



Universität
Salzburg

Modularization & Software Architectures

Vorlesung Software Engineering II Sommersemester 2004 Universität Salzburg

O.Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Pree
© Copyright Wolfgang Pree, All Rights Reserved

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.



Context

- Motivation & basic terminology
- Essential concepts for architecture analysis
 - | Architectural styles
 - | Architecture description
 - | (Re)design strategies for (OO) architectures
- Architecture analysis
 - | SAAM (Software Architecture Analysis Method)
 - | ATAM (Architecture Tradeoff Analysis Method)
- Analysis for rearchitecting legacy systems



Motivation & basic terminology

Definition of *software architecture* (I)

Software architecture :=
software components +
the relationships among them

- top-level design of a system
- identification of core subsystems/abstractions

Definition of software architecture (II)

closely related terms/concepts:

- | domain-specific software architectures, frameworks, product lines (→ reusable architectures)
- | architectural styles
- | design patterns (some of the GoF patterns focus on architecture, eg, Facade, Mediator)
- | component-based software

5

© 2002, W. Pree

Benefits of architecture analysis

- **Communication among stakeholders** based on an explicit description of high-level abstractions of the system under development
- **Early design decisions**, influenced by driving quality attributes.
- **Transferable abstraction of a system**; can promote large-scale reuse.

6

© 2002, W. Pree

Essential concepts for architecture analysis

- | Architectural styles
- | Architecture description
- | (Re)design strategies for (OO) architectures

Architectural styles

Architecture description based on architectural styles *)

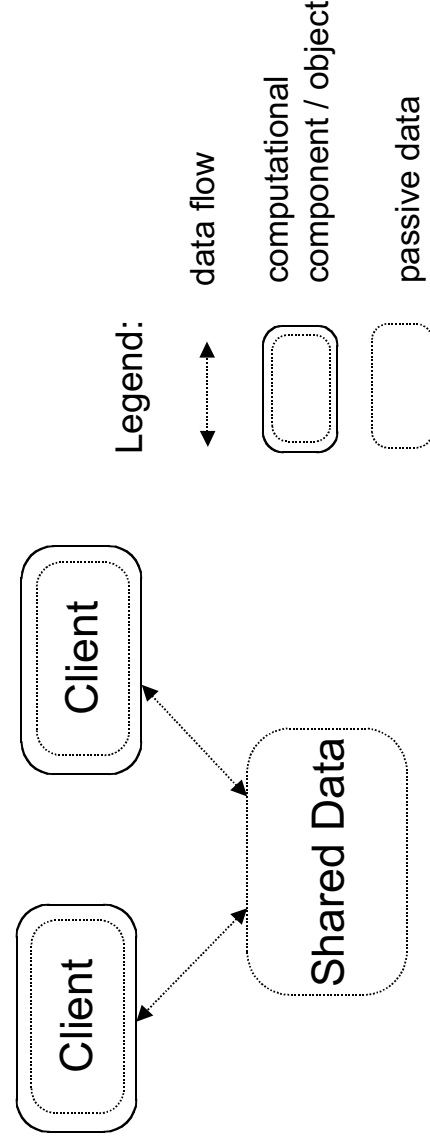
- | **Data-centered:**
 - » Virtual machine:
 - Interpreter
 - Rule-based
 - » Independent components:
 - Communicating processes
 - Event systems
 - implicit invocation
 - explicit invocation
- | **Data-flow:**
 - | Pipes & filters
 - | Batch/sequential
- | **Call-and-return:**
 - | Top down
 - | OO
 - | layered

*) The presentation is based on *Software Architecture in Practice* (Bass et al.; Addison-Wesley, 1998) and *Software Architecture: Perspectives on an Emerging Discipline* (Shaw, Gatlan; Prentice Hall, 1996)

© 2002, W. Pree

Sample architectural style: Data-centered (I)

Access to shared data represents the core characteristic of data-centered architectures. The data integrability forms the principal goal of such systems.



© 2002, W. Pree

Sample architectural style: Data-centered (II)

The means of communication between the components distinguishes the subtypes of the data-centered architectural style:

- **Repository: passive data** (see schematic representation of previous slide)
- **Blackboard: active data**
A blackboard sends notification to subscribers when relevant data change (→ Observer pattern)

11

© 2002, W. Pree

Sample architectural style: Data-centered (III)

- + clients are quite independent of each other
=> clients can be modified without affecting others
coupling between clients might increase performance but lessen this benefit
- + new clients can be easily added

No rigid separation of styles: When clients are independently executing processes: client/server architectural style

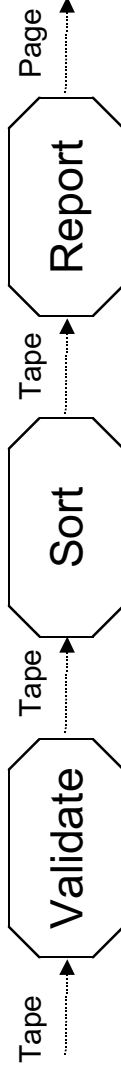
12

© 2002, W. Pree

Sample architectural style: Data-flow

The system consists of a **series of transformations on successive pieces of (input) data.**

Reuse and modifiability form the principal goals of such architectures.



- **Batch sequential**
- **Pipe-and-filter** (→ incremental)

Architecture description

Architectural styles are often insufficient to describe a system's architecture

Architectural styles do not clearly categorize architectures. Thus they do not suffice to describe architectures as a whole.

Consider the following sample cases:

- The layers in a layered architecture might be objects/ADTs.
- Commercial client/server systems with a CORBA-based infrastructure could be described as **layered object-based process systems**, i.e., a hybrid of three styles.

Choose an appropriate mix of various notations + informal description

As a consequence, one has to decide on the appropriate description, which will be a mix of the following principal options:

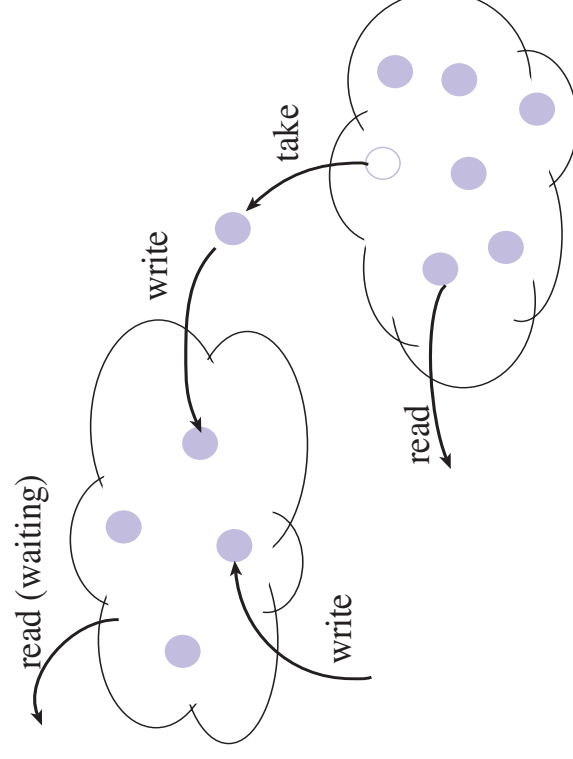
- schematic representation according to CMU/SEI
- 4+1 View Model of Architecture (Kruchten , 1995)
- UML:
 - subsystem/package-diagrams
 - class-diagrams
 - interaction-/object-diagrams
 - state-diagrams
- any schematic figures that help; informal text as glue
- source code fragments of coarse-grained components or component interfaces
- formal specifications, eg, with architecture description languages

