ES Software Engineering

Lecture 1
Developing software for large projects of the ES industry

Subject matter

The subject covers the following topics:

- nature of contemporary embedded systems and the direction of the field
- problem of creating, expanding and managing complex projects of class "system of systems"
- methods and tools for modeling systems
- communication methods between designer and engineers teams
- types of software licenses
- porting a software layer to a new hardware architecture
- project management tools
- hardware layer emulators
- domestic market of embedded systems

Plan of the lectures

Lecture 1 – Developing large software projects
Lecture 2 – Requirements specification, testing and defects
Lecture 3 – Unified Modeling Language
Lecture 4 – State Machine Diagrams
Lecture 5 – Use case modeling and Application logic design
Lecture 6 – Implementation and testing
Lecture 7 – System and Data architecture design / Software licence
Lecture 8 – ES Developer tools + Credit

Lecture 1

1. History of the development of embedded systems
2. Embedded systems today
3. Multiprocessor systems
4. Systems of systems
5. Software life cycles

Embedded systems

What characterizes an embedded system?

1. **Location**: software within a dedicated hardware
2. **Function**: implementation of a specific task – not a general purpose system
3. **Functionality**: interaction with a dedicated environment

Embedded systems

The history of embedded systems:

1944-1952 – Whirlwind I

1971 – Intel 4004 (4-bit) (2300 transistors)

1972 – Intel 8008 (8-bit) (2500 transistors)

The 70s – automotive industry (controlling fuel mixes)

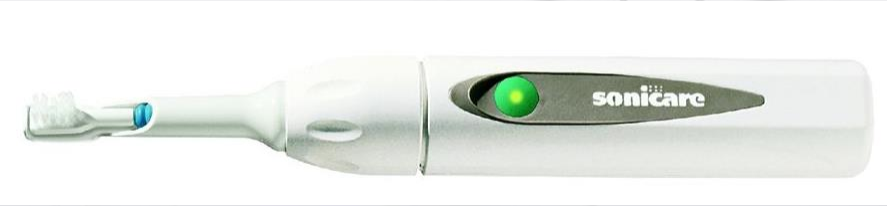
1984 – Intel 8080 (8-bit) (2MHz, ±5V, +12V, 72 instructions)

2014 – Intel Xeon (4.31 billion transistors)

Embedded systems today

Nowadays

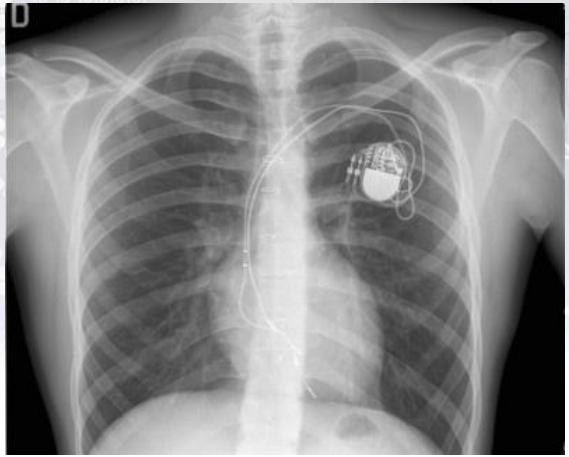
Linux in Lernstift pen



Sonicare Plus toothbrush with
an 8-bit Zilog Z8 microprocessor

Embedded systems

Nowadays



Embedded systems

NASA's Twin Mars Rovers

Microprocessor 20MHz

Commercial RT OS

OS and software developed during the flight to Mars and downloaded via space.

Embedded systems

I’m driving or I’m carried:

