

Introduction to Component Based Development CBD

Agenda

- What is a component?
- What is a software component?
- UML notation for components.
- Why use components?
- How is CBD different from OOP?
- How to use components?

What is a Component?

- An identifiable part of the whole
- They are only a part, not the whole
- Components are possibly:
 - Replaceable
 - Interchangeable
 - A means of supplying variation
- The word *component* comes from the Latin word *componere*, meaning “*to put together*”.

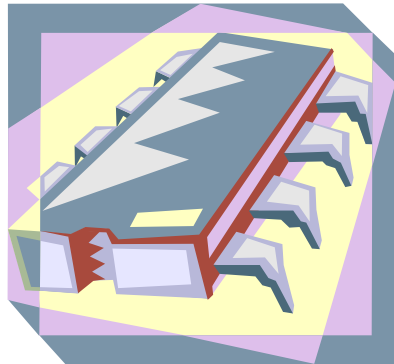
Examples of Components

- Light bulb
- Lego[®] blocks
- Hard disk
- All of these are replaceable and provides possible variation

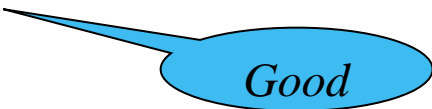

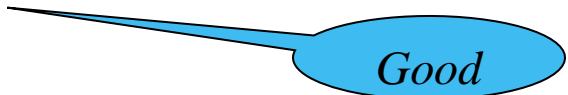
Why Use Components?

- Components simplify the process of building systems
- Promote reuse of both design and implementation
- Allow specialization
 - Lamp manufactures need not also produce light bulbs
 - Computer manufactures need not also produce harddisks

WHAT IS A SOFTWARE COMPONENT?

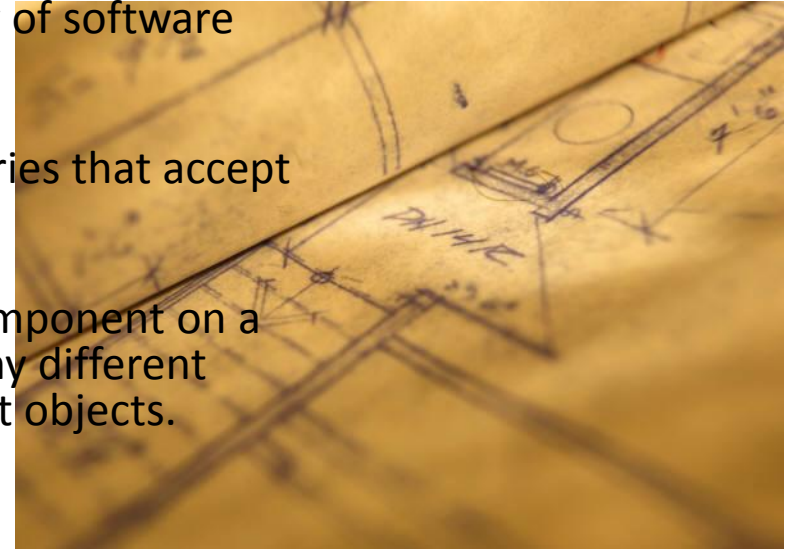
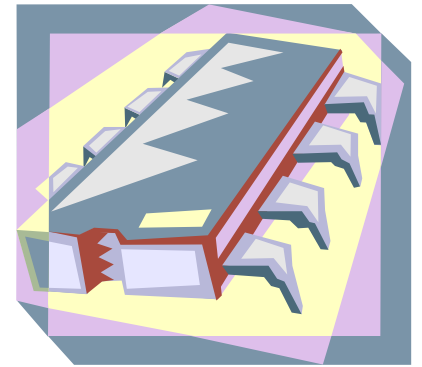


What is a software component?

- A component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces.
[Booch 1998]

- Component: A physical, replaceable part of a system that packages implementation and provides the realization of a set of interfaces. A component represents a physical piece of implementation of a system, including software code (source, binary or executable) or equivalent such as scripts or command files.
[The UML 1.3 specification – OMG 1999]

- Software components are binary units of independent production, acquisition, and deployment that interact to form a functioning system.
[Szyperski 1999]


The nature of software components

- Software components were initially considered to be analogous to hardware components.
- E.g.: integrated circuits, stereo equipment, gears, nuts and bolts, and even Lego blocks.
- But software is different from products in all other engineering disciplines!
- Rather than delivering the final product, delivery of software means delivering the blueprints for products.
- Computers can be seen as fully automated factories that accept such blueprints and instantiate them.
- Normally there is only installed one copy of a component on a computer, but that one copy can be used by many different programs, and may instantiate lots of component objects.



Components characteristic properties

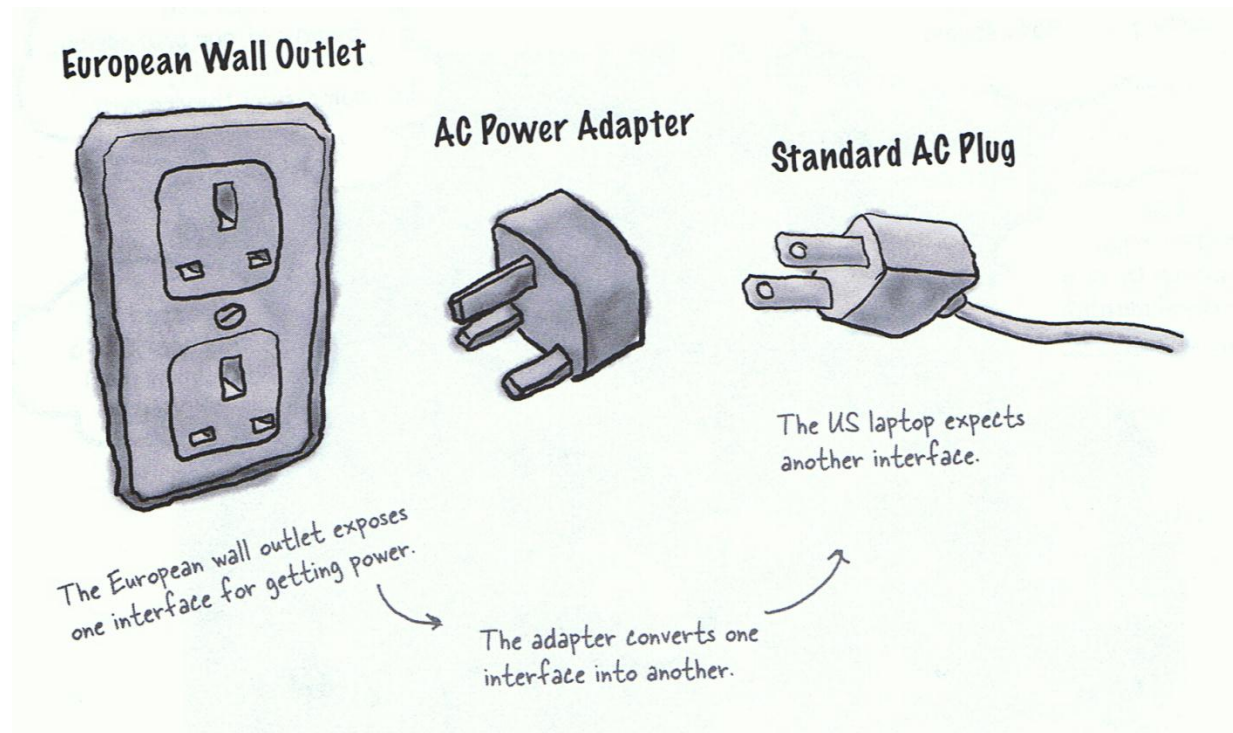
- Szyperski:

