Introduction to Component-Based Software Engineering

Ivica Crnkovic
Mälardalen University,
Department of Computer Engineering
Sweden
ivica.crnkovic@mdh.se
http://www.idt.mdh.se/~icc
Topic overview

1 The challenges of SW- how can CBD help?
2 What is a software component?
3 Basic principles of component-based approach
4 Component-based Software Development Process
5 Problems and research issues
6 References
The challenges of software development
- how can component software help?
Challanges of Software Engineering

Problems of Software Engineering

• The size & complexity of software increases rapidly
• Single products become part of product families
• Software is upgraded after deployment
• The time-to-market must decrease significantly
• The cost of products must be reduced

(The author of slides with blue background: Michel Chaudron, Tue)
Observations on the practice of SE

About 80% of software engineering deals with changing existing software

It is not the strongest of the species that survive, nor the most intelligent, but the ones most responsive to change.  

-- Charles Darwin

Time to market is an important competitive advantage: incorporate successful innovations quickly

→ Systems should be built to facilitate change
→ easy removal and addition of functionality
Answer: Component-based Development

- **Idea:**
  - Build software systems from pre-existing components (like building cars from existing components)
  - Building components that can be reused in different applications
  - Separate development of components from development of systems
Component-Based Software Engineering (CBSE)

- Provides methods and tools for
  - Building systems from components
  - Building components as reusable units
  - Performing maintenance by replacement of components and introducing new components into the system
Component-based software construction (1)
Concentration on the business parts

“30 % of SW development effort is spent on infrastructure that adds no value”

Application specific

Business issues

- GUI
- Communication
- Data model
- Deployment

Standard Reusable parts

INFRASTRUCTURE
Is CBD the same as OOP?

- Object-oriented programming

Are objects the same as components?
Side remark: OO and reuse

Object orientation is not primarily concerned with reuse, but with appropriate domain/problem representation using the technological enablers

- Objects, classes, inheritance, polymorphism

Experience has shown that the use of OO does not necessarily produce reusable SW

CBD

- scale reusable entities: Component = many objects in collaboration
- reusable parts on the execution level (plug-in)
- Additional services provided by component models
What is a software component?
The software architecture of a program or computing system is the structure or structures of the system, which comprise software components [and connectors], the externally visible properties of those components [and connectors] and the relationships among them.”

Another example
Object Management Architecture Overview

CORBA (Common Object Request Broker Architecture)

CORBAapps
CORBAdomains
CORBAfacilities

Transactions
Event
Security
Naming

CORBA services
Corba component model

Component
Stock_Exchange reference
(Supported interface)

Facet implementation

Component
Stock_Exhange

Provided interface (facet)

Stock_Coject

Event sink

Buy_Offers

Sell_Offers

Attributes

Event source

Price_change

Event sink implementations
Some successful components: In the past...

- **Mathematical libraries**
  - NAGLIB - Fortran Library
  - Mathematical and physical functions

- **Characteristics**
  + Well defined theory behind the functions - very well standardized
  + Simple Interface - *procedural type* of communication between client (application) and server (component)
  + Well defined input and output
  + Relative good error handling
  - Difficult for adaptation (not flexible)
Some successful components: The big ones...

Client - server type

- **Database**
  - Relational databases, (Object-oriented databases, hierarchical databases)
  - Standard API - SQL
  - Different dialects of the standard

- **X-windows**
  - Standard API, callback type of communication
  - High level of adaptation
  - Too general - difficult to use it
Even bigger components: Operating systems

- Example - Unix
  - A general purpose OS, used as a platform for dedicated purposes
  - Standard API - POSIX
  - Commands used as components in a shell-process
    Example: sort out words from text files:
    ```
    $ cat file1 file2 file3 ... | sed 's/ /\n/g' | sort -u >words.txt
    ```
  - Different variants, POSIX is not sufficient
  - Not a real component behavior (difficult to replace or update)

- MS Windows...
Frameworks - building “the real components”

- Component Object Management - COM, Active X
- Enterprise JavaBeans
- CORBA components
- .NET

Late binding - easy replacement
Example: The architecture of a car control system