7-series BMW, S-class Mercedess – about 100 processors

Low-profile Volvo – about 50-60 processors

„econobox” – a few dozen different microprocessors

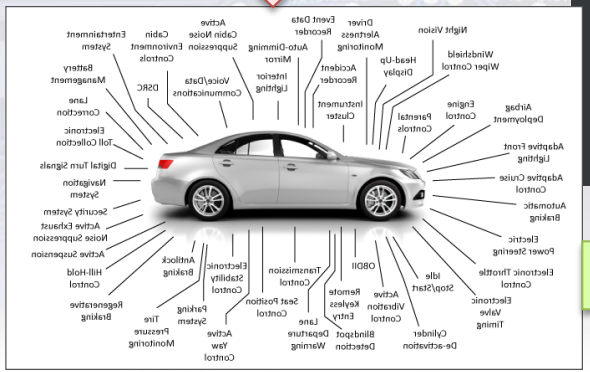
Embedded systems

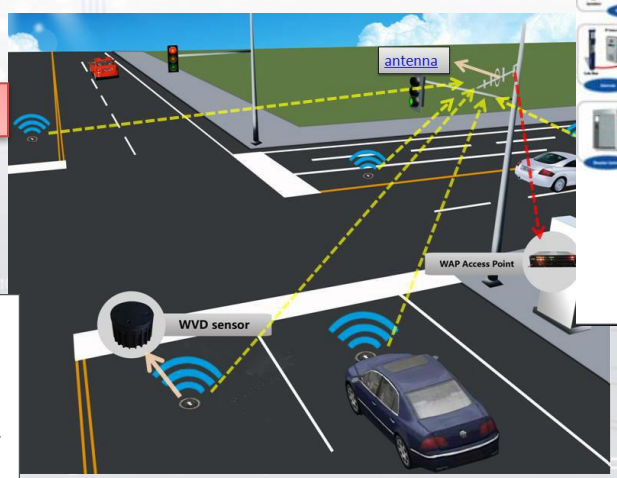
In multiprocessor systems each module can have a different hardware layer, and thus also a different port for the software layer. Are we able to manage such a project without a proper plan/model/design/tools? How to go about such a project anyway?

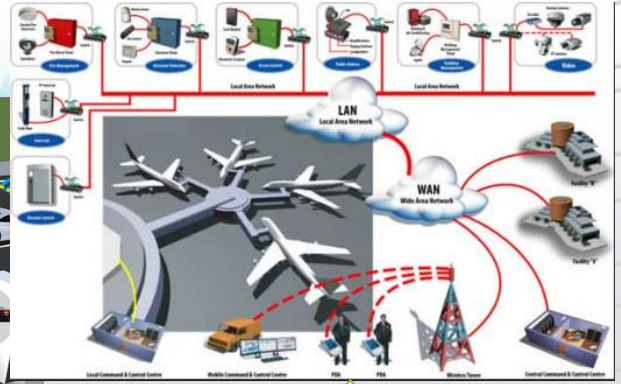
And how about a SYSTEM OF SYSTEMS?

