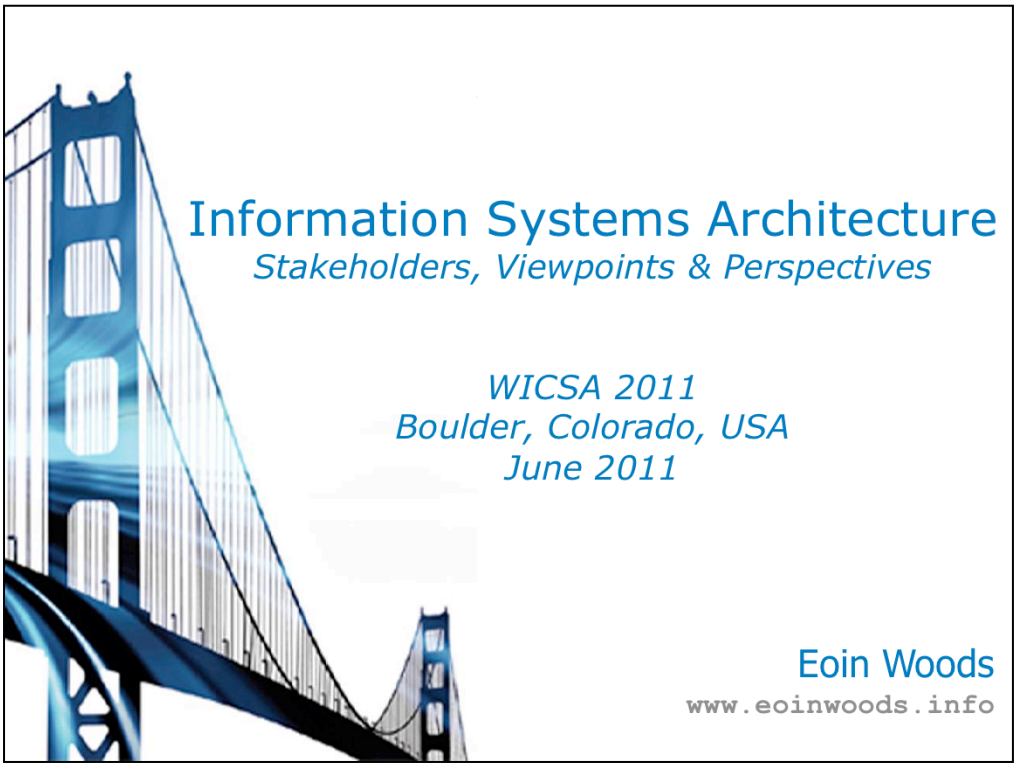


Information Systems Architecture

Stakeholders, Viewpoints & Perspectives

WICSA 2011
Boulder, Colorado, USA
June 2011

Eoin Woods
www.eoinwoods.info

Content

- Introducing Software Architecture
- Stakeholders
- The Software Architecture Problem
- Using Viewpoints for Architectural Structures
- Using Perspectives for Architectural Qualities
- Example Application
- Uses for Viewpoints and Perspectives

Defining Software Architecture

- A common definition …

  *The software architecture of a program or computing system is the **structure or structures** of the system, which comprise software **elements** the externally visible **qualities** of those elements, and the **relationships** among them*

  Len Bass, Paul Clements and Rick Kazman (SEI)
  Software Architecture in Practice, 2nd Edition

3

There are a vast number of definitions of software architecture (the SEI have collected dozens on their web site http://www.sei.cmu.edu/architecture/definitions.html) but we use the Bass, Clements and Kazman definition as a starting point.

The key points are:

(1) Architecture defines structures of elements;
(2) Architecture defines the relationships between elements;
(3) Architecture results in a system exhibiting a set of quality properties, derived from the properties of its constitent parts.

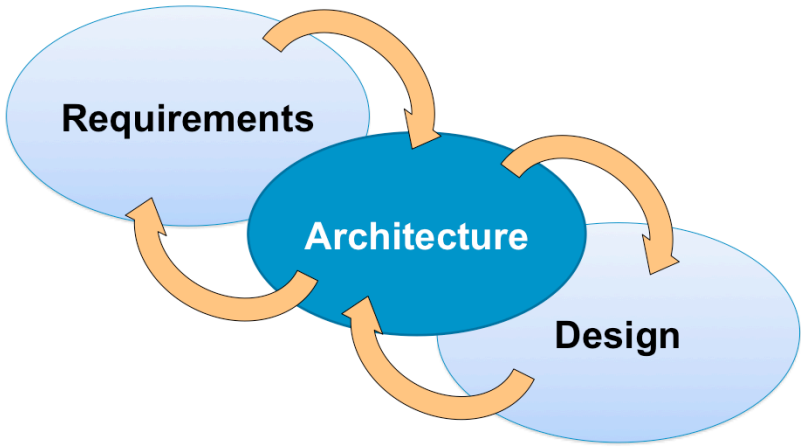
Defining Software Architecture

- Alternative definitions …
  - *The set of system design decisions that dictate the fundamental structure and properties of a system Thus, the set of decisions that will cause the system to fail if made incorrectly*
  - *The set of design decisions that, if made wrongly, cause your project to be cancelled!*

4

Role of Software Architecture

The bridge between requirements and design



5

Software architecture doesn‘t exist in isolation, but contains elements of both requirements analysis and system design. It also has its own unique elements of course.

The key point is that architecture acts as the bridge between the problem-centric world of requirements (“what we want”) and the solution-centric world of design (“how should we do it”). People who are experts in one of these areas are rarely experts (or even all that interested) in the other. Architects act as a bridge between these two areas. Architects understand enough about the requirements to know what is important and what tradeoffs are possible, and perform enough design to define a structure that can meet those requirements and inside which the detailed design decisions can be made.

Architecture & Requirements

- Requirements are an input to architecture
  - Requirements frame the architectural problem
  - Stakeholder needs and desires
- Architecture must influence requirements
  - "The art of the possible"
  - Stakeholder understanding of risk/cost
  - Stakeholder understanding of possibilities

6

While it's pretty obvious that requirements are fed into the architecture process, it's not always understood that this is a two way thing. By performing architectural design, architects quickly gain an understanding of what is and isn't possible with respect to a set of requirements. They can use this knowledge to feed back into the requirements process and help stakeholders make decisions about what should be built.

Architects help stakeholders to make these decisions by explaining the relative risks and costs of different requirements to them. This helps stakeholders to work out what they really *need*, as opposed to what they'd *like* if it were possible easily and cheaply.

As Otto von Bismark's said of politics, software architecture can be seen as the "art of the possible", with architects being best placed to understand what is and isn't possible and helping to communicate this to stakeholders, so helping both to manage expectations and open up new possibilities.

Quality Properties

- Non-functional characteristics ("-illities")
  - Performance, Security, Availability, …
- Often crucial to stakeholders
  - Slow functions don't get used
  - Unavailable systems stop the business
  - Security problems cause headlines
- Yet often an after-thought
  - Addressing qualities is a key architectural task
  - Making the key trade-offs to allow delivery
  - Avoiding expensive "retro-fit"

7

When people start their software design careers, they normally focus on what the software needs to do. More experienced software designers know that while functionality is part of the puzzle, it's by no means always the most important part. Quality properties (also known as "non functional requirements") are a crucial part of

Many systems have failed terribly because they failed to exhibit one or more important quality property, such as security, availability, maintainability and so on. There is an old saying that there is no such thing as bad publicity, but the headline "XYZ Inc's Customer Details Sold on Internet" is probably an exception. For Internet facing systems, it's often the case that qualities such as security and availability are crucial to avoid negative publicity for the system's owner.

The problem we have often seen is where quality properties aren't considered until late in the day and it can then be very difficult to retro-fit these properties to a system because they cause fundamental architectural changes which are expensive to do late in the lifecycle.

It is the job of the architect to ensure that your system exhibits an acceptable set of quality properties. It is unlikely that you can achieve the perfect set (as, if asked, any system acquirer would like their system to be 100% reliable, delivered at zero cost, with zero time to delivery and infinitely scalable and maintainable). Because you can't achieve the perfect set at an acceptable cost, you need to understand what is really needed and what is really possible and try to match these two, often by making trade-offs between conflicting qualities. For example, the overheads of achieving a high degree of security often conflicts with efficiency or performance goals and can make a system significantly less usable so you need to achieve an acceptable balance between them.

What you must try to avoid is needing to change the set of fundamental system quality properties late in the development lifecycle as this is usually expensive and disruptive due to the fundamental architectural changes needed to achieve this.

Architecture & Design

- Architecture frames design
  - architecture *is* part of the design process
- Captures the system-wide decisions
  - what has to be consistent or constant
- Importance of role increases with scale
- Perfectly compatible with agile
  - even XP with the right approach

8

Software architecture provides a context for detailed software design. It acts as the first step in the design process, but it is only the first step and the expertise of many other people is usually required in order to design a complete system. An important part of the architect's role is to integrate their work with the more detailed software design in order to ensure the relevance of the architecture and the integrity of the detailed design.

The key point is that the architecture of a software system must capture those design decisions that apply right across the system in order to ensure its technical integrity.

While every system has an architecture, the importance of explicit software architecture increases quite rapidly as systems get larger. It's not difficult for a 4 or 6 person team to ensure conceptual integrity for their system quite informally; it's much more difficult for 20 or 30 people to do so.

A common point of contention is the role of software architecture when developing software using agile methods. While some agile methods (particularly XP) do de-emphasise the role of architectural design, a sound architectural design can provide a solid framework in which agile teams can incrementally understand and develop the details of their subsystems. Software architecture can never address all of the details that emerge during detailed design and an iterative, incremental, agile approach is ideal for understanding the detailed requirements and developing a solution for them.

Distinguishing Architecture

"*All architecture is design but not all design is architecture*" …

- Architecture addresses system qualities
- Architecture addresses cross-cutting concerns
- Architecture provides the system skeleton
- Architecture decisions are hard to change
- Architecture addresses risks the system faces
- Architecture constrains as much as defines

9

As Paul Clements has said, *all architecture is design, but not all design is architecture*. So it can be difficult to work out what "architectural" design is, in order to know what to concentrate on. We have found that some of the distinguishing characteristics of architectural design are:

• The architectural design is the set of design decisions that are made in order to achieve the system's quality properties (e.g. scalability) as opposed to its detailed functions

• Architectural design decisions are the decisions that apply on a system wide basis rather than to an individual component (e.g. how the system will be monitored and controlled)

• Architectural design provides the skeleton of the system, inside which the more detailed design of the system fleshes out the specifics of how each part of the system will work (e.g. the fact that the system will comprise a client, a server, a batch processing engine and a single database).

• Architectural design decisions are those that are difficult to change once implemented in the system (e.g. embedding a large number of business rules in a rules engine)

• Architectural design decisions are those made to address one or more of the risks that the system faces (e.g. using a data access layer library for all database access if there are concerns about the scalability of a single database to support the system)

• Architectural design is often expressed in terms of constraints to guide more detailed design decisions rather than prescriptive decisions (e.g. defining where transaction boundaries must be enforced in all modules, without mandating how the transaction should be initiated or completed to allow for different detailed design options).

When is Architecture Important?

- Small Solution Space
  - hard problems with few solutions
- High Failure Risk
  - loss of life, organisational criticality, monetary loss, …
- Difficult Quality Attributes
  - scale, security, speed, modifiability, …
- New Problem or Technology Domain
  - unfamiliar problems need careful attention
- Product Line Development
  - enabling clear commonality and variation

[Dr George Fairbanks]

10

In his book Just Enough Software Architecture, Dr George Fairbanks has identified 5 situations which suggest that software architecture will be important:
• Where you're solving a hard problem with few solutions
• Where getting it wrong has serious implications
• When the system needs to provide some difficult combination of quality attributes
• When you aren't familiar with the problem domain or the technology you need to use to build the system
• When creating a software product line and so you need to be very clear what is common and what varies for each member of the product line

How Much Architecture is Enough?

- That depends on the risks you're running
- Architecture is there to mitigate risks
- So continue architecture until your risks are acceptably mitigated
  - partial architectural designs are OK
  - if your risks are under control
- When your stakeholders are confident then you're probably done
  - remember you're one of the stakeholders!

Read *Just Enough Software Architecture* – George Fairbanks 11

Content

- Introducing Software Architecture
- Stakeholders
- The Software Architecture Challenge
- Using Viewpoints for Architectural Structures
- Using Perspectives for Architectural Qualities
- Example Application
- Uses for Viewpoints and Perspectives

Stakeholders

- Identifying Stakeholders
  - People, Groups, Entities
  - Those who have an interest in or concerns about the realisation of the architecture
- Importance of Stakeholders
  - Architectures are built for stakeholders
  - Decisions must reflect stakeholder needs
  - Involving a wide stakeholder community increases your chances of success

13

An important concept within software architecture is that of the “stakeholder”, who is any person, organisation or group with an interest in the system being developed. When you think about it, we only build systems because stakeholders need them and so we must make sure that all of our architectural activities are directed at meeting the needs of at least one stakeholder. If we’re doing something that doesn’t meet the needs of a stakeholder, then why are we doing it?

In order to deploy a system successfully, it is nearly always important to identify and engage with your stakeholders as early as possible and make sure that you have a good, representative set of stakeholders (see next slide).

Talking Point: Stakeholders

- Who cares whether your systems get built?
- Why do they care?
- Are they positively or negatively impacted?

Create your system's stakeholder list

Stakeholders

- Attributes of a good stakeholder
  - Informed, to allow them to make good decisions
  - Committed, to the process and willing to make themselves available in a constructive manner, even if decisions are hard
  - Authorised, to make decisions
  - Representative, of their stakeholder group so that they present its views validly

15

It is important that you try to engage effective stakeholders who will participate in the architectural process positively and make a net contribution. Of course, such people are not always easy to find, so when considering stakeholders we look for people who are informed, committed, authorised and representative (RACI – “racy”).

Stakeholder Groups

- Acquirers pay for the system
- Assessors check it for compliance
- Communicators create documents and training
- Developers create it
- Maintainers evolve and fix the system
- Suppliers provide system components
- Support Staff help people to use the system
- System Administrators keep it running
- Testers verify that it works
- Users have to use the system directly

16

It is also important to spread your net widely when considering stakeholders. Many stakeholders aren't immediately obvious and indeed, some of them (such as acquirers or assessors) may never use the system. Stakeholders can be generally divided into two groups: those who want the system and those who actively or passively want to resist it. For example, many internal auditors and regulation specialists (members of our "Assessors" group) are goaled on reducing and managing risk and they may consider your system to actively imperil their goals. Such stakeholders can be just as important to consider as the more obvious ones (like end users and software developers) as they may have the power to prevent your system going live if their needs are not met.

This slide shows some of the more important stakeholder groups for enterprise information systems.

Content

- Introducing Software Architecture
- Stakeholders
- The Software Architecture Challenge
- Using Viewpoints for Architectural Structures
- Using Perspectives for Architectural Qualities
- Example Application
- Uses for Viewpoints and Perspectives

17

The Challenge

- Essential difficulties
  - Multi-dimensional problem
    - Highly complex mix of people and technology
    - Diverse stakeholder community to serve
  - Making trade-offs is essential but hard
    - Often no "right" answer
- Accidental difficulties
  - Little standardisation in architectural description
    - Difficult to compare and discuss alternatives
  - Little sharing of proven practice and known problems and their solutions
  - No framework for handling quality properties

18

As architects, we'd probably all agree that our role is a challenging one and its useful to stop for a moment to consider why. Some of the challenges are fundamental to the role (essential difficulties) while others are the result of the current state of practice (accidental difficulties).

