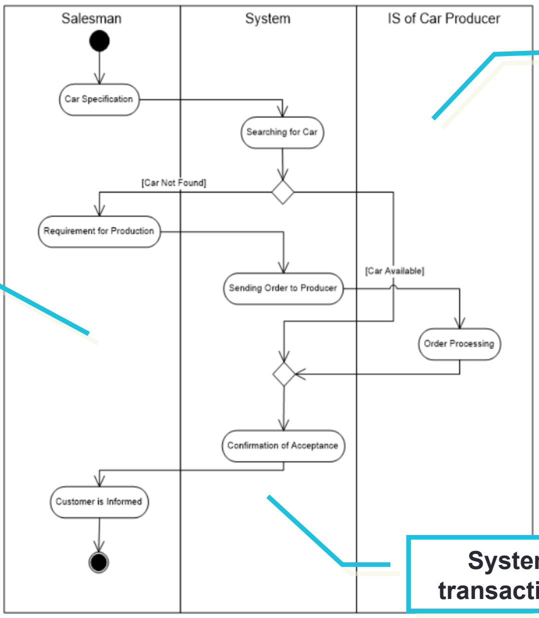
Structuring Actors

There might be many actors as well and they can use the same use cases. If this situation occurs the diagram would looks like a mess. There would be a lot of lines connecting particular actors with their use cases. For example, Salesman have a use case car ordering, Accountant have a use case Payment Checking. Both these actors has a use case logon validation. In case that we would like to show that in one diagram the diagram might start to be unreadable. Hence, we can use a structuring between actors.

A **generalization** is the relationship between a more general actor (the parent) and a more specific actor (the child) that is fully consistent with first actor.

We can even add a new (sometimes abstract like here) actor - user, that is connected to the use case Logon Validation. Actors Salesman and Accountant are specialization (or we can say – they inherit from the actor) of the actor User.

Elaborate Functionality of Car Ordering



Actor’s responsibility

Actor’s responsibility

System transactions

Elaborate Functionality of Car Ordering

On this picture, we can see what the final requested functionality of the system is. The salesman performs Car Specification, system then searches for the car. If the car is not found, the system informs salesman that the production must be done. Salesman confirms that and the system should send the order to the producer, Car producer has to process the order and confirm it. If the car is available, when the system searches for it, the flow continues to the confirmation directly. Then the system informs a salesman about the order and the salesman has to inform the customer. The process ends. That is the final derived functionality of one requirement.

Elements of a Use Case

Depending on how in depth and complex you want or need to get, use cases describe a combination of the following elements:

- **Actor** – anyone or anything that performs a behavior (who is using the system)
- **Stakeholder** – someone or something with vested interests in the behavior of the system under discussion (SUD)
- **Primary Actor** – stakeholder who initiates an interaction with the system to achieve a goal
- **Preconditions** – what must be true or happen before and after the use case runs.
- **Triggers** – this is the event that causes the use case to be initiated.
- **Main success scenarios [Basic Flow]** – use case in which nothing goes wrong.
- **Alternative paths [Alternative Flow]** – these paths are a variation on the main theme. These exceptions are what happen when things go wrong at the system level.

Use Case - Fully dressed

Cockburn describes a more detailed structure for a use case but permits it to be simplified when less detail is needed. His fully dressed use case template lists the following fields:

- Title: "an active-verb goal phrase that names the goal of the primary actor"[25]
- Primary Actor
- Goal in Context
- Scope
- Level
- Stakeholders and Interests
- Precondition
- Minimal Guarantees
- Success Guarantees
- Trigger
- Main Success Scenario
- Extensions
- Technology & Data Variations List

Use Case – Example

- **Use Case:** Edit an article
- **Primary Actor**: Member (Registered User)
- **Scope**: a Wiki system
- **Level**: ! (User goal or sea level)
- **Brief**: (equivalent to a user story or an epic)
- The member edits any part (the entire article or just a section) of an article they are reading. Preview and changes comparison are allowed during the editing.
- **Stakeholders**
  - …
- **Postconditions**
  - **Minimal Guarantees:**
  - **Success Guarantees:**
    - The article is saved and an updated view is shown.
    - An edit record for the article is created by the system, so watchers of the article can be informed of the update later.
- **Preconditions**:
  - The article with editing enabled is presented to the member.

Triggers:

The member invokes an edit request (for the full article or just one section) on the article.

Basic flow:

1) The system pravides a new editor area/box filled with all the article's relevant content with an informative edit summary for the member to edit. If the member just wants to edit a section of the report, only the original content of the section is shown, with the section title automatically filled out in the edit summary.

2) The member modifies the article's content until the member is satisfied.

3) The member fills out the edit summary, tells the system if they want to watch this article, and submits the edit.

4) The system saves the article, logs the edit event, and finishes any necessary post-processing.

5) The system presents the updated view of the article to the member.

Use Case – Example

- **Extensions**:
- 2–3.
  - a) Show preview:
    1) The member selects Show preview which submits the modified content.
    2) The system reruns step 1 with the addition of the rendered updated content for preview, and informs the member that his/her edits have not been saved yet, then continues.
  - b) Show changes:
    1) The member selects Show changes which submits the modified content.
    2) The system reruns step 1 with the addition of showing the results of comparing the differences between the current edits by the member and the most recent saved version of the article, then continues.
  - c) Cancel the edit:
    1) The member selects Cancel.
    2) The system discards any change the member has made, then goes to step 5.