Embedded systems

System of systems

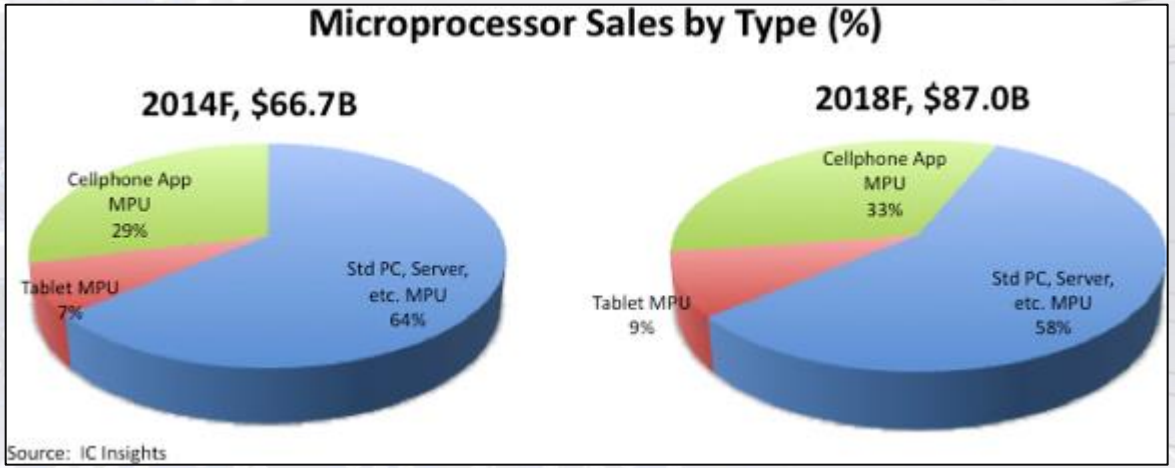






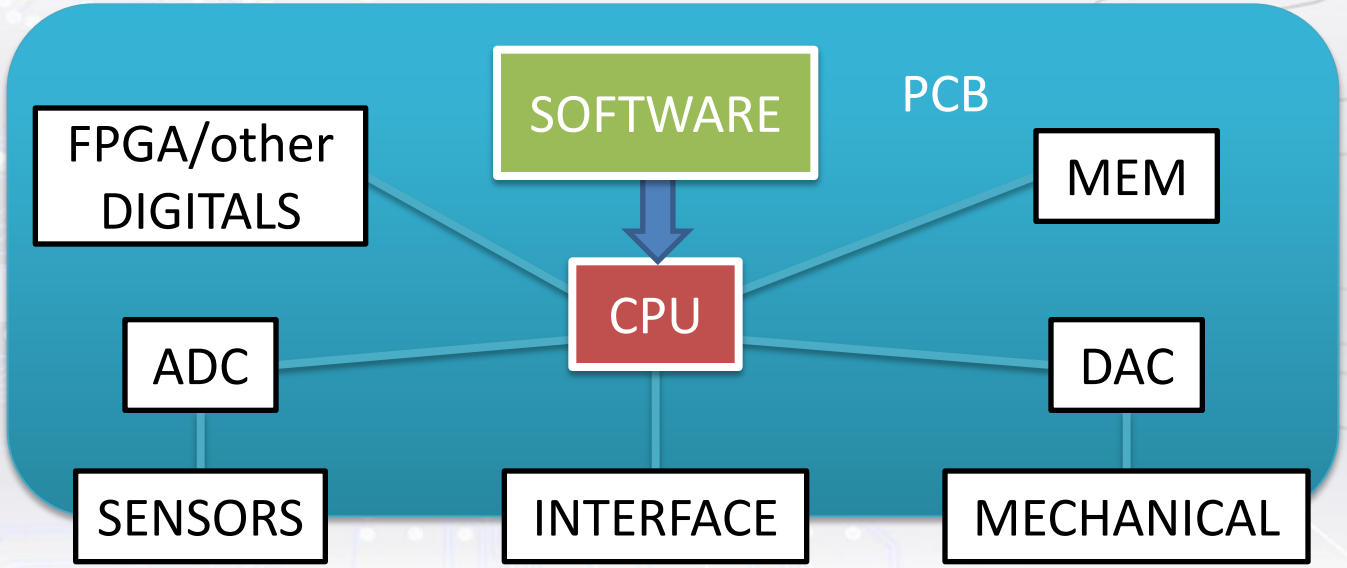
Embedded systems processors

Microprocessors sales by type [3]



Developing software

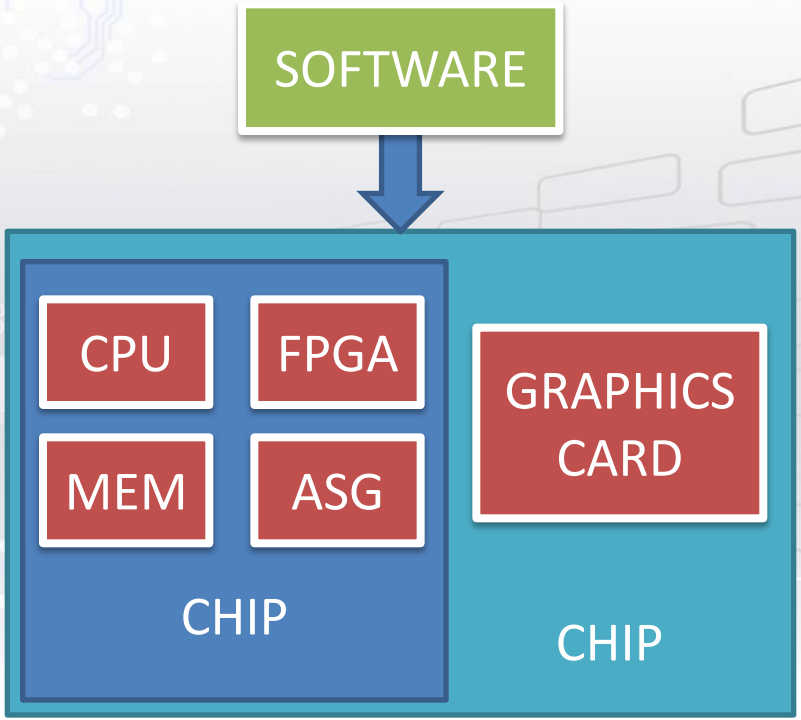
Traditional approach:



Developing software

Current approach:

INTEGRATED



Developing Embedded Systems

Conclusion:

1.Multiprocessor systems dominating

2.In the nearest future systems of systems will be developed

3.Entering a market requires knowing various technologies, many hardware architectures, multiple programming languages and multiple software development tools