ECU – Electronic Control Unit
The architectural design challenge

Vehicle stability  Suspension  Drive by wire  ……

Complex functions

Local Control Functions

Sensor  Actuator

Local Control Functions

Sensor  Actuator

Basic functions

How to implement complex functions based on local control functions?
Problem: resource sharing

Can functions of different criticality be allowed to share resources?
Challenge – open and dependable platform

Vehicle stability

- Collision detection
- Antispin
- Cruise control

Local functions
- Engine Control
- Local brake Control
- Transmission
- ……

Global (complex) functions

Sensors

Actuators

Applications

Middleware

Input/output drivers

Hardware

SOFTWARE COMPONENTS

ECU

Vehicle
Challenge – open and dependable platform

**Requirements**
- Separation of hw from SW development
- Separation of SW component development
Szyperski: Software Component Definition

Szyperski (Component Software beyond OO programming)

- **A software component is**
  - a unit of composition
  - with contractually specified interfaces
  - and explicit context dependencies only.

- **A software component**
  - can be deployed independently
  - it is subject to composition by third party.
Composition unit

A software component is a **unit of composition** with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party. –Clemens Szyperski

*How much components fit together?*

*How much costs the glue code?*
What is a contract?

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party.

- Interface – component specification

- Contract - A specification attached to an interface that mutually binds the clients and providers of the components.
  - Functional Aspects (API)
  - Pre- and post-conditions for the operations specified by API.
  - Non functional aspects (different constrains, environment requirements, etc.)
What is an explicit context dependency?

A software component is a unit of composition with contractually specified interfaces and **explicit context dependencies only**. A software component can be deployed independently and is subject to composition by third party.

- Provided and Required Interface

- Context dependencies - Specification of the deployment environment and run-time environment

  - Example: Which tools, platforms, resources or other components are required?
What does it mean deployed independently?

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party.

- Late binding - dependencies are resolved at load or run-time.

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

Page 30, 23 January 2012
Example: Interface description: (M)IDL

(Microsoft) Interface Definition Language

```
[ uuid(00112233-ABBA-ABBA-ABBA-BADBADDBAD),
  object
]

interface IAddressList {
  HRESULT addAddress ([in] name, [in] address);
  HRESULT deleteAddress ([in] name, [in] address);
}
```

- language independent interface specification
- can be compiled into language dependent code skeletons
Components and Interfaces - UML definition

Component – a set of interfaces required (in-interfaces) provided (out-interfaces)

Interface – set of operations
Operations – input and output parameters of certain type
Contractually specified interfaces in a UML metamodel

**Diagram:**

- **Constraint**
  - *1*
  - *1*

- **Component**
  - *1*
  - *1*

- **State**
  - *2*

- **Interface**
  - *1*
  - *1*

- **Invariant**
  - *1*

- **PreCondition**
  - *1*

- **Operation**
  - *1*

- **PostCondition**
  - *1*

- **InParameter**
  - *1*

- **OutParameter**
  - *1*

- **In-interfaces**
  - *1*

- **out-interfaces**
  - *1*
Is Szyperski definition enough?
Component specification

- Components are described by their interfaces
- (A black box character)
Nice components that can be composed (put together)
CBD in Practice

Lego + Fisher Technik + Meccano + Ministek + ...
A component can be used within the scope of a component–model
Another definition

- **A software component is a software element that**
  - confirms a component model
  - can be independently deployed
  - composed without modification according to a composition standard.

- **A component model defines specific interaction and composition standards.**

  G. Heineman, W. Councel, Component-based software engineering, putting the peaces together, Addison Wesley, 2001
Variety of Component Models

- Different application(domain)s have different demands on component-based systems.
- Different extra-functional properties
  - Reliability, Resource use, Performance
  - Scalability, Adaptability, Extensibility, backward-compatible

Corba Components:
  distributed, language independent, supports OO

Enterprise Java Beans:
  distributed, Java-oriented, transaction-facilities

Robocup:
  single machine, C/C++, low resource use
Architecture of Component Models

Component platform

OS	Middleware

HW platform

Platform Services
more run-time flexibility
⇒ more elaborate platform

J2EE:
rich services in platform

CORBA:
emphasis on middleware;
integration of legacy
Design Choices in Component Models

- control flow
- concurrency model
- distribution
- interaction style
  - data exchange format
- mobility
- topology
- binding time
- binding type
- platform features
- life-cycle management:
  - instantiation,
  - (de)activation,
  - removal
Interaction Style and Coupling

Component Models often use one particular interaction style. Interaction styles imply some type of coupling!

