

Comp2110

Software Design

Comp2510

Software Design for Software Engineers

13 August 2008

Srinivas Chemboli
Department of Computer Science
ANU College of Engineering and Computer Science
The Australian National University

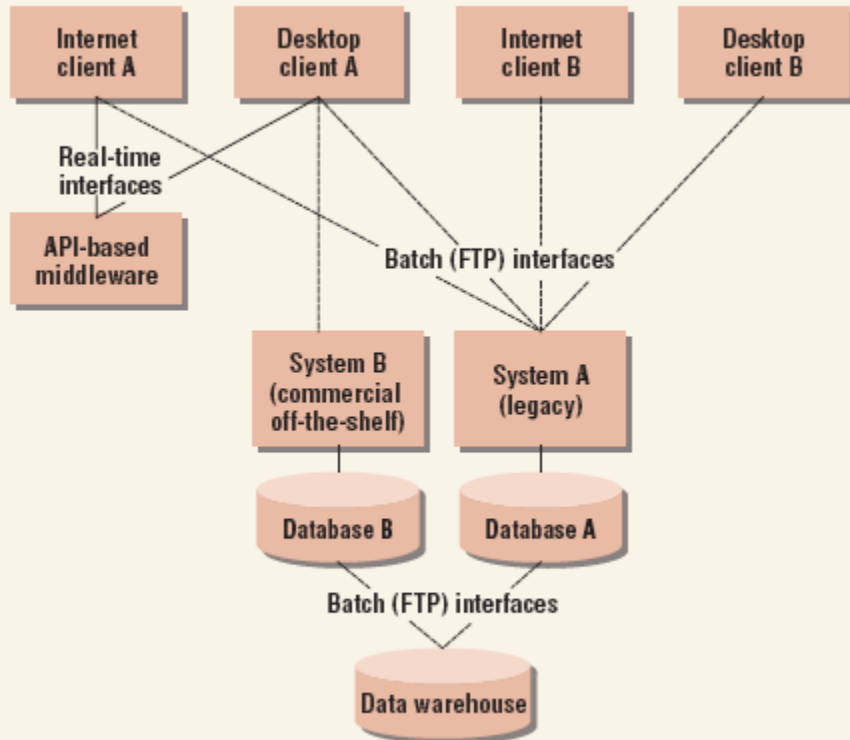
In This Session...

- **A descriptive understanding of Software Architectures**
- **A look at existing software architectures**
- **Thinking about software architecture**

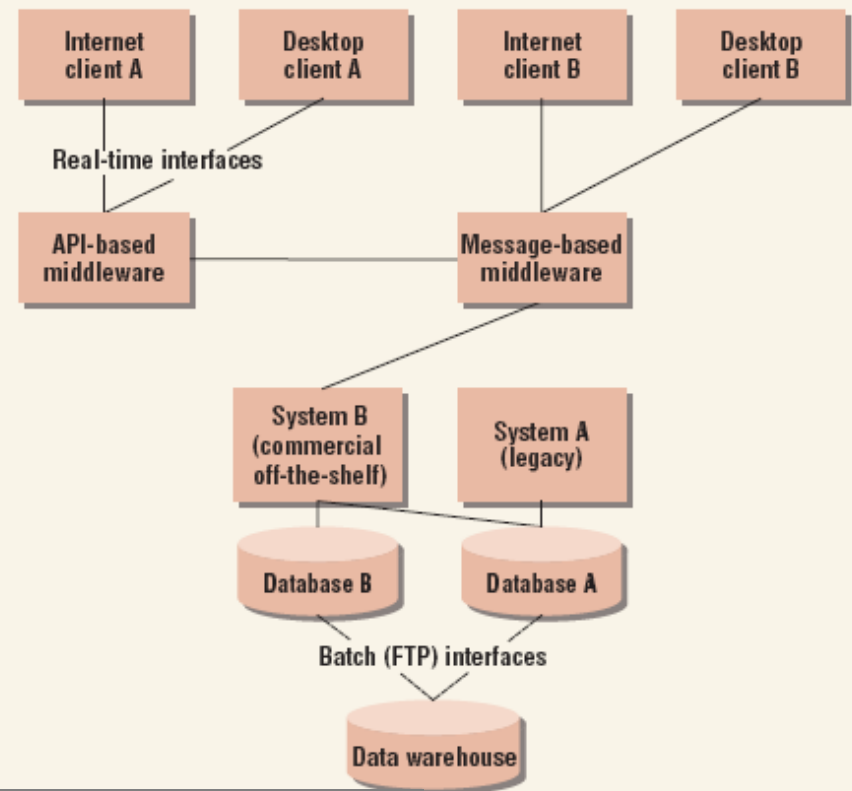
Software Architecture (Recap)

- **Decomposition of design into groups of abstract modules**
 - **Modules are concepts [with names]**
- **Identify potential for generality and reuse in other projects**
- **Coherence**
 - **Stronger / more relationships between classes inside module boundary**
- **Coupling**
 - **Weaker relationships going in and out through the boundary**

Design Description using Architecture



Current Architecture



Future Architecture

Figures 1 and 3 from Jeffy Tyree and Art Ackerman: *Architecture Decisions: Demystifying Architecture* [IEEE Software, Mar/Apr 2005 pp19-27]

Elements of Software Architecture

- Description of elements from which software systems are built
- Interactions between those elements
- Patterns that guide their composition
- Constraints on those patterns
- **In general, a system is defined in terms of**
 - A collection of components
 - Interactions between those components