The characteristic properties of a component are that it:

- **Is a unit of independent deployment**
- **Is a unit of third-party composition**
- **Has no (externally) observable state**

What Is an Interface?

- Hard disk
 - IDE, SCSI, SAS, ESDI, ATA (PATA), SATA
- Electrical Power
 - Outlet



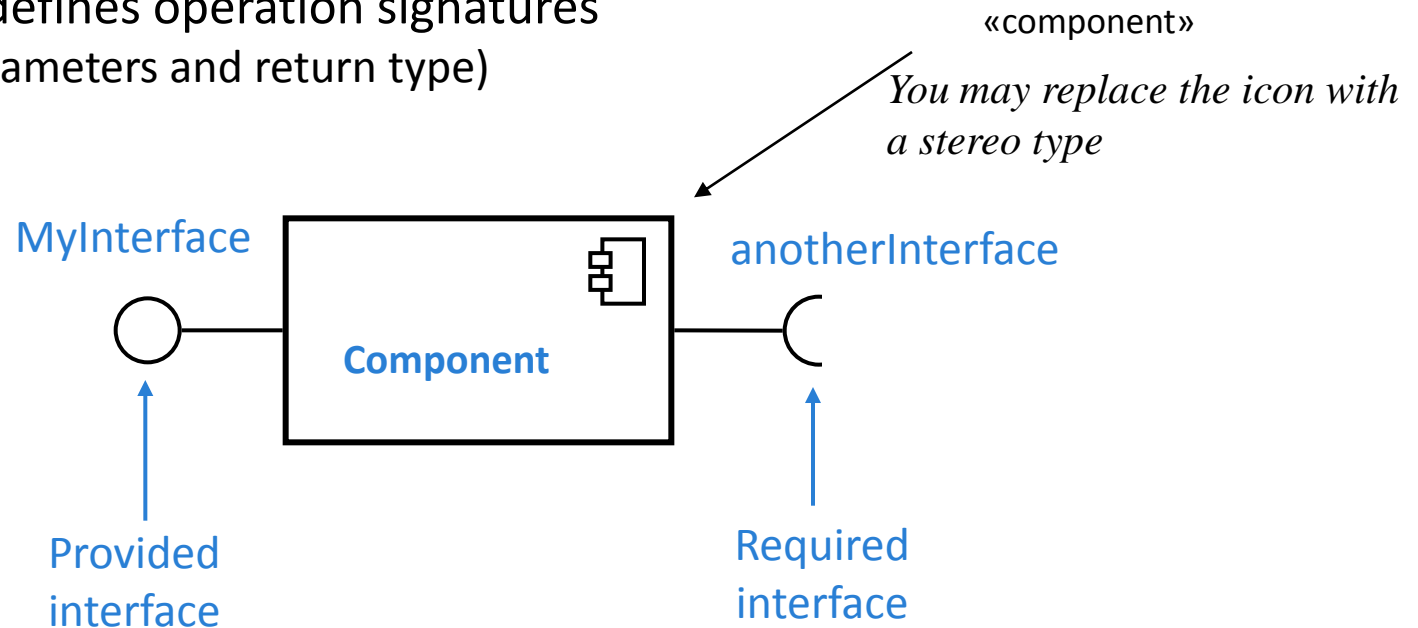
What Is a Software Interface?

- “Abstraction of a service that only defines the operations supported by that service (publicly accessible variables, procedures or methods) but not their implementation.”
 - Szyperski, C.,
Component Software [Addison-Wesley, 1998]

UML NOTATION FOR COMPONENTS

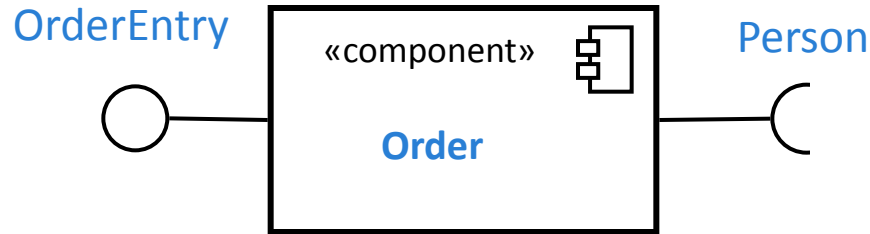
UML Component Notation (1)

- A **component** describes a software unit at run-time e.g.:
 - An executable file (exe)
 - A component (ocx/dll)
 - A table (database, data in ROM etc.)
 - A file (e.g. a configuration file or a html fil)
- An **interface** defines operation signatures
 - (name, parameters and return type)

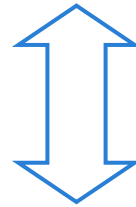


UML 2.0 symbols

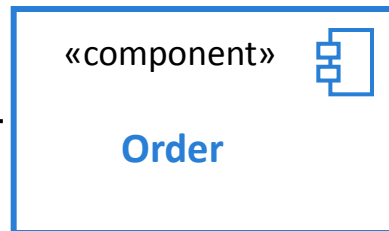
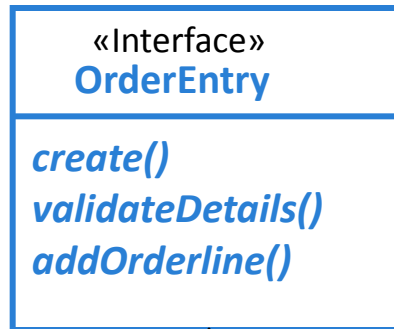
UML Component Notation (2)



