

# Panel Discussion – What is Software Architecture?

*At TOOLS Europe 2000, June 8 – 2pm.*

**Audience briefing notes.**

Mark Collins-Cope

Ratio Group Ltd.  
17/19 The Broadway  
Ealing W5 3NH

Email: [info@ratio.co.uk](mailto:info@ratio.co.uk)  
Web: [www.ratio.co.uk](http://www.ratio.co.uk)



*We Know the Object*

## Table of Contents

|           |   |           |
|-----------|---|-----------|
| <b>1.</b> | <b>INTRODUCTION.....</b>  | <b>3</b>  |
| 1.1.      | BACKGROUND .....  | 3         |
| 1.2.      | OBJECTIVES OF DISCUSSION.....                                     | 3         |
| 1.3.      | WHAT IS SOFTWARE ARCHTECTURE? .....                               | 3         |
| <b>2.</b> | <b>THE PANEL .....</b>  | <b>6</b>  |
| 2.1.      | NIGEL BARNES – ANDERSEN CONSULTING – PANELLIST .....              | 6         |
| 2.1.1.    | <i>Brief Bio</i> .....  | 6         |
| 2.1.2.    | <i>Position Statement</i> .....                                   | 6         |
| 2.2.      | ALAN CAMERON WILLS - TRIREME INTERNATIONAL LTD. – PANELLIST ..... | 8         |
| 2.2.1.    | <i>Brief Bio</i> .....  | 8         |
| 2.2.2.    | <i>Position Statement</i> .....                                   | 8         |
| 2.3.      | MARK COLLINS-COPE – RATIO GROUP LTD. – PANEL CHAIR .....          | 9         |
| 2.4.      | JOHN DANIELS – SYNTROPY LTD. – PANELLIST .....                    | 10        |
| 2.4.1.    | <i>Brief Bio</i> .....  | 10        |
| 2.4.2.    | <i>Position Statement</i> .....                                   | 10        |
| 2.5.      | ALAN O’CALLAGHAN – DEMONTFORD UNIVERSITY – PANELLIST.....         | 12        |
| 2.5.1.    | <i>Brief Bio</i> .....  | 12        |
| 2.5.2.    | <i>Position Statement</i> .....                                   | 12        |
| 2.6.      | WOLFGANG PREE – UNIVERSITY OF CONSTANCE, GERMANY – PANELIST ..... | 14        |
| 2.6.1.    | <i>Brief Bio</i> .....  | 14        |
| 2.6.2.    | <i>Position Statement</i> .....                                   | 14        |
| <b>3.</b> | <b>THE DISCUSSION .....</b>                                       | <b>16</b> |



## 1. Introduction

### 1.1. Background

Though much talked about, there appears to be no clear definition or agreement of what constitutes software architecture. This is apparent from many newsgroup discussions on the subject, from the position statements submitted by panellists, and from the material extracted from the CMU SEI website below.

### 1.2. Objectives of discussion

The objective of the panel discussion at TOOLS 2000 is to probe this issue with a view to:

- (a) seeing if we can come up with a consensus on what should constitute a software architecture (which is perhaps unlikely in a one hour discussion), or more likely,
- (b) to decide whether the differences in opinion that exist reflect a genuine difference of emphasis on what is important in software development, or alternatively merely indicate a difference in naming conventions – i.e. are the issues discussed under the title of software architecture all equally important, but just given different names?

### 1.3. What is software architecture?

By way of background information, here are some of the definitions and associated discussions of software architecture that can be found on the CMU software engineering institute website. Panellist position statements can be found later in this paper.

- **Booch, Rumbaugh, and Jacobson, 1999**

An architecture is the set of significant decisions about the organisation of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behaviour as specified in the collaborations among those elements, the composition of these structural and behavioural elements into progressively larger subsystems, and the architectural style that guides this organisation---these elements and their interfaces, their collaborations, and their composition (The UML Modelling Language User Guide, Addison-Wesley, 1999).

- **Garlan and Shaw, 1993**

Mary Shaw and David Garlan suggest that software architecture is a level of design concerned with issues... “beyond the algorithms and data structures of the computation; designing and specifying the overall system structure emerges as a new kind of problem. Structural issues include gross organisation and global control structure; protocols for communication, synchronisation, and data access; assignment of functionality to design elements; physical



distribution; composition of design elements; scaling and performance; and selection among design alternatives.”

