Programming Technologies

Kollár, Lajos

Kocsis, Gergely (English version)
Software products

• General purpose products
  • Standalone systems designed to be able to be solved to as much users as possible
  • E.g. graphical and office applications, billing applications or ticketing systems

• Software products for given purposes
  • Systems to cover the needs of a given customer
  • E.g. Embedded control systems, air-traffic control system, road traffic monitoring system
Software product specification

• General purpose products
  • The specification is the property of the developer. He decides about later changes in the software.

• Software products for given purposes
  • The specification is the property of the customer. He can decide if a modification is required or not.
### FAQ (1)

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is software?</td>
<td>Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.</td>
</tr>
<tr>
<td>What are the attributes of good software?</td>
<td>Good software should deliver the required functionality and performance to the user and should be maintainable, dependable, and usable.</td>
</tr>
<tr>
<td>What is software engineering?</td>
<td>Software engineering is an engineering discipline that is concerned with all aspects of software production.</td>
</tr>
<tr>
<td>What are the fundamental software engineering activities?</td>
<td>Software specification, software development, software validation, and software evolution.</td>
</tr>
<tr>
<td>What is the difference between software engineering and computer science?</td>
<td>Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.</td>
</tr>
<tr>
<td>What is the difference between software engineering and system engineering?</td>
<td>System engineering is concerned with all aspects of computer-based systems development including hardware, software, and process engineering. Software engineering is part of this more general process.</td>
</tr>
<tr>
<td>Question</td>
<td>Answer</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>What are the key challenges facing software engineering?</td>
<td>Coping with increasing diversity, demands for reduced delivery times, and developing trustworthy software.</td>
</tr>
<tr>
<td>What are the costs of software engineering?</td>
<td>Roughly 60% of software costs are development costs; 40% are testing costs. For custom software, evolution costs often exceed development costs.</td>
</tr>
<tr>
<td>What are the best software engineering techniques and methods?</td>
<td>While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can’t, therefore, say that one method is better than another.</td>
</tr>
<tr>
<td>What differences has the Web made to software engineering?</td>
<td>The Web has led to the availability of software services and the possibility of developing highly distributed service-based systems. Web-based systems development has led to important advances in programming languages and software reuse.</td>
</tr>
</tbody>
</table>
Essential attributes of good software

• Maintainability
  Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment

• Dependability and security
  Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system
Essential attributes of good software

• Efficiency
  Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilization, etc.

• Acceptability
  Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.
General issues affecting software

• Heterogeneity
  Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.

• Business and social change
  Business and society are changing incredibly quickly as emerging economies develop and new technologies become available.

• Security and trust
  As software is intertwined with all aspects of our lives, it is essential that we can trust that software.
Types of applications

• **Standalone applications**
  These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network. Examples of such applications are office applications on a PC, CAD programs, photo manipulation software, etc.

• **Interactive transaction-based applications**
  These are applications that execute on a remote computer and that are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications where you can interact with a remote system to buy goods and services.

• **Embedded control systems**
  These are business systems that are designed to process data in large batches. Examples of embedded systems include the software in a mobile (cell) phone, software that controls anti-lock braking in a car, and software in a microwave oven to control the cooking process.

• **Batch processing systems**
  These are business systems that are designed to process data in large batches. Examples include periodic billing systems, such as phone billing systems, and salary payment systems.
Types of applications

• Entertainment systems
  These are systems that are primarily for personal use and which are intended to entertain the user

• Systems for modeling and simulation
  These are systems that are developed by scientists and engineers to model physical processes or situations,

• Data collection systems
  These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.

• Systems of systems
  These are systems that are composed of a number of other software systems.
Abstraction, models

- Arts
- Religion
- Philosophy
- Science (mathematics, physics, engineering sciences)
- Computer science
  - Data abstraction: data model
  - Procedural abstraction: functional model
How to define models

• Natural languages
• Formal languages
• Programming languages
• Visual languages
• Mathematics (algebra, logic)
Languages

• Syntax:
  • abstract
    • E.g.  
      / \ 
      2   3
  • exact
    • infix: 2 + 3
    • prefix: + 2 3
    • postfix: 2 3 +
    • JVM:  bipush 2
             bipush 3
             iadd
    • english: the sum of 2 and 3
    • magyar: 2 és 3 összege
Languages

• „Colorless green ideas sleep furiously” (Noam Chomsky)

• Semantics
  • static: rules than can be checked during compilation
    • E.g. all the identifiers are declared, different branches of a conditional statement consist different statements
  • dynamic: how some given constructions make their effect
First modeling tools ('50)

- Automata
- Machine language
- Assembly language
- FORTRAN
- COBOL
- Flowchart
Meta languages ('60)

How can we define formally a description language?
• (E)BNF ((Extended) Backus-Naur Form)

```
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<sign> ::= + | -
<whole_number> ::= [ <sign> ] <digit> { <digit> }
```
Meta languages ('60)

How can we define formally a description language?