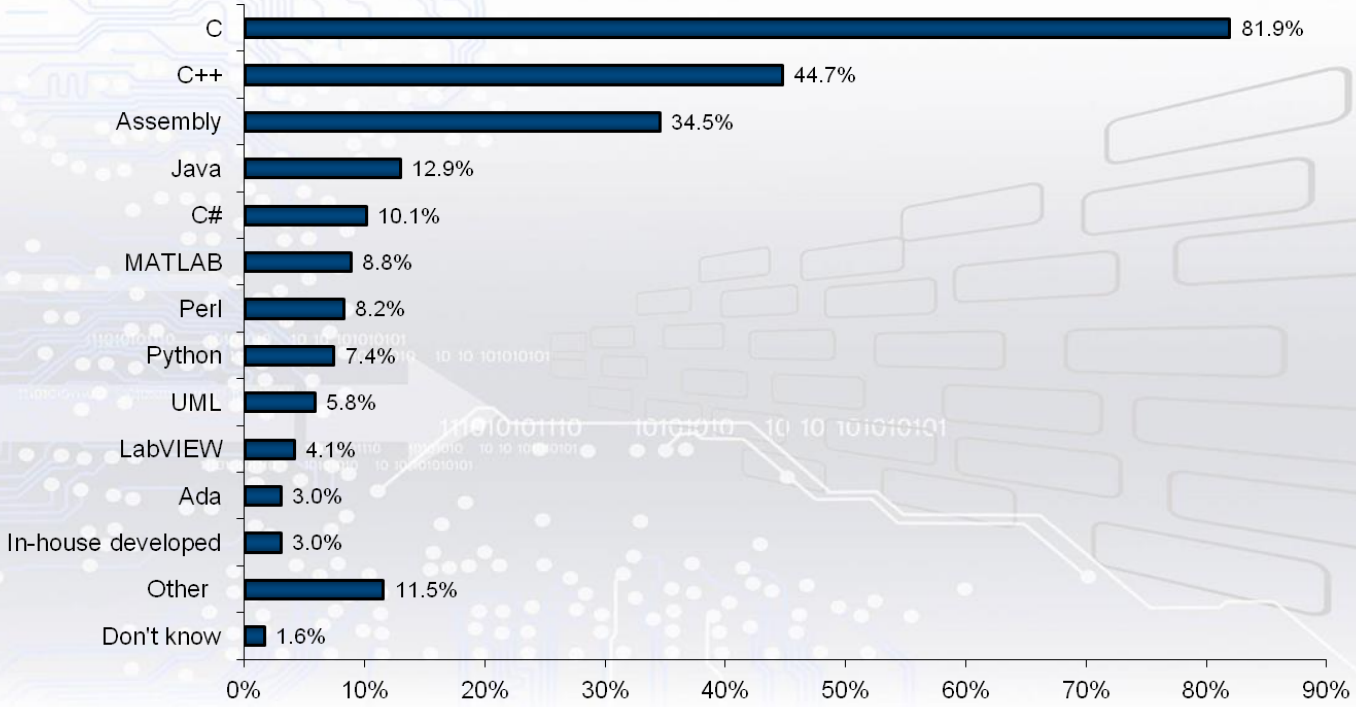
4.Commercial projects are created by many teams (sometimes corporations) and are characterized by a high degree of complexity

5.Communication between teams requires using a top-down model and tools to develop a project

6.To control such a project, a way of managing it should be defined in advance and should stick to assumptions

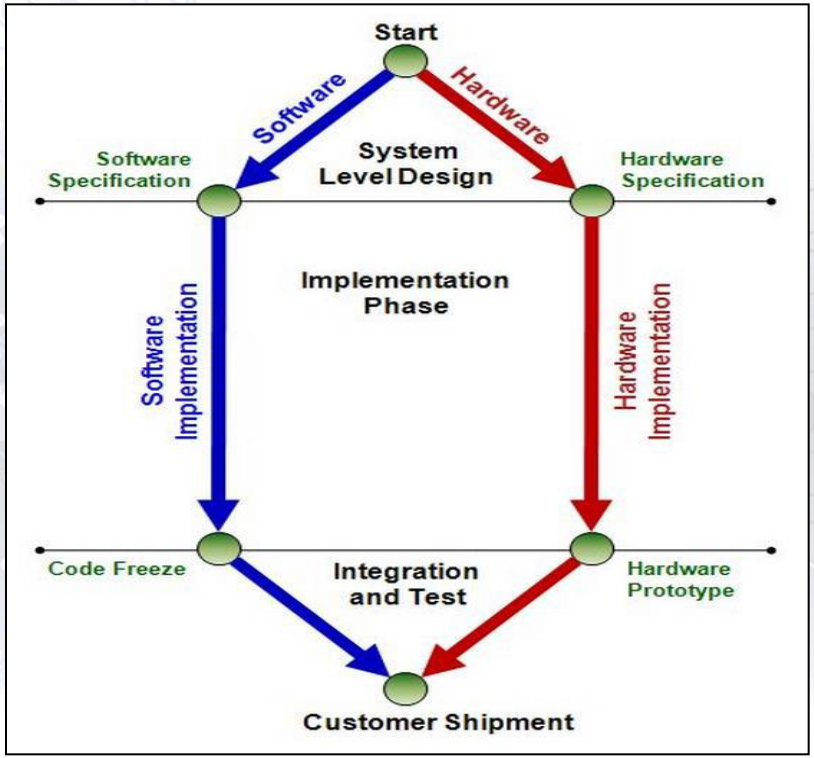
7.The adopted method of administration defines the so-called software lifecycle