A Few Existing Software Architectures

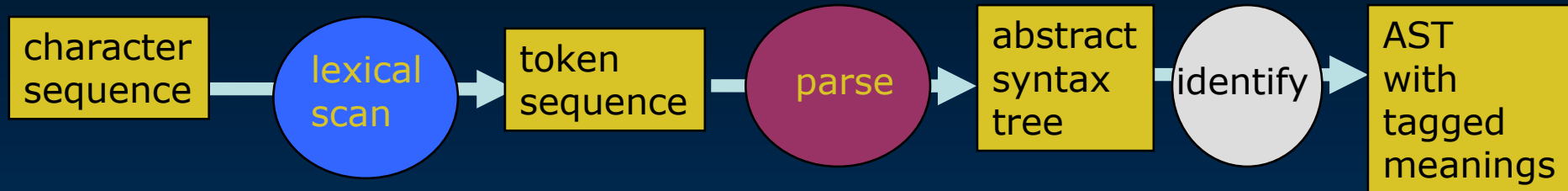
- **Data-flow systems**
 - **Batch sequential**
 - **Pipes and filters**
- **Call and return systems**
 - **Main program and subroutines**
 - **Hierarchical layers**
- **Independent components**
 - **Communicating systems**
- **Model View Controller**

Example: Lexical Analysis, Parsing and Identification

- **An application to process complex sequences of character input**
- **3 preliminary stages**
 - **Lexical analysis**
 - **Parsing**
 - **Identification of words (identifiers)**
- **Examples**
 - **Compiling and interpreting programming languages**
 - **Parsing HTML, XML for rendering, editing, transforming**

Example: lex, parse, identify

- Abstract dataflow model:



- 2 alternatives for implementing these 3 processing stages:
 - Batch processing
 - Pipeline processing

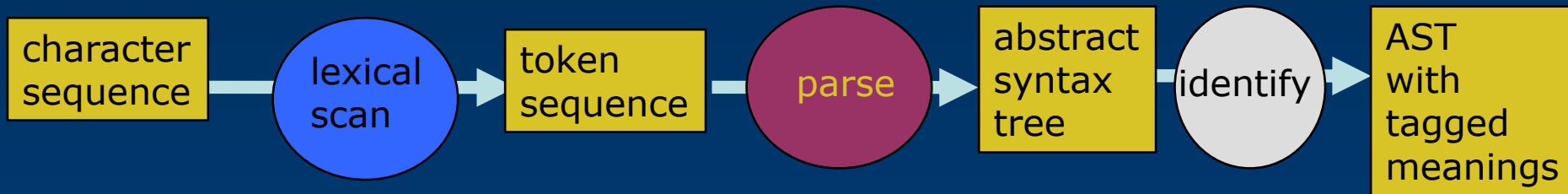
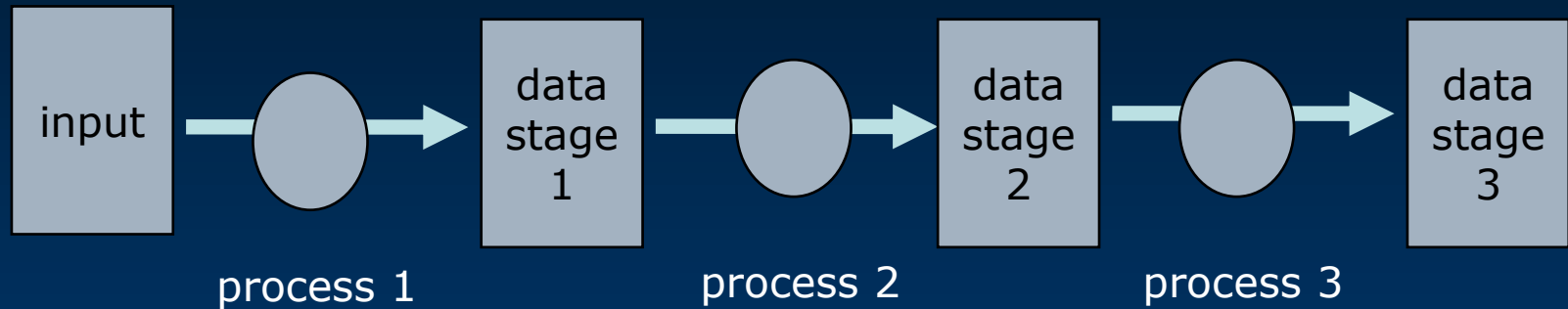


***Batch processing* uses the schema:**

Divide the processing into a number of processing stages that can be done in sequence.

- **process all of the input in a batch as a whole through process 1, to produce a batch of intermediate data stage 1**
- **process all of the intermediate data stage 1 through process 2... to produce a batch of intermediate data stage 2**
- **in general**
process data stage i through process stage $i+1$ to produce data stage $i+1$

batch architecture



Stage 1 – lexical scan

input
HTML source

```
<p color="black"  
font="Times">  
This is it <pre>  
no more code!  
<!-- but then... -->  
no more  
</pre> that was it </p>
```

**lexical
analysis**

output
sequence of tokens

```
STARTTAG "p"  
ATTR "color"  
ATTRVAL "black"  
ATTR "font"  
ATTRVAL "Times"  
DATA "This is it "  
STARTTAG "pre"  
DATA " no more code!"  
COMMENT " but then..."  
DATA " no more"  
ENDTAG "pre"  
DATA "that was it"  
ENDTAG "p"
```

Stage 2 – parse

input
sequence of tokens

```
STARTTAG "p"  
ATTR "color"  
ATTRVAL "black"  
ATTR "font"  
ATTRVAL "Times"  
DATA "This is it "  
STARTTAG "pre"  
DATA " no more code!"  
COMMENT " but then..."  
DATA " no more"  
ENDTAG "pre"  
DATA "that was it"  
ENDTAG "p"
```

parse

phrase structure
(syntax tree)