· **Bass, et al., 1994**

Writing about a method to evaluate architectures with respect to the quality attributes they instil in a system, Bass and his colleagues write that... “the architectural design of a system can be described from (at least) three perspectives -- functional partitioning of its domain of interest, its structure, and the allocation of domain function to that structure.”

· **Hayes-Roth, 1994**

Writing for the ARPA Domain-Specific Software Architecture (DSSA) program, Hayes-Roth says that software architecture is ... an abstract system specification consisting primarily of functional components described in terms of their behaviours and interfaces and component-component interconnections.

· **Garlan and Perry, 1995**

David Garlan and Dewayne Perry have adopted the following definition for their guest editorial to the April 1995 IEEE Transactions on Software Engineering devoted to software architecture:

The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time.

· **Boehm, et al., 1995**

Barry Boehm and his students at the USC Centre for Software Engineering write that: A software system architecture comprises: A collection of software and system components, connections, and constraints. A collection of system stakeholders' need statements. A rationale which demonstrates that the components, connections, and constraints define a system that, if implemented, would satisfy the collection of system stakeholders' need statements.

· **Soni, Nord, and Hofmeister, 1995**

Soni, Nord, and Hofmeister of Siemens Corporate Research write that, based on structures found to be prevalent and influential in the development environment of industrial projects they studied, software architecture has at least four distinct incarnations: Within each category, the structures describe the system from a different perspective:

- The conceptual architecture describes the system in terms of its major design elements and the relationships among them.
- The module interconnection architecture encompasses two orthogonal structures: functional decomposition and layers.
- The execution architecture describes the dynamic structure of a system.



- The code architecture describes how the source code, binaries, and libraries are organised in the development environment.

*[Finally, the website concludes...]*

These views do not preclude each other, nor do they really represent a fundamental conflict about what software architecture is. Instead, they represent a spectrum in the software architecture research community about the emphasis that should be placed on architecture -- its constituent parts, the whole entity, the way it behaves once built, or the building of it. Taken together, they form a consensus view of software architecture[!]

## 2. The Panel

This section gives introductory position statements and a brief biography of each the panellists, as well as a brief biography of the panel chairman.

### 2.1. Nigel Barnes – Andersen Consulting – Panellist

#### 2.1.1. Brief Bio

Nigel Barnes is an Associate Partner in Andersen Consulting' Financial Services Solution Centre. He has extensive experience of large web, client-server and distributed component based architectures and has been involved in many of AC's largest development projects. Most recently he has been the lead architect for a >20 000 manday project to design and develop the next generation of settlement services for a major European financial institution. Previously he has worked in Utilities, Defence and Telecommunications. He is technology leader for AC's UK dot com launch centre initiative. He has an MEng in Electronic Systems Engineering. This is his first time on such an august panel.

#### 2.1.2. Position Statement

Architecture has been defined as the art or science of building - it is both about designing something and then constructing something. Well architected solutions are based on sound theoretical basis but must also be practical to produce within the overall guidelines of cost. It must be possible to construct the solution with the tools and processes defined. Large systems building projects are only possible if a solid blueprint for the system can be established early in the lifecycle and is maintained and enforced during development.

In such projects the architecture team typically must:

- develop and maintain the appropriate high level models of system (e.g. the Component Model, the Logical Data Model).
- develop and enforce the standards which will ensure the overall consistency of the solution (e.g. style of user interface)
- ensure the overall technical quality targets (such as performance and operability) are achievable
- select the platforms on which applications will be developed (e.g. hardware, database, common base classes, frameworks)
- manage change to the baseline during the development of the solution.