Referential coupling:
- Broadcast: Radio, TV
- Connection-oriented: telephone

Blackboard coupling:
- Blackboard: Notice board
- Asynchronous: connection, E-mail

Temporal coupling
Referential coupling: sender has reference to receiver’s name
Temporal coupling: sender and receiver synchronize in time
Endogenous versus Exogenous composition

Endogenous composition: composition is built into the items that are composed

Exogenous composition: composition is defined outside of the items that are composed.
- localises changes in binding
- reduces dependencies between application components

Benefits: consider changes in A or B or in the connections.
Summary CBSE – basic definitions

- **The basis is the** Component
- **Components can be assembled according to the rules specified by the** component model
- **Components are assembled through their interfaces**
- **A Component Composition is the process of assembling components to form an assembly, a larger component or an application**
- **Component are performing in the context of a component framework**
- **All parts conform to the** component model
- **A component technology is a concrete implementation of a component model**
Component Technology

- Supporting Tool
- Component Framework
- Components
- Platform
- Repository
Software Architecture and Software Components
The software architecture of a program or computing system is the structure or structures of the system, which comprise software components [and connectors], the externally visible properties of those components [and connectors] and the relationships among them.”
Aspects of Software Architecture

- Elements and Form
- (UniCon notation)
Two Tier Architecture

Presentation

---------------
Business Logic

Database

Driver

Presentation

---------------
Business Logic

Presentation / Business Layer

Data Layer

Tier Boundary
N-Tier Architecture

Presentation Logic

Database Driver

Business Logic

Business Logic

Business Logic

Presentation Layer

Business Layer

Data Layer

Tier Boundary

Database
Different architecture view in different phases

- Phase I
  - System architecture - Decomposition of the system
System Design – Phase 2

- Implementation Architecture - Component Identification
System Design – Phase 3

- Deployment architecture

![Deployment Architecture Diagram]

- Server
  - :ComA
  - :SysX
  - :ComC

- DataServer
  - :ComB
  - :ComD
Basic principles of Component-based approach
Main principles: (1) Reusability

- Reusing components in different systems

- The desire to reuse a component poses few technical constraints.
  - Good documentation (component specification…)
  - a well-organized reuse process
  - Similar architecture
  - ....

[Diagram showing component reuse across different applications]
Main principles: (2) Substitutability

- Alternative implementations of a component may be used.
- The system should meet its requirements irrespective of which component is used.
- Substitution principles
  - Function level
  - Non-functional level
- Added technical challenges
  - Design-time: precise definition of interfaces & specification
  - Run-time: replacement mechanism
Substitution principle

- Substituting a component Y for a component X is said to be safe if:
  - All systems that work with X will also work with Y

- From a syntax viewpoint, a component can safely be replaced if:
  - The new component implements at least the same interfaces as the older components

- From semantic point of view?
  - Contractual interface holds (pre-, postconditions and invariants)
Substitution principle

Principle:
- A component can be replaced if the new component
  - Provide a sub-range of the output
  - Can accept larger range of input

CONDITION.
Everything which comes from the first Tube fits to the second
Main principles: (3) Extensibility

- Comes in two flavors:
  - Extending components that are part of a system
  - Increase the functionality of individual components

- Added technical challenges:
  - Design-time: extensible architecture
  - Run-time: mechanism for discovering new functionality
Main principles: (4) Composability

- Composition of components
  - \( P(c_1 \circ c_2) = P(c_1) \circ P(c_2) \) ??
  - Composition of functions
  - Composition of extra-functional properties

- Many challenges
  - How to reason about a system composed from components?
    - Different type of properties
    - Different principles of compositions
Compositional Reasoning

- Calculating properties of a system by combining properties of its constituents.

- If $S = C_1 \circ C_2$
- Then $P(S) = P(C_1) \times P(C_2)$

- ‘Traditionally’ $P(C_i)$ denotes the meaning of $C_i$
Compositional Reasoning: Functions

- Meaning $P(C)$ of program $C$ can be a function from inputs to outputs

- Then composition is nicely modelled by function composition
Predictable Assembly

• Same question, now for extra-functional properties.