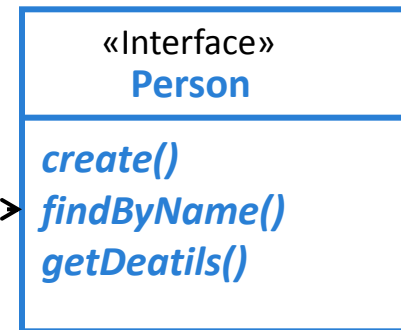
Compact view



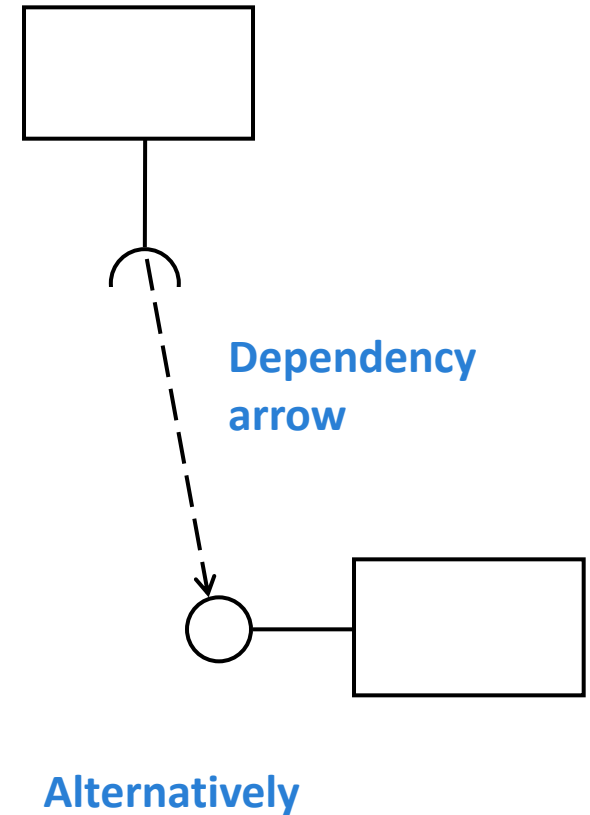
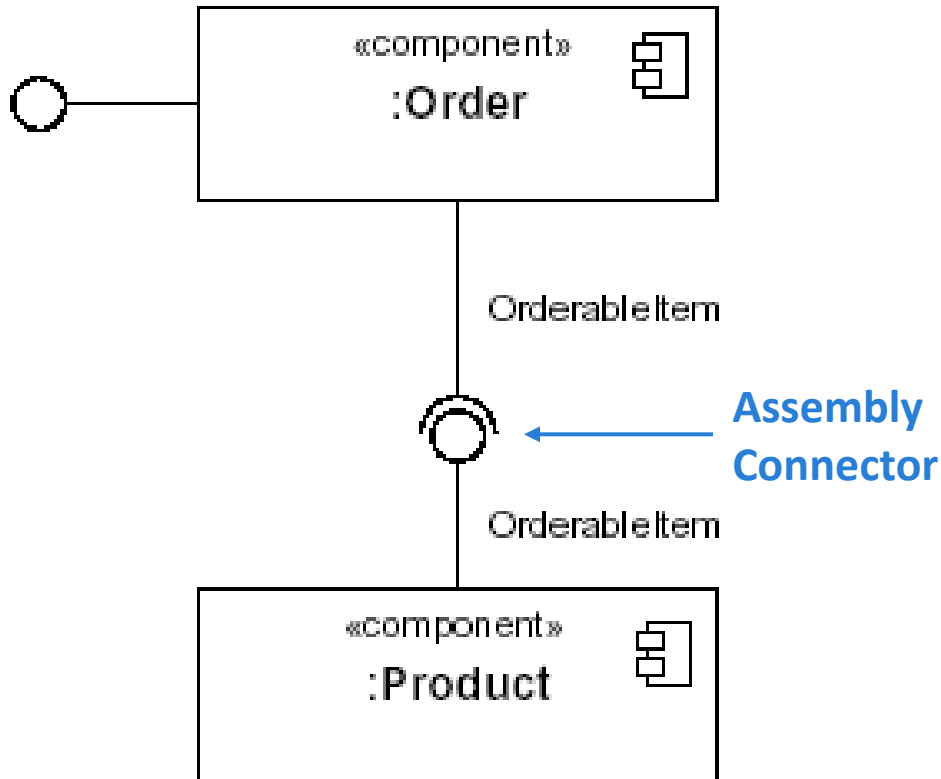
Detailed view