- (E)BNF ((Extended) Backus-Naur Form)
- COBOL-like

\[
\begin{align*}
\text{DIGIT} & : \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\
\text{SIGN} & : \{+\} \\
\text{WHOLE NUMBER} & : [\text{SIGN}] \text{ DIGIT}...
\end{align*}
\]
Meta languages ('60)

How can we define formally a description language?

- (E)BNF ((Extended) Backus-Naur Form)
- COBOL-like
- Syntax graph
Meta languages ('60)

How can we define formally a description language?

• (E)BNF ((Extended) Backus-Naur Form)
• COBOL-like
• Syntax graph
• Hybrid

digit : { 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 }
whole number : [ { + | - } ] digit [digit]...
Meta languages ('60)

How can we define formally a description language?

• (E)BNF ((Extended) Backus-Naur Form)
• COBOL-like
• Syntax graph
• Hybrid
• ...

• Formal languages (syntax and semantics)
Standards, standardization organizations

• de jure/de iure
  • ISO: International Standardization Organization
  • ANSI: American National Standardization Institute
  • MSZT: Magyar Szabványügyi Testület
  • ...

• de facto
  • OMG: Object Management Group
  • W3C: World Wide Web Consortium
  • ECMA: European Computer Manufacturers Association
  • ODMG: Object Data Management Group
  • ...
Paradigms

• Procedural
• Functional
• Object oriented
• Logical
• Data driven

• Generic
Attributes

• Abstraction tools
• Expressiveness
• Handling data and functional model
• Support for reuse
• Dynamism
• Handling parallelism
Data models

• Relational (’70)
• ER/EER (’76)
  Just the structure, the behavior is modeled procedurally

• OO (’80)
• OR (’90)
  Behavior as well
Meta-models ('90)

• Model of the model
• The model is an abstraction of the real world. The meta-model is an abstraction of the model
  • It highlights the attributes of the model
• OMG MOF (Meta Object Facility)
• OMG CWM (Common Warehouse Metamodel)
Meta modeling

• Even higher level of abstraction
• We describe the attributes of the model
• Terminology: the model *conforms to* its meta-model
Meta Object Facility (MOF)

• MOF uses a 4 layer meta-modeling architecture:
  • closed: the M3 layer represents itself
  • strict: all model elements of all layers are in relation with exactly one model element of the layer above

• With the use of MOF we can describe the structure (the abstract syntax) of a language or a datum

• MOF is a domain specific language (DSL) used to define meta-models
Meta-modeling

M3 (MOF)

M2 (UML)

M1 (User model)

M0 (Run-time instances)
Meta Object Facility (MOF)

• The aim is to provide a type system for CORBA entities and also to provide a collection of interfaces with which these types can be created, stored and manipulated

• Models...
  • can be exported from applications
  • can be imported to applications
  • can be transported through the network
  • can be stored in the storage and then can be reached there
  • can be converted to different formats (e.g. XMI – XML Metadata Interchange)
Methodologies

• „A methodology, in software engineering terms, is a collection of methods based on a common philosophy that fit together in a framework called the systems development life cycle“ (Ken Orr, 1989)
• Who, when, what, how
• Historically:
  • structured
  • Object oriented
  • agile
Software process

• A software process is a set of related activities that leads to the production of a software product.

• There are many different software processes but all must include four activities that are fundamental to software engineering:
  • **Specification**: The functionality of the software and constraints on its operation must be defined.
  • **Design and implementation** The software to meet the specification must be produced.
  • **Validation** The software must be validated to ensure that it does what the customer wants.
  • **Evolution** The software must evolve to meet changing customer needs.
Description of software engineering

• Providing data-model, designing the user interface etc.
• It is needed also to specify the order of these
• The description can also include:
  • Products, which are the outcomes of a process activity.
  • Roles, which reflect the responsibilities of the people involved in the process. (e.g. project manager, programmer etc.)
  • Pre- and post-conditions, which are statements that are true before and after a process activity has been enacted or a product produced.
Plan driven and agile processes

• Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.

• In agile processes planning is incremental and it is easier to change the process to reflect changing customer requirements.

• In practice most processes also use plan based and agile elements

• There is no good or bad software process
„No silver bullet!”

Software process models

• The waterfall model
  A plan driven process that takes the fundamental process as separate process phases

• Incremental development
  This approach interleaves the activities of specification, development, and validation. The system is developed as a series of versions (increments), with each version adding functionality to the previous version.