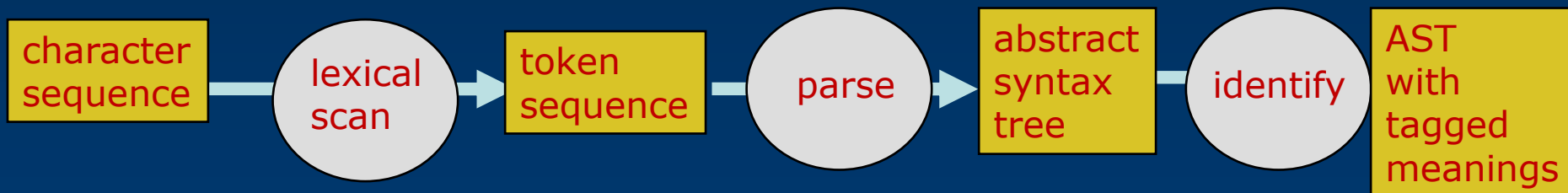
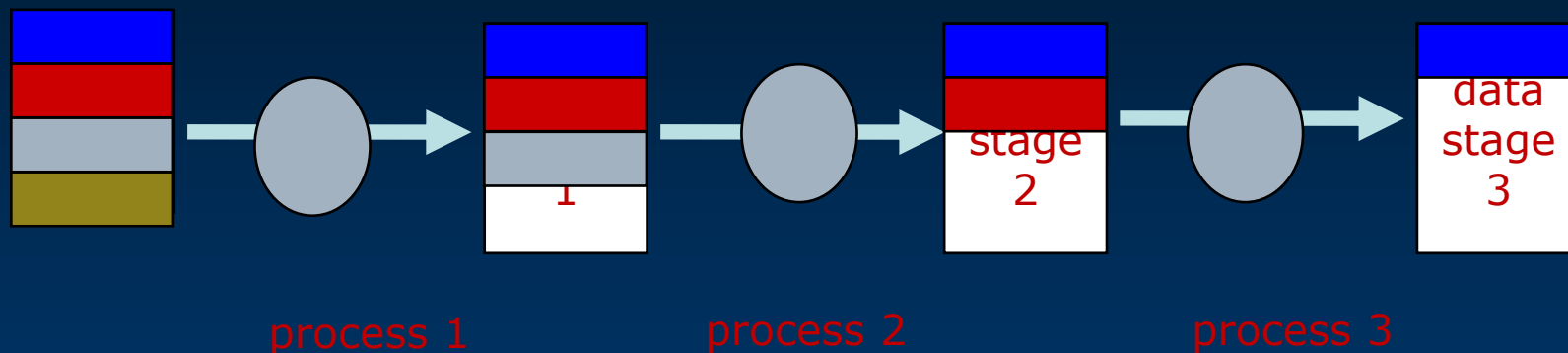
```
ATTRVAL "black"  
ATTR "color"  
ATTRVAL "Times"  
ATTR "font"  
ATTRIBSEQ 2  
DATA "This is it"  
DATA " no more code! "  
COMMENT " but then..."  
DATA " no more"  
ELEMENT "pre" 3  
DATA "that was it"  
ELEMENT "p" 4
```

*(this is a syntax tree, in postfix notation:
indentation added for human reader)*

Alternative: Pipeline architecture

- in a pipeline, the processes produce the same input and output as the batch processes
but as elements in sequences, not as complete batches
The processes operate incrementally: later processes can start before the earlier ones have finished.
- the batch architecture is a limiting case of the pipeline

pipeline architecture



Pipeline architecture

- the processes are not always in one for one lock step:
downstream may have to wait
 - E.g. the scanner will need to consume a variable number of characters for each token
 - the parser may need to consume several tokens before it can produce any parse tree
- the architectures can be mixed: e.g in one HTML display application: *WebView*

lexical scanner-> parser

-> renderer *Pipeline*
 Batch

batch vs pipeline

- **performance: space (batch: big) and time**
- **responsiveness: pipeline has fast first output**
- **may need whole structure to apply variety of last stage processes – suggests batch at the last**
- **may need overview across whole of input**
- **pipeline may be too complex to program**

Model-View-Controller Architecture

Recall the example of a spreadsheet program

Model - the core application part

- underlying data, functional core of application
- The Model Component also *registers* the dependent views and controllers
- *Notifies* views and controllers when data in the model changes

View - the graphical part of the interface

- *creates* its controllers
- displays information (about the model) to the user
- *updates* itself when it is *notified* that the model changes by retrieving data (*gets state*) from the model and re-doing its display

Controller - the decision-making part

- accepts user inputs and other events
- translates events into service requests for the model or display requests for the view

Using a named architecture in a high-level design

- name the architectural model you use
e.g. Model View Controller
- name which modules are playing which roles
e.g. the Model is module CentralControlStates
the View is module UserManager
the Controller is the ControlLogicManager module
- state any major differences from the standard architecture