Programming languages



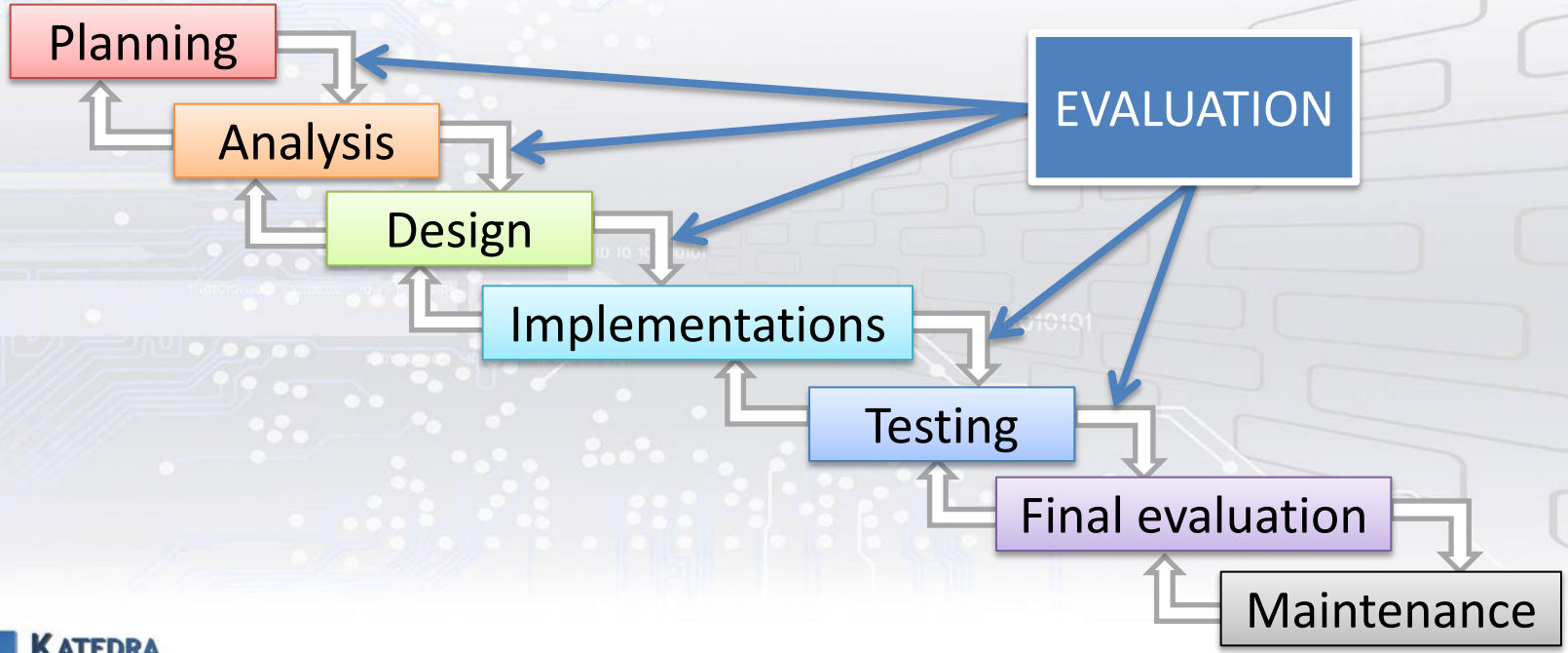
Note: Percentages sum to over 100% due to multiple responses.

Developing software



Software lifecycle

„Waterfall" model (Herbert D. Benington 1956)



Waterfall model

Planning and Analysis (steps 1 and 2)

- *The result of steps is the specification of system functions, its interface, the way it relates to the environment, as well as system parameters, e.g. accuracy.*
- *The documentation also defines how to implement the system – modules and software planned to be used.*
- *Guidelines should be kept during later stages.*

Waterfall model

Design (step 3)

1. *An initial distribution of software components assigning them their functions*
2. *Planning tests*
3. *Preparing a project specification divided into modules and algorithms*
4. *Defining data structures*
5. *Preparing an initial documentation of module tests*

Waterfall model

Implementation (step 4)

Programming and testing modules

The result is:

1. *source code with a listing of programs*
2. *instructions for performing tests*

Testing (step 5)

1. *report concerning testing components (not modules)*
2. *updated software source code and a new project specification*
3. *configuration unit testing instruction*

Waterfall model

Final evaluation (step 6)

The result of this stage is a ready-to-sell product (the designed system)

Maintenance (step 7)

The purpose of maintenance is to adapt the system to the needs of an individual customer.