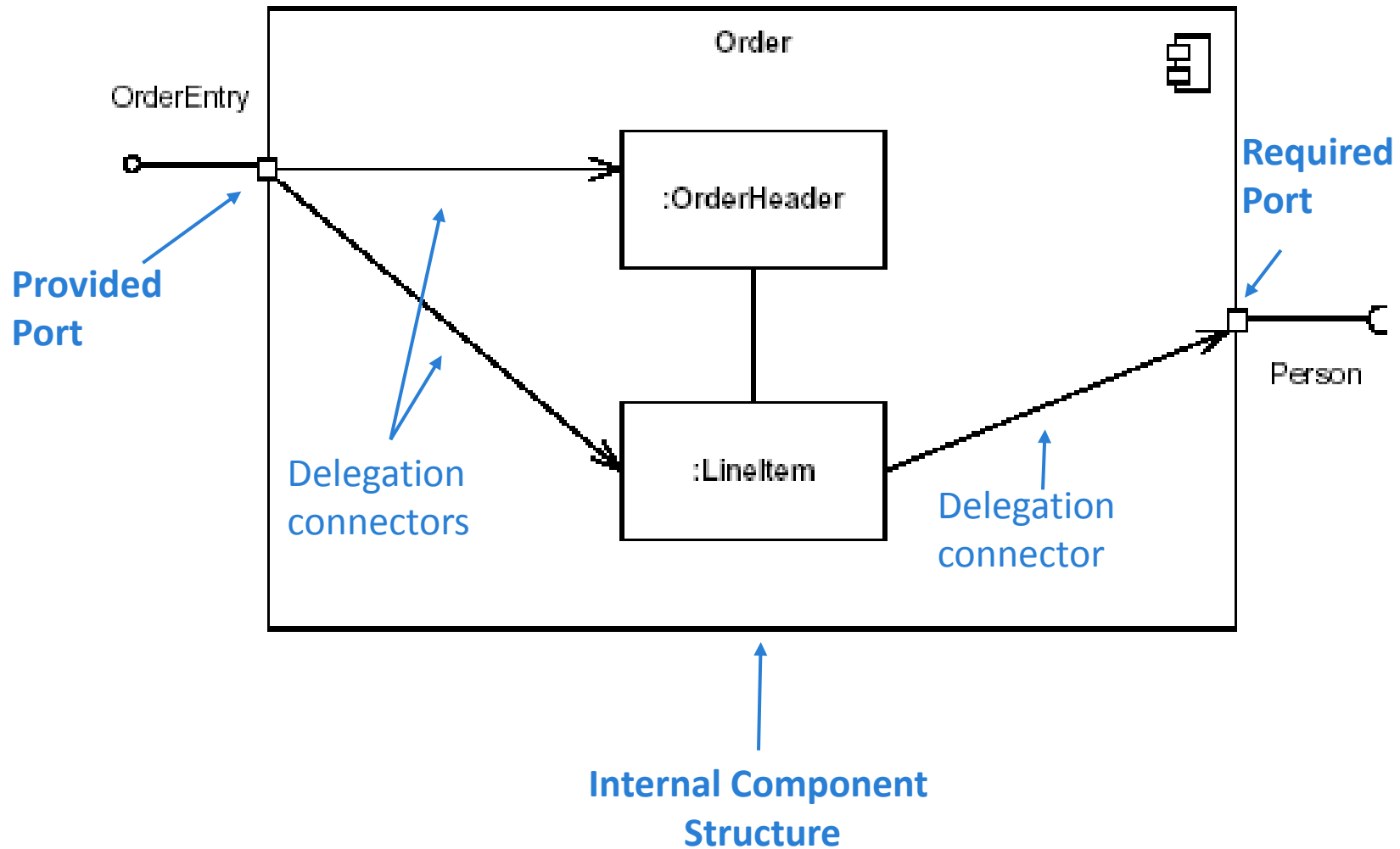
«use»



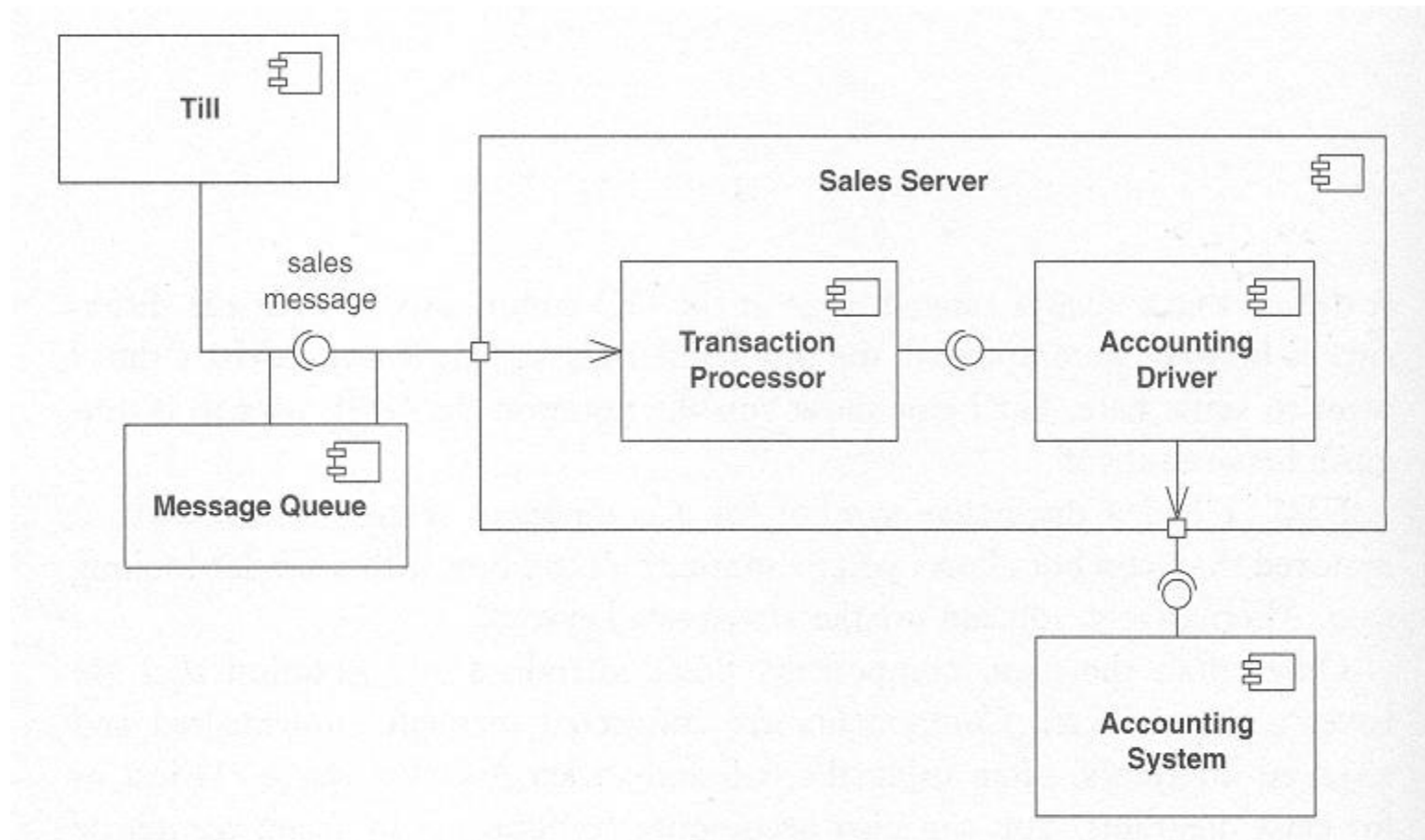
UML Component Notation (3)



UML Component Notation (4)



UML Component Diagram



Why use software components?



Why use components?

- Reuse!
 - To reduce development cost
 - To shorten the time to market
- As early as 1968 McIlroy envisioned “*software integrated circuits*” as the solution to the software crisis.
- All other engineering disciplines introduced components as they became mature – and still use them.
- Components can be seen as the next step after OOP in the software evolution towards a higher level of reuse.
- Szyperski’s rule of thumb:
 - Components must be reused 3 times to get a positive return on investment (*this factor is highly dependent on how much extra plumbing the component framework require*).