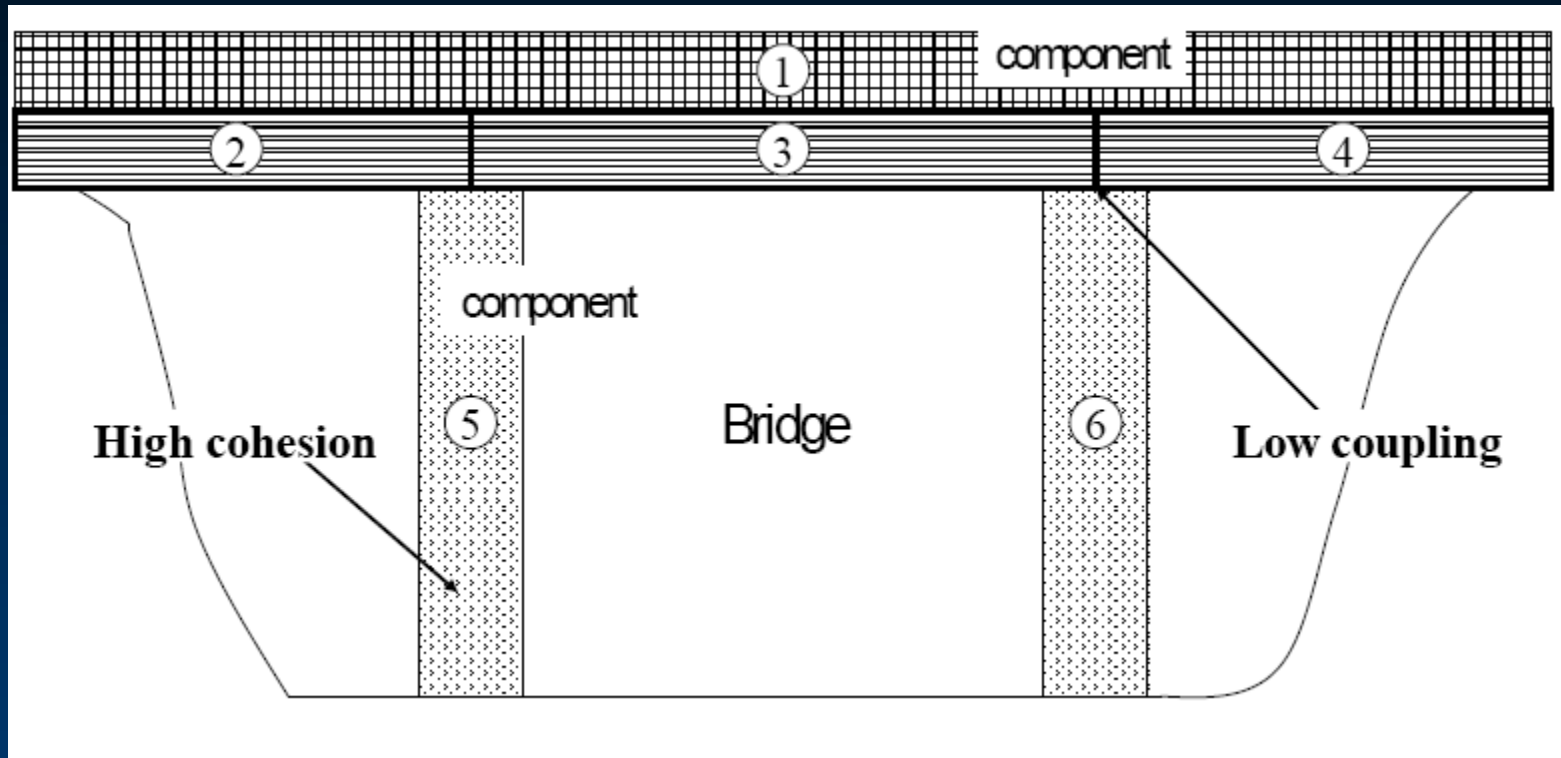
Selecting a basic architecture...

- 1. Develop a mental model of the application at a high level – *creative invention***
 - as if it were a small application
 - e.g., personal finance application ...*
 - ... “works by receiving money or paying out money, in any order, controlled through a user interface”.*
- 2. Decompose into the required components**
 - look for **high cohesion & low coupling**
 - e.g., personal finance application ...*
 - ... decomposes into Assets, Suppliers, & Interface.*
- 3. Repeat this process for the components**

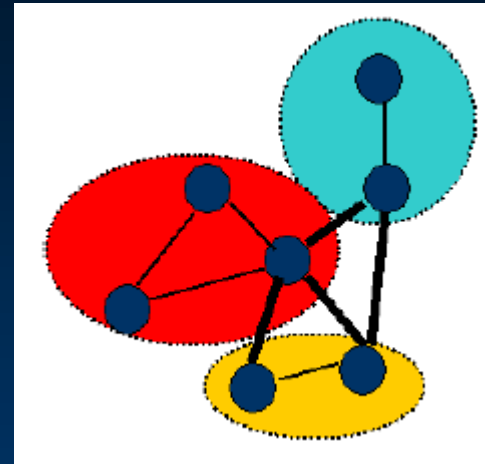
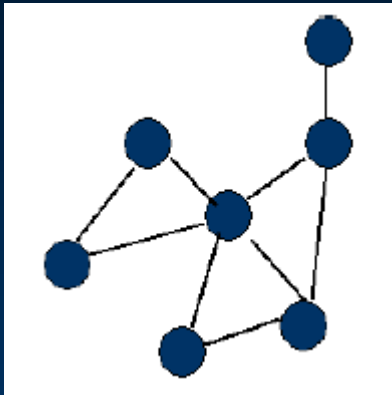
Cohesion and Coupling

- a principle of quality design
how to divide systems into modules
- modules should have **strong cohesion**:
 - “hang together” logically,
 - have a common theme,
 - have more interaction (associations, uses relationships)
between things within the same module
than with things outside
- and there should be **weak coupling** between modules
 - small interfaces
 - thin interfaces
 - few dependencies – easy to separate, reuse, replace