Maintenance may constitute more than 50% of a company income of software development companies.

Waterfall model

Waterfall model

Pros

- Well organized process
- Complete model
- Milestones
- Well documented

Cons

- Long process
- Costly
- Frequent turn-backs
- Requirements hard to be completely specified
- Uncertainty of success

Waterfall model

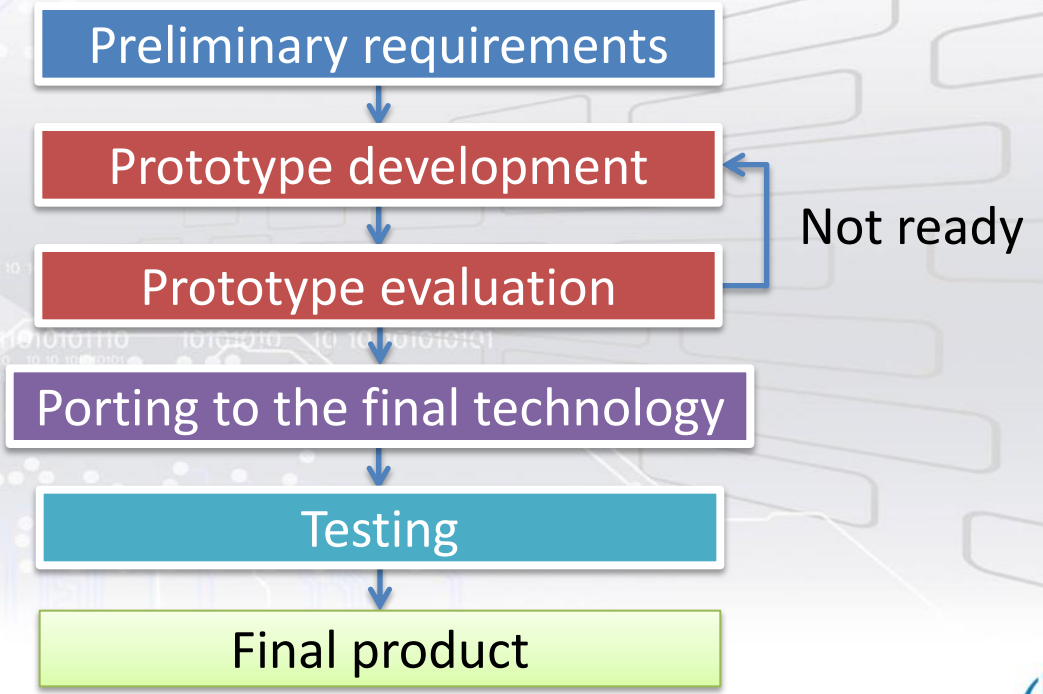
Most common errors:

- wrongly defined input data
- underestimating the role of the first phases of the model
- none or unreliable assessment of the successive stages of the project
- not using project management tools or CAD tools for developing software
- no documentation of subsequent stages of project development
- changes in the assumptions of the project during implementing

Software prototyping

Model basing on prototype (s.c. software prototyping):