Two key areas are more important than ever:

- Streamlining the development process  
Increasing the productivity of development teams is an important goal, both to reduce cost but also to improve time to market. The architect has several responsibilities here:



- ensuring that the "frictional cost" of translating design into code is minimised by ensuring that only the right design deliverables are produced, and that the need for "glue" to link business logic to the underlying system software (e.g. persistence) is minimised.
  - ensuring that the design of the system enables construction to be partitioned into parallel build activities
  - ensuring that common components are built once and reused
  - ensuring that the complex parts of the system are treated specially
  - ensuring that the development environment can support the development effort envisaged (e.g. what granularity the software is placed under configuration management, amount of CPU required to support timely compilation of the application)
- Designing for scalability and performance  
Scalability and performance are important to correctness of the solution. As the dependency on third party products increases (e.g. middleware, commercial base classes) - it is vital that the architect understands the performance characteristics of these "black boxes". It is not feasible to wait until the system is built to determine its performance characteristics by testing - the architect must be able to model the likely behaviour early enough in the process to avoid problems later.

Good application of architecture can result in one or more of the following:

- preservation of investments in applications and technology by isolating each from changes in the other (e.g. upgrades in hardware or third-party software do not impact applications)
- leveraging scarce technical skills (e.g. the need for people with detailed skills in a specific communications protocol or aspects of SQL).
- Serves as a construction blueprint and discussion agenda and ensures consistency across systems. This can have a big impact on the operability and maintenance of the delivered application

## 2.2. Alan Cameron Wills - Trireme International Ltd. – Panellist

### 2.2.1. Brief Bio

Alan Cameron Wills is technical director of TriReme International Ltd., consulting and training clients in many fields, including banking, telecommunications, and manufacturing. Dr. Wills has worked on methods and tools since 1982, and specialises in making frontline research practical and available for mainstream software engineering. Dr. Wills is co-author of “Objects, Components and Frameworks with UML, The Catalysis Approach” published by Addison Wesley.

### 2.2.2. Position Statement

The architecture of a design is the set of rules that ensure consistency across all of its parts. By setting out how a range of design decisions should be made across a development, it reduces the effort needed in the design of each part; and at the same time makes the design more flexible and more usable by ensuring uniformity in appropriate aspects.

Architecture isn't just the design of the large pieces. While the distinctions are somewhat arbitrary, we take design to mean creating a larger part from some smaller ones -- each of which could itself be specified and given to separate people to design. Whether the pieces we're talking about are grand distributed systems composed from substantial subsystems, or whether they are little subroutines made up from a few program statements, design means working out how the smaller parts make up the bigger one.

If each piece in a big project were given to someone to design in complete isolation, the resulting complete system would look a bit incoherent. The architecture of a system is the set of guidelines followed by all the designers, making the parts coherent.

By analogy, think of the development of a large building. Each room is necessarily somewhat different in its exact shape and function, and each is designed by a separate designer. But (in a sensible building) there is a coherence of style that guides the decisions that each designer might otherwise make arbitrarily: for example, the height of the rooms, the choice of window frame. These rules make it easier to fit the parts together; they save us from having to procure hundreds of different windows, and also make life easier for the occupants.

In families of products, the architecture is crucial in ensuring different family members can readily be formed by composing the components in different configurations. The same is true in enterprise integration.



### **2.3. Mark Collins-Cope – Ratio Group Ltd. – Panel Chair**

Mark Collins-Cope is technical director of Ratio Group Ltd., a UK based organisation specialising in the provision of training, consultancy, software tools and recruitment in the object and component technology arena. During his eighteen years in commercial software development, Mr. Collins-Cope has undertaken many roles including that of technical architect and project manager on a number of system developments. As well as his managerial responsibilities, he undertakes consultancy assignments specialising in project management, component based development and the application of use case analysis.



## 2.4. John Daniels – Syntropy Ltd. – Panellist

### 2.4.1. Brief Bio

John Daniels is a consultant at Syntropy Limited, providing help with system architectures and development processes to a number of large corporations. He was previously Application and Technical Architect for Bankers Trust in London, and before that Managing Director of consulting and training company Object Designers Limited. John is an object technology pioneer, with more than 15 years experience of object modelling and implementation in a range of industrial and commercial applications.

John gave the first training course on object-oriented design in the UK in 1986. Together with Steve Cook he developed the highly influential Syntropy object method, the subject of a jointly-authored book published in 1994. Aspects of Syntropy have subsequently appeared in the Unified Modelling Language and numerous methods including Catalysis. Syntropy was the direct ancestor of the UML's Object Constraint Language. He is the joint author, with John Cheesman, of a new book on component-based design to be published in October 2000.

### 2.4.2. Position Statement

We need to make a distinction between software architecture and software architects.