Cohesion and Coupling



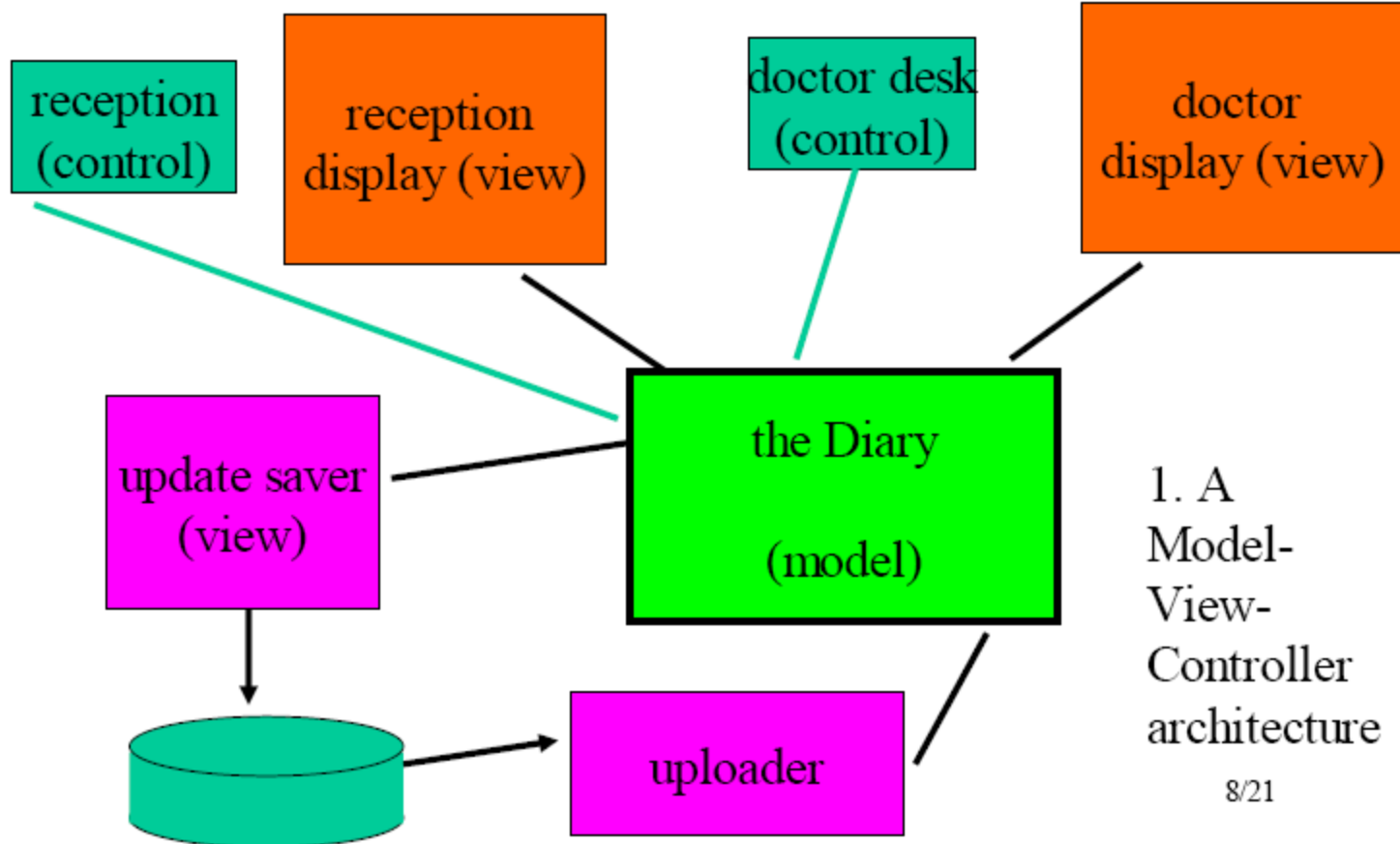
Cohesion and Coupling





High Level Design: Medical Center Diary

it is always best to have two or more alternatives to consider

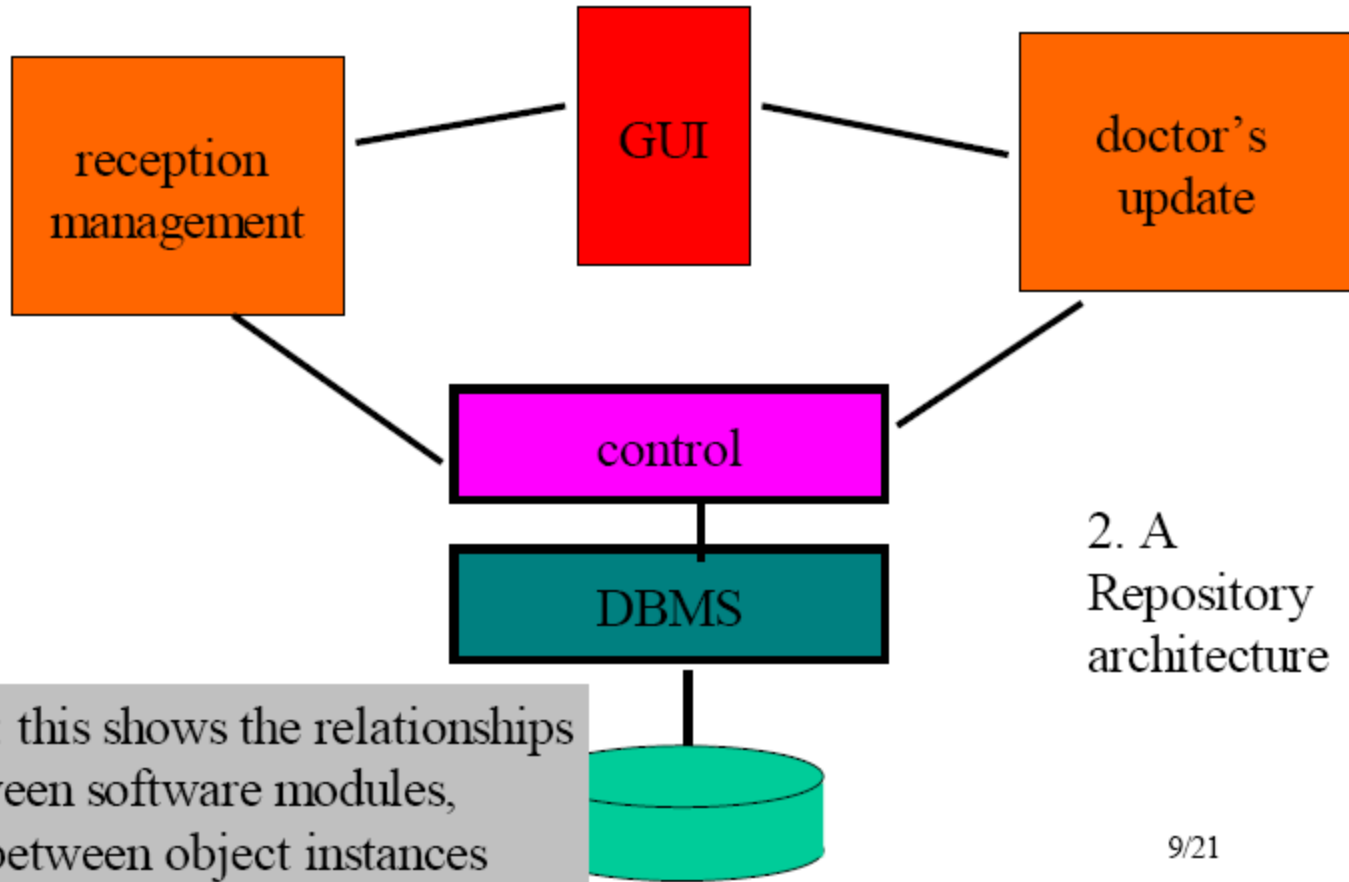


1. A Model-View-Controller architecture