Sample architectural description of JavaSpaces

JavaSpaces characteristics (from an architectural point of view):

- | data-centered, mainly a repository architectural style, sometimes black-board architectural style
- | main design goals
 - | **extensibility** through loose coupling of distributed processes and distributed Java components
 - | **simplicity** from a reuser's point of view

JavaSpaces architecture overview



Characteristics of JavaSpaces (JS)

- **high-level coordination tool** for gluing processes and components together in a distributed system **without message passing and remote method invocation**
- a space is a **shared, network-accessible repository for objects**
- instead of communicating directly, JS apps consist of processes that coordinate by exchanging objects through spaces

Sample JS use scenarios (I)

Example isn't another way to teach, it is the only way to teach (Albert Einstein)

(1) Space acting as “auction room”:

Sellers deposit for-sale items with descriptions and asking prices as objects into the space.

Buyers monitor the space for items that interest them. If an item interest them they put bid objects into the space.

Sellers monitor the space for bids.
etc.

Sample JS use scenarios (II)

(2) Compute-intensive jobs

A series of tasks—for example, rendering a frame in a computer animation represents a task—are written into a space.

Participating graphic workstations search the space for rendering tasks. Each one finding tasks to be done, removes it, accomplishes it, writes the result back into the space.

Interfaces of JS key abstractions and their usage

The rest of the architecture description could be a series of documented **source code** and **commented UML class** and **interaction diagrams** that illustrate the simplicity of reusing the JS architecture.

```
public class SampleMsg implements Entry { // empty JS interface
    ...
}

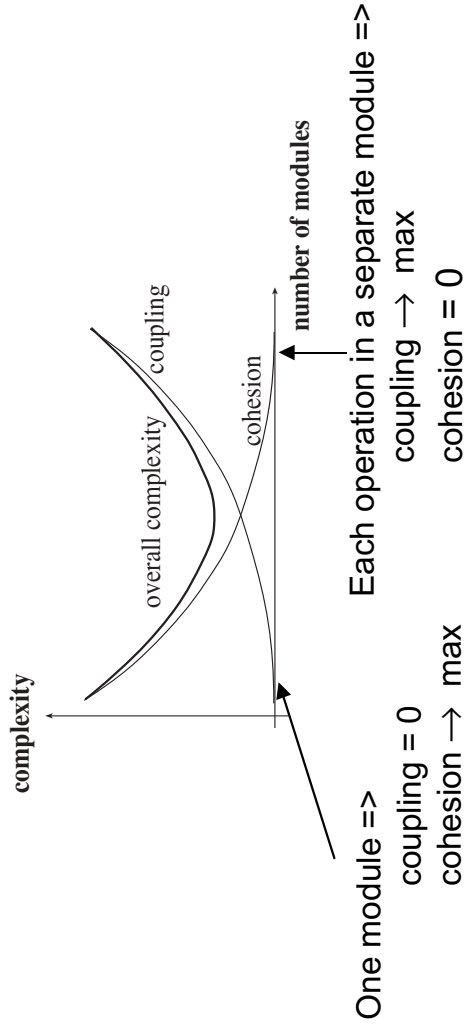
// putting an object into a space
SampleMsg msg= new SampleMsg();
JavaSpace space= SpaceAccessor.getSpace();
space.write(msg, ...); // other parameters omitted
...
```

Architecture analysis: The SAAM

Appropriate modularization in theory (I)

Balance between

- maximizing the cohesion within a component
- minimizing the coupling between components



When and Why To Analyze Architecture -1

- Analyzing for system qualities early in the life cycle allows for a comparison of architectural options.
- When building a system
 - Architecture is the earliest artifact where trade-offs are visible.
 - Analysis should be done when deciding on architecture.
 - The reality is that analysis is often done during damage control, later in the project.

When and Why To Analyze Architecture -2.

- When acquiring a system
 - Architectural analysis is useful if the system will have a long lifetime within organization.
 - Analysis provides a mechanism for understanding how the system will evolve.
 - Analysis can also provide insight into other visible qualities.

Qualities Are Too Vague for Analysis

- Is the following system modifiable?
 - Background color of the user interface is changed merely by modifying a resource file.
 - Dozens of components must be changed to accommodate a new data file format.
- A reasonable answer is
 - **yes** with respect to changing background color
 - **no** with respect to changing file format

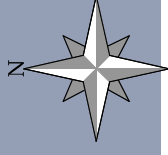
Qualities Are Too Vague for Analysis

- Qualities only have meaning within a context.
- SAAM specifies context through scenarios.

Scenarios

- A scenario is a brief description of a stakeholder's interaction with a system.
- When creating scenarios, it is important to consider all stakeholders, especially
 - end users
 - developers
 - maintainers
 - system administrators

Steps of a SAAM Evaluation



- Identify and assemble stakeholders
- Develop and prioritize scenarios
- Describe candidate architecture(s)
- Classify scenarios as direct or indirect
- Perform scenario evaluation
- Reveal scenario interactions
- Generate overall evaluation

Step 1: Identify and Assemble Stakeholders -1

Stakeholder	Interest
Customer	Schedule and budget; usefulness of system; meeting customers' (or market's) expectations
End user	Functionality, usability
Developer	Clarity and completeness of architecture; high cohesion and limited coupling of parts; clear interaction mechanisms
Maintainer	Maintainability; ability to locate places of change

Step 1: Identify and Assemble Stakeholders -2

Stakeholder	Interest
System administrator	Ease in finding sources of operational problems
Network administrator	Network performance, predictability
Integrator	Clarity and completeness of architecture; high cohesion and limited coupling of parts; clear interaction mechanisms

Step 1: Identify and Assemble Stakeholders -3.

Stakeholder	Interest
Tester	Integrated, consistent error-handling; limited component coupling; component cohesion;
high conceptual integrity	Architectural clarity, completeness; interaction mechanisms; simple tailoring mechanisms
Application builder (if product line architecture)	Interoperability
Representative of the domain	

33

© 2002, W. Pree

Step 2: Stakeholders Develop and Prioritize Scenarios

- Scenarios should be typical of the kinds of evolution that the system must support:
 - functionality
 - development activities
 - change activities
- Scenarios also can be chosen to give insight into the system structure.
- Scenarios should represent tasks relevant to all stakeholders.
- Rule of thumb: 10-15 prioritized scenarios

34

© 2002, W. Pree

Step 3: Describe Candidate Architectures

- It is frequently necessary to elicit appropriate architectural descriptions.
- Structures chosen to describe the architecture will depend on the type of qualities to be evaluated.
- Code and functional structures are primarily used to evaluate modification scenarios.