By analogy with building architecture, a software architecture is a style, a shape. A software architecture is independent of the details of the problem domains to which it is applied. It is possible to look at a software system and determine the style it follows, but a description of the design of a particular system, no matter what level of abstraction is applied, is not an architecture. It is inherent in software architectures that they can be applied to many systems.

A software architecture tells you:

- the basis on which software is partitioned into pieces
- the form and nature of connections between those pieces
- the ways in which the pieces use the connections
- optionally, the best technology to use for each aspect

The best way to document an architecture is to:

- describe the end result you want
- provide a set of principles or rules that will guide designers into solutions that look like the desired end result

There are two valid interpretations of "software architect". The first is that a software architect is someone who creates software architectures, as described above. However, by analogy with building architects we can arrive at a second and quite different interpretation. This is that a software architect is someone who:



- Knows what is possible, given the state of the technology
- Acts as a conduit between the client and the builder, translating their needs into solution ideas.

It would help if these interpretations had different names. For the purposes of this position paper I'll call the first interpretation "architectural stylist".

In my experience too much of the money spent on "software architecture" goes to architectural stylists and not enough to software architects. The return on investment in architectural stylists is often hard to realise. In many cases it is better to let the style naturally emerge from design rather than attempt to create it in advance.

## 2.5. Alan O'Callaghan – DeMontford University – Panellist

### 2.5.1. Brief Bio

Alan O'Callaghan is a Senior Lecturer and researcher at De Montfort University's Software Technology Research Laboratory. In his industry-based consultancy and training work he has pioneered the use of patterns in the migration of legacy systems to object technology and component-based systems, authoring the ADAPTOR pattern language. He is a member of the non-profit OPEN Consortium, the IEEE and the World Wide Institute of Software Architects. He has written two books and more than forty journal articles including a regular column in *Application Development Advisor*.

### 2.5.2. Position Statement

As the pace of technological innovation increases, and begins to intersect with radically new means of doing business through e-commerce, the web, as computing systems are more and more impacted by issues of scale and distribution, there is an increasingly urgent interest in software architecture. The books by Garlan and Shaw, Best et al., and products and processes promoted by the SEI at Carnegie Mellon University (e.g., the Architectural Tradeoff Analysis Method, the Architectural Business Cycle etc.) have undoubtedly increased our ability to analyze, evaluate and generally reason about the high-level structure of constructed systems. That is, given a system Z, we can now use one of a variety of Architectural Description Languages (ADLs) to describe the static connections and dynamic interactions between Z's components. On the basis of what the chosen ADL reveals we can evaluate the system's fitness for purpose at a high level of abstraction. But there is also an accelerating trend toward reductionist thinking involved. Is analyzing structure all there is to 'Software Architecture'? Is there any evidence that we are getting better at *creating* 'good' systems on the basis of our new found understanding?

The metaphor is long established. Delegates to the famous 1968 NATO conference on Software Engineering remember many contributions which referred to the notion of Software Architecture and the role of the Software Architect. Yet the analogy was considered too fanciful to be included in the official proceedings. In 1975 in his now famous *Mythical Man Month* articles, Fred Brooks Jr. popularized the idea while crediting first use to G A Blaauw five years beforehand. Brooks championed two critical ideas fundamental to what architecture is considered to be in the built environment: first, the architect is the client's agent and second, the most important thing is to maintain the 'conceptual integrity' of the system. We are in serious danger of losing sight of these fundamental aspects just at the point we need them most. An 'architect' is not a 'structural engineer' and even less so a 'buildings inspector' (whose sole job is to police systems for conformity to standards and regulations). Rather she is the champion of a design rationale, which enables other designers' creative and inventive capacities to deliver a really useful (to the client) and usable solution to meet some perceived need.



Fundamentally software development is a design activity in which, in common with all recognized design disciplines, we seek to transform and, hopefully, improve the pre-existing environment. If this is so then maybe it is about time that we really engaged with the architecture of the built environment and learned the concrete lessons of hundreds of years of accumulated design wisdom?