High Level Design: Medical Center Diary

it is always best to have two or more alternatives to consider



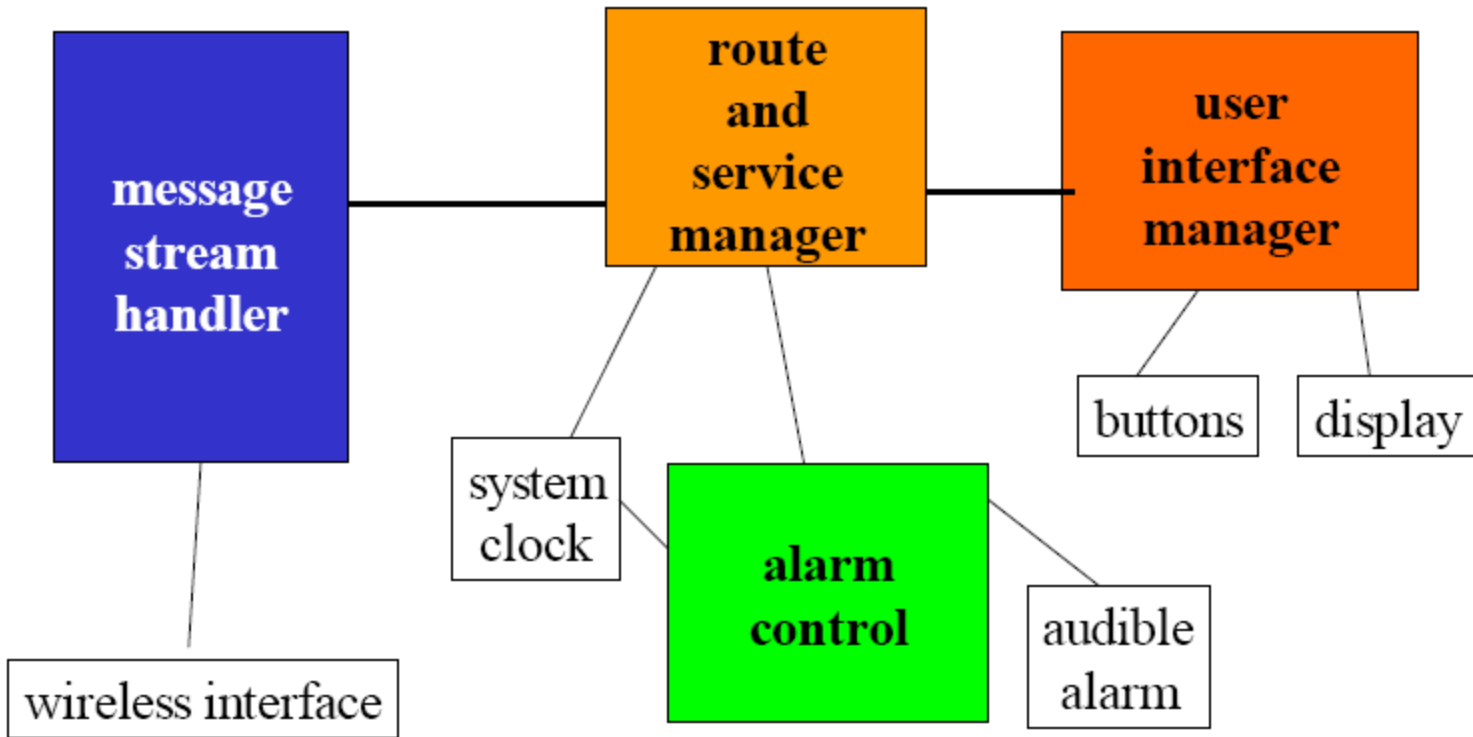
2. A Repository architecture

note: this shows the relationships between software modules, not between object instances



