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. Component Architecture
4. Specific component models
5. Component-based Software Development Process
6. Problems and research issues
7. References
Part 1
The challenges of software development
- how can component software help?

Software problems
How to develop high quality software in an efficient and inexpensive way?

The “software crisis” (1968) still exists:

- SW is late
- SW is buggy
- SW is expensive
- SW is difficult to understand
- Software is difficult to maintain (problems with software life cycle)
SW problems and CBD

- SW is late
- SW is buggy
- SW is expensive

... because too often applications are created instead of constructed, requiring
- re-invention
- re-coding
- re-testing

Component-based software engineering can help because it focuses on reuse of
- subsystems
- infrastructure
Component-based software construction (2)

Application 1

Components

Component #1

Component #2

Component #3

Component #4

Component-based software construction (3)

Application

Components

Component #1

Component #2

Component #3

Component #4

New Component #4

Update
Concentration on the business parts

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

Business issues

<table>
<thead>
<tr>
<th>Standard Reusable parts</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI</td>
</tr>
<tr>
<td>Communication</td>
</tr>
<tr>
<td>Data model</td>
</tr>
<tr>
<td>Deployment</td>
</tr>
</tbody>
</table>

INFRASTRUCTURE

Application specific

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

Part 2
What is a component?
Different understandings…

Architectural point of view

- 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 architectural components

- Elements and Form

![Diagram showing architectural components]

Different understandings – Automotive Industry

Functional components

- Vehicle stability
- Suspension
- Drive by wire
- ..........

Implementations

- Component – combination of hardware and software (ECU)

- Local control function
- Sensor
- Actuator
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 uses 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)

Frameworks - building “the real components”

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

Page 10
Component definition (Szyperski)

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.

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.

- 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.)

A component may provide / implement several interfaces
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.

- Context dependencies - Specification of the deployment environment and run-time environment
  - Example: Which tools, platforms, resources or other components are required?

Do existing component models have support for declaring the dependencies?
Is it possible to verify if the environment comply with the context 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.
- Replacing of a component independent of the client (main application) if the contract is not broken.
- Delegation - interaction with a weak coupling (for example no inheritance).

Which problems can occur in relation to late binding?
How can we guarantee that a replacement of a component will not affect other parts of the system?
### CBS vs. subsystem structuring

<table>
<thead>
<tr>
<th>Component-based system</th>
<th>System with subsystems</th>
</tr>
</thead>
<tbody>
<tr>
<td>focus is on interfaces between units</td>
<td>focus is on functional units</td>
</tr>
<tr>
<td>multiple instances of one component (class) in the running system</td>
<td>structuring, top-down</td>
</tr>
<tr>
<td>composition, bottom-up</td>
<td></td>
</tr>
</tbody>
</table>

### Part 2 - Summary

**What is component?**

- No clearly defined concept
- No revolutionary technology

**Most important features can be summarized as**

- interfaces
- Infrastructure
- Independent deployment
Part 3
Component Architecture

N-Tier Architecture
N-Tier Architecture - and Components

Different architecture view in different phases

- **Phase I**
  - System architecture - Decomposition of the system

  ![Diagram](image-url)
System Design – Phase 2

- Implementation Architecture - Component Identification

System Design – Phase 3

- Deployment architecture
Part 4
Specific component models

Component models (technologies)

- There are many component technologies
  - In-house developed
  - Domain specific
  - Platform specific
  - Language-related
- Standards and de facto standards
  - COM
  - .NET
  - JavaBeans, Enterprise JavaBeans
  - CORBA, CCM
  - ......
Part 5
Component-based software development process

Time to Market – “Classical” Development Process?

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

Product Lifecycle

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Specification</th>
<th>Design</th>
<th>Implementation</th>
<th>Test</th>
<th>Operation &amp; Maintenance</th>
</tr>
</thead>
</table>
Development process

- COTS and outsourcing require different development processes

Requirements → Specification → Design → Implementation → Test → Deploy → Find & Select → Adapt

Development process – emphasize reuse

- Managing COTS in the early stage of the development process

Requirements → Specification → Design → Implementation → Test → Deploy → Find & Select → Test
CBD – separation of development processes

Requirements
Specification
Design
Implementation
Test
Deploy

Find & Select
Adapt

Application development
Component development

Part 6
Problems and research issues
CBSE research and the SW life-cycle

- SW development process
  - certification
  - team structure
  - configuration management

Project Management

Quality Management

Application

Components

- Analysis
  - Design
  - Implementation
  - Testing
  - Deployment

- Analysis
  - Design
  - Implementation
  - Testing
  - Deployment

- development methods
- frameworks
- assembly
- finding
- trusting
- distribution
- glue code
- run-time infrastructures

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?
- ...
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

Organizational changes

“Through at least 2002, most IS organizations will be unable to achieve anticipated CBD payback due to factors such as undefined goals, conflicts of purpose, inflexibility in component design, domain scope or turf wars, and inadequate reuse infrastructure”

- New mind-set: create -> construct
- Allocate reuse management staff (repository, training)
- Accept short term productivity reduction
- Accept new business model: first use of a component designed for reuse may not recover production cost
- More interdepartmental cooperation and trust necessary

[Source: Gartner]
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
- Documentation 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
  - sandboxing: 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