This is best approached perhaps by asking what is architectural knowledge as opposed to (vernacular) design knowledge? Architectural knowledge has variously been described as 'design imagination', configurational knowledge about how to 'do' design, design rationale etc. The common theme seems to be that it not only describes 'What' a system is in terms of its structure but also 'Why' that particular structure. It is also clear that architecture is not restricted to gross structure but dictates detailed design issues too if it is to be consistent, and impacts both upon the process by which a system is delivered and the organization of the work that needs to be carried out. Architecture is present even in vernacular design, but usually only implicitly so. Architectural knowledge tends to be put at risk (i.e. open to challenge) only when different architectural approaches are compared and contrasted. Improved quality of systems, be they buildings or computer systems, requires that architecture be made explicit and that it be placed on the critical path of projects. The dissemination of architectural knowledge therefore requires forms which permit the answers to the 'Why' questions to be placed alongside the answers to the more traditional 'What' and 'How' questions of software development. Here the historic contribution of software patterns to software development needs to be noted. From this perspective it is clear that software architecture is expressed through structure but cannot be reduced to it.

## 2.6. Wolfgang Pree – University of Constance, Germany – Panelist

### 2.6.1. Brief Bio

Wolfgang Pree is a professor of computer science at the University of Constance, Germany. His research covers various areas of software engineering, in particular object and component technology, software architectures, and human-computer interaction. Wolfgang is the author of *Design Patterns for Object-Oriented Software Development* (Addison-Wesley, 1995). Wolfgang was a Visiting Assistant Professor at Washington University in St. Louis, and a guest scientist at Siemens AG Munich.

### 2.6.2. Position Statement

A common understanding of software architecture is that it comprises the top-level design of a software system and the identification of core subsystems/abstractions. We often encounter the following definition:

Software architecture := software components + the relationships among the components

Some thoughts on that:

- Isn't software architecture just a synonym for system modularization?
- What does top-level design mean? It should be a coarse-grained view that can be explained to someone who doesn't know a system within a reasonable time (3-5 hours), independent of the overall size. On the other hand, the architecture description should not be too coarse-grained so that changes of the architecture have an effect on quality/flexibility/functionality.
- The visualisation of the software architecture (eg, boxes and lines) is probably helpful but shouldn't be absolutely necessary. Instead, e.g. prose source code of module interfaces should suffice.

The definition of software architecture above requires the definition of the term component which is used in quite different ways. In the nineties, the term componentware has become the vogue in the software engineering community. Component-based software development as buzzword is associated with a shift from statement-oriented coding to system building by plugging together components. The idea is not new. Module-oriented and object-oriented languages have already tried to make this vision come true. What are the commonalities and differences between a module, a class, an object and a component? They have in common that they all rely on information hiding. All that is required for that is a means to define a programming interface for a piece of software.

Thus we define the term component as follows:

Component := A piece of software with a programming interface



The module-oriented language Modula fulfils this requirement through its language construct definition module. Component standards such as Microsoft 's Component Object Model (COM), Object Management Group 's CORBA (Common Object Request Broker Architecture) allow this through their interface definition languages (IDLs). Object-oriented languages and the JavaBeans component standard provide the class construct.

The following is a list of terms/concepts that are closely related to the term software architecture:

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

Benefits of software architecture/software 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

### 3. The Discussion

The discussion will proceed as follows:

- Brief (<5 ins) introduction to the subject by Mark Collins-Cope, including introduction of the panellists.
- A number of questions - some pre-prepared, and some from the audience. For the pre-prepared questions, the starting point will be to ask each panellist to state their position in <=2 minutes, in the context of the wider objectives their position seeks to address. Other questions may include:
  - what exactly would you expect to find in a software architecture document that conformed to your position?
  - is the analogy between software architect and building architect a useful one?
  - if you applied for a job as a software architect, do you think you'd be doing what you stated in your position?
  - how does your view of software architecture relate to software development process?
  - how does your view of software architecture relate to organisational structure?
  - has the recent upsurge in interest in component based or product line development affected your position? If so, how?
- Towards the end of the discussion the chairman will bring the focus back onto the themes outlined in the background section – can we agree, and if not, are the differences between our positions merely “naming” issues - with a view to trying to reach some form of conclusion from the discussion.

**Audience members wishing to submit a question to the panel in advance of the conference may do by emailing the question to: [markcc@ratio.co.uk](mailto:markcc@ratio.co.uk).**