The essential difficulties of software architecture are that we have to deal with a highly multi dimensional problem (functionality, performance, scalability, security, delivery times, budgets, ...) in a complex environment where many different specialists and complex pieces of technology must work together in order to deliver a system. We also typically have to deal with a diverse stakeholder community, with overlapping and conflicting needs.

In such an environment, we have to accept that there isn't a single right answer and we may well have to be prepared to settle for an acceptably bad one. Philippe Kruchten summed the problem up nicely in one of his papers when he said "*The life of a software architect is a long and rapid succession of suboptimal design decisions taken partly in the dark*"!

However, these are all fundamental aspects of the job of the architect and we shouldn't expect any of these challenges to recede significantly due to technological or methodological change.

In contrast, there are some challenges that most software architects face that are more likely to be solved by improving the practice of software architecture.

At present, there is little standardisation in how architectures are described and this means that it is often difficult to compare and discuss alternatives without huge amounts of face to face communication. We also tend to go about the process of software architecture in individual ways, with relatively little shared language or process for how we do it. There is also a limited amount of knowledge sharing between architects in terms of sharing proven practice and the inevitable sets of pitfalls and solutions that experienced architects learn over time. Finally, while there are some frameworks for structuring architectural descriptions, these have tended to focus on architectural structures rather than qualities and so we tend to handle quality properties in a relatively ad-hoc manner, which can be unfortunate.

Solving the Accidental Difficulties

- Organise the architectural design process
  - define roles & activities
  - define relationship to requirements & design
- Define the use of architecture artefacts
  - which models? when? why?
- Capture, classify and share knowledge
  - proven practices and solutions
  - problems and pitfalls

19

To help to meet the challenges of software architecture, the following approaches would appear to be potentially useful:

- Organise the architectural design process according to a standard model so that, even if it is a simplification of reality, we have some standard language and understanding of how to go about the process.
- Similarly, we can define a set of commonly used architecture artefacts and capture lessons learned that can guide architects to use these artefacts at the right time, for the right purposes.
- More generally, if we can capture and share knowledge between architects, then we will have a better chance of learning how to overcome the challenges of the job from other people‘s experiences.

Content

- Introducing Software Architecture
- Stakeholders
- The Software Architecture Challenge
- Using Viewpoints for Architectural Structures
- Using Perspectives for Architectural Qualities
- Example Application
- Uses for Viewpoints and Perspectives

Architectural Viewpoints

- Help to deal with architectural structure
- Decompose architectural description to views
  - each view addresses one aspect of the system
  - functional view, deployment view, …
- Guide development of views via viewpoints
  - viewpoint contains proven practice, pitfalls, …
  - each view defined by one viewpoint
- Organises the process and the artefacts

21

The use of Viewpoints and Views is an existing approach, that we have used successfully, for dealing with complex architectural structures.

Both ideas are simple but have proved to be effective:

- Views are used to structure the architectural description into a number of pieces, each describing one aspect of the system (the functional structure, the deployment environment, the development constraints and so on). The architectural description is made up of a set of views.
- Viewpoints provide templates for the views and as such a particular viewpoint is used to develop each view. A viewpoint provides the architect with guidance by defining what the corresponding view should or may contain, how to represent it, how to go about developing it, potential problems to be aware of and their solution, and so on.

The relationship between view and viewpoint is similar to that of object and class.

Using viewpoints and views helps to organise both the process being followed (the viewpoints providing implcit structure and explicit guidance) and the artefacts produced (the views being an organisation of the architectural description).

Architectural Viewpoints

- Well understood, widely applied, many exist
  - RUP/Kruchten "4+1" set (1995)
  - RM-ODP set (1995)
  - Siemens set (1999)
  - Garland and Anthony set (2003)
  - Rozanski & Woods set (2005)
  - Conceptual basis in ISO 42010 (2011)
    - formerly IEEE 1471 (2000)

22

The architectural viewpoints idea isn't all that new, having academic roots back in the 1970s from David Parnas and more recently in the 1990s from Dewayne Perry and Alex Wolf. Widespread awareness of viewpoints started to spread in the mid-1990s and since then a number of sets of viewpoints have been developed. Some of the important ones are listed on this slide:

- 4+1 – Philippe Kruchten and the Rational Corporation, published in IEEE Software in 1995, probably the earliest mainstream description.
- RM-ODP is an ISO standard for describing distributed object systems and their viewpoint set was published as part of the standard in 1995 too.
- Christine Hofmeister, Rod Nord and Dilip Soni defined a set for real-time and embedded systems while working at Siemens Research, based on the way that Siemens software architects worked. Documented in their book "Applied Software Architecture" in 1999.
- Jeff Garland and Richard Anthony defined a set of viewpoints for information systems, using UML as the base description notation across the views, documenting the set in their book "Large Scale Software Architecture" in 2003.
- We defined a set of viewpoints, based on the 4+1 set in our book "Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives" in 2005. Conceptually, our set is like G & A's, being practitioner focused, the result of our own experience and aimed at information systems, although the set is a lot smaller.

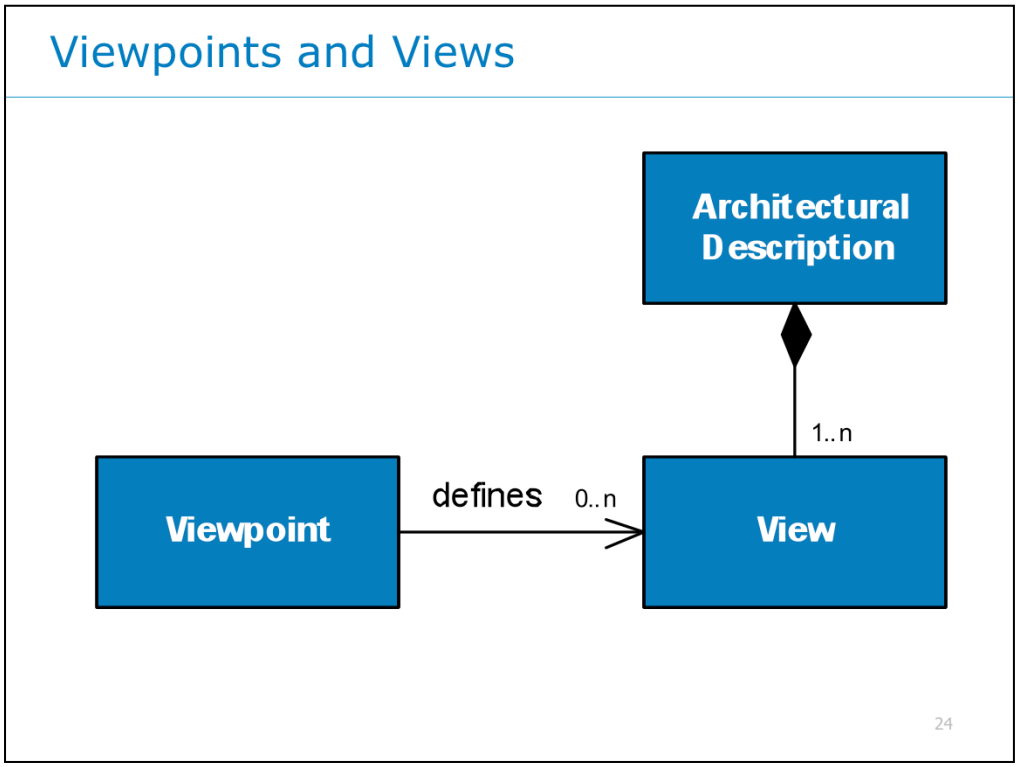
- A conceptual model for how viewpoints and views relate to each other and their environment (systems, architects, stakeholders and so on) forms the basis of ISO Standard 42010 (which is a development of the previous IEEE Standard 1471).

Viewpoints and Views

- IEEE 1471 provides standard definitions
  - A viewpoint is a collection of patterns, templates and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint, and guidelines and principles and template models for constructing its views.
  - A view is a representation of all or part of an architecture, from the perspective of one or more concerns which are held by one or more of its stakeholders.
  - from IEEE Standard 1471 – Recommended Practice for Architectural Description (2000)

23

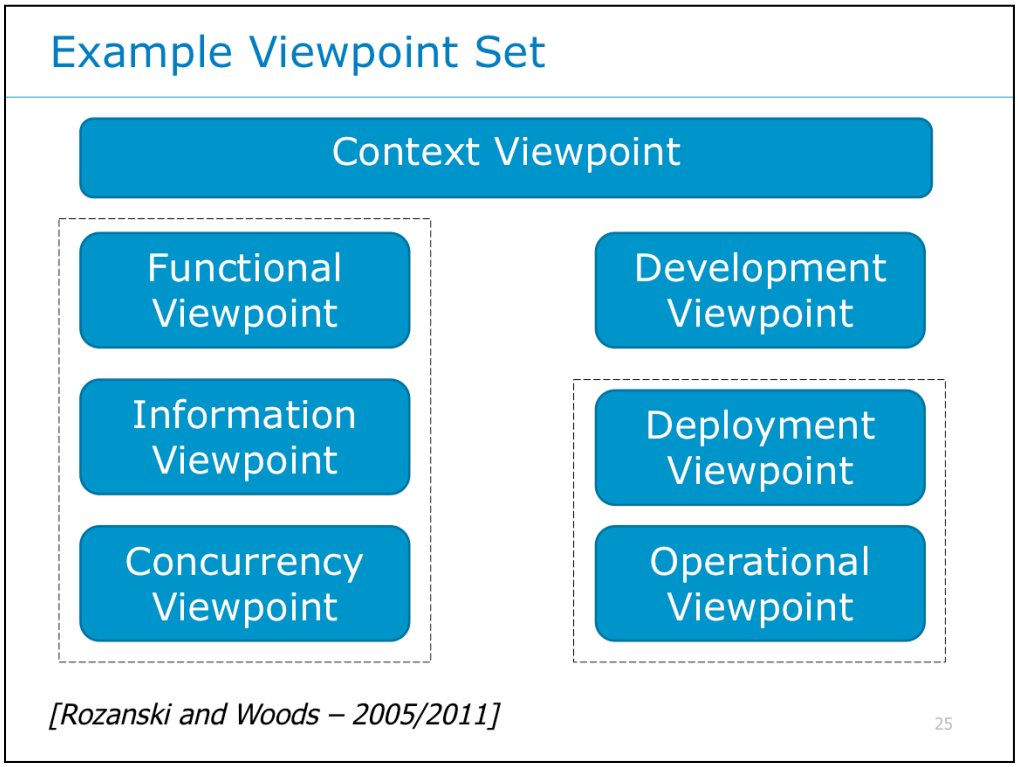
IEEE Standard 1471, Recommended Practice for Architectural Description, was published in 2000 and provides nice definitions for all of the conceptual entities that it discusses. The later ISO 42010 standard was based on 1471 but generalised the definitions greatly, defining a viewpoint to be a "work product establishing the conventions for the construction, interpretation and use of architecture views to frame specific system concerns" and a view as a "work product expressing the architecture of a system from the perspective of specific system concerns". Hence we retain the older, more specific, 1471 definitions above.



This UML class diagram presents part of the conceptual model for viewpoints and views:

- An architectural description is a collection of one or more views.
- A viewpoint is used to define (the structure and content of) zero or more views.
- The structure and content of a particular view is defined by exactly one viewpoint.

Normally, only one view corresponding to a particular viewpoint would appear in an architectural description. (In UML, this could be defined as the OCL constraint: `Context ArchitecturalDescription inv UniqueViews self.View->isUnique (Viewpoint)`).



An example of a viewpoint set for information systems work is the one we defined in our book. Briefly, our viewpoints are:

- Context – system environment, scope and external connections and dependencies
- Functional – functional structure, elements, responsibilities, connectors, interactions.
- Information – stored data, ownership, information models for interfaces (e.g. messaging), information latency and so on.
- Concurrency – packaging of elements into runtime processes and threads, with coordination as required.
- Development – architectural constraints on software development (CM, design patterns, layering, tiers, ...)
- Deployment – runtime environment, nodes, links, software and hardware dependencies.
- Operational – strategies for migration, installation, back-out, parallel run, operational control, support and so on.

The first three really define the design of the software itself, the development viewpoint guides it being built, while deployment and operational define the environment it requires in order to run in production.

Example Viewpoint Set

- Setting the scene
  - Context
    - scope, relationship with external environment
- Core architectural structures
  - Functional
    - elements, connectors, interfaces, responsibilities, interactions
  - Information
    - entities, constraints, relationships, timeliness, usage, ownership
  - Concurrency
    - processes, threads, coordination, element to process mapping

26

The context view is used to define the scope of the system and the relationships that it has with its environment.

The functional, information and concurrency views define the core architectural structures of the software. Their important concepts are outlined on the slide.

Example Viewpoint Set

- Building the system
  - Development
    - layers, module structure, standard design, codeline
- Moving towards deployment
  - Deployment
    - hardware, network, software dependencies, process to node mapping
  - Operational
    - installation, migration, administration, support
- See appendix for outline definitions

27

The development view is where the architect defines the (smallest possible!) set of constraints on software development.

The deployment and operational views define the required runtime environment and how the system will get there and be run and supported, respectively.

Example Viewpoint Set

- Rozanski/Woods Viewpoint Set
- Aimed at large scale information systems
- Extension and refinement of the "4+1" set
  - renamed "Logical", "Process" and "Physical"
  - added "Context", "Information" and "Operational"
- Standard content for viewpoints
  - applicability, concerns, models, stakeholders, problems & pitfalls, solutions, checklists

28

Our viewpoint set is aimed at architects working on large scale, mainstream, information systems (and specifically not embedded systems).

We started with Philippe Kruchten / RUP's 4+1 set, applied it for some time and then decided to improve it. Some improvements are trivial (such as renaming views), others are quite substantial (such as adding an operational view).

We use a standard structure for our viewpoints, namely:

- Applicability - where is the content of this viewpoint useful and relevant?
- Concerns – what are the architectural concerns that views based on this viewpoint address?
- Models – what models should a view based on this viewpoint contain?
- Stakeholders – who are the stakeholders who are likely to be interested in the content of views based on this viewpoint?
- Problems and Pitfalls – what is likely to go wrong in this area and what should you do about it?
- Checklists – what do you need to remember in order to avoid problems?

Talking Point: Views

- Which views would be useful for your systems?
- Who would be interested in each view?
- Which views wouldn't be useful – why?

Draw up a list of which views you would use

Who would be interested in each ?

Can you sketch fragments of one or two?

Choosing Views

- Standard JEE/.NET Enterprise System
  - Context, Functional, Information, Development, Deployment
  - Maybe Operational
- Enterprise middleware or compute engines
  - Functional, Concurrency, Deployment, Development
- DSS / MIS / Data-Warehouse
  - Context, Information, Deployment
  - Maybe Functional, Operational

Choose the views you really need

30

Viewpoints and Views Recap

- Viewpoints
  - A store of knowledge and experience
  - A guide to the architect
  - Templates to guide the process
- Views
  - A structure for description
  - A separation of concerns
  - Aid to stakeholder communication

31

So, to recap …

•Viewpoints are the store of knowledge, that define the content of views of a particular type and guide the architect to create them.
•Views are the specific partial system descriptions, each describing one aspect of the system, the collection of which forms the architectural description.

Limitations of Viewpoints and Views

- Quality properties are critical
  - most viewpoint sets don't explicitly consider qualities
- Quality properties usually need cross-view consideration
  - viewpoints are relatively independent
- Viewpoint focus may lead to late consideration of quality properties
  - architects focus on desired structures not qualities
  - qualities are often expensive to add later

32

We've found viewpoints and views to be a very effective approach for architectural description and guiding the architectural process. However, when applying existing sets, we've found one major limitation: they don't address quality properties effectively.