• Let’s consider dynamic memory-use
• Given the dynamic memory-use of C1 and C2.
• Now what is the dynamic memory use of S?
Complicating Factors in Compositional Reasoning about Extra–Functional Properties

- The property is not determined by the components only

- But also by the platform
  - platform may be OS + run-time environment
  - in particular the resource management
    - Scheduling
    - Memory management

- The information supplied by $C_1$ and $C_2$ is not sufficient to reason about the composition of their extra–functional properties.
Components in Unified Modelling Language (UML)
Component diagram

- Three representations for a component

- But access points are required
  - Utilisation of interfaces
  - Utilisation of port
Interfaces

- **Role:**
  - Specification of the access point
  - Required functionalities
  - Provided functionalities

- **2 existing representation**
  - The most descriptive
  - The compact

```
<<interface>>
ProvidedItf
```

```
<<component>>
C
```

```
<<interface>>
RequiredItf
```

```
<<component>>
C
```

```
ProvidedItf
```

```
RequiredItf
```
Ports

- **Role:**
  - Access point to the internal structure of the component
  - Can have 0 or several interfaces

- **Representation:**
Relationship between components

- Use the notion of connector
  - Roughly a way to link components together & make them "communicate" via a request of services

- Generalisation of the means of communication
  - Example:
    - Client-server
    - Pipe&filter
    - Message exchange
  - Can also be called horizontal composition
Vertical composition

- Can also be called hierarchical composition

- Role
  - To increase the component granularity
  - To expose the content of the component

- Use the notion of delegation connector (between two ports)
Profile UML

- Extension of the UML model in order to adapt it to the particular requirements of a context

- Uses
  - Stereotypes
  - Tagged values
  - OCL Constraints

- Examples:
  - Profile for EJB components
  - Profile for a software architecture
EJB Profile

Source: norme UML 2.0
The component model  PICOLO

UML Metamodel

Component-based software development process
Time to Market – “Classical” Development Process?

Product Lifecycle

Requirements → Specification → Design → Implementation → Test → Operation & Maintenance

Problems:
• Time To Market
• High Costs
• Meeting deadlines
• Visibility

TIME
Development process

- COTS and outsourcing require different development processes
Development process – emphasize reuse

- Managing COTS in the early stage of the development process

![Diagram showing development process with stages: Find & Select, Test, Adapt, Deploy, Requirements, Specification, Design, Implementation, Test.]

Page 79, 23 January 2012
CBD – separation of development processes

- Requirements
- Specification
- Design
- Implementation
- Test
- Deploy
- Operation & Maintenance
- Find & Select
- Component development
- Test
- Adap
Types of component-based development

- Internal components as structural entities
- Reusable components developed in-house
- COTS (commercial off the shelf) components
Product Line Architecture

- Core components building a core functionality
  (Basic platform)
- A set of configurable components combined building different products
Platform-based products

- Basic services
- Middleware / infrastructure
- Application layer

Platform layer
Advantages of Software Product Lines

- Using existing infrastructure
  - Savings 30%-40% of efforts per product
  - Time to Market - improved
- Larger variety of products
- Uniform and recognizable functionality/interface
- Better scalability (not obvious!)
- Better maintainability (not obvious!)
- Better possibility of component replacement
Building platforms

The Cathedral and the Bazaar?

La Sagrada Familia, Barcelona

Building Started:
On March 19, 1882

Still not completed

Is it worth to build it in that way?

Similar with platform-based
And component-based development
Is it worth?
Problems and research issues
CBSE research and the SW life-cycle

- Analysis
- Design
- Implementation
- Testing
- Deployment

Components

- design for customization/variability
- wrapping
- specification/contracts

- development methods
- frameworks

- assembly
- finding
- trusting
- distribution
- glue code

- certification
- team structure
- configuration management

Project Management

Quality Management

Application

- SW development process
- run-time infrastructures

- analysis
- design
- implementation
- testing
- deployment

- storage
- documentation

MÄLARDALLEN REAL-TIME RESEARCH CENTRE
Specification

Are more than interface method definitions

- How to specify?
  - Interfaces, behavior (pre-/post conditions, invariants)
  - dependencies (required interfaces)
  - quality of service

- How to test/verify component specifications?
- How to document component specifications?
- How to automatically connect components in builder tools using their specification?
- How to verify the correctness of a composite system?
- ...