Step 4: Classify Scenarios

- There are two classes of scenarios.
 - Direct scenarios are those that can be executed by the system without modification.
 - Indirect scenarios are those that require modifications to the system.
- The classification depends upon both the scenario and the architecture.
- For indirect scenarios we gauge the order of difficulty of each change: e.g. a person-day, person-week, person-month, person-year.

Step 5: Perform Scenario Evaluation

- For each indirect scenario
 - ▮ identify the components, data connections, control connections, and interfaces that must be added, deleted, or modified
 - ▮ estimate the difficulty of modification
- Difficulty of modification is elicited from the architect and is based on the number of components to be modified and the effect of the modifications.
- A monolithic system will score well on this step, but not on next step.

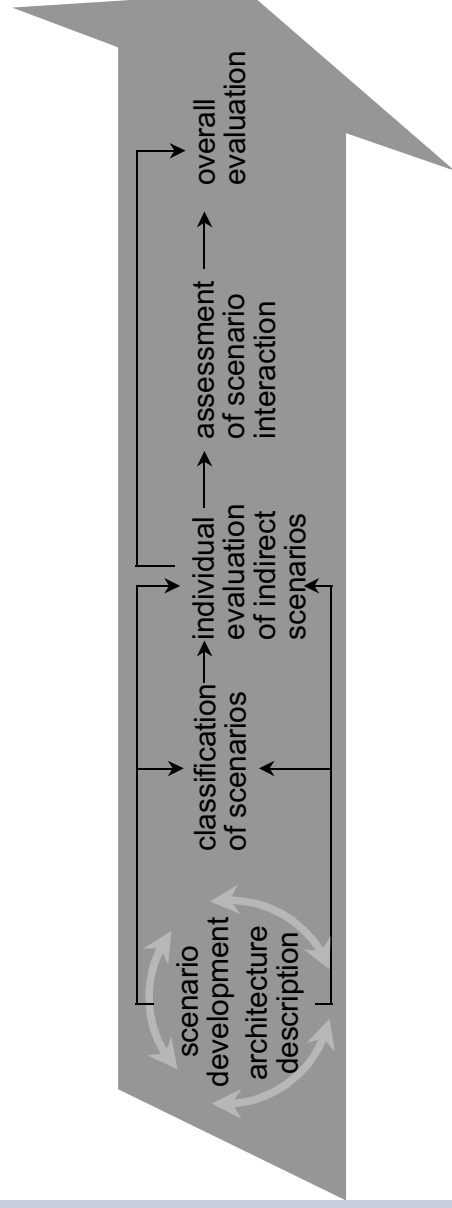
Step 6: Reveal Scenario Interactions

- When multiple indirect scenarios affect the same components, this could indicate a problem.
 - ▮ could be good, if scenarios are variants of each other
 - ▮ change background color to green
 - ▮ change background color to red
 - ▮ could be bad, indicating a potentially poor separation of concerns
 - ▮ change background color to red
 - ▮ port system to a different platform

Step 7: Generate Overall Evaluation

- Not all scenarios are equal.
- The organization must determine which scenarios are most important.
- Then the organization must decide as to whether the design is acceptable “as is” or if it must be modified.

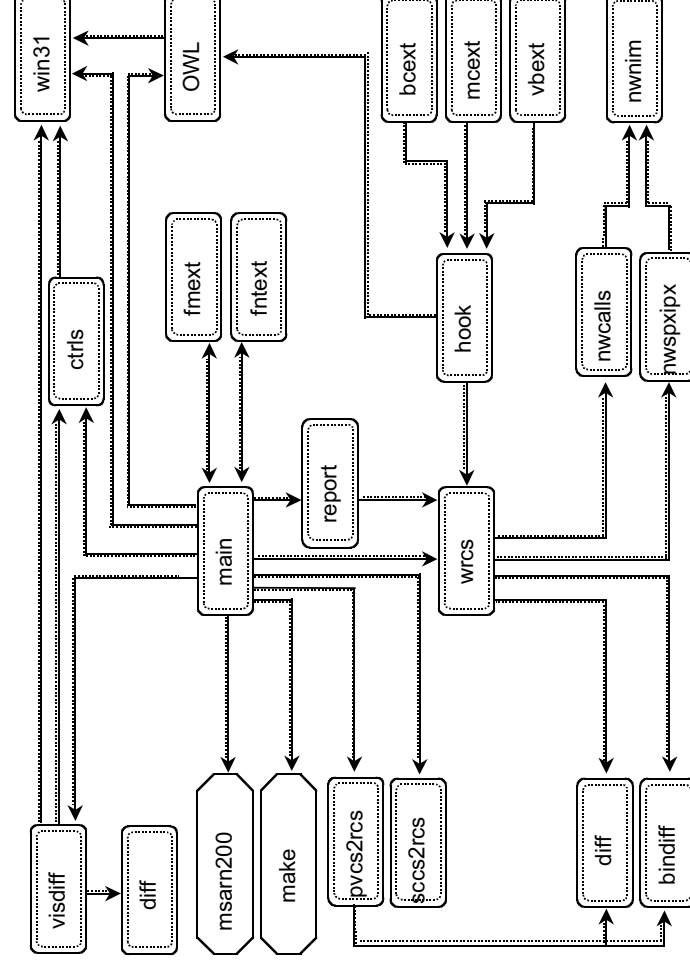
Interaction of SAAM Steps



Example: SAAM Applied to Revision Control System

- “WRCS” is a large, commercially-available revision control system.
- No documented system architecture existed prior to the evaluation.
- The purpose of the evaluation was to assess the impact of anticipated future changes.
- Three iterations were required to develop a satisfactory representation, alternating between
 - development of scenarios
 - representation of architecture

Architectural Representation of WRCS



Scenarios Used in WRCS

- User scenarios
 - ┆ compare binary file representations
 - ┆ configure the product's toolbar
- Maintainer
 - ┆ port to another operating system
 - ┆ make minor modifications to the user interface
- Administrator
 - ┆ change access permissions for a project
 - ┆ integrate with a new development environment

Scenario Classification

- User scenarios
 - ┆ compare binary file representations: *indirect*
 - ┆ configure the product's toolbar: *direct*
- Maintainer
 - ┆ port to another operating system: *indirect*
 - ┆ make minor modifications to the user interface: *indirect*
- Administrator
 - ┆ change access permissions for a project: *direct*
 - ┆ integrate with a new development environment: *indirect*