As the authors of IEEE 1471 point out, there is nothing to stop you developing a viewpoint for a quality property (e.g. security) and then creating a view of your system based upon it. However, in practice, we didn't find that this worked well. We found we could define the viewpoint reasonably easily, but when we came to create the view, it inevitably overlapped with lots of the other views.

To take security as an example, a "security view" is probably going to need to include information duplicated from the deployment view, development view, possibly the functional view, possibly the information view and so on, in order to explain effectively how the system to be made secure.

There is also the practical point that all of the existing viewpoint sets contain viewpoints for particular architectural structures. None of them contain viewpoints for quality properties. This means that when using these sets, there is a very natural tendency to focus on structures first and think about their properties later, which as we know can cause problems.

Content

- Introducing Software Architecture
- Stakeholders
- The Software Architecture Challenge
- Using Viewpoints for Architectural Structures
- Using Perspectives for Architectural Qualities
- Example Application
- Uses for Viewpoints and Perspectives

Talking Point: Qualities

- Which quality properties are crucial to your systems?
- Which views are going to be impacted by considering each such quality property?

Create a matrix showing which qualities are likely to impact which of your views

Possible Qualities to Consider

- Performance
- Scalability
- Security
- Availability
- Resilience
- Evolution
- Maintainability
- Supportability
- Geographical distribution
- Localisation
- Efficiency
- Development constraints
- Time to deliver
- Reliability
- Cost
- Safety

35

Dealing with Quality Properties

- A new concept could help
  - Allowing cross-view focus
  - Being quality rather than structure oriented
  - Providing similar organisation and guidance to a viewpoint but for a quality property
  - That can be used in tandem with viewpoints
- We call this new concept a "perspective"
  - or "architectural perspective" in full

36

We found it difficult to use viewpoints and views for quality properties, so we considered adding a new concept to our architectural approach. We realised that we needed some way of achieving a cross-view focus, that is so common when considering quality properties, and we needed to guide architects to consider quality properties much more explicitly. We wanted many of the features of a viewpoint and for our new concept to be naturally usable with viewpoints (as we still found them to be very effective for architectural structures).

We called our new concept an "architectural perspective" (normally shortened to "perspective").

Architectural Perspectives

*An architectural **perspective** is a collection of **activities, checklists, tactics and guidelines** to guide the process of ensuring that a **system exhibits** a particular set of closely related **quality properties** that require consideration across a number of the system's architectural views.*

Rozanski and Woods, 2005

37

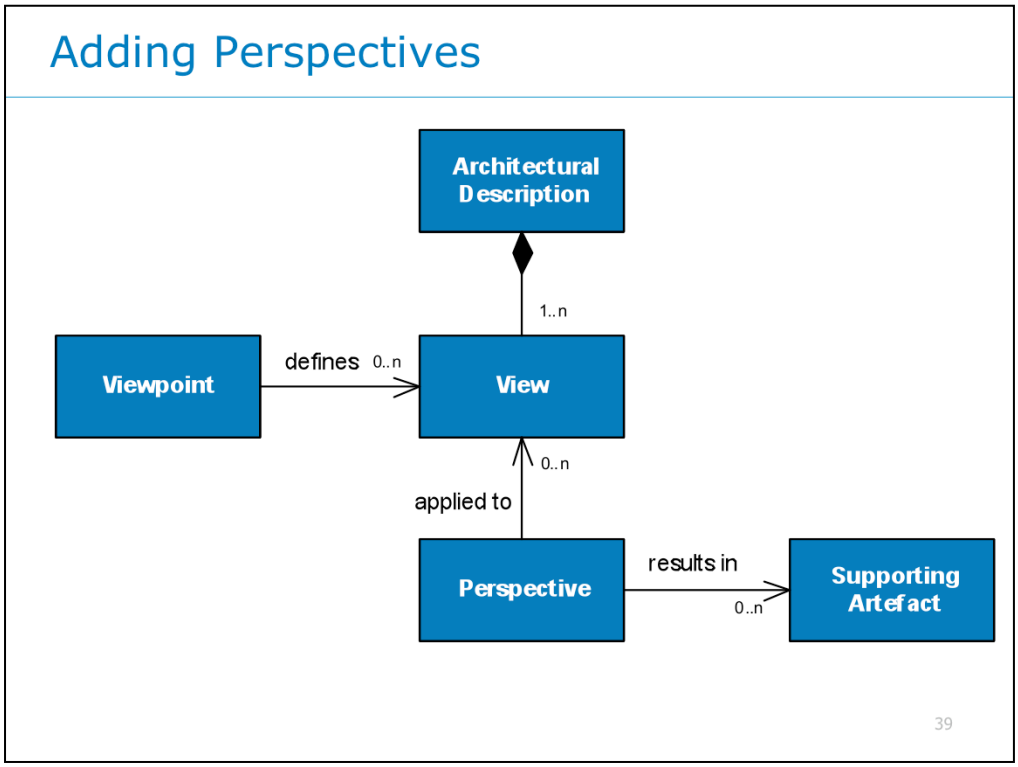
The definition of a perspective is deliberately fashioned after the IEEE 1471 definition of a viewpoint, to help people relate the two.
The perspective is a collection of architectural guidance, in terms of activities to be performed, checklists to check, tactics to consider applying and sets of pitfalls to be aware of when attempting to create a system that exhibits a particular quality property (or a very small closely related set, if this makes sense).
Perspectives are described in our recent book, "*Software Systems Architecture: Working With Stakeholders Using Viewpoints & Perspectives*", Nick Rozanski & Eoin Woods, Addison Wesley, 2005.

Architectural Perspectives

- A guide for dealing with quality properties
  - Guide the architect in achieving the required quality properties
  - Suggest changes to the existing views
  - Avoid possible redundancy between quality and structural views
- A new concept to use with viewpoints
  - Related to and extends SEI tactics work
  - Adds more context and advice to tactics

38

A set of perspectives provides an architect with a guide for dealing with quality properties in their system. Each perspective guides the architect through the process of ensuring that their system will exhibit the quality property that the perspective in question addresses.
The perspectives suggest a process for the architect to follow and catalogue architectural tactics that can be applied to the system in order to achieve the quality. Applying these tactics is likely to involve changes to a number of the views describing the system. The advantage of this approach, as opposed to creating a new view per quality property, is that it avoids the high degree of redundancy between views that a view-per-quality-property approach would inevitably result in.
While they are a new concept, perspectives dovetail neatly with the SEI's tactics work, as each perspective lists a number of tactics that can be applied in order to achieve the property in question. However, perspectives are much richer than tactics, as they provide much more context and guidance than a simple list of tactics can.



Adding perspectives to our conceptual model results in the additional statements that:

- A perspective is applied to zero or more views (almost certainly one or more if the quality property is important).
- A perspective can result in a supporting artefact (such as a performance model for example)

An important point to note is that the perspectives cause changes to be made to the existing views that describe the system. They do not appear themselves in the architectural description, as the perspectives are guides to the architect, analogous in many ways to viewpoints.

Architectural Perspectives

- A simple but effective idea
  - A store of knowledge and experience
  - A guide to the architect based on proven practice
  - Templates to guide the process
- Analogous to viewpoints but for quality properties, rather than structures
- Perspectives "applied" to views to assess qualities and guide changes needed

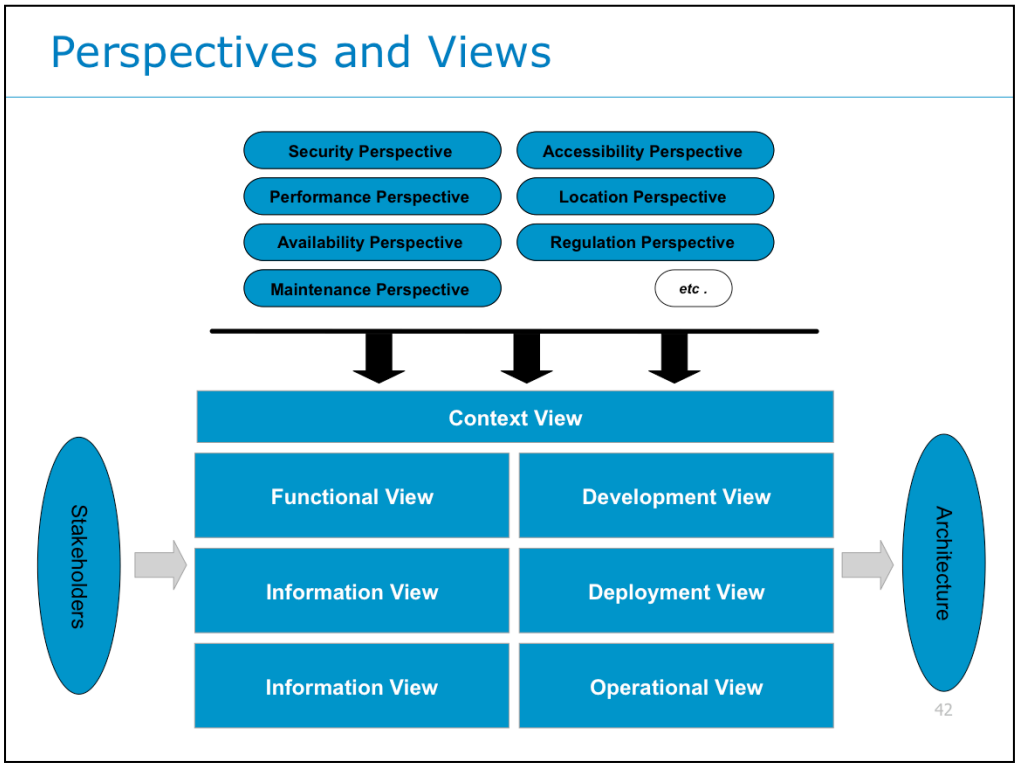
40

To summarise …
Perspectives are a simple idea, but we've found them to work well in practice, providing a guide to the architect by acting as a store of knowledge and experience and providing suggested activities to guide aspects of the architectural process.
We say that a perspective is "applied" to the views in order to modify them so that the system exhibits the quality property required.

Perspectives vs. Viewpoints

	Perspective	Viewpoint
Focus	▪ A quality property	▪ A type of structure
Result	▪ Changes to views ▪ Supporting artefacts	▪ A view – model(s) ▪ A primary arch structure
Guidance	▪ A process for application ▪ Advice based on practice	▪ Models to create ▪ Advice based on practice
Versus Views	▪ 1 perspective : n views	▪ 1 viewpoint : 1 view

41



This figure provides an overview of how views and perspectives work together to produce an architecture.

> An architecture is designed, based on stakeholder inputs and needs and this results in a candidate architecture.
>
> Then, the perspectives relevant to this system can be applied to the architecture (as defined by the views), and the architecture modified as required in order to ensure that the system will exhibit the quality property in question.
>
> During this process, the architect obviously has to balance conflicting needs implied by different quality property requirements (e.g. security often being in conflict with performance).
>
> The result of this process should be an architecture that meets the needs of the system's stakeholders.

Of course, this is a greatly simplified process. In reality, experienced architects consider quality properties throughout the architecture process and perform a lot of these activities concurrently. Never the less, we find the simplification useful when explaining how viewpoints and perspectives can be used to create an architecture.

Architectural Perspectives

- Our initial core set for information systems
  - Performance and Scalability
  - Security
  - Availability and Resilience
  - Evolution
  - Also: Location, I18N, Usability, Regulation, …
- Different sets in different domains
- See appendix for outline definitions

43

Like viewpoints, perspectives exist in sets, a particular set being aimed at a particular domain (such as information systems, embedded systems, mobile systems and so on).

We have defined a set of perspectives, to work with out viewpoint set, aimed at large scale information systems. Our core perspectives are:

- Performance and Scalability - will the system have the capacity and performance today, and be able to scale to tomorrow's demand?
- Security - can the resource owners in the system control access to the system and can the system recognise and recover from security breaches?
- Availability and Resilience - will the system's functions be available when people need to use them? Can the system's availability survive the failure of one or more elements?
- Evolution - can the system be changed over time as required?

We chose these four as our core set as they are relevant to nearly all information systems. Other perspectives, that we have provided outline definitions for, that may be relevant to some systems include Location, Internationalisation, Usability, Development Resource, Regulation and so on.

Other Perspectives

Accessibility	Can the system be used by people with disabilities?
Development Resource	Can the system be built within people, time and budget constraints?
Internationalisation	Is the system independent of language, country and culture?
Location	Will the system work, given its required geographical constraints?
Regulation	Does the system meet any required regulatory constraints?
Usability	Can people use the system effectively?

44

This slide summarises the other information systems perspectives that we have outlined in our book. These perspectives are not relevant to all information systems, but many systems may need to consider one or more of them.

- The Accessibility perspective examines the ability of the system to be used by people with disabilities.
- The Development Resource perspective examines the ability of the system to be designed, built, deployed, and operated within known constraints related to people, budget, time, and materials.
- The Internationalisation perspective examines the ability of the system to be independent from any particular language, country, or cultural group.
- The Location perspective examines the ability of the system to overcome problems brought about by the absolute location of its elements and the distances between them .
- The Regulation perspective examines the ability of the system to conform to local and international laws, quasi-legal regulations, company policies, and other rules and standards.
- The Usability perspective examines the ease with which people who interact with the system can work effectively.

Talking Point: Perspectives

- Going back to your qualities/views matrix
- Which perspectives would help you to achieve your quality properties?
  - Are they in the primary or secondary sets?
  - Useful feedback for us if in secondary
- Where may you have conflicts between advice in different relevant perspectives?

45

Talking Point: Perspectives

- Decide on the most important qualities for the system you've been considering
  - Limit this to the top 3
- For the most important (or interesting) property identify the likely impact of applying its perspective
  - How does achieving that quality affect the architecture?