ROI of COTS

- The Return on Investment on Commercial off-the-shelf (COTS) Software Components.
Extract from

<http://www.componentsource.com/Services/ROI on COTS Components White Paper.pdf>

Table 1 - Sample of study results

Component	SLOC	Language	% Component Used	SLOC Avoided	Time Avoided (Person Months)	Cost Avoided	I Developer License	ROI (x:1)	Component Type
Sax ActiveX Zip Objects	31,000	VC++	10%	3,100	11	\$108,500	\$399	272	Data Compression Components
Xceed Zip Compression Library	72,773	VC++	10%	7,277	25	\$254,706	\$300	849	Data Compression Components
Dart PowerTCP Zip Compression Tool	29,994	VC++	10%	2,999	10	\$104,979	\$249	422	Data Compression Components
Desaware StorageTools	131,000	C++,	10%	13,100	46	\$458,500	\$199	2,304	Data Storage Components
Dart PowerTCP Mail Tool	44,991	VC++	10%	4,499	16	\$157,469	\$499	316	Email Components
Xceed Encryption Library	42,338	VC++	10%	4,234	15	\$148,183	\$300	494	Encryption Components
Dart PowerTCP SSL Tool	123,843	VC++	10%	12,384	43	\$433,451	\$999	434	Encryption Components
Desaware File Property Component	11,000	C++	10%	1,100	4	\$38,500	\$79	487	File Handling Components

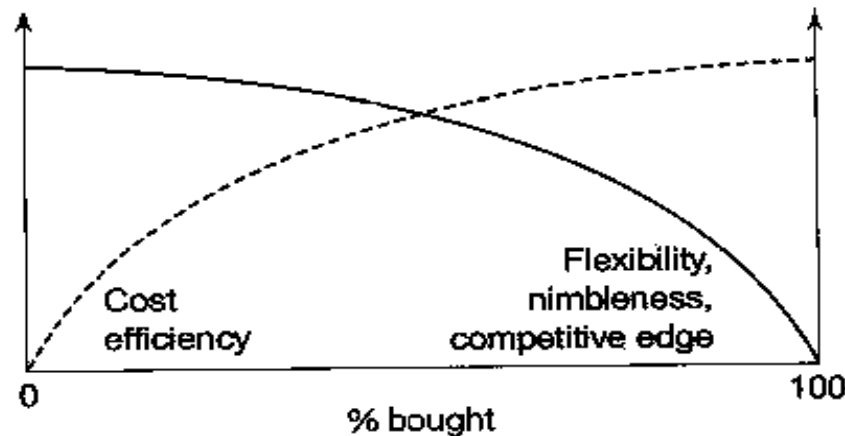
Custom-made versus standard 1/3

- The 2 extremes in traditional software development:
 - Full custom made:
All the software is developed entirely from scratch, with the help of only programming tools and libraries.
 - Standard software:
All the software is bought and just parameterized to provide a solution that is “close enough” to what is needed.

Custom-made versus standard 2/3

	Custom made	Standard
Pros	Can be optimally designed to the corporate business process, and can provide the competitive edge.	Relatively cheap - fixed price. Fewer bugs. Short time to adapt.
Cons	Very expensive. Long development time – is often to late. Tend to have many sub-optimal solutions in areas with lack of expertise.	May demand a reorganization of the business process. The competitors have is as well. No control of it's development – what direction and when to move forward.

Custom-made versus standard 3/3



The concept of component software represents a middle path that could solve this problem.

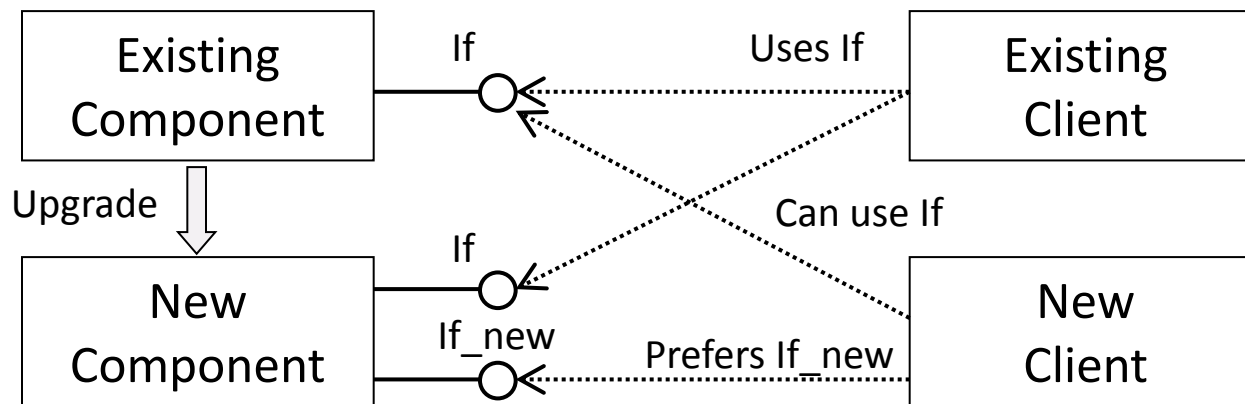
Although each bought component is a standardized product, with all the attached advantages, the process of component assembly allows the opportunity for significant customization.

It is likely that components of different qualities (level of performance, resource efficiency, robustness, degree of certification, and so on) will be available at different prices. It is thus possible to set individual priorities when assembling based on a fixed budget.

In addition, some individual components can be custom-made to suit specific requirements or to foster strategic advantages

Why use components: Managing Changes

- Requirements change because customers change, the environment change etc.
- Managing changes during the lifetime of the software is one of the big challenges.
- Software components are well suited to handle changes because they have a strong encapsulation and uses interfaces.
- These characteristics allow a component to be upgraded or replaced with minimal impact on the clients of that component.



Why use components: Managing Big Projects

- Components may also be used to brake up a large project into smaller independent parts, suitable for independent development by smaller teams (Scrum).