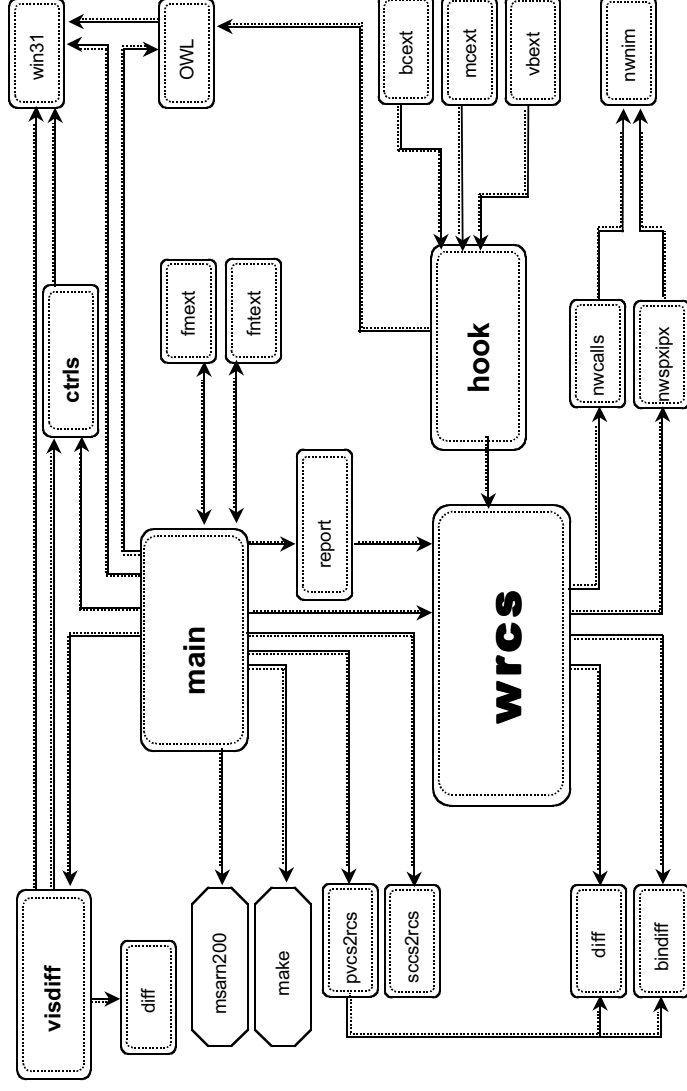
Scenario Interactions

- Each indirect scenario necessitated a change in some modules. This can be represented either tabularly or visually.
- The number of scenarios that affected each module can be shown with a table or graphically, with a fish-eye view.
- A fish-eye view uses size to represent areas of interest.

Scenario Interaction Table

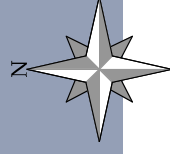
<u>Module</u>	<u>No. changes</u>
main	4
wrcs	7
diff	1
bindiff	1
pvc2rcs	1
sccs2rcs	1
nwcalls	1
nwspixpx	1
nwnlm	1
hook	4
report	1
visdiff	3
ctrls	2

Scenario Interaction Fish-Eye



47

© 2002, W. Pree



Lessons Learned from WRCs

- Granularity of architectural description
- Interpretation of scenario interactions

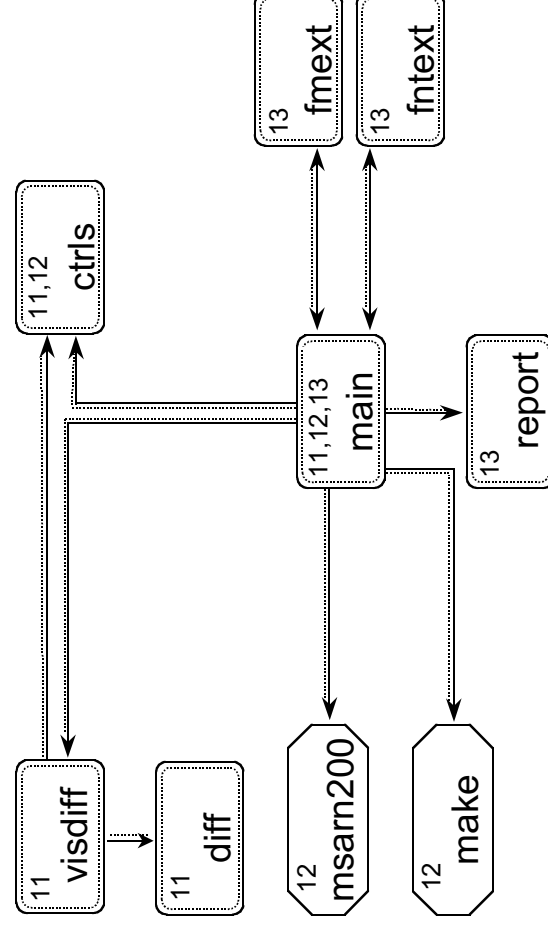
48

© 2002, W. Pree

Proper Granularity of Architectural Description

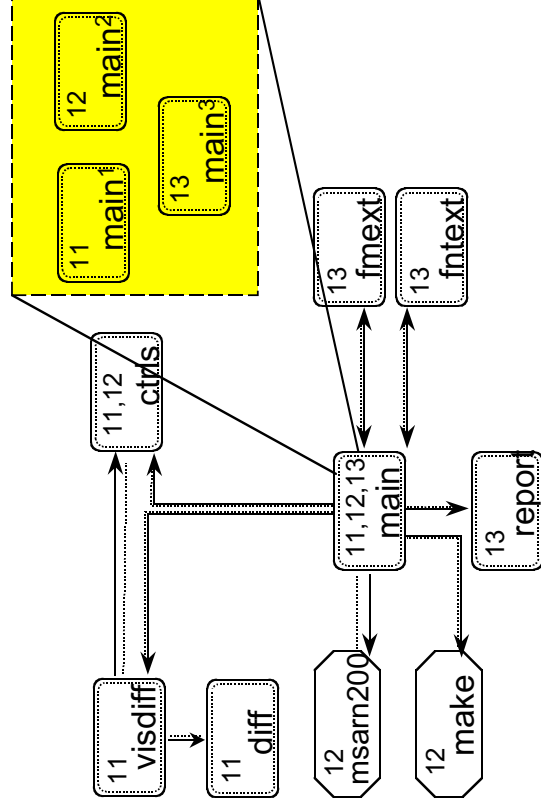
- The level of detail of architectural description is determined by the scenarios chosen.
- The next slide shows what an architect thought was an appropriate level of detail.
- Components are annotated with the numbers of indirect scenarios that affect them.

Original Representation of WRCS



The “main” Scenario Interactions

- Possibilities:
 - Scenarios are all of the same class.
 - Scenarios are of different classes and “main” cannot be subdivided.
 - Scenarios are of different classes, and “main” can be subdivided.



WRCS: What did we learn?

- We identified severe limitations in achieving the desired portability and modifiability. A major system redesign was recommended.
- The evaluation itself obtained mixed results.
 - Senior developers/managers found it important and useful.
 - Developers regarded this as just an academic exercise.
- SAAM allowed insight into capabilities and limitations that weren't easily achieved otherwise.
- This was accomplished with only scant knowledge of the internal workings of WRCS.

Lessons from SAAM -1

- Direct scenarios provide a
 - ▮ first-order differentiation mechanism for competing architectures
 - ▮ mechanism for eliciting and understanding structures of architectures (both static and dynamic)
- It is important to have stakeholders present at evaluation meetings.
 - ▮ Stakeholders find it to be educational.
 - ▮ Architectural evaluators may not have the experience to keep presenters “honest.”