Content

- Introducing Software Architecture
- Stakeholders
- The Software Architecture Challenge
- Using Viewpoints for Architectural Structures
- Using Perspectives for Architectural Qualities
- Example Application
- Uses for Viewpoints and Perspectives

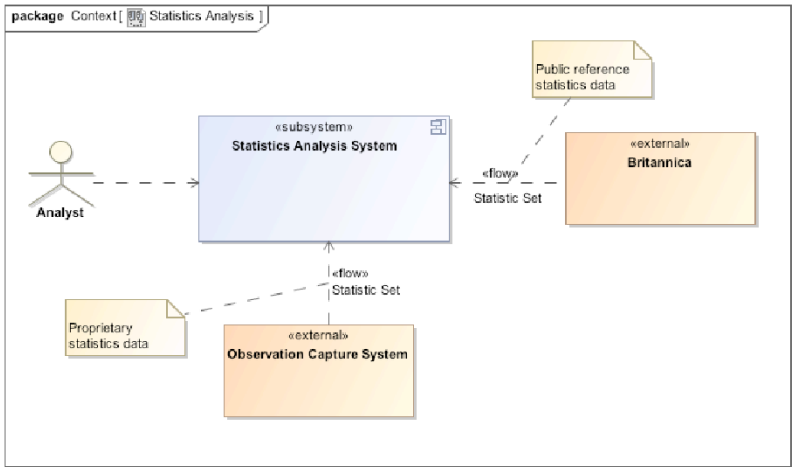
47

Example Application

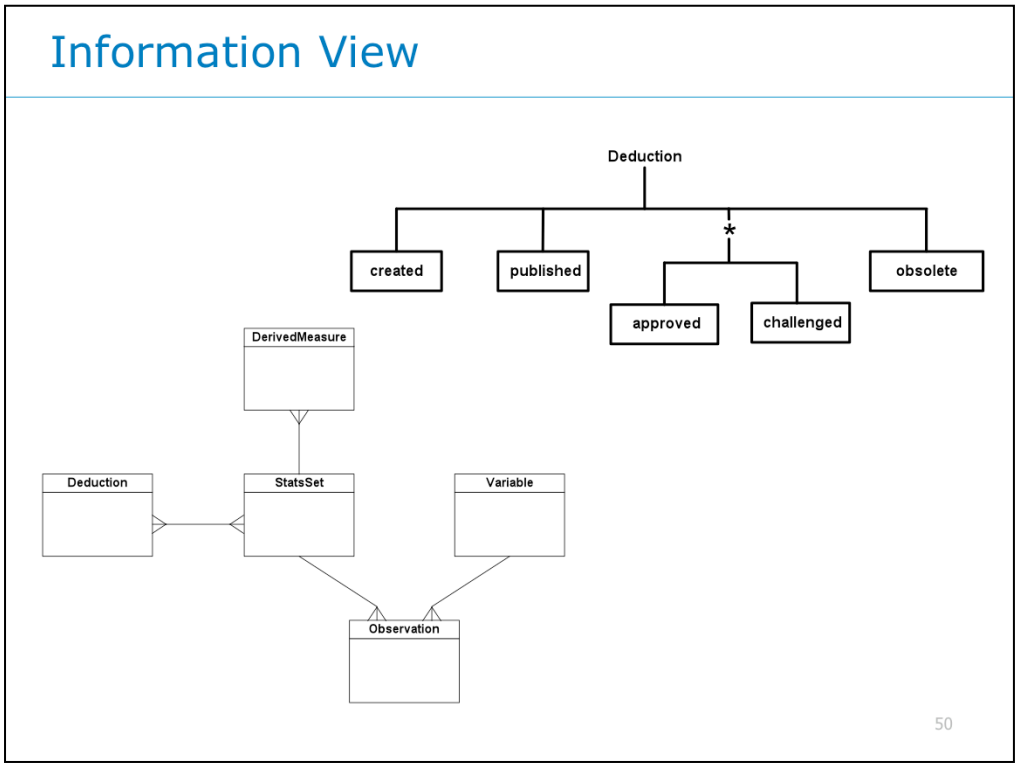
- Simple example of viewpoints and perspectives
- Used throughout the tutorial materials
- Statistics storage and processing system
  - Data loaded into the database
  - Derived measures calculated automatically
  - Statisticians view and report on the data
  - Deductions recorded and reviewed manually

48

Context View



49



Fragment of the system's information view reproduced here from earlier in the presentation.

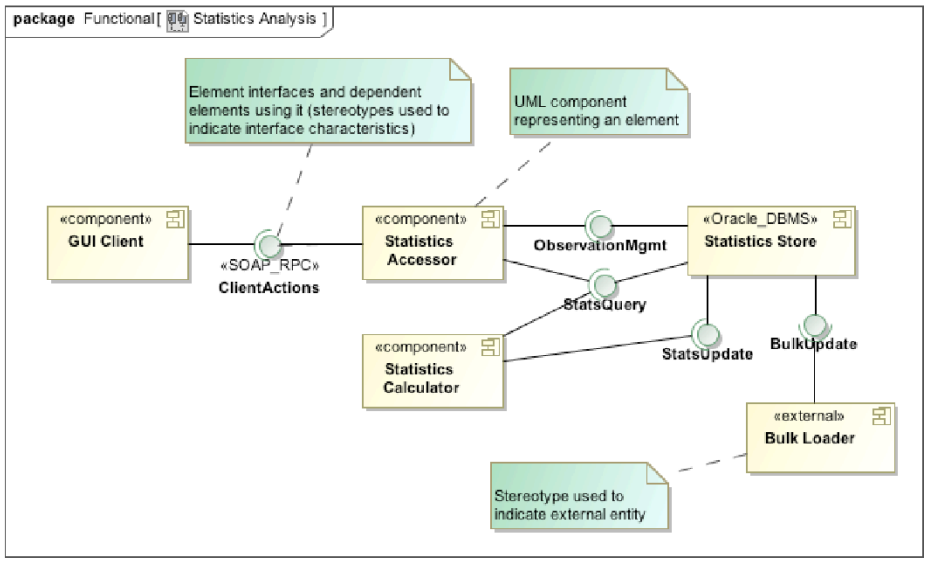
ERD shows us:

- The system stores definitions of *Variables* it is monitoring.
- *Observations* exist for a variable, each observation is a value captured at a point in time.
- A *Statistics Set* collects a set of *Observations* that are related (presumably captured at the same time).
- A *Derived Measure* is a derived statistic created by running a statistical calculation on the *Statistics Set*.
- A *Deduction* can be made manually from one or more *Statistics Sets* and related to them.

The Entity Life History shows us:

- A deduction is initially *created*.
- Then, a deduction has to be *published* before being visible to other users.
- The deduction can then repeatedly be *approved* or *challenged* by other users. It can only be in one state or the other (so challenging a deduction prevents it being approved).
- If the deduction is no longer relevant, it can be marked as *obsolete* (but note, it cannot be destroyed).

Functional View

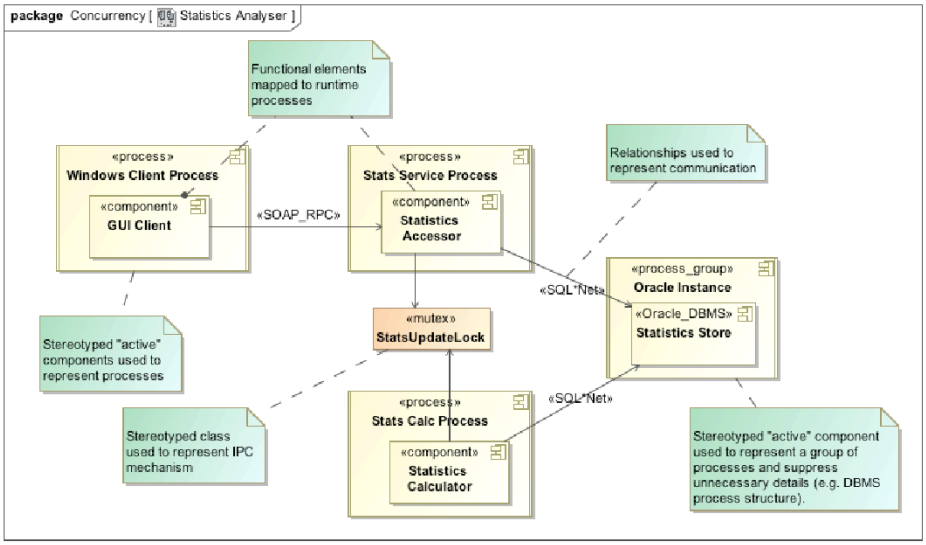


51

Fragment of the system's functional view reproduced here from earlier in the presentation.

•*GUI Clients* access the system via the *Statistics Accessor* server element/component's *ClientActions* interface (which would need defined properly elsewhere). The interface they use is obviously some sort of web services interface as it uses the SOAP protocol, according to the tagged value.
•Statistics are stored in the *Statistics Store* element, which offers three distinct interfaces, one to query statistics, one to update statistics and one to allow management (create, retrieve, delete, …) observations,
•The *Statistics Calculator* element is responsible for calculating the derived measures and so both reads and writes statistics via the *StatsQuery* and *StatsUpdate* interfaces.
•The *Bulk Loader* element is marked as "external" and so is an element that interacts with our system (and is probably something like Oracle's SQL*Loader or Sybase's BCP bulk loading utility programs).

Concurrency View

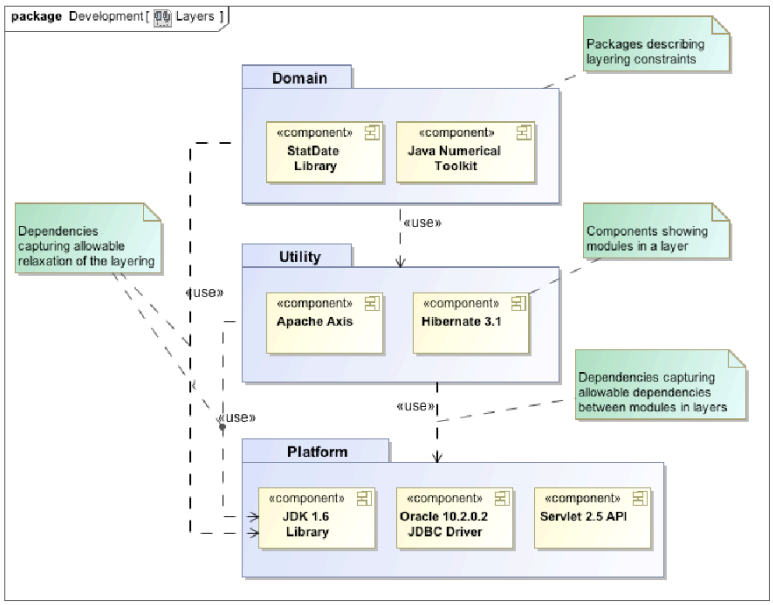


52

Fragment of the system's concurrency view reproduced here from earlier in the presentation.

•The *GUI Client* element is packaged into a *Stats_Client* process, of which many run concurrently.
•The *Stats_Client* processes all interact with a single *Stats_Server* process, containing the *Statistics Accessor*.
•The *Statistics Calculator* has been packaged into a *Calculator* process, which coordinates its access to the *Statistics Store* with the *Stats_Server* process using the *ExclAccessMutex*.
•The *Statistics Store* is packaged as a group of processes, *DBMS_Process_Group* (presumably the processes of a commercial DBMS).
•The *Bulk Loader* runs as its own process, *Loader*.
•The *Stats_Client* communicates with the *Stats_Server* using SOAP over HTTP, while the other processes communicate using Oracle SQL*Net.

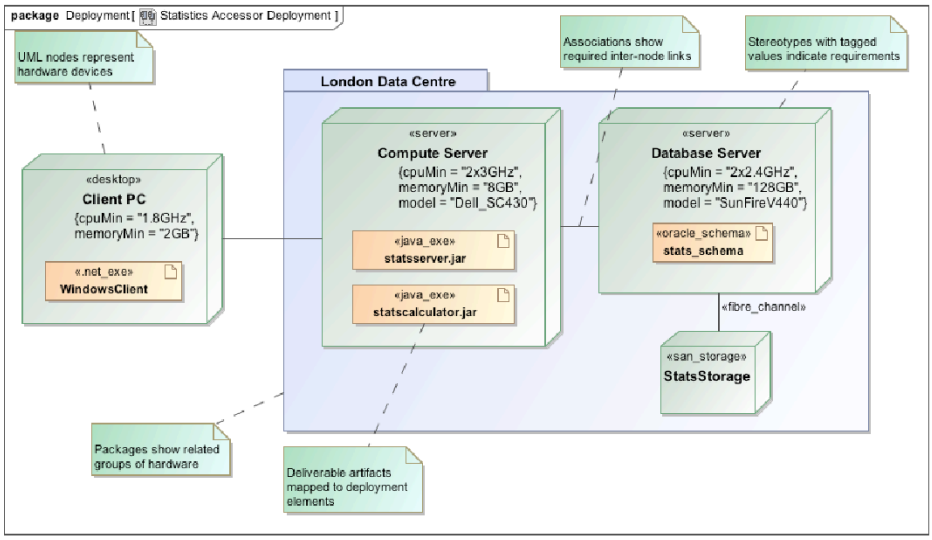
Development View



Fragment of the system's development view reproduced here from earlier in the presentation.

•The code modules in the system have been separated into three layers of abstraction: *Domain*, *Utility* and *Platform*.
•The *Domain* modules can access all of the *Utility* modules and the *Java 1.4 Library* module in the *Platform* layer.
•The *Utility* modules can access all of the modules in the *Platform* layer.

Deployment View



Fragment of the system's deployment view reproduced here from earlier in the presentation.

•The *Stats_Client* processes all run on *Client PC* nodes, with at least 500MB of memory and 1.8GHz processors (presumably WinTel PCs, although that's not stated … see next slide).
•The *Primary Server*, *Database Server* and *Disk Array* nodes all run in a data centre.
•The *Primary Server* hosts the *Stats_Server* and *Calculator*, running on a specific model of Dell server with the specified memory and CPU resources.
•The *Database Server* hosts the *DBMS_Process_Group* and the *Loader* process, running on a specific Sun server model, again with specified memory, CPU and specialised IO interface resources.
•The *Disk Array* is connected to the *Database Server* via a fibre channel interface (according to the tagged values) and is a specific model with specific capacity (but no specified disk layout, at least here).

We're assuming certain standard network specifications between the machines, but if this was complex or critical, we would create a network model to clearly communicate the connectivity we require.

Deployment View (ii)

Client PC	▪ Windows XP SP1 ▪ Java JRE 1.4.2_06 or later ▪ Internet Explorer 6.0 SP1
Primary Server	▪ Windows 2003 server, w/sec patches ▪ Java SDK 1.4.2_06 or later ▪ Apache Tomcat 5.5.9 or later

Database Server	▪ Solaris 9.0 w/Aug05 patch cluster
	▪ Oracle 9.2.0.2 Std Edition
	  ▪ 10GB buffer cache, auto sized SGA
	  ▪ auto storage management, 2 table spaces
	▪ OEM 9.2.0.2 installed and working

55

Fragment of the system's deployment view reproduced here from earlier in the presentation.

•The Client PC nodes run Windows XP, SP1 and need particular versions of IE and the JRE.
•The Primary Server node runs Windows 2003 Server with unspecified security patches (presumably the latest recommended at all times) and needs a specific JDK and Tomcat installed.
•The Database Server runs Solaris 9, with a specific set of patches applied. A specific version of Oracle should be installed and a couple of critical configuration items are noted along with the need to install the Oracle Enterprise Manager tool as well as the core DBMS.

Recap: Operational View

- Omitted from slides for space reasons.
- Would include:
  - Operational CM approach
  - Monitoring and Control
  - Operational Needs
  - Installation / migration / backout strategies

56

We omit the operational view here, as we did in the previous slides, because operational views tend to be bulky “text and tables” views (usually captured in dedicated documents and just summarised in the main AD). However, for this system we’d expect it to address any required operational configuration management, how the system will be monitored and controlled, any operational needs it has (routine or exceptional) and how it will be installed, migration managed and/or backed out if it all goes horribly wrong.

Example: Applying Perspectives

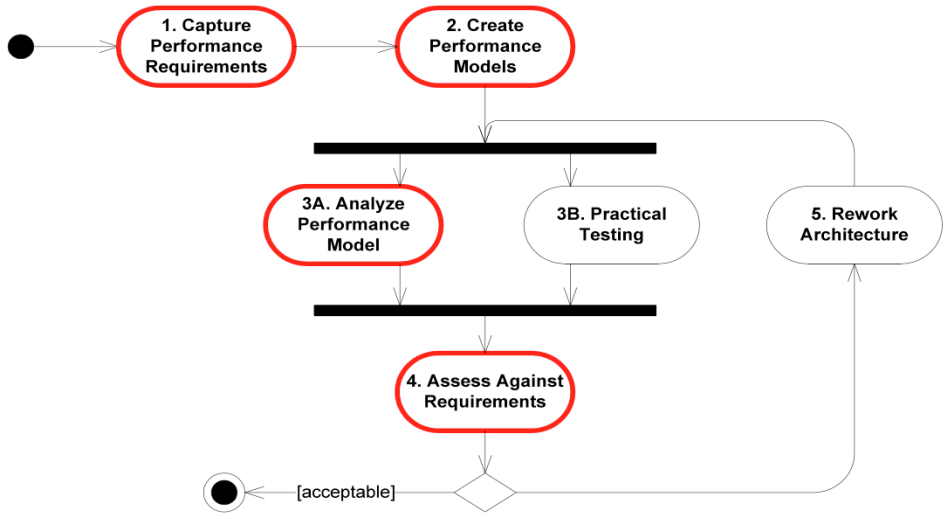
- Performance and Scalability
  - Capture P & S Requirements
  - Create Performance Models
  - Analyse Models
  - Perform Practical Testing
  - Assess Against Requirements
  - Rework Architecture (apply tactics)
- What affect will this have on our system?