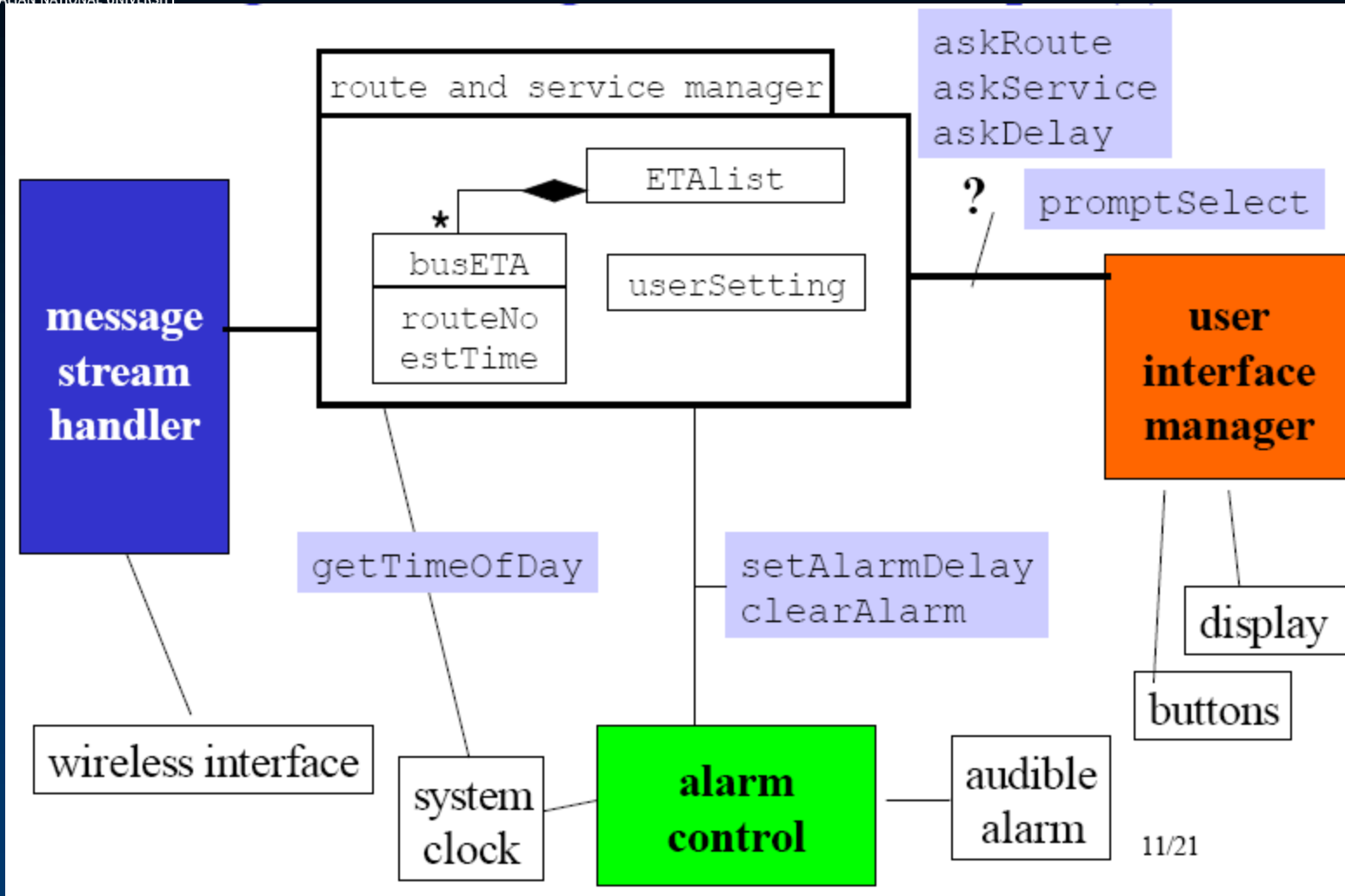
High Level Design: Module Example

BusCatcher



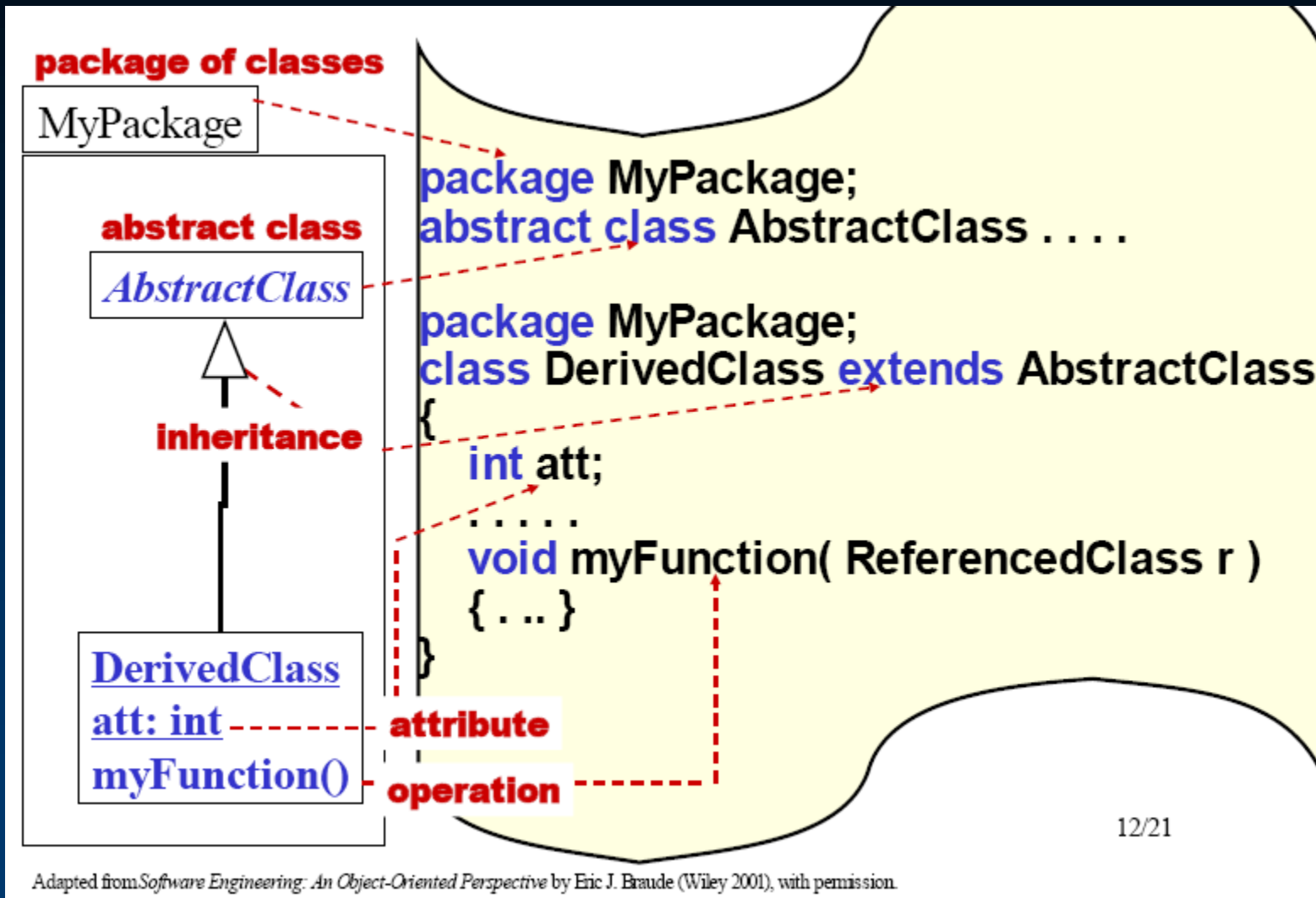


High Level Design: Module Example



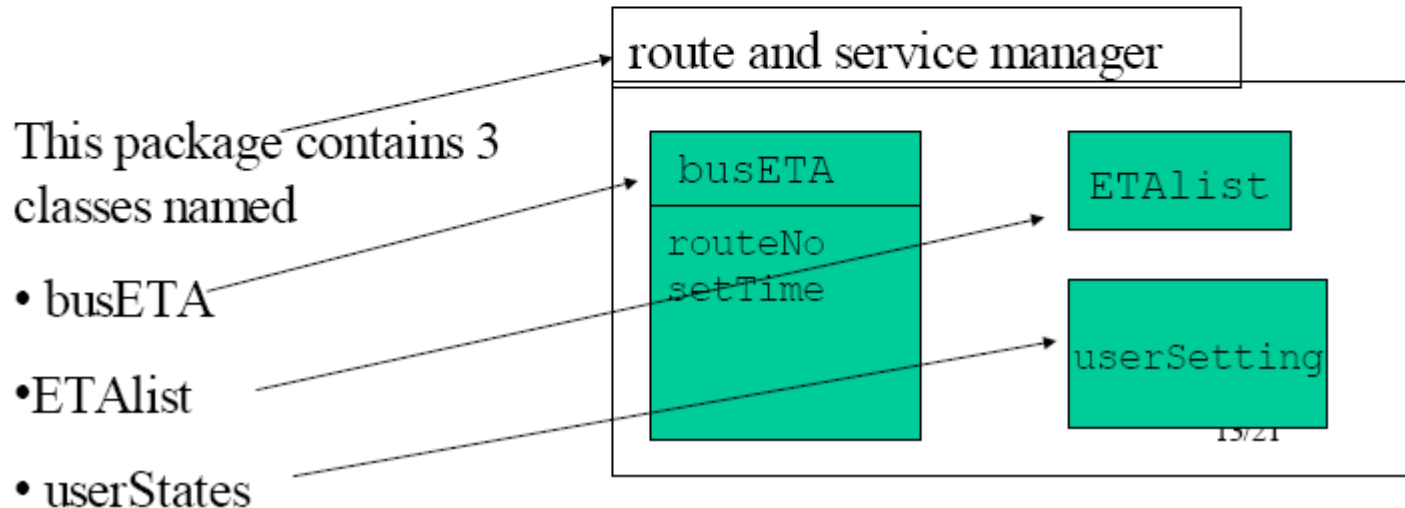


UML Notation ... and ... Typical Implementation



Modules, Packages & High Level Design

- what Parnas refers to as “abstract modules” and design standards call “modules” may be seen in OO languages as
 - “packages” - Java
 - “clusters” – Eiffel
 - packages – UML
- *or they may be groupings in the design document and in the file system, with no language structure*



- **Use patterns to**
 - **Promote communication**
 - **Streamline documentation**
 - **Increase developmental efficiency**
 - **Support software reuse**
 - **Provide design ideas**

The Origin of Patterns...

- **Patterns in Architecture**
- **Christopher Alexander: Notes on the Synthesis of Form**
- **Research on a Language of Patterns**
- **Led to the Patterns Movement in software design**

Patterns in Context

C (contextual)	C.E (Contextual-Explicit)	← C.I (Contextual-Implicit)
NC (non-contextual)	↑ N.C.E (Non-contextual-Explicit)	↖ ↗ N.C.I (Non-contextual-Implicit)
	Explicit	Implicit

The Context Dynamics Matrix [1]

- **A descriptive understanding of Software Architectures**
- **A look at existing software architectures**
- **Thinking about software architecture**