Lessons from SAAM -2.

- SAAM and traditional architectural metrics
 - ▮ Coupling and cohesion metrics do not represent different patterns of use.
 - ▮ High scenario interaction shows low cohesion.
 - ▮ A scenario with widespread hits shows high coupling.
 - ▮ Both are tied to the context of use.
 - ▮ SAAM provides a means of sharpening the use of coupling and cohesion metrics.

Summary

- A SAAM evaluation produces
 - technical results: provides insight into system capabilities
 - social results
 - | forces some documentation of architecture
 - | acts as communication vehicle among stakeholders

Summary and conclusion

Where Have We Been?

Hopefully, by now, you are convinced that:

- it is a good idea to have a software architecture
- this architecture should be documented
- this architecture should be communicated to the system's stakeholders
- this architecture should be analyzed

Why Analyze an Architecture? - 1

To get insights into a system's qualities that could not be achieved via design inspections, code inspections,...

We can analyze to understand:

- how to rearchitect a system
- the eventual modifiability of a system
- where architectural tradeoffs occur

Why Analyze an Architecture? - 2

- Architecture analysis is easy to do.
- Architecture analysis is (relatively) cheap.
- It can be done early, when discovered problems are still fixable at a reasonable cost.
- It improves the quality of an architecture as well as the stakeholders' understanding and confidence.
- It is a way of enhancing team communication.
- It is an effective risk mitigation technique.

Appendix A

Overview of architectural styles

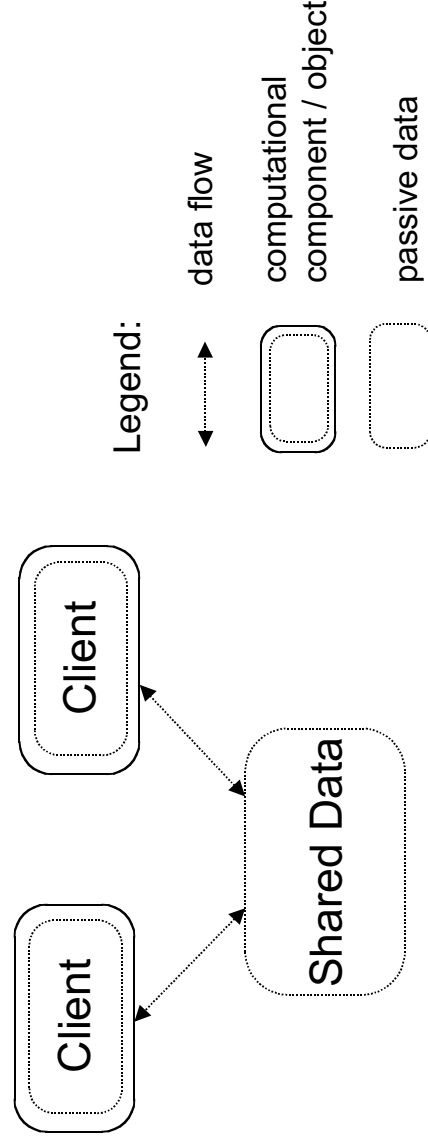
Overview of architectural styles^{*)}

- Data-centered:
 - | Repository
 - | Blackboard
- Data-flow:
 - | Pipes & filters
 - | Batch/sequential
- Call-and-return:
 - | Top down
 - | OO
 - | layered
- » Virtual machine:
 - Interpreter
 - Rule-based
- » Independent components:
 - Communicating processes
 - Event systems
 - implicit invocation
 - explicit invocation

*) The presentation is based on *Software Architecture in Practice* (Bass et al.; Addison-Wesley, 1998) and *Software Architecture: Perspectives on an Emerging Discipline* (Shaw, Garlan; Prentice Hall, 1996)

Data-centered (I)

Access to shared data represents the core characteristic of data-centered architectures. The data integrability forms the principal goal of such systems.



Data-centered (II)

The means of communication between the components distinguishes the subtypes of the data-centered architectural style:

- **Repository: passive data** (see schematic representation of previous slide)
- **Blackboard: active data**
A blackboard sends notification to subscribers when relevant data change (→ Observer pattern)

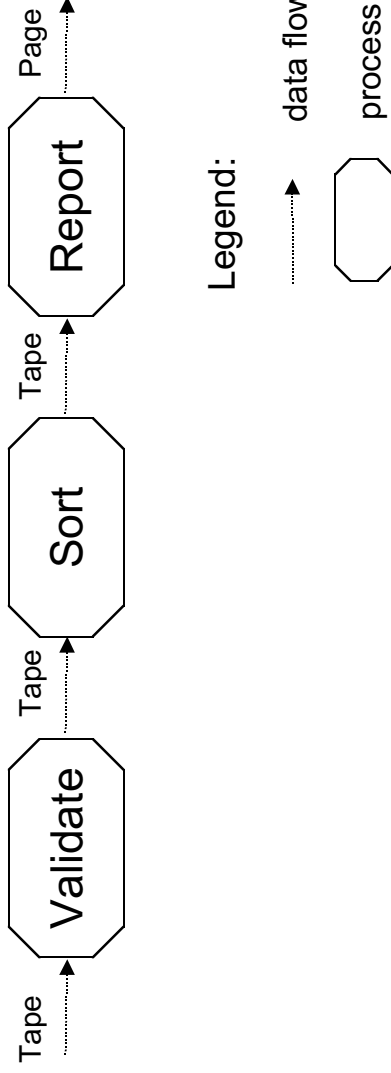
Data-centered (III)

- + clients are quite independent of each other
=> clients can be modified without affecting others
coupling between clients might increase performance but lessen this benefit
- + new clients can be easily added

No rigid separation of styles: When clients are independently executing processes: client/server architectural style

Data-flow

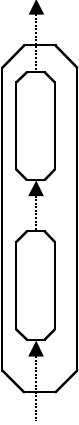
The system consists of a **series of transformations on successive pieces of (input) data**. Reuse and modifiability form the principal goals of such architectures.



Data-flow substyles

- **Batch sequential** (→ sample on previous slide)
 - components (= processing steps) are independent programs
 - **each step runs to completion before the next step starts**, i.e., each batch of data is transmitted as a whole between steps
- **Pipe-and-filter** (→ UNIX pipes & filters)
 - **incremental transformation of data** based on streams
 - filters are stream transducers and use little contextual information and retain no state information between instantiations
 - pipes are stateless and just move data between filters

Pros and cons of pipes-and-filters

- + no complex component interactions to manage
 - + filters are black boxes
 - + pipes and filters can be hierarchically composed
- 
- batch mentality => hardly suitable for interactive applications
 - filter ordering can be difficult; filters cannot interact cooperatively to solve a problem
 - performance is often poor
 - parsing/unparsing overhead due to lowest common denominator data representation
 - filters which require all input for output production have to create unlimited buffers

67

© 2002, W. Pree

Virtual machine (I)

Virtual machines **simulate some functionality that is not native to the hardware/software on which it is implemented**. This supports achieving the quality attribute of **portability**.