model – incomplete system designed for testing a planned functionality



Software prototyping

Software prototyping

Pros

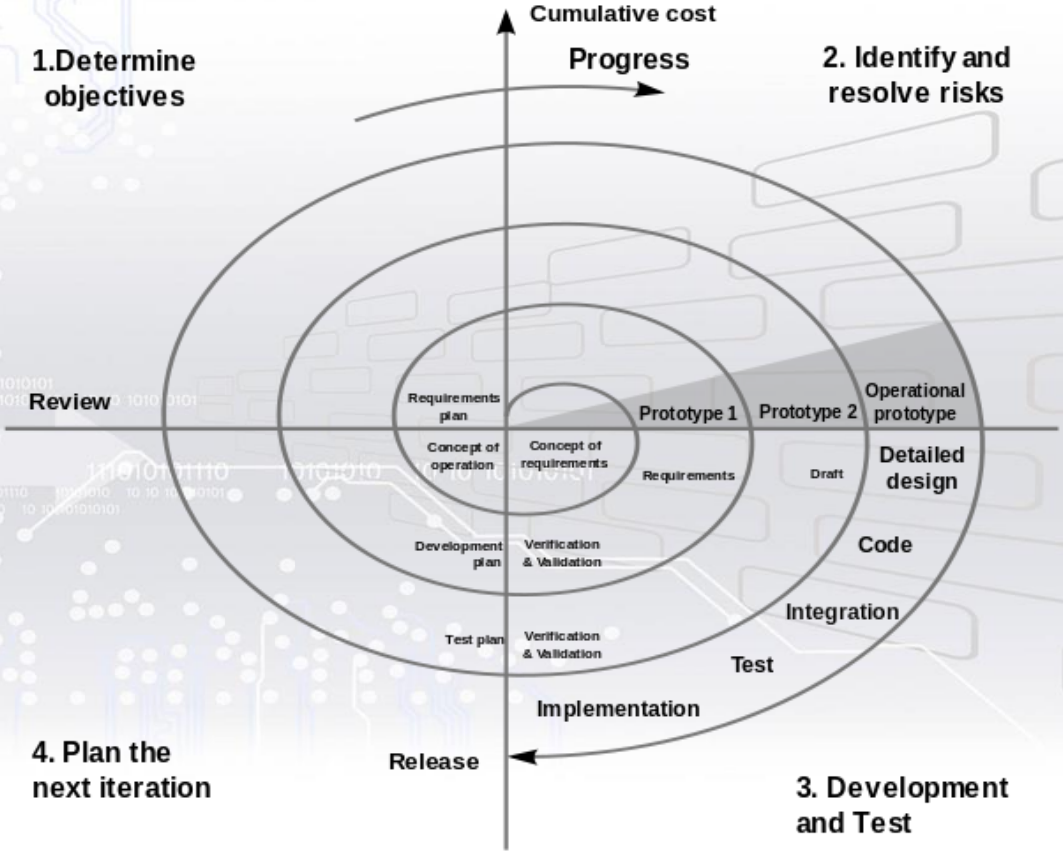
- Better requirements recognition
- Easy to change prototype
- Lower risk
- Visualisation at the beginning

Cons

- High costs
- Misunderstanding – prototype treaded as the final version
- Final version may not be optimal

Spiral iterative model

Spiral model
(B. Boehm 1986):



Spiral iterative model

Spiral iterative model

Pros

- Complete requirements achieved in many cycles
- Lower risk according to evaluation
- There is always a possibility for developing the project
- The assessment phases make it possible to detect errors earlier

Cons

- Customer feels a lack of benefits from continuous improvement
- Large cost of error elimination in the finally used software
- New versions with new errors

Rational Unified Process

RUP (Ken Hartman i B. Boehm – the 80s and the 90s.; official version by Philippe Kruchten in 1998.)

- Iterative software development, developed by the Rational Software Corporation (now acquired by IBM).
- Rational Unified Process is also a name of software available from IBM. The RUP process is also defined in the Rational Method Composer tool and its lighter version in the Eclipse Process Framework (both sponsored by IBM)

Rational Unified Process

RUP – genesis

Ken Hartman and Barry Boehm in their research – analyzed mistakes made during developing software The most common are:

- no requirements management
- ambiguous communication
- Brittle software architecture
- redundant software complexity
- lack of proper care during testing or too small number of tests
- subjective tests assessment
- no risk management
- no automation during project development

Rational Unified Process

What is RUP:

1. Iterative Development
2. Requirement Management
3. Component-based architecture
4. Graphical modeling and visual design
5. Quality Assurance
6. Change Management

Rational Unified Process

RUP –> Iterative Development

- one master phase plan and many iteration plans
- each iteration improves software architecture
- including stakeholders at each milestone
- cheaper and simpler software integration step by step
- easy management of changes in requirements
- each iteration allows to detect threats

Rational Unified Process

RUP –> Requirement Management

- identification, detecting changes and meeting the needs of users
- project covers an important function for the user
- RUP involves understanding the needs of shareholders
- Scope Management
- SRS in a more detailed view (Software Requirements Specification)
- Requirements change management

Rational Unified Process

RUP –> Component-based architecture

- Component = a collection of related objects, in the context of object-oriented programming
- Extensible, understandable and a re-usable system
- Simple architecture in early iterations – prototype
- Implementing components in technologies CORBA, COM, JEE
- Components often become separate products

Rational Unified Process

RUP –> Graphical Software Modeling

- Unified Modeling Language (UML)
  - modeling with diagrams
  - well known and understandable
  - better understanding of the structure of a solution
- Visual software design
  - easy to use visual tools (not UML diagrams)

Rational Unified Process

RUP –> Quality Assurance

- Quality control is often the weakest point of the process – performed after the construction of the system and operated by another team
- RUP assumes a location of the control during the whole process
- RUP assumes each team member working with the control
- Early error detection and elimination (costs and benefits)