Potential Problems

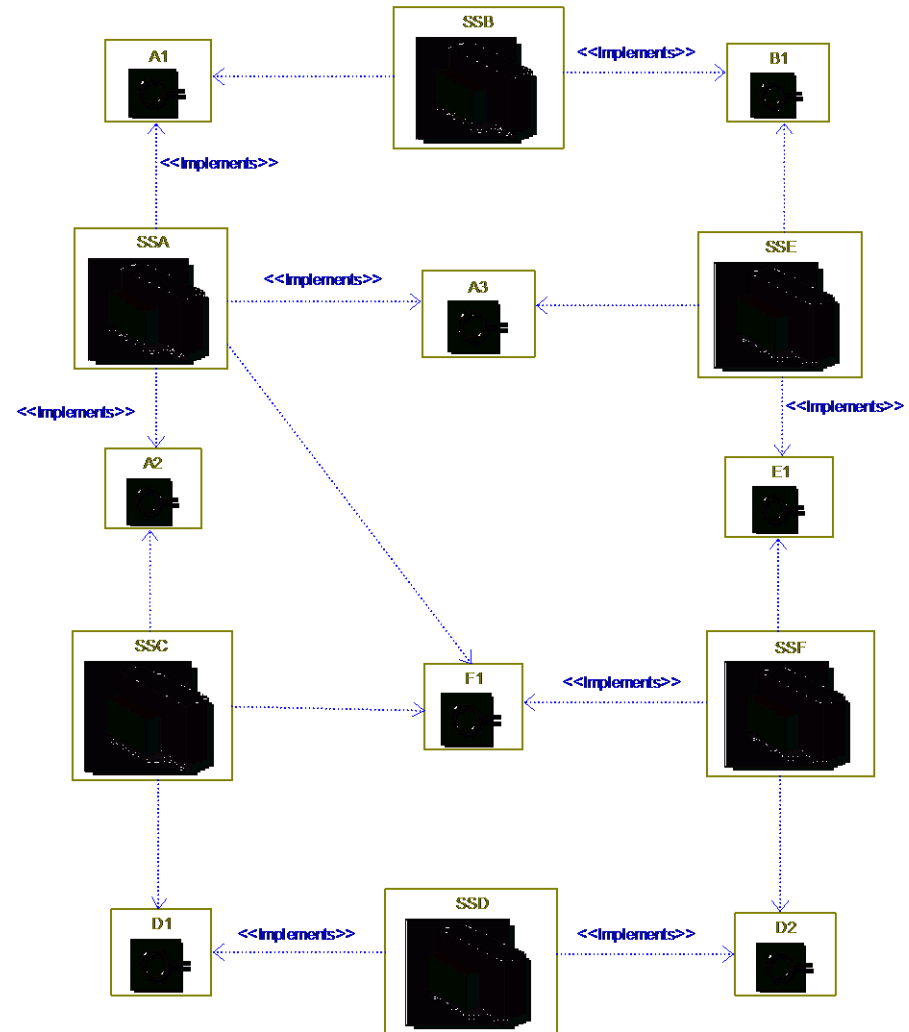
- If we don't take care in putting together a good structure for our huge project, then we'll find the following problems start to appear over time:
 - The main problem is that there will be too many dependencies between model elements, the more the dependencies, the longer it will take to generate code. After a simple modification is made to just one class, many more will need to be recompiled.
 - Developers will want to work on the same elements at the same time resulting in frustration or the need for complicated merging later.
 - For Case tools: As the model gets very big, it takes a long time to load, save and to manipulate it. Because of the many dependencies, in order to generate code, almost all the model must be loaded.

Component Based development

- One solution to all of these problems is to use component based development
- The idea here is to break down the project into large independent units - sometimes called Sub-Systems.
- All the Sub-Systems will communicate only via interface classes between themselves.
- Each Sub-System will be mapped to a single component.
- Each developer / group will work independently on a Sub-System / component.

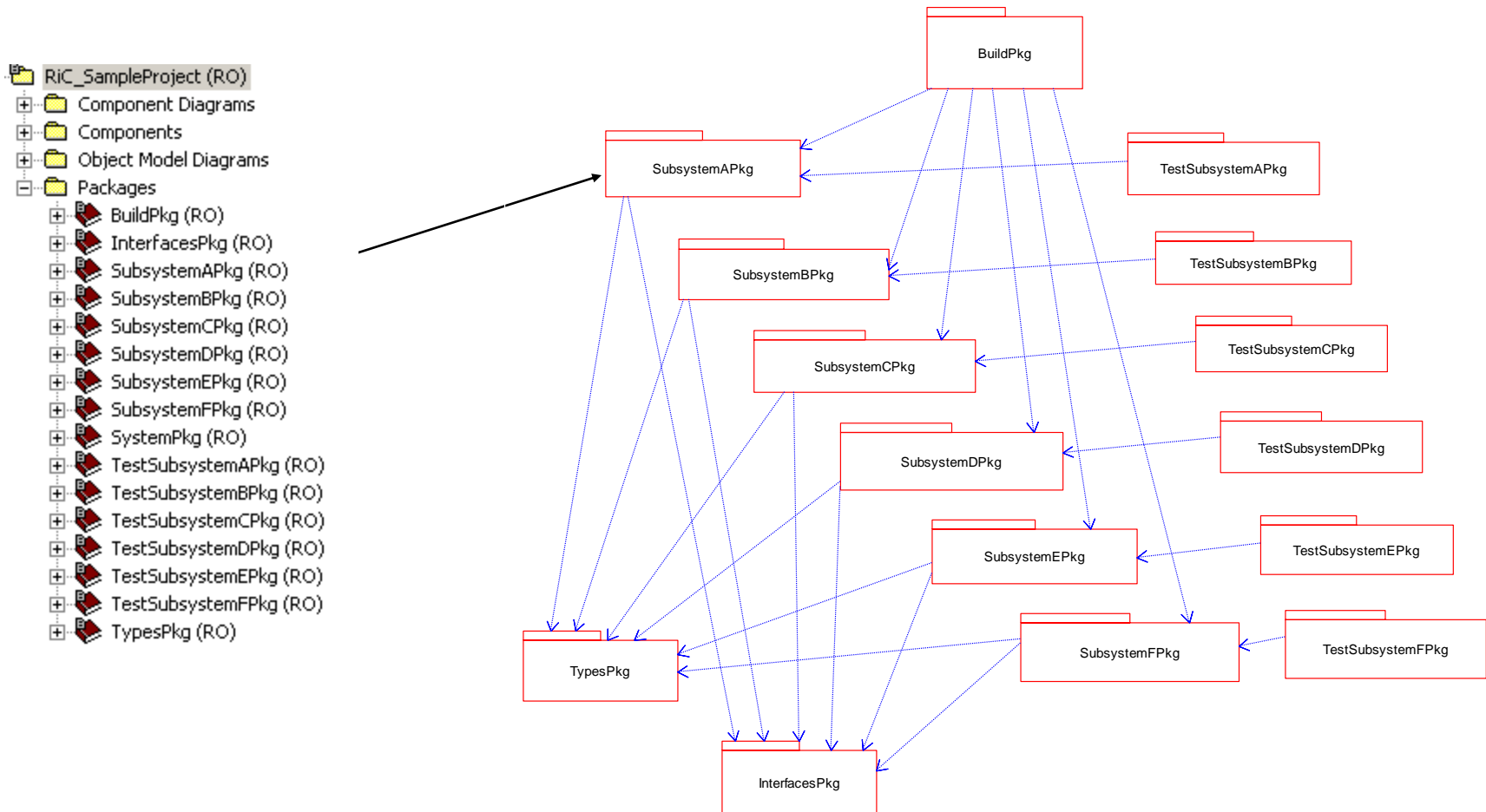
A system split up in independent component