Examples:

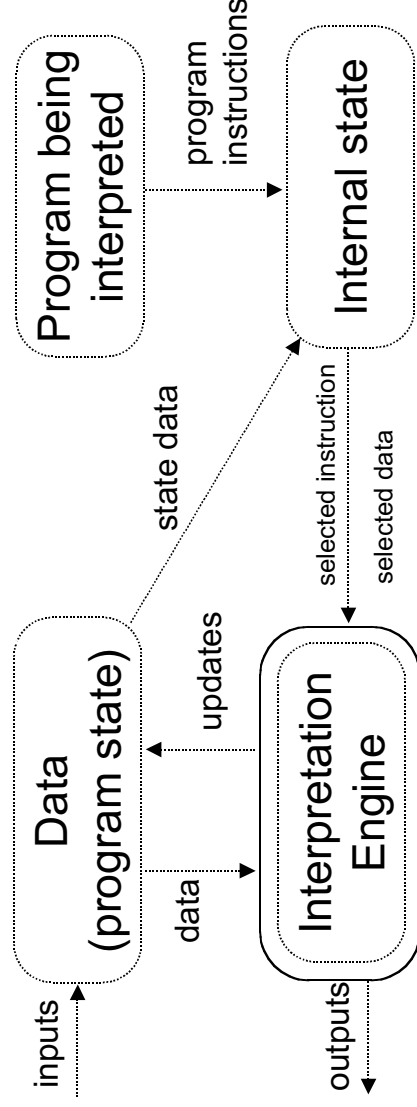
- interpreters
- command language processors
- rule-based systems

68

© 2002, W. Pree

Virtual machine (II)

Schematic representation:



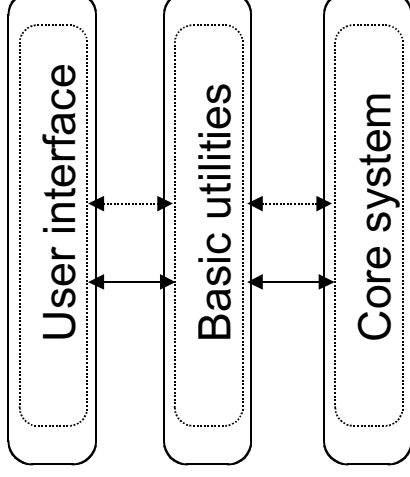
Call-and-return

Call-and-return architectures rely on the well-known abstraction of procedures/functions/methods. Shaw and Garlan discern between the following substyles:

- main-program-and-subroutine style
- remote-procedure-call systems also belong to this category but are decomposed in parts that live on computers connected via a network
- object-oriented or abstract-data-type style
- layered style

Layered style

Components belong to layers. In pure layered systems **each level should communicate only with its immediate neighbors.**

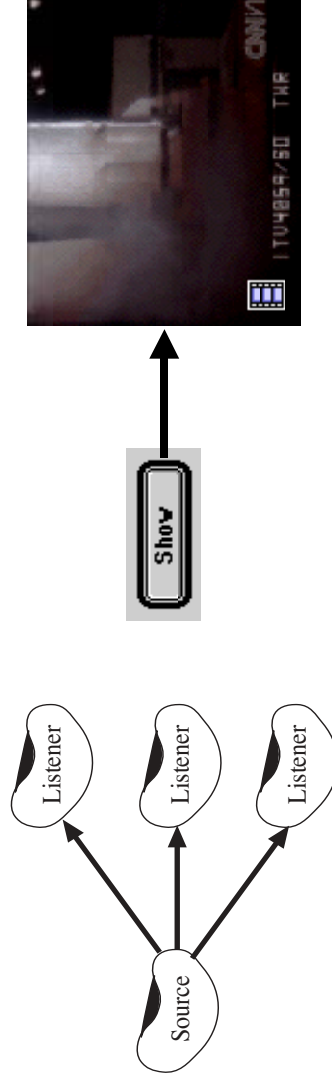


Each successive layer is built on its predecessor, hiding the lower layer and providing some services that the upper layers make use of. Upper layers often form virtual machines.

Event systems

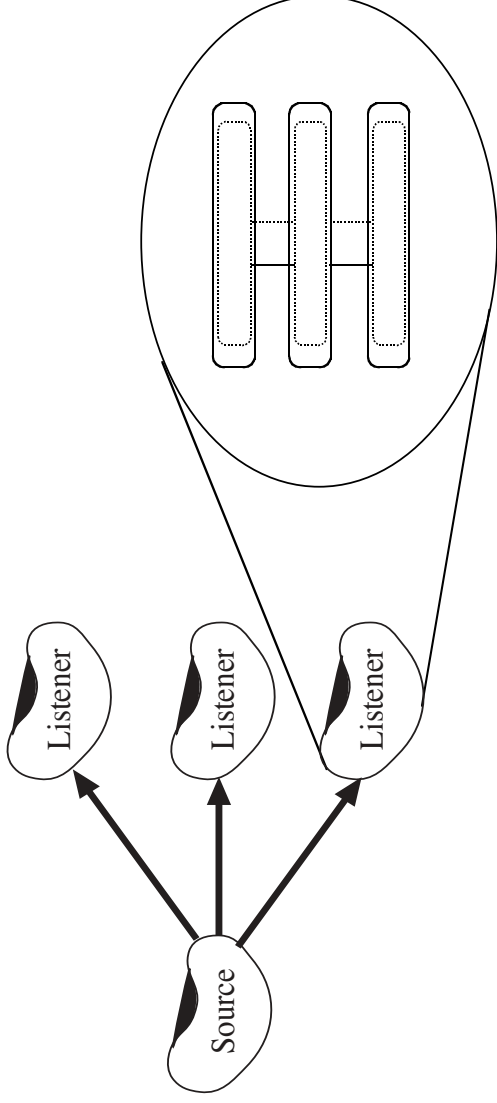
Publish/subscribe (observer) pattern: Components can register an interest in notifications.

Example: coupling between JavaBeans



Heterogeneous styles (I)

Example: event system + layered style



Heterogeneous styles (II)

In general, the presented architectural styles do not clearly categorize architectures. Styles exist as cognitive aids and communication cues.

- The data-centered style, composed out of thread-independent clients is like an independent component architecture.
- The layers in a layered architecture might be objects/ADTs.
- The components in a pipe-and-filter architecture are usually independently operating processes and thus also correspond to an independent component architecture.
- Commercial client/server systems with a CORBA-based infrastructure could be described as layered object-based process systems, i.e., a hybrid of three styles.