• Reuse-oriented software engineering
  During the development we focus on how we can integrate already existing components rather than developing new ones. It can be plan driven as well as agile

• These process models are just abstractions. In practice most systems are developed using processes that contain elements from all of the above process models
Waterfall model

- Requirements Definition
- System and Software Design
- Implementation and Unit Testing
- Integration and System Testing
- Operation and Maintenance
Incremental development

Concurrent Activities

Outline Description
Specification
Development
Validation
Initial Version
Intermediate Versions
Final Version
Reuse-oriented software engineering

Requirements Specification → Component Analysis → Requirements Modification → System Design with Reuse

Development and Integration → System Validation
Life-cycle of software development

- Vision
- Requirements specification
- Analysis
- Architectural planning
- Planning
- Implementation
- Testing
- Deployment
- Operation
- Maintenance
- Evolution
- Putting out of service
UML – Unified Modeling Language

• Grady Booch, Jim Rumbaugh, Ivar Jacobson
• '95 UML 0.9
• '97 UML 1.1 OMG standard
• '01 UML 1.4, then 1.5 (executable UML)
• '05 UML 2.0
• '07 UML 2.1.2
• '09 UML 2.2
• '10 UML 2.3
• '11 UML 2.4, UML 2.4.1
• '15 UML 2.5
UML planning goals

• modularity
• layered structure
• partitioning
• expressiveness
• reusability
UML Infrastructure
The role of the Core package

- UML
- MOF
- CWM
- Profiles
Subpackages of the Core package
UML Superstructure

• Layer 0.: class based modeling structures
• Layer 1.: use cases, interactions, actions and activities
• Layer 2.: deployment, state machine modeling, profiles
• Layer 3.: information flow, templates, packages
UML diagram types

- Structure Diagram
  - Class Diagram
  - Object Diagram
  - Package Diagram
  - Composite Structure Diagram
  - Component Diagram
  - Deployment Diagram
  - Profile Diagram

- Behavior Diagram
  - UseCase Diagram
  - Activity Diagram
  - State Machine Diagram
  - Interaction Diagram
    - Sequence Diagram
    - Communication Diagram
    - Timing Diagram
    - Interaction Overview Diagram
Namespaces diagram, Kernel package
Multiplicities diagram, Kernel package
Expressions diagram, Kernel package
Classifiers diagram,
Kernel package
Classes diagram
Kernel package
Packages diagram
Kernel package

A csomagok absztrakt szintakszisa
Package

Exact syntax
Diagram elements

- stereotype
  
    <<name>>

- Dependency
  
    ------>

- constraint
  
    {constraint}
Comments

Exact syntax

Abstract syntax

Ownerships

Comments
Class diagram

A fully connected graph in which
• Nodes are classes
• Edges are relations between classes

These relations may be:
• Associations (one or two directions)
• aggregations
• compositions
• Inheritance relations
• Dependency relations
• realizations
<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
</tr>
<tr>
<td>operations (just specifications)</td>
</tr>
</tbody>
</table>
Class

attribute:

\[ \text{name[}: \text{type}][ = \text{starting value}] \]

operation:

\[ \text{name}(\text{fp1})[ : \text{type}] \]

dp (formal parameter):

\[ [ \text{mode}] \text{name} : \text{type} [ = \text{starting value}] \]

mode:

\{ \text{IN} | \text{OUT} | \text{INOUT} | \text{RETURN} \} \]

visibility:

\{ + | # | - | ~} (\text{public, protected, private, package private}) \]
Class diagram

association, relation

Class diagram

- **Role**
  - Mandatory for recursive associations
- **Multiplicity**
  - 0..1, *, 0..*, 1..*, 2..4, 3..17, 5..*
- **Navigability**
  - One or two directions
Class diagram

aggregation, composition
Inheritance / interface realization

Class diagram

Shape
- Polygon
- Ellipse

Shape
- getArea(): double

RectangularShape
- width: int
- height: int
- area: double
- contains(p: Point): boolean
- getArea(): double

Rectangle
- x: int
- y: int
- contains(p: Point): boolean
- distance(r: Rectangle): double
a) GeneralizationSet sharing same general Classifier using the same generalization relationship names.

b) GeneralizationSet designation by subtypes sharing a common generalization arrowhead.

c) GeneralizationSet sharing same general Classifier using the dashed-line notation.
Object diagram

A graph in which
• nodes are objects
• edges are relations between objects
Object diagram

Entity

<table>
<thead>
<tr>
<th>[object name] : Class name</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute = value</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>deik : Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>postal_code = 4028</td>
</tr>
<tr>
<td>city= &quot;Debrecen&quot;</td>
</tr>
<tr>
<td>address = &quot;Kassai út 26.&quot;</td>
</tr>
</tbody>
</table>
Object diagram example

```
    Party
      * children
        location

    Person
    Organization

    engineering : Organization
      location = "Boston"

    tools : Organization
      location = "Chicago"

    apps : Organization
      location = "Saba"

    Don : Person
      location = "Champaign"

    John : Person
      location = "Champaign"
```