- The diagram shows how a system has been divided into six components each of which implement some interfaces and use other interfaces.
- Each component can be developed concurrently and independent of the other components
 - **when all the interfaces have been specified!**



Example Package structure

- Each Sub-System maps to a different package / component



What's the difference between CBD and OOP?

Synonymer

Komponentbaseret programudvikling

kaldes på engelsk

Component Based Development, CBD

eller

Component-based Software Engineering CBSE

What is CBD?

Component Based Development / Component-based Software Engineering

Brown and Wallnau (1998):

“CBSE is coherent engineering practice, but we still haven’t fully identified just what it is.”

This quote is from David Budgens book: “Software Design” 2003!

From a purely formal point of view there is nothing that could be done with components that could not be done without them.

The difference are concepts such as reuse, time to market, quality and viability.

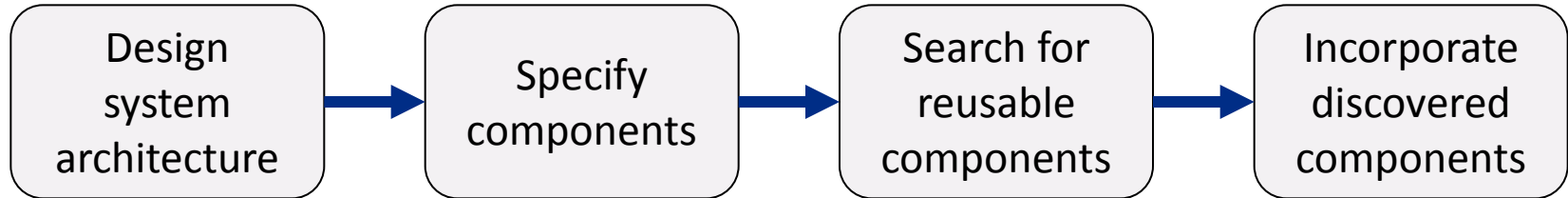
All of these are of a non-mathematical nature and value, and are generally of more interest for the software business than for the computer science faculties.

How is CBD different from OOP?

- CBD shares all the methodologies of OOP.
- But CBD also addresses:
 - Packaging
 - Deployment
 - Licensing
 - And potentially Security
- OOP takes code to the point where it runs on the developer's machine.
- CBD takes it to the point where it runs on the end-user's machine.

How to use software components

An opportunistic reuse process



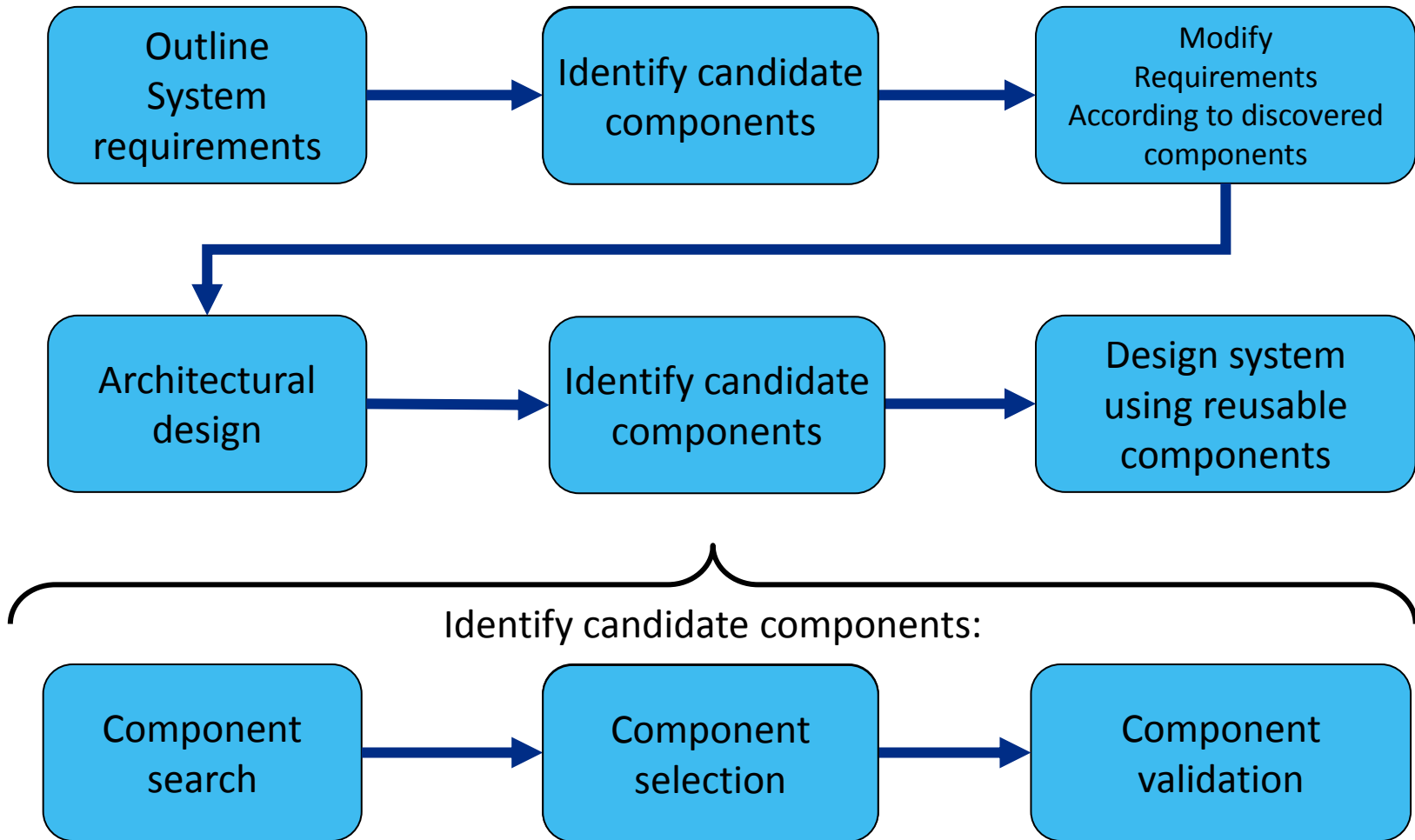
If you make a detailed specification of the components you want, then you are unlikely to find any!