Appendix B—Bibliography (I)

- Bass L., Clements P., Kazman R. (1998) *Software Architecture in Practice*, Addison-Wesley
- Fayad M., Schmidt D., Johnson R. (1999) *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, Wiley
- Fayad M., Schmidt D., Johnson R. (1999) *Implementing Application Frameworks: Object-Oriented Frameworks at Work*, Wiley
- Fayad M., Schmidt D., Johnson R. (1999) *Domain-Specific Application Frameworks: Manufacturing, Networking, Distributed Systems, and Software Development*, Wiley
- Gabriel R.P. (1996). *Patterns of Software—Tales from the Software Community*. New York: Oxford University Press
- Gamma E., Helm R., Johnson R. and Vlissides J. (1995) *Design Patterns—Elements of Reusable OO Software*. Reading, MA: Addison-Wesley (also available as CD)
- Prece W. (1995) *Design Patterns for Object-Oriented Software Development*. Reading, Massachusetts: Addison-Wesley/ACM Press
- Szyperski C. (1998) *Component Software—Beyond Object-Oriented Programming*, Addison-Wesley.
- Shaw M., Garlan D. (1996) *Software Architecture—Perspectives on an Emerging Discipline*. Prentice-Hall
- comprehensive architecture descriptions of real-world software systems:
- Freeman E, Hupfer S, Arnold K (1999) *JavaSpaces—Principles, Patterns, and Practice*, Addison-Wesley
- Wirth N, Gutknecht J. (1993) *Project Oberon—The Design of an Operating System and Compiler*, Addison-Wesley

Appendix B—Bibliography (II)

Bibliography on Software Architecture Analysis (<http://www.fit.ac.jp/~zhao/pub/isa.html>), maintained by Jianjun Zhao

This is the bibliography on software architecture analysis, with special emphasis on architectural-level understanding, testing, debugging, reverse engineering, re-engineering, maintenance, and complexity measurement.

- R. Balzer, "Instrumenting, Monitoring and Debugging Software Architectures."
- P. Bengsson and J. Bosch, "Scenario-Based Software Architecture Reengineering," *Proc. 5th International Conference on Software Reuse (ICSR5)*, pp.308-317, IEEE Computer Society Press, Victoria, B.C. Canada, June 1998.
- P. Bengsson, "Towards Maintainability Metrics on Software Architecture: An Adaptation of Object-Oriented Metrics," *First Nordic Workshop on Software Architecture (NOISA'98)*, Romeby, August 1998.
- P. Bengsson and J. Bosch, "Architecture Level Prediction of Software Maintenance," *Proc. 3rd European Conference on Maintenance and Reengineering (CSMR99)*, Amsterdam, The Netherlands, March 1999.
- L. Bass, P. Clements, and R. Kazman, "Software Architecture in Practice," Published by Addison-Wesley in the SEI Series, 1998.
- A. Bertolino, P. Inverardi, H. Muccini, and A. Rosetti, "An Approach to Integration Testing Based on Architectural Descriptions," *Proc. Third IEEE International Conference on Engineering of Complex Computer Systems (ICECCS97)*, pp:77-84, Como, Italy, September 1997.
- G. Cantora, A. De Lucia, G. di Luccia, and A. Fasolino, "Recovering the Architectural Design for Software Comprehension," *Proc. IEEE Third Workshop on Program Comprehension*, Washington, DC, November 1994.
- S. J. Carriere and R. Kazman, "The Perils of Reconstructing Architectures," *Proc. 3rd International Software Architecture Workshop (ISAW3)*, pp.13-16, ACM SIGSOFT, Orlando, Florida, USA, November 1998.
- S. J. Carriere, R. Kazman, and S. Woods, "Assessing and Maintaining Architectural Quality," *Proc. 3rd European Conference on Maintenance and Reengineering (CSMR99)*, Amsterdam, The Netherlands, March 1999.
- P. Clements, R. Krut, E. Morris, and K. Wallnau, "The Godfly: An Approach to Architectural-Level System Comprehension," *Proc. 4th International Workshop on Program Comprehension (IWPC96)*, IEEE Computer Society Press, pp.178-186, 1996.
- J. F. Girard and R. Koschke, "Finding Components in a Hierarchy of Modules: A Step towards Architectural Understanding," *Proc. International Conference on Software Maintenance (ICSM97)*, IEEE Computer Society Press, pp.58-65, Bari, Italy, October 1997.

Appendix B—Bibliography (III)

- G. Y. Guo, J. M. Allee, and R. Kazman, "A Software Architecture Reconstruction Method," *Proc. First Working IFIP Conference on Software Architecture (IWCSA1)*, San Antonio, TX, USA, February 1999.
- D. Harris, H. Reubenstein, and A. S. Yeh, "Reverse Engineering to the Architectural Level," *Proc. International Conference on Software Engineering (ICSE95)*, pp.186-195, IEEE Computer Society Press, July 1995.
- S. Henry and D. Kafura, "Software Structure Metrics Based on Information Flow," *IEEE Transactions on Software Engineering*, 7(5), September 1981.
- P. Invernardi and A. L. Wolf, "Formal Specification and Analysis of Software Architectures using the Chemical Abstract Machine Model," *IEEE Transactions on Software Engineering*, 21(4):373-386, April 1995.
- R. Kazman, "Tool Support for Architectural Analysis and Design," *Proc. 2nd Software Architecture Workshop (SAW2)*, pp.94-97, San Francisco, CA, October 1996.
- R. Kazman, G. Abowd, L. Bass, and P. Clements, "Scenario-Based Analysis of Software Architecture," *IEEE Software*, pp.47-55, November 1996.
- R. Kazman and M. Burth, "Assessing Architectural Complexity," *Proc. 2nd EuroMicro Working Conference on Software Maintenance and Reengineering (CSMR98)*, pp.104-112, IEEE Computer Society Press, Florence, Italy, March 1998.
- R. Kazman and S. J. Carriere, "View Extraction and View Fusion in Architectural Understanding," *Proc. 5th International Conference on Software Reuse (ICSR5)*, pp.290-299, IEEE Computer Society Press, Victoria, B.C., Canada, June 1998.
- R. Kazman, M. Klein, M. Barbacci, H. Lipson, T. Longstaff, and S. J. Carriere, "The Architecture Tradeoff Analysis Method," *Proc. Fourth IEEE International Conference on Engineering of Complex Computer Systems (ICECS98)*, pp.68-78, Monterey, USA, August 1998.
- R. Kazman, S. Woods, and S. J. Carriere, "Requirements for Integrating Software Architecture and Reengineering Models: CORUM II", *Proc. 5th Working Conference on Reverse Engineering (WCRE98)*, pp.154-163, Honolulu, HI, October 1998.
- R. Kazman and S. J. Carriere, "Playing Detective: Reconstructing Software Architecture from Available Evidence", *Journal of Automated Software Engineering*, April 1999. (to appear)
- T. H. Kim, Y. T. Song, L. Chung, and D. Hynh, "Software Architecture Analysis Using Dynamic Slicing", *Proc. ACM/IAO/M CS'99*, August 1999.
- T. H. Kim, Y. T. Song, L. Chung, and D. Hynh, "Dynamic Software Architecture Slicing", *Proc. 23th IEEE Annual International Computer Software and Applications Conference (COMPSAC99)*, October 1999. (to appear)
- J. Kramer and J. Magee, "Analysing Dynamic Change in Software Architectures: A Case Study", *Proc. IEEE 4th International Conference on Configurable Distributed Systems (CDS 98)*, pp.91-100, Annapolis, May 1998.
- R.L. Krikhaar, R.P. de Jong, J.P. Medema, and L.M.G. Feijs, "Architecture Comprehension Tools for a PBX System", *Proc. 3rd European Conference on Maintenance and Reengineering (CSMR99)*, Amsterdam, The Netherlands, March 1999.
- D.C. Luckham, J.J. Kenney, L.M. Augustin, J. Vera, D. Bryan, and W. Mann, "Specification and Analysis of System Architecture Using Rapide," *IEEE Transactions on Software Engineering*, Vol.21, No.4, pp.336-355, April 1995.
- C. H. Lung, S. Bot, K. Kalachelvan, and R. Kazman, "An Approach to Software Architecture Analysis for Evolution and Reusability," *Proc. of CASCON 97*, November 1997.