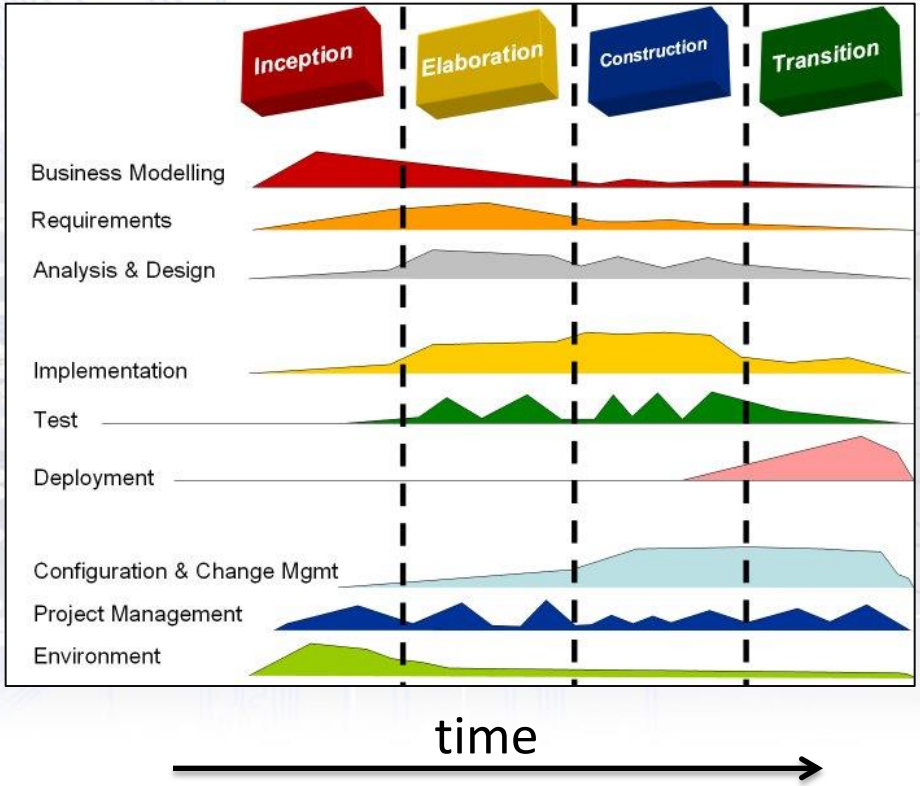
Rational Unified Process

RUP –> Change Management

- Methods of tracking, recording and controlling changes
- *Secure workspaces* – definition of areas within which no changes should threaten the system one is creating
- A concept of *secure workspaces* is closely associated with the component-oriented architecture

Rational Unified Process

RUP phases



RUP phases

1. Inception

- Business goals analysis
- Use-cases model
- Project plan
- Preliminary risk analysis
- Preliminary requirements specification

2. Elaboration

- System analysis and design
- Completeness of the Use Case model at 80%
- Designed system architecture
- Checking objectives reachability and possible risks
- Detailed schedule for the entire project

3. Construction

- Implementation and testing
- Stable product – ready for deployment
- Customers ready for product acceptance
- Actual costs and time are within acceptable limits

3. Transition

- From development to deployment
- User training and beta testing
- Actual costs and time are acceptable
- Main milestone: satisfaction of the users/customers

Alternatives

Alternative approaches

- Open UP (Open Unified Process)
- Eclipse Process Framework (EPF)
- Enterprise Unified Process
- Agile programming

Bibliography

[1] Ignacy Pardyka, „Systemy wbudowane. 01.ES: Wprowadzenie”, Ignacy Pardyka, UJK Kielce
[2] http://www.mhealthtalk.com/smart-toilets-a-royal-flush-for-home-healthcare/
[3] http://electroiq.com/blog/2014/01/microprocessor-sales-growth-will-strengthen-slightly-in-2014/
[4] http://wazniak.mimuw.edu.pl/
[5] Jarosław Kuchta, „Embedded Systems Software Engineering – Software lifecycle”, 2015
[6] https://en.wikipedia.org/wiki/Spiral_model
[7] https://pl.wikipedia.org/wiki/Rational_Unified_Process
[8] Roger S. Pressman, „Software Engineering. A Practitioner's Approach”
[9] IBM, „Rational Unified Process. Best Practices for Software Development Teams”
[10] OMG.ORG, „Unified Modeling Language”
[11] Eclipse.org, „Introduction to OpenUP”
[12] Scott W. Ambler, „Introduction to the Enterprise Unified Process”
[13] Agile Manifesto (HTML)
[14] Kent Beck, „Extreme Programming Explained. Embrace Change” (book with Cynthia Andres)