If you want to use COTS components, you have to investigate the market for relevant components while designing software architecture and sometimes during the initial requirement analysis.

The CBSE process

Development with reuse

From: I. Sommerville "Software Engineering"



The process for selecting COTS components

From Ted Faisons "Component-Based Development with Visual C#"

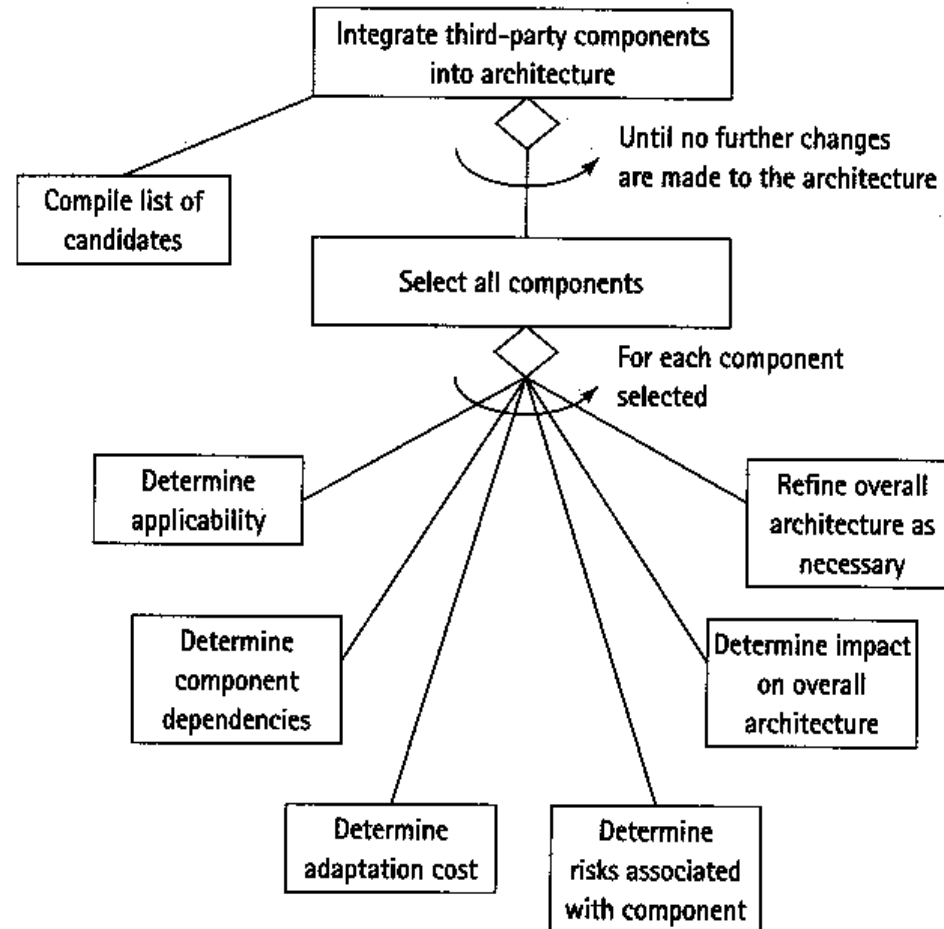


Figure 1-5: The process for selecting third-party components

Architecture

- **“The right architectures and properly managed architecture evolution are probably the most important and challenging aspects of component based software engineering.”**

Szyperski, C.,
Component Software [Addison-Wesley, 1998]

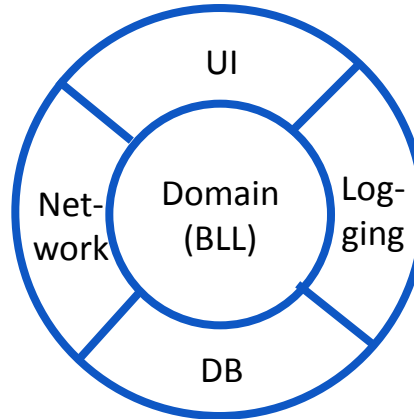
Architecture

- “Overall design of a system. An architecture integrates separate but interfering issues of a system, such as provisions for independent evolution and openness combined with overall reliability and performance requirements.”
- “An architecture defines guidelines that together help to achieve the overall targets without having to invent ad hoc compromises during system composition.
- **An architecture must be carefully evolved to avoid deterioration as the system itself evolves and the requirements change.”**

Szyperski, C.,
Component Software [Addison-Wesley, 1998]

Michael Feathers:

- If one of your goals is to make a system that users can extend with components, it helps to view your architecture as a big circle.
 - In the center you have your domain code.
 - The boundary of the circle is where you interface with the world (databases, devices, UI, etc).



- Every time you add a new capability to your system, you have to change your domain classes and work your way outward to express the new capability at the edge of the circle.
- This is one of the reasons why most components are graphical rather than NonGraphicalComponents.
- Decreasing the amount of code that has to change from the center to the edge of the circle makes extensibility easier.
- One of the ways to avoid code change is to use data and metadata extensively. For instance, if you use XML, you can add components closer to the center of the system and give them a data channel to the UI.

Ralph Johnson:

- Components are about customers.
 - It is the people who pay for software who are driving components. That is why sales and marketing people are talking about them. They know that people want this. So, they tell us to make it, or they tell us they have it already and our management buys it and tells us to use it.
- In fact, making good components requires more than a particular technology like .Net, COM or EJB. It requires understanding the problem domain and figuring out how to carve it into the right kind of pieces.
- Component software requires some sort of underlying structure like .Net, COM or EJB, but that is just the start. You also have to develop standard interfaces on top of it, and they have to be the right kind of interfaces.

Commercial Components

- There are several marked places for components and many vendors
- Component Source
<http://www.componentsource.com/index.html>
- DevExpress
<http://www.devexpress.com/>
- Open source
 - <http://sourceforge.net/>
 - <http://www.codeproject.com/>
 - <http://www.codeplex.com/>
 - ...