Appendix B—Bibliography (IV)

- C. H. Lung and K. Kalachelvan, "A Quantitative Approach to Software Architecture Sensitivity Analysis", *Proc. of the 10th International Conference on Software Engineering and Knowledge Engineering*, pp. 185-192, June 1998.
- C. H. Lung, "Software Architecture Recovery and Restructuring through Clustering Techniques," *Proc. 3rd International Software Architecture Workshop (ISAW3)*, pp.101-104, ACM SIGSOFT, Orlando, Florida, USA, November 1998.
- J. Magee, J. Kramer, and D. Giannakopoulou, "Analysing the Behaviour of Distributed Software Architectures: a Case Study", *Proc. 5th IEEE Workshop on Future Trends in Distributed Computing Systems (FTDCS97)*, pp.240-247, Tunisia, October 1997.
- J. Magee, J. Kramer, and D. Giannakopoulou, "Software Architecture Directed Behavior Analysis," *Proc. Ninth International Workshop on Software Specification and Design (WSSD9)*, pp.144-146, IEEE Computer Society Press, Ise-Shima, Japan, April 1998.
- J. Magee, J. Kramer and D. Giannakopoulou, "Behaviour Analysis of Software Architectures" *Proc. First Working IFIP Conference on Software Architecture (IWCSA1)*, San Antonio, Texas, February 1999.
- T. J. McCabe and C. W. Butler, "Design Complexity Measurement and Testing," *Communications of ACM*, Vol.32, No.12, pp.1415-1425, 1989.
- G. Naumovich, G.S. Avrunin, L.A. Clarke, and L.J. Osterweil, "Applying Static Analysis to Software Architectures," *Proc. the Sixth European Software Engineering Conference Held Jointly with the 5th ACM SIGSOFT Symposium on Foundations of Software Engineering*, pp.77-93, Lecture Notes in Computer Science, Vol.1301, Springer-Verlag, 1997.
- D. E. Perry and A. L. Wolf, "Foundations for the Study of Software Architecture", *ACM SIGSOFT Software Engineering Notes*, pp.40-52, Vol.17, No.4, October 1992.
- J. Peterson and M. Sulzmann, "Analysis of Architectures using Constraint-Based Types," *Proc. First Working IFIP Conference on Software Architecture (IWCSA1)*, San Antonio, TX, USA, February 1999.
- M. D. Rice and S. B. Seidman, "An Approach to Architectural Analysis and Testing," *Proc. 3rd International Software Architecture Workshop (ISAW3)*, pp.121-123, ACM SIGSOFT, Orlando, Florida, USA, November 1998.
- D.J. Richardson and A. L. Wolf, "Software Testing at the Architectural Level," *Proc. 2nd International Software Architecture Workshop (ISAW2)*, pp.68-71, San Francisco, California, October 1996.
- M. Shaw and D. Garlan, "Software Architecture: Perspectives on an Emerging Discipline," Prentice Hall, 1996.
- J.A. Stafford, D.J. Richardson, and A. L. Wolf, "Chaining: A Software Architecture Dependence Analysis Technique," Technical Report CU-CS-845-97, University of Colorado, September 1997.
- J.A. Stafford, D.J. Richardson, and A. L. Wolf, "Aladdin: A Tool for Architecture-level Dependence Analysis of Software Systems," University of Colorado Technical Report, CU-CS-858-98, 1998.
- J.A. Stafford and A. L. Wolf, "Architectural-level Dependence Analysis in Support of Software Maintenance," *Proc. 3rd International Software Architecture Workshop (ISAW3)*, pp.129-132, ACM SIGSOFT, Orlando, Florida, USA, November 1998.
- W. Tracz, "Testing and Analysis of Software Architectures," *Proc. ACM International Symposium on Software Testing and Analysis (ISSTA96)*, S.Diego, USA, January 1996.
- V. Tzepos and R.C. Holt, "The Orphan Adoption problem in Architecture Maintenance," *Proc. Working Conference on Reverse Engineering (WCRE97)*, Amsterdam, The Netherlands, October 1997.

Appendix B—Bibliography (V)

- C. Williams, "Software Architecture: Implications for Computer Science Research," *Proc. First Working FIP Conference on Software Architecture (WCSA)*, San Antonio, TX, USA, February 1999.
- J. Zhao, "Software Architecture Slicing," *Proc. 14th Conference of Japan Society for Software Science and Technology (JSSST'97)*, pp.49-52, Ishikawa, Japan, September 1997.
- J. Zhao, "Using Dependence Analysis to Support Software Architecture Understanding," in M. Li (Ed.), "New Technologies on Computer Software," pp.135-142, International Academic Publishers, September 1997.
- J. Zhao, "Applying Slicing Technique to Software Architectures," *Proc. Fourth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'98)*, pp.87-98, August 1998.
- J. Zhao, "On Assessing the Complexity of Software Architectures," *Proc. 3rd International Software Architecture Workshop (ISAW3)*, pp.163-166, ACM SIGSOFT, Orlando, Florida, USA, November 1998.
- J. Zhao, "Extracting Reusable Software Architectures: A Slicing-Based Approach," *Proc. ESEC/FSE-99 Workshop on Object-Oriented Reengineering*, Toulouse, France, September 1999. (to appear)

Other Links on Software Architecture

Bibliographies:

- Ric Holt's Annotated Bibliography on Software Architecture <http://plg.uwaterloo.ca/~holt/cs/746/98biblio.html>
- Rick Kazman's Software Architecture Bibliography <http://www.cgi.uwaterloo.ca/~rnkazman/SA-bib.html>
- Kamran Sartipi's Software Architecture Bibliography <http://se.math.uwaterloo.ca:80/~ksartipi/papers/sa-bib.ps>
- SEI Bibliography on Software Architecture <http://www.sei.cmu.edu/architecture/bibpart1.html>

Others:

- Dewayne Perry's Web Page on Software Architecture <http://www.bellhabs.com/user/dep/work/swa/>
- Software Architecture Technology Guide <http://www-ast.tds-gn.lmco.com/arch/guide.html>
- On-line Proceedings of the International Workshop on the Role of Software Architecture in Testing and Analysis (ROSATEA) <http://www.ics.uci.edu/~djfrosata/>

Maintained by Jianjun Zhao (zhao@cs.ftr.ac.jp)