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE
Design for reuse

Design for reuse requires additional effort

- What is the best level of reuse (component granularity)?
- How can the benefit of reuse be measured?
- Development and documentation of component usage patterns
Repositories

- How to store components?
- How to classify and describe components?
- How to find components?
  - fast
  - different aspects
    - interfaces
    - functionality
    - component model
    - certification level
    - previous usage, trust
  - negotiable requirements
Software development process

- Current approach
  - requirements - analyses - design - implementation - test

- CBSE approach must include
  - reuse component selection
  - component test
  - requirements reconciliation

- CBSE must be supported by
  - modeling formalisms and tools
  - development tools
Developing a component market

Imperative feature for component success

- Have to establish framework for …?
  - legal aspects (licensing and warranties)
  - technical abilities
  - economic forces
- Proven business case
- Repositories, precise descriptions and search engines
- Documentations and application support
Versioning and configuration management

- Is more complex than usually (DLL hell)
  - especially in dynamic environments
- Dependencies and composition constraints have to be resolved almost automatically
  - consider systems comprising thousands of components
- How to do safe exchange of components e.g. upgrade, without contractual specification and proof?
- All of the issues above are prerequisite for uploading and downloading of components
Security

- Requires trust and certification
- Complicated by large group of (small) vendors
- ‘Mobile code security’ important
  - Not user access control but code access control
- Current mechanisms
  - Sandbox: restricted functionality, restricted availability
  - Codesigning: not necessarily suitable to establish trust
    - Prove of problem origin
    - Difficulty of persecution
Problems and research issues - Summary

- Contracts and documentation
- Design for reuse
- Repositories
- Software development process
- Organizational changes
- Developing a component market
- Versioning and configuration management
- Security
- Component models for embedded systems
Information sources
This presentations is based on:

- Ivica Crnkovic, Magnus Larsson: *Building reliable component-based systems*

  **Chapters:**
  - **PART 1** *The Definition and Specification of Components*
    - Chapter 1 Basic Concepts in Component-Based Software Engineering
    - Chapter 2 On the Specification of Components
  - **PART 2** *SOFTWARE ARCHITECTURE AND COMPONENTS*
    - Chapter 3 Architecting Component-based Systems
    - Chapter 4 Component Models and Technology
  - **PART 3** *DEVELOPING SOFTWARE COMPONENTS*
    - Chapter 6 Semantic Integrity in Component Based Development

- Ivica Crnkovic: *CBSE - New Challenges in Software Development*

- Ivica Crnkovic et al: *Specification, Implementation and Deployment of Components*
Books

- Ivica Crnkovic & Magnus Larsson: *CBSE - Building reliable component--based systems*


- Alan W. Brown: Large-Scale Component-Based Development

- Betrand Meyer: Object-Oriented Software Construction, 2nd

- G.T. Heineman, W. Councill: CBSE Putting the Pieces Together

- J. Cheesmam, J. Daniels: UML Components

- K. Wallnau: Building Systems form Commercial Components
Journals

- IEEE Computer
- IEEE Software
- IEEE Internet Computing
- IEEE Transactions on Software Engineering
- IEEE Transactions on Computers
- ACM Transactions on Programming Languages and Systems
- languages and programming systems.
- ACM Transactions on Software Engineering and Methodology
- ACM Transactions on Computer Systems
- Software Development (www.sdmagazine.com)
- … all major SW development magazines
Conferences

- International Conference on Software engineering (ICSE)
- International Symposium of Component-based Software Engineering (CBSE)
- Euromicro Conference on Software Engineering and Advanced Application (SEAA) – track MOCS – Model-based development, components and services
- International Workshop on Component-Oriented Programming (WCOP)
- Symposium on Generative and Component-Based Software Engineering
- Technology of Object-Oriented Languages and Systems (TOOLS) (www.tools-conferences.com)
- International Conference on Software Reuse (ICSR)
- ESEC/FSE