57

As an exercise, let's consider what would happen to our views if we applied some perspectives.

Consider the performance and scalability perspective first. Let's assume that the system needs to support a concurrent load of 10s of users and a potential load of 100s of concurrent users. Is the current implementation suitable? How will you assess that? What will you do if not?

Example: Applying Perspectives



58

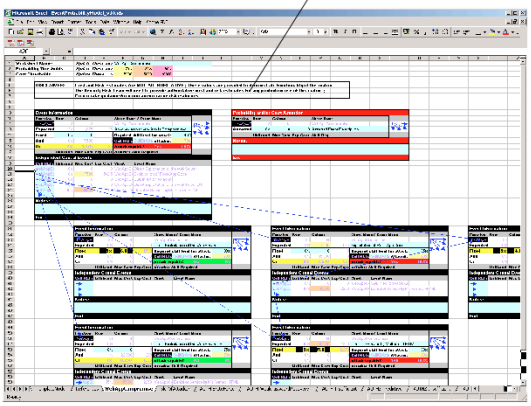
The process captured in the performance and scalability perspective is reproduced here for convenience. The activities highlighted in red are those that we focus on here.

Example: Applying Perspectives

Measure	Value
Network A	100Mb
DB access (ro)	20ms
DB access (rw)	50ms
Single derived calc	1400ms
Client network	10Mb
Bulk load 100K	2500ms
Online load users	100
Memory per user	1MB

Calibration measures

Performance model



59

•The first step is to make sure we have performance requirements and to validate them. You may have good performance requirements gathered for you and you just need to sanity check them. Conversely you may well need to gather your own performance requirements because no one else has thought of doing this!
•The second activity to consider is that we need to do some performance modelling, which we illustrate on this slide. Typically this involves creating a quantitative performance model of the system's essential operations (e.g. in Excel) and then calibrating this model with actual measures taken from a real or representative test environment. The output of this process will be some performance estimates for the system's most important operations.
•We'd use the model for various sorts of analysis to check how sensitive the system is likely to be to variable factors (e.g. database size or network latency). This allows us to assess performance risks.
•In parallel with performance modelling and analysis, we'd do practical testing to validate it and discover more about our likely performance characteristics.
•At the end of this process we'd be in a position to consider whether our likely performance is going to meet our requirements or not. The answer is usually "no" and we need to rework the architecture in some way, but for the purposes of this example let's assume we've been lucky (or very clever!) and we'll easily meet our performance requirements and move onto the next quality property.

Example: Applying Perspectives

- Security
  - Identify Sensitive Resources
  - Define Security Policy
  - Identify Threats to the System
  - Design Security Implementation (apply tactics)
  - Assess Security Risks
- What affect will this have on our system?

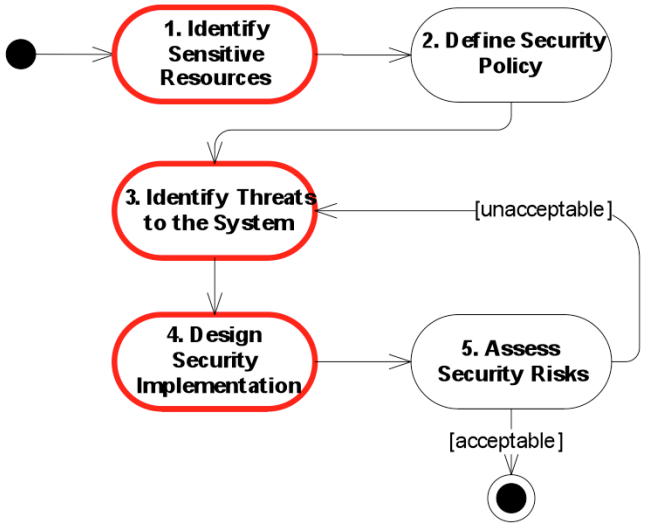
60

Continuing the exercise, let us move on to consider security.

So far, we've talked about the system in very generic terms, without mentioning the kind of information it stores. Now let's address this and assume that this system is actually an intelligence analysis system for a government agency.

Is the system "secure" enough? What resources are sensitive? (Names, addresses, operational details?) What policy is needed? What threats does the system face? (Operators taking backups away? Administrators accessing all data in the database? Internal network attacks? Bribing investigating officers?) What countermeasures are possible?

Example: Applying Perspectives



61

The process captured in the security perspective is reproduced here for convenience. The activities highlighted in red are those that we focus on here.

Example: Applying Perspectives

- Sensitive Resources
  - The data in the database
- Security Threats
  - Operators stealing backups
  - Administrators querying data, seeing names
  - Bribing investigating officers
  - Internal attack on the database via network

62

•The first step in applying the security perspective is to work out what resources in the system are valuable or sensitive and so under threat of misuse. For our system it's the data in the database (as it doesn't perform any operations that affect the outside world).
•We then need to work out what threats the system faces that could allow misuse of the database data. Some of the main ones are:

•Operators taking backups of the database offsite (e.g. on a flash drive these days!) and passing them to other interested parties who could restore them and access the data.

•Administrators seeing the names of suspects and targets as a side effect of querying the data in the database when performing routine maintenance or troubleshooting. The problem with this, even if we trust the administrators, is simply information leakage. Given the types of organisation and individual monitored by this database, even knowing their existence could be hazardous to health!

•A simple social engineering attack by bribing or blackmailing an investigating officer is probably the simplest way to get some of the data out of the system. The investigating officers can legitimately run a wide variety of reports against the system and if they can be persuaded to extract these without a high chance of detection then this is a very promising attack channel. Many people become vulnerable to bribery or blackmail at some time.

•The system runs in the organisation's data centre and so sends data over the organisational network so anyone with physical access to the network could potentially attack the database using a range of techniques including password guessing, known vulnerability exploitation or brute force attacks.

Example: Applying Perspectives

- Security Countermeasures
  - Backups: encrypt data in the database
    - How about performance?
    - Does this make availability (DR) harder?
  - Hiding names: use codes instead of names, protect the underlying names at a higher security level
    - More development complexity
    - Possible performance impact

63

•We then consider possible countermeasures against the threats we've identified.

•To avoid backups being stolen, we could consider encrypting the data in the database so that the backup is difficult to use. However this may slow down the database significantly (particularly for large pieces of data) and encryption implies the need for keys. The right keys (which presumably need to be changed regularly) need to be available in any disaster recovery environment, which is another thing to go wrong during failover.

•To avoid system administrators from accidentally seeing subject and target names then we could isolate the names in a separate table that most of the system's operations don't access (so needs little administrative attention) and we could protect that table at a higher level to other data. However, this adds development complexity and could slow down data retrieval where the names are needed because of the need to access another table.

Example: Applying Perspectives

- Security Countermeasures
  - Bribery: add audit trail for data access
    - Possible performance impact
    - More complexity
    - Protecting / using the audit trail
  - Network Attacks: harden database, firewalls, IDS
    - More deployment & administrative complexity
    - More hardware and operational cost
    - Operational impact if IDS trips

64

•More countermeasures …

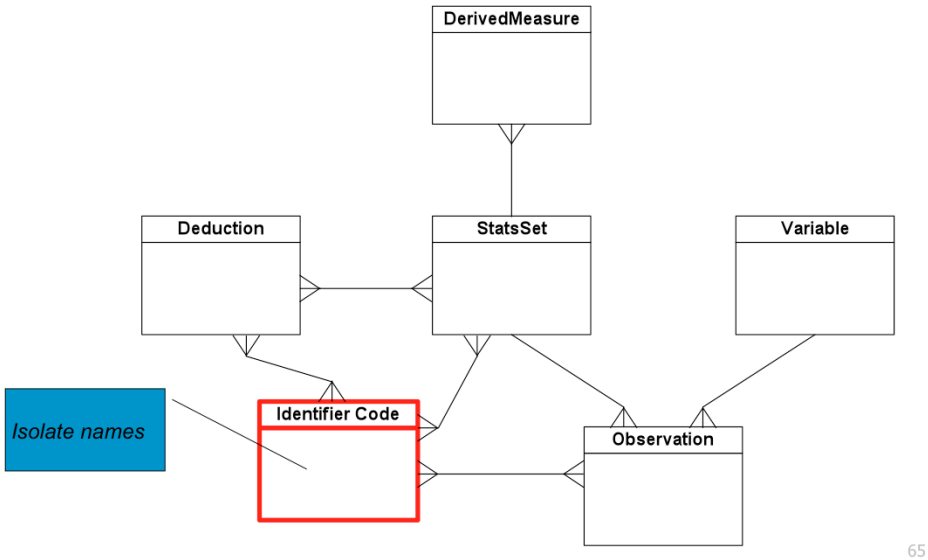
•To guard against the possibility of bribery and blackmail, we need to make sure that investigating officers can't retrieve data from the system without being detected. So we need to add an audit trail that can be checked regularly for odd patterns of access that would allow further investigation. However audit trails are themselves hard to implement securely and often add a significant runtime overhead as well as adding effort and complexity to the development process.

•To avoid the danger of the database being attacked, we need to harden the database and make sure that it's security configuration and patch level are monitored and updated regularly. We should also employ firewalls to filter the traffic that can get to the database and an intrusion detection system on all of the server hosts to spot possible suspicious activity. However, as you might expect, all of this can be expensive and costs money to implement, monitor and run.

We now turn to consider how the system's architecture is affected by our application of the perspectives. Luckily, in this case, the performance of the system appears to be fine, but its security needs some attention.

Example: Applying Perspectives

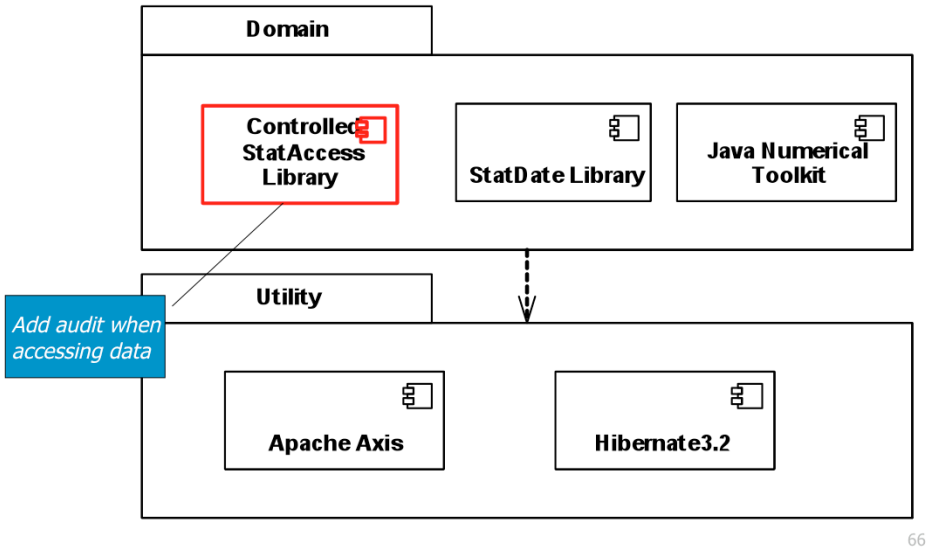
Impact on information view …



Firstly, considering the information view, an example of a change needed is going to be a change to the core data model to introduce the name/identifier code map and reference this from other tables in order to isolate subject and target names in a separate part of the database.

Example: Applying Perspectives

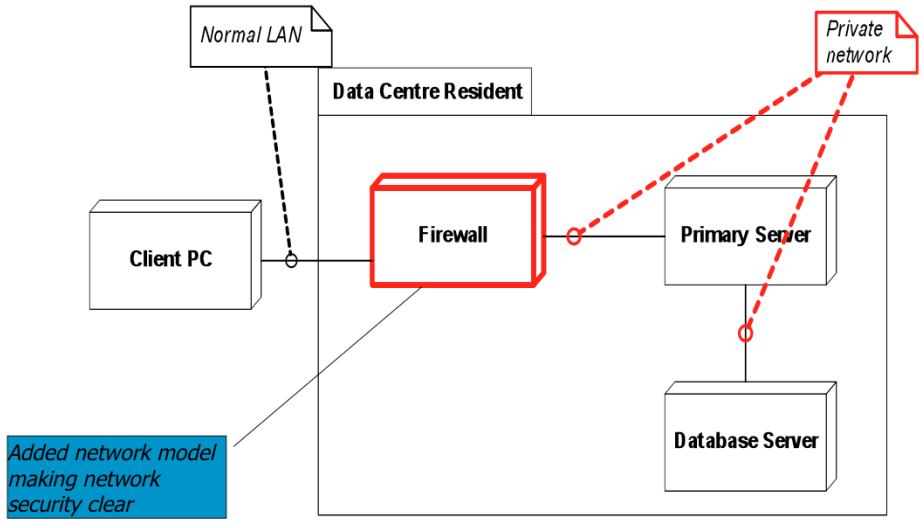
Impact on development view …



In the development view, an example of a change needed is the need to audit access to statistics and reports means that we'll need to enhance our StatAccess library in order to add auditing to it.

Example: Applying Perspectives

Impact on deployment view …



Moving on to the deployment view, an example of changes needed are to add the network requirements to improve the security of the database (adding a firewall and isolating the LAN sections linking the system's servers together).

Example: Applying Perspectives

- Other Impact
  - Need IDS added to Development view
  - Database security
  - Need to capture impact on Operational view
  - Need to consider impact on availability
  - Need to re-work performance models to allow for database encryption, audit, …
- Note the need to change many views to address security needs

68

Of course we really are just skimming the surface here in the interests of clarity and time. There would be a lot more activity than these few small changes, but they give you a flavour of the sort of impact that applying a perspective tends to have. It usually results in a list of related changes right across the architecture. These changes themselves can then have a knock on effect, needing further analysis, such as updating performance models to allow for performance compromises that have been introduced in order to meet other quality requirements.

Content

- Introducing Software Architecture
- Stakeholders
- The Software Architecture Challenge
- Using Viewpoints for Architectural Structures
- Using Perspectives for Architectural Qualities
- Example Application
- Uses for Viewpoints and Perspectives

Using Viewpoints & Perspectives

- A framework for organising work
- A store of knowledge
  - Document proven practice
  - Help standardise language and approach
  - Help to standardise languages and approaches
- Applicable at different career stages
  - Mentor novice architects
  - Guide working architects
  - Support expert architects

70

Fundamentally, viewpoints and perspectives provide two benefits: they can act as a framework to organise the architectural design process and they can act as a store of proven architectural design knowledge.
We have also found that viewpoints and perspectives can be useful to architects of different levels of experience, from novices to experts.

Using Viewpoints & Perspectives

- For Novice Architects
  - An introduction to each area of knowledge
  - A guide to what is important
  - A structure for the process
  - Definitions of standards and norms
  - Repository of proven practice and tactics
  - Pitfalls and solutions to avoid common errors
  - Checklist to ensure nothing is forgotten

71

For a novice architect, a set of viewpoints and perspectives can provide them with a learning framework that provides an overview of the core knowledge that they are likely to need in order to be successful. They can guide the architect's focus as they learn and provide reliable definitions of standard terms and approaches. The proven practice and tactics in the set provides the architect with a set of solutions to common situations that they will face, while the pitfalls (and solutions) and the checklists allow them to learn from the experience of others.

Using Viewpoints & Perspectives

- For Working Architects
  - A reminder of what is important
  - A guide to new or rarely used areas of practice
  - Repository of proven practice and tactics
  - Pitfalls and solutions to avoid common errors
  - Checklist to ensure nothing is forgotten

72

The fairly experienced working architect will find that viewpoints and perspectives are a useful reminder of what to focus on when considering a type of architectural structure of a particular quality property. They can also help extend the architect's knowledge in to new areas as their experience grows and again, the set of proven practice and tactics provide a basic knowledge base to work from. Perhaps most usefully, the pitfalls (and solutions) and checklists help the architect to avoid the mistakes that other heads with more grey hairs have already made.

Using Viewpoints & Perspectives

- For Expert Architects
  - A framework to allow knowledge sharing
  - An aid to tutoring and mentoring
  - Checklists to ensure nothing is forgotten

73

The experienced expert architect may well also find viewpoints and perspectives to be useful in their work. Part of the expert's role is to lead and mentor others, sharing best practice. A viewpoint and perspective set helps them to do this by providing a knowledge sharing framework and a concrete set of artefacts to use when mentoring less experienced architects. They may also find that the checklists provided are useful for themselves, particularly to avoid problems with half-remembered previous experiences!

Summary

- Architecture Essential Difficulties
  - Multi-dimensional problem, no right answer
  - Stakeholder needs conflict
  - Complex mix of people and technology
  - Tradeoffs are inevitable
- Architecture Accidental Difficulties
  - Lack of standardisation (approach, artefacts)
  - Little sharing of knowledge and experience

74

So to summarise what we've covered in this session …

Firstly, software architecture is an essentially complex business that is difficult to do well. Software architecture is a complex, multi-dimensional problem covering a range of factors including functional structure, information, security, concurrency, deployment, performance, maintainability and more. It is the sort of problem, typical to engineering systems design, where you don't really have a "right" answer, but rather a series of possibly good enough ones to choose from. Part of this problem stems from the fact that the system's stakeholders all have different, usually conflicting, agendas. Developing a system involves a large, complex mix of people and technology, any of which often has the ability to cause your system to fail and you need to manage this mix.
Given this environment, making tradeoffs is an inevitable, but difficult and often unenviable part of the role.

There are also some accidental difficulties that arise from our state of practice. In particular, we've only recently started to standardise very much across the industry (even at the conceptual level, of IEEE 1471, where it might help communication and understanding without being too constraining). The result of this is that every architect ends up inventing their own architecture process and defining their own set of useful architectural artefacts. A related problem is that we're not very good at sharing hard-won knowledge and experience. This means that we don't tend to use standard solutions to common problems but rather each architect tends to end up learning their own lessons, often at the cost of our stakeholders.

Summary (ii)

- Viewpoints and Views
  - Views provide a convenient approach for effective architectural description
  - Viewpoints standardise views by defining their content
  - Viewpoints contain proven architectural knowledge for a particular domain
  - Viewpoints and views can address many accidental difficulties of software architecture

75

Many of the accidental difficulties of software architecture can be at least partially addressed by using an approach based on viewpoints and views.

Views provide us with the fundamental structuring mechanism to allow us to describe our architectures as a coherent set of related, but distinct, descriptions, unified into an architectural description. However, by themselves, views are little more than a simple documentation convention and still leave much to the individual architect's intuition and experience.

Viewpoints strengthen the approach by providing a set of standard templates, applicable to a particular domain, that guide the development of each view. Viewpoints share architectural knowledge by communicating effective models to use in views of particular types, explaining how to go about building a view, highlighting potential problems in that area (and suggesting solutions) and providing focus by defining the stakeholders interest in the view in question.

That said, to date, viewpoints have only been used to address the question of designing and describing architectural structures, with system quality properties being handled by the architect's intuition and experience – the very problem we are trying to assist with. In theory, views (and viewpoints) can be created for quality properties but, to the best of our knowledge, no one has actually done this and as we explained earlier, we don't think it's a good thing to try.

Summary (iii)

- Viewpoints for Information Systems
  - Context
  - Functional
  - Information
  - Concurrency
  - Development
  - Deployment
  - Operational

76

A possible viewpoint set to guide the architectural design of large scale information systems can be found in our book (see `http://www.viewpoints-and-perspectives.info`), which contains the following viewpoints:
•Context – the relationship of the system and its external environment
•Functional – the functional structure, elements, responsibilities, interfaces and interactions of the system.
•Information – the information structure, ownership, flow, latency and access in the system.
•Concurrency – how the functional structure will be packaged as processes and threads to allow it to be executed.
•Development – the architectural constraints that are important to impose on the software development and integration process.
•Deployment – the system's runtime environment in terms of nodes, links, process to node mappings and hardware/software dependencies on each node.
•Operational – the set of strategies and requirements that will allow the system to be installed, migrated to, operated, controlled and supported.

Summary (iv)

- Perspectives
  - Viewpoints handle structure well, less convinced about quality properties
  - Perspectives provide similar guidance and knowledge sharing for quality properties
  - Perspectives suggest the design activities required to achieve a property
  - Perspectives provide proven practice, pitfalls, solutions and checklists to share experience
  - ***Applying perspectives modifies views***

77

We did try to create views for quality properties (and viewpoints to define their structure) but we kept running into the problem that addressing a quality property typically impacts a number of architectural structures (improving security might affect the functional, deployment and development views for example). What kept happening was that our "quality views" duplicated a lot of information in other views, as well as resulting in changes to the views themselves. For anything other than a trivial system, this duplication rapidly meant that the architectural description became very hard to change and so just fell out of use.

Our solution to this problem was to introduce a new concept: the architectural perspective. Perspectives have similar goals to viewpoints, around standardising approaches and sharing proven knowledge, to avoid mistakes that have already been widely made.
A particular perspective suggests a simple set of activities that the architect should perform in order to ensure that their system exhibits a particular quality property. These activities generally involve understanding the requirements, analysing the system against those requirements (often by creating an ancillary artefact in the shape of a model) and applying architectural tactics until the quality related behaviour of the system is acceptable.
Perspectives provide proven practice in the shape of architectural tactics to apply, activities to perform, pitfalls to be aware of, solutions to try if the pitfalls emerge and checklists to help the architect to avoid overlooking important factors.

Perspectives are intended to be defined in sets, a set being applicable to a particular systems domain, with each perspective in the set addressing a particular quality property (e.g. performance or security). Applying a set of perspectives to a system will almost certainly result in the views describing the system being modified in order to address the quality properties under consideration. Note that the perspectives *don't* appear in the architectural description – they are guides to modifying the existing views in the AD.

Summary (v)

- Perspectives for Information Systems
  - Availability and Resilience
  - Evolution
  - Performance and Scalability
  - Security
  - Others
    - Accessibility, Development Resource, Geographical Location, I18N, Regulation, Usability

78

We have found four perspectives critical for most large information systems:
•Availability and Resilience – will the system be available when the stakeholders need it, even if things go wrong?
•Evolution – will the system be amenable to change, when needed?
•Performance and Scalability – will the system be able to process its workload quickly enough and can you increase its capacity at reasonable cost and effort?
•Security – can the owners of sensitive resources in the system control access to them reliably, tell when security has been breached and recover from this?

We have produced complete definitions of these core perspectives in our book.

A large number of other perspectives could be used with information systems, of which we have produced outline definitions of 6 in the book.
•Accessibility – can the system be used by everyone who needs to use it?
•Development Resource – can the system be created given the resource constraints you face?
•Geographical Location – can the system operate from the geographical locations that it will need to be installed in?
•Internationalisation – is the system independent of currencies, formats, languages and other locale specific aspects and can these facets of the system be changed easily for localisation?
•Regulation – will the system meet the regulatory constraints that it is expected to operate under?
•Usability – can the users of the system use the system effectively for their tasks?

Summary (iv)

- Using Viewpoints and Perspectives
  - Novices
    - Overview, guides, focuses attention
    - Proven practice, pitfalls, solutions and checklists
  - Working Architects
    - Reminder of existing knowledge
    - Aid in new areas
  - Experts
    - Mentoring and communication vehicle
    - Reminders of hard won lessons

79

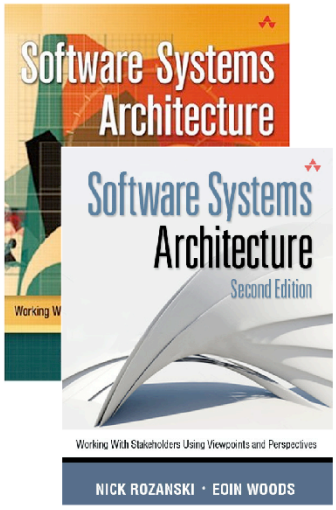
We have found that viewpoints and perspectives can be used by architects of vastly differing experience and knowledge.

•Novices use them primarily as educational and learning aids, guiding their development and helping them to avoid mistakes.
•Working architects use them to reinforce existing knowledge and expand their competence to new areas, as well as being useful aide memoirs when working.
•Experts find that they can use viewpoints and perspectives to capture their hard won knowledge, to allow it to be used for mentoring and teaching less experienced architects, as well as being useful aide memoirs to themselves when working.

To Learn More

Software Systems Architecture
Working With Stakeholders Using Viewpoints and Perspectives

Nick Rozanski & Eoin Woods
Addison Wesley, 2005/2011

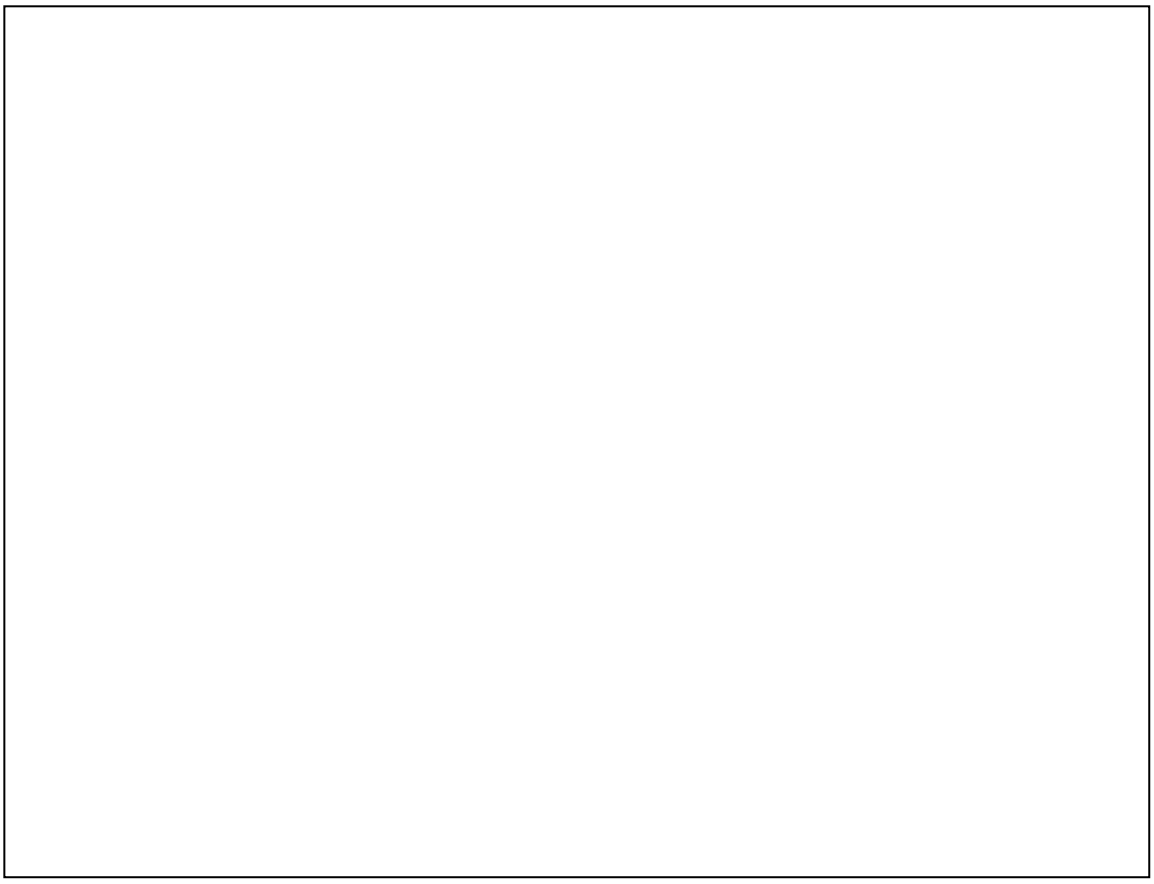


http://www.viewpoints-and-perspectives.info

Eoin Woods
contact@eoinwoods.info
www.eoinwoods.info

Comments and Questions?

Nick Rozanski
nick@rozanski.org.uk
www.nick.rozanski.org.uk



Appendix

Viewpoint Descriptions

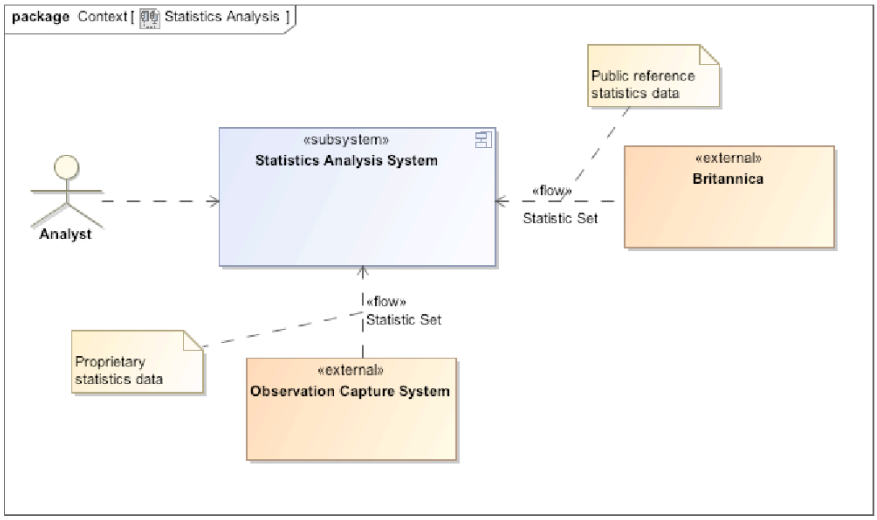
Context Viewpoint

Focus	scope and external environment of the system
Content	design of runtime functional elements and their responsibilities, interfaces, and primary interactions
Concerns	▪ scope and responsibilities ▪ external entities, services and data dependencies ▪ external interfaces ▪ impact of system on its environment
Models	▪ context model, interaction scenarios
Pitfalls	▪ missing or incomplete external elements ▪ uneven focus across the environment ▪ loose or incomplete interface definitions ▪ Unnecessarily complicated interactions [...]

83

The complete pitfall list for the Context Viewpoint is:
• Missing or incorrect context model elements
• Uneven focus
• Inappropriate level of detail
• Scope creep
• Implicit or assumed scope or requirements
• Missing implicit dependencies
• Loose or inaccurate interface descriptions
• Unnecessarily complicated interactions
• Overuse of jargon

Context View Fragment



84

Functional Viewpoint

Focus	functional structure of the system
Content	design of runtime functional elements and their responsibilities, interfaces, and primary interactions
Concerns	▪ functional capabilities ▪ external interfaces ▪ internal structure ▪ design qualities
Models	▪ functional structure model
Pitfalls	▪ poorly defined interfaces / responsibilities ▪ infrastructure modelled as functional elements ▪ unintentionally overloaded view ▪ diagrams without element definitions [...]

85

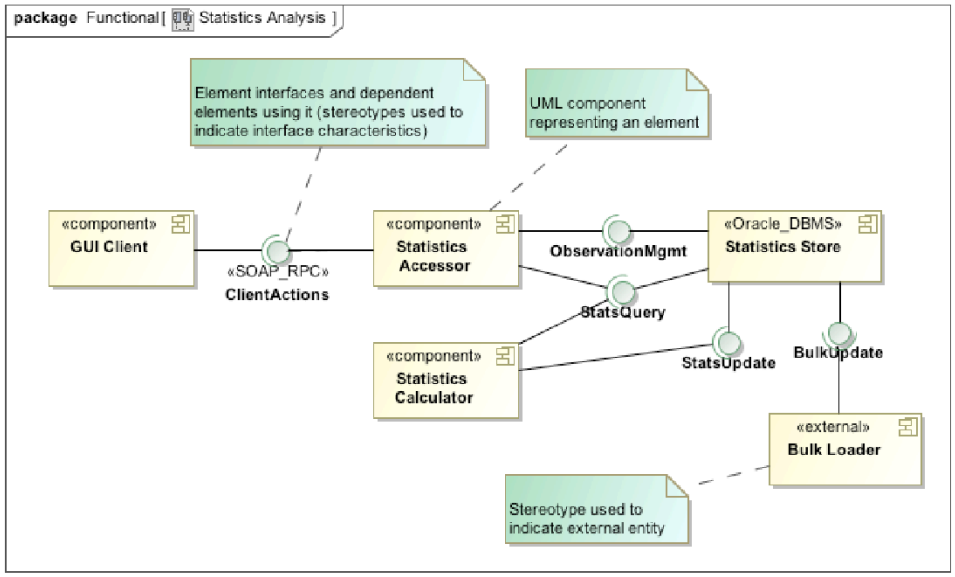
The complete list of functional pitfalls is:

- Poorly defined interfaces
- Poorly understood responsibilities
- Infrastructure modelled as functional elements
- Unintentionally overloaded view
- Diagrams without element definitions
- Difficulty in reconciling the needs of multiple stakeholders
- Including an inappropriate level of detail
- Ending up with “God elements”
- Having too many dependencies between elements

A useful set of questions to ask yourself in order to validate your functional view is:

- Do you have fewer than 15–20 top-level elements?
- Do all elements have a name, clear responsibilities, and clearly defined interfaces?
- Do all element interactions take place via well-defined interfaces and connectors
- Do your elements exhibit an appropriate level of cohesion and coupling?
- Have you identified the key usage scenarios and used these to validate the view?
- Have you checked the functional coverage of the architecture to ensure it meets its requirements?
- Have you considered how the architecture is likely to cope with possible change in the future?
- Does the presentation of the view take into account the concerns and capabilities of all interested stakeholder groups? Will the view act as an effective communication vehicle for all of these groups? If not, what alternative representations will you use?

Functional View Fragment



86

An example of a fragment of a functional view is illustrated on this slide. We suggest the use of a UML component diagram to show functional structure, showing system elements, the offered and required interfaces and any known constraints on the structure via tagged values. If there are different sorts of functional element, we use stereotypes to indicate this. Obviously, behind the picture, there needs to be a lot of text defining the model elements.

Note the use of stereotypes and tagged values. Given the relatively limited notation available in the UML component diagram, we have always found the need to embellish the model with stereotypes and tagged values to help people understand the details of the model we are trying to communicate. The potential pitfall with this approach is the amount of clutter that can end up on the diagram if you try to explain everything. The key is to pick out a few key details that need to be communicated and to explain the rest elsewhere (such as in the textual definitions that go with the diagram). An example in this model is the use of a tagged value to indicate that a SOAP interface is used between the Statistics Accessor and its clients, however the types of the other interfaces aren't tagged because we judged this information to be less important.

Information Viewpoint

Focus	information structure, ownership and processing
Content	design of storage, manipulation, management, and distribution of information
Concerns	▪ information structure and content ▪ information flow ▪ data ownership and quality ▪ timeliness, latency, and age ▪ references and mappings ▪ transaction management and recovery ▪ data volumes ▪ archives and data retention ▪ regulation

87

First half of the information viewpoint summary.

Information Viewpoint (ii)

Models	▪ static data and metadata structure models ▪ information flow models ▪ information lifecycle models ▪ data ownership and access models ▪ volumetric models
Pitfalls	▪ data incompatibilities ▪ poor data quality ▪ unavoidable multiple updaters ▪ key matching deficiencies ▪ poor information latency ▪ interface complexity ▪ inadequate volumetrics

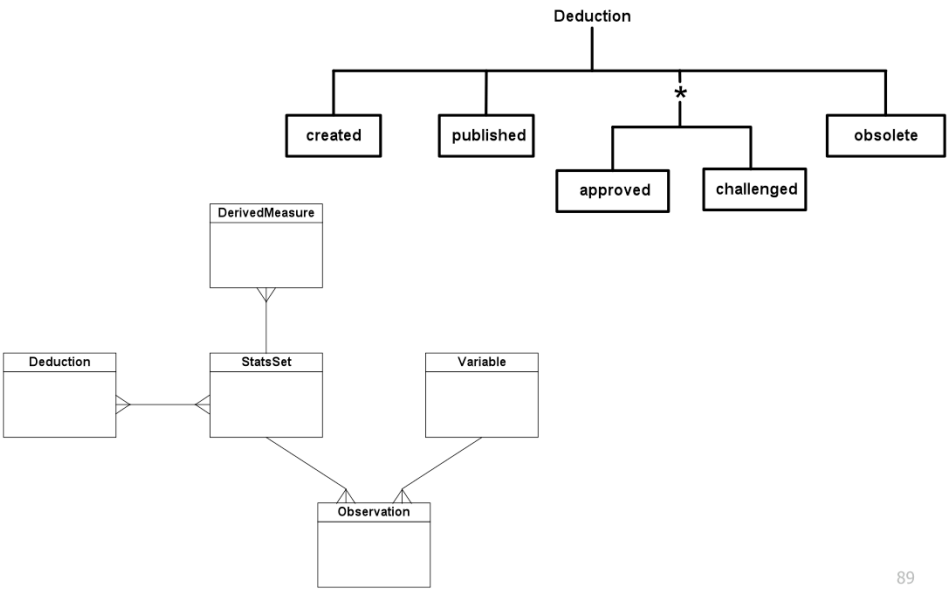
88

Second half of the information viewpoint summary.

A useful set of questions to ask yourself in order to validate your information view is:

- Do you have an appropriate level of detail in your data models (e.g., no more than about 20 entities)?
- Are keys clearly identified for all important entities?
- Where an entity is distributed across multiple systems or locations with different keys, are the mappings between these keys defined? Do you have processes for maintaining these mappings when data items are created?
- Have you defined strategies for resolving data ownership conflicts, particularly where there are multiple creators or updaters?
- Are latency requirements clearly identified, and are mechanisms in place to ensure these are achieved?
- Do you have clear strategies for transactional consistency across distributed data stores, and do these balance this need with the cost in terms of performance and complexity?
- Do you have mechanisms in place for validating migrated data and dealing with errors?
- Have you defined sufficient storage and processing capacity for archiving? And for restoring archived data?
- Has a data quality assessment been done? How do you deal with poor-quality data?

Information View Fragments

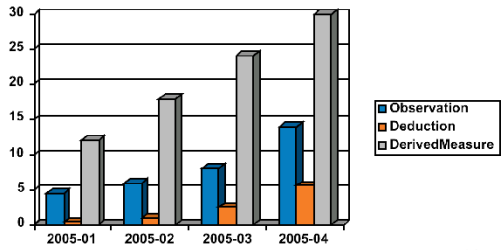


Example fragments of information view content are shown here.

- An entity relationship diagram (ERD) defining the key stored data and inter-relationships for the system.
- An entity life history (ELH) diagram, showing the states that the “Deduction” entity can pass through and the ordering constraints on these states. (In UML, you could use a state chart for this too … we use ELHs because they’re quite accessible, and for entity state changes, seem to be more easily understood by many people than state charts).

Information View Fragments (ii)

	Observation	Deduction	Derived Measure
Statistics Accessor	R	C,R,U,D	R
Statistics Calculator	-	-	C,U,D
Bulk Loader	C,U,D	-	-



90

Further example fragments of information view content are shown here.

•An access matrix for key information entities (showing which elements Create, Read, Update and/or Delete instances of a particular entity). This helps you to understand information ownership and to spot problematic situations such as having multiple updaters of key pieces of information.

•A volumetrics data set, capturing the number of each key entity that are expected over time, allowing you to validate this with key stakeholders (such as database administrators and domain experts) and to see the likely data scalability required.

Concurrency Viewpoint

Focus	packaging elements into processes and threads
Content	the concurrency structure, mapping functional elements to concurrency units to clearly identify the parts of the system that can execute concurrently, and how this is coordinated and controlled
Concerns	▪ task structure ▪ mapping of functional elements to tasks ▪ inter-process communication & re-entrancy ▪ state management ▪ synchronization and integrity ▪ task startup, shutdown and recovery from failure

91

First half of the concurrency viewpoint summary.

Concurrency Viewpoint (ii)

Models	▪ system-level concurrency model ▪ system-level state model
Pitfalls	▪ modelling of the wrong concurrency ▪ excessive complexity ▪ resource contention ▪ deadlock and race conditions

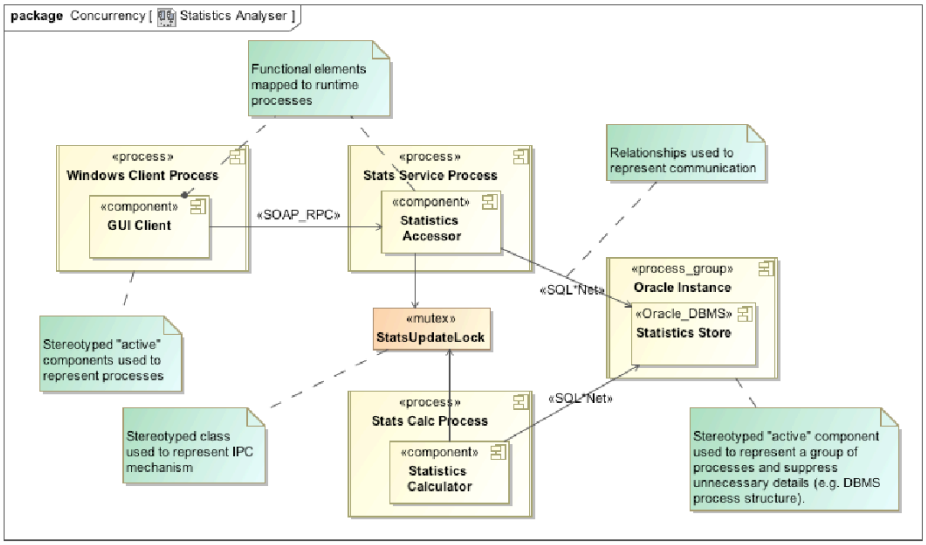
92

Second half of the information viewpoint summary.

A useful list of questions to ask yourself in order to validate your concurrency view is:

- Is there a clear system-level concurrency model?
- Are your models at the right level of abstraction? Have you focused on the architecturally significant aspects?
- Can you simplify your concurrency design?
- Do all interested parties understand the overall concurrency strategy?
- Have you mapped all functional elements to a process (and thread if necessary)?
- Do you have a state model for at least one functional element in each process and thread? If not, are you sure the processes and threads will interact safely?
- Have you defined a suitable set of inter-process communication mechanisms to support the inter-element interactions defined in the Functional view?
- Are all shared resources protected from corruption?
- Have you minimized the inter-task communication and synchronization required?
- Do you have any resource hot spots in your system? If so, have you estimated the likely throughput, and is it high enough? Do you know how you would reduce contention at these points if forced to later?

Concurrency View Fragment



93

An example of a fragment of a concurrency view is illustrated on this slide.

- A UML class diagram is used to show how the system's functional elements are packaged into processes, so that they can be executed.
- Stereotypes are used to indicate processes, threads and process groups (no threads are shown here).
- Interconnections between processes are shown, with tagged values being used to indicate the particular inter-process communication mechanisms to be used.
- Coordination mechanisms are shown as stereotyped classes, with associations being used to indicate use of the mechanism by particular processes.

Development Viewpoint

Focus	architectural constraints on the development process
Content	architectural design that supports and constraints the software development process
Concerns	▪ module organization ▪ codeline organization ▪ common processing ▪ standardization of design and testing ▪ instrumentation
Models	▪ module structure models ▪ common design models ▪ codeline models
Pitfalls	▪ too much detail or lack of precision ▪ overburdening the architectural description ▪ uneven focus or lack of developer focus ▪ problems with the specified environment

94

A useful list of questions to ask yourself in order to validate your development view is:

Have you defined a clear strategy for organizing the source code modules in your system?

Have you defined a general set of rules governing the dependencies that can exist between code modules at different abstraction levels?

Have you identified all of the aspects of element implementation that need to be standardized across the system?

Have you clearly defined how any standard processing should be performed?

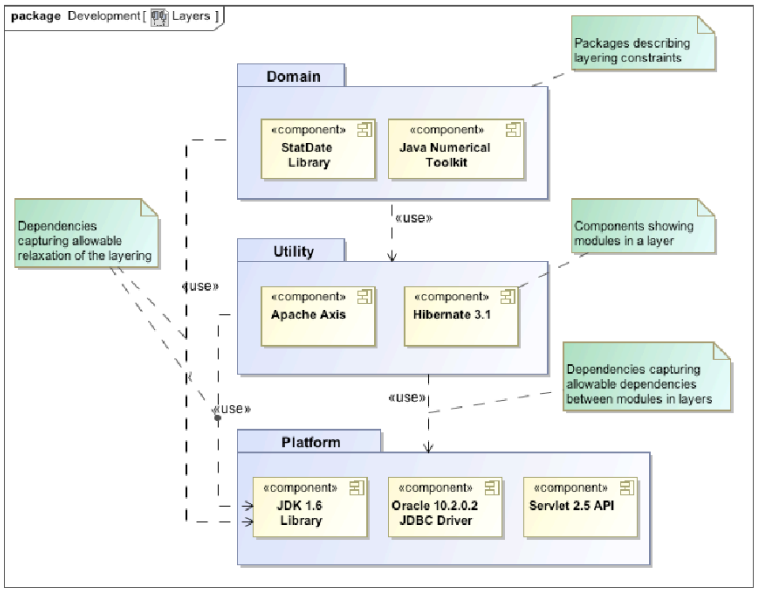
Have you identified any standard approaches to design that you need all element designers and implementers to follow? If so, do your software developers accept and understand these approaches?

Will a clear set of standard third-party software elements be used across all element implementations? Have you defined the way they should be used?

Is this view as minimal as possible?

Is the presentation of this view in the AD appropriate?

Development View Fragment



95

This slide shows a fragment of a development view, in this case a layer model for the software to be developed. The layer model groups code modules into packages, which represent logical layers of abstraction. The dependencies used show which layers depend on which other layers. Note the "relaxed" layering, where "Utility" and "Domain" modules are allowed to access the "Java 1.4 Library" directly, without going through the layering. (In reality, this may be such an obvious dependency that you might eliminate it from the model, but it serves as a useful example here.)

Some other aspects of the development view (such as the configuration management approach and codeline design) can be described using UML bit it normally isn't worth the bother so a simple "text and tables" approach is often used. Other aspects (such as standard design patterns) are often described using UML design models, as well as sample code.

The development view often gets very large and it's normally only of interest to software developers and testers, so it is often best captured as a separate document (perhaps called "Development Standards") with a short summary of the key points being captured in the AD.

Deployment Viewpoint

Focus	runtime environment structure and the distribution of software across it
Content	design of the environment into which the system will be deployed, including the system's runtime dependencies
Concerns	▪ types of hardware required ▪ specification and quantity of hardware required ▪ third-party software requirements ▪ technology compatibility ▪ network requirements ▪ network capacity required ▪ physical constraints

96

First half of the deployment viewpoint summary.

Deployment Viewpoint (ii)

Models	▪ runtime platform models ▪ network models ▪ technology dependency models
Pitfalls	▪ unclear or inaccurate dependencies ▪ unproven technology ▪ lack of specialist technical knowledge ▪ late consideration of the deployment environment

97

Second half of the deployment viewpoint summary.

A useful list of questions to ask yourself in order to validate your deployment view is:

•Have you mapped all of the system's functional elements to a type of hardware device? Have you mapped them to specific hardware devices if appropriate?

•Is the role of each hardware element in the system fully understood? Is the specified hardware suitable for the role?

•Have you established detailed specifications for the system's hardware devices? Do you know exactly how many of each device are required?

•Have you identified all required third-party software and documented all the dependencies between system elements and third-party software?

•Is the network topology required by the system understood and documented?

•Have you estimated and validated the required network capacity? Can the proposed network topology be built to support this capacity?

•Have network specialists validated that the required network can be built?

•Have you performed compatibility testing when evaluating your architectural options to ensure that the elements of the proposed deployment environment can be combined as desired?

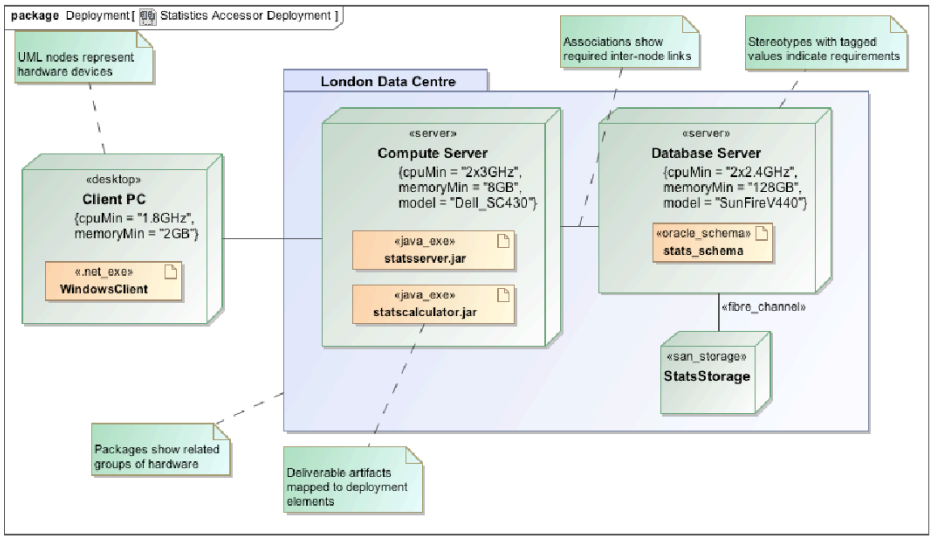
•Have you used enough prototypes, benchmarks, and other practical tests when evaluating your architectural options to validate the critical aspects of the proposed deployment environment?

•Can you create a realistic test environment that is representative of the proposed deployment environment?

•Are you confident that the deployment environment will work as designed? Have you obtained external review to validate this opinion?

•Are the assessors satisfied that the deployment environment meets their requirements in terms of standards, risks, and costs?

Deployment View Fragment



98

This slide shows a fragment of a deployment view. The view is a UML deployment diagram, using a package to show which nodes (machines) are located in the data centre and which are outside. The nodes each have a tagged value specification of the minimum specification required and the processes from the concurrency view are mapped to the nodes to show where each runs. In cases where the functional element to process mapping is trivial or obvious, the functional elements can be mapped directly to the nodes.

The nodes are also connected via associations that indicate the kind of physical interconnection required, again with tagged values used to indicate specific or non-standard requirements (such as the Database Server to Disk Array link being fibre channel in the example here).

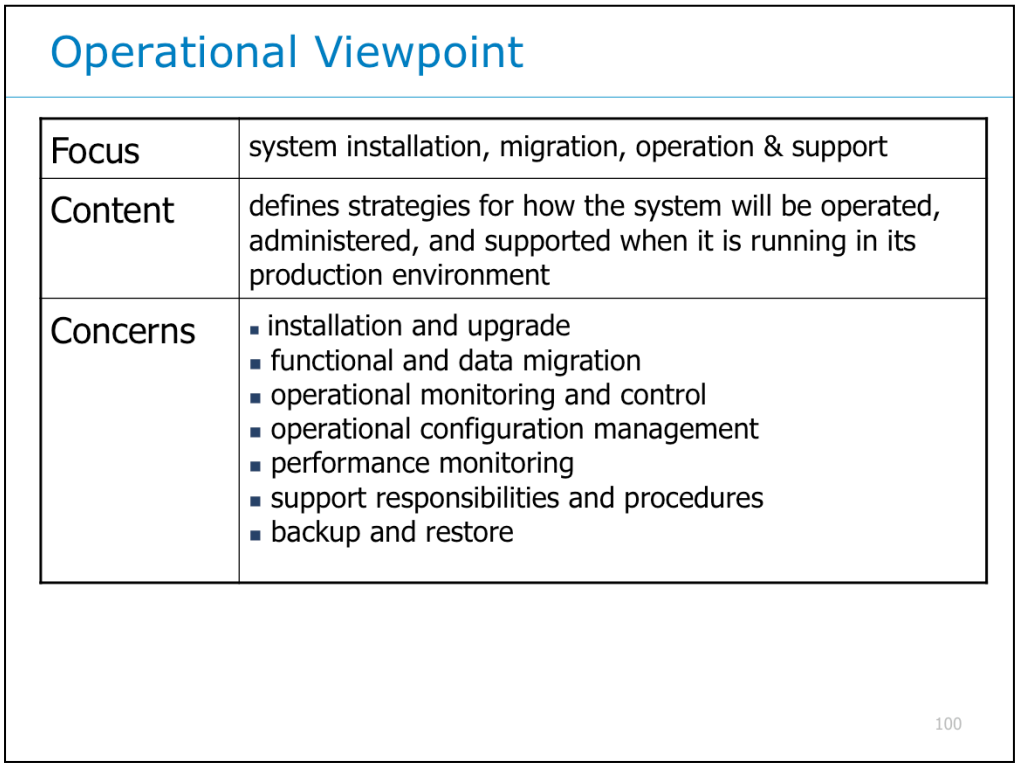
Deployment View Fragment (ii)

Client PC	▪ Windows XP SP1 ▪ Java JRE 1.4.2_06 or later ▪ Internet Explorer 6.0 SP1
Primary Server	▪ Windows 2003 server, w/sec patches ▪ Java SDK 1.4.2_06 or later ▪ Apache Tomcat 5.5.9 or later

Database Server	▪ Solaris 9.0 w/Aug05 patch cluster
	▪ Oracle 9.2.0.2 Std Edition
	  ▪ 10GB buffer cache, auto sized SGA
	  ▪ auto storage management, 2 table spaces
	▪ OEM 9.2.0.2 installed and working

99

Another deployment view fragment is shown in this slide, namely a software dependencies table, showing the supporting software required on each node (or node type) in the system. Note the detailed dependencies used, to avoid the “you need Oracle and Java” type of specification that often causes problems later.



First half of the operational viewpoint summary.

Operational Viewpoint (ii)

Models	▪ installation models ▪ migration models ▪ configuration management models ▪ administration models ▪ support and escalation models
Pitfalls	▪ lack of engagement with the operational staff ▪ lack of migration & backout planning ▪ insufficient migration window ▪ missing management tools ▪ lack of integration into the production environment ▪ inadequate backup and recovery modelling

101

Second half of the operational viewpoint summary.

A useful list of questions to ask yourself in order to validate your operational view is:

- Do you know what it takes to install your system?
- Do you have a plan for backing out a failed installation?
- Can you upgrade an existing version of the system (if required)?
- How will information be moved from the existing environment into the new system?
- Do you have a clear migration strategy to move workload to the new system?
- How will the system be backed up? Is restore possible in an acceptable time period?
- Are the administrators confident that they can monitor and control the system and do they have a clear understanding of operational procedures?
- How will performance metrics be captured for the system's elements?
- Can you manage the configuration of all of the system's elements?
- How will support be provided for the system? Is it suitable for those who will be supported?
- Have you cross-referenced the requirements of the administration model back to the development view to ensure that they will be implemented consistently?

Operational View Content

- Installation Model
  - Installation groups
  - Dependencies and constraints
  - Backout strategy
- Operational CM Model
  - Configuration groups and dependencies
  - Configuration parameter sets
  - Operational control (switching between sets)

102

The operational view content tends to be represented as “text and tables” and even for a simple example system tends to be quite bulky. As this sort of information is difficult to present via slides, we have just listed the sort of information you might expect to find in the operational view for a system such as the one being described here.

Installation Model
•What needs to be installed? How are things grouped for easy installation?
•What dependencies and constraints exist between installation groups?
•How will you “undo” the installation and back out if it proves to be difficult?

Operational Configuration Management Model
•What sets of configuration settings need to be controlled? Are there dependencies between them? (e.g. changing database configuration needs operating system changes at the same time)
•What are the different sets of configuration settings that need to be applied? (e.g. overnight, online day, end of month, …)
•How will you actually perform the process of applying the different sets?

Operational View Content (ii)

- Administration Model
  - Monitoring and control facilities required and provided
  - Required routine operational procedures
  - Required operational action in case of error conditions

103

Further possible operational view content:

Administration Model
• What facilities are required in the environment to monitor and control the various parts of the system? (e.g. system management frameworks)
• What facilities will your system be providing for monitoring and control? (e.g. plugins to frameworks, scripts and so on)
• What routine operational procedures will you need performed for your system?
• What action do you expect the operational staff to be able to perform if things go wrong? How will they recognise these conditions?

Appendix

Perspective Descriptions

Performance and Scalability

Quality	ability of the system to predictably execute within its mandated performance profile and to handle increased processing volumes
Concerns	▪ processing volume ▪ response time ▪ responsiveness ▪ throughput ▪ predictability
Tactics	▪ optimize repeated processing ▪ reduce contention via replication ▪ prioritize processing ▪ consolidate related workloads ▪ distribute processing over time ▪ [...]

105

First part of Performance and Scalability perspective summary.

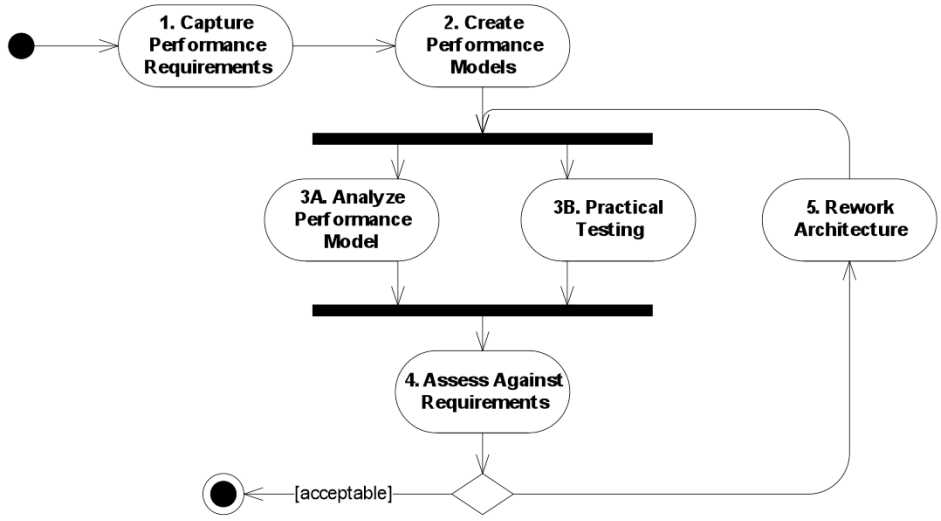
Performance and Scalability (ii)

Tactics (cont.)	▪ distribute processing over time ▪ minimize the use of shared resources ▪ partition and parallelize ▪ use asynchronous processing ▪ make design compromises
Pitfalls	▪ imprecise performance and scalability goals ▪ unrealistic models ▪ use of simple measures for complex cases ▪ inappropriate partitioning ▪ invalid environment and platform assumptions ▪ too much indirection ▪ concurrency-related contention ▪ careless allocation of resources ▪ disregard for network and in-process differences

106

Second part of Performance and Scalability perspective summary.

P & S Perspective Activities



107

This slide shows the set of activities suggested for applying the performance and scalability perspective to a system. It involves capturing and validating requirements, building performance models to check likely performance, using the models for analysis, in parallel with validation via practical testing and assessing the results against the requirements, changing the architecture if required.

Security

Quality	ability of the system to reliably control, monitor, and audit who can perform actions on resources and to detect and recover from security failures
Concerns	▪ policies ▪ threats ▪ mechanisms ▪ accountability ▪ availability ▪ detection and recovery
Tactics	▪ apply recognized security principles ▪ authenticate the principals ▪ authorize access ▪ ensure information secrecy ▪ ensure information integrity ▪ [...]

108

First part of Security perspective summary.

The full list of Security tactics is:

- •Apply recognized security principles
- •Authenticate the principals
- •Authorize access
- •Ensure information secrecy
- •Ensure information integrity
- •Ensure accountability
- •Protect availability
- •Integrate security technologies
- •Provide security administration
- •Use third-party security infrastructure

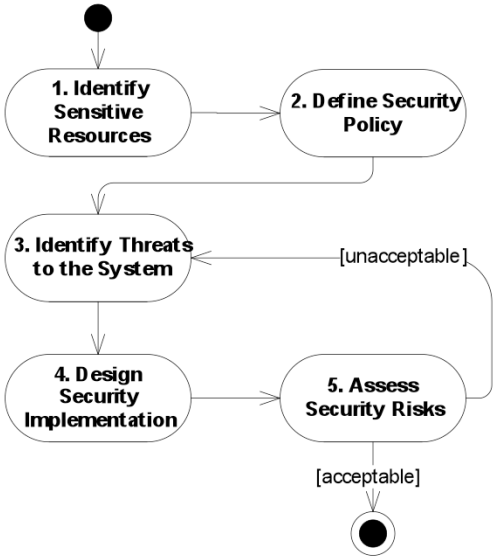
Security (ii)

Tactics (cont.)	▪ vulnerability analysis ▪ application of security technology
Pitfalls	▪ no clear requirements or models ▪ complex security policies ▪ unproven or ad-hoc security technologies ▪ not designing for failure ▪ lack of security administration facilities ▪ technology-driven approach (or over-reliance) ▪ failure to consider time sources ▪ security as an afterthought ▪ security embedded in the application code ▪ piecemeal security

109

Second part of Security perspective summary.

Security Perspective Activities



110

This slide shows the set of activities suggested for applying the security perspective to a system. It involves working out what the sensitive resources in the system are and what security policy needs to be enforced on them. Then, threats to the enforcement of the policy are identified and a security implementation is designed to meet these threats, before assessing the risks remaining and reconsidering the threats and design as required until an acceptable level of risk is reached.

Availability and Resilience

Quality	ability of the system to be fully or partly operational as and when required and to effectively handle failures that could affect system availability
Concerns	▪ classes of service ▪ planned / unplanned downtime ▪ mean time between failures & mean time to repair ▪ disaster recovery ▪ redundancy, clustering, failover
Tactics	▪ select fault-tolerant hardware ▪ use hardware clustering and load balancing ▪ log transactions ▪ apply software availability solutions ▪ select or create fault-tolerant software ▪ identify backup and disaster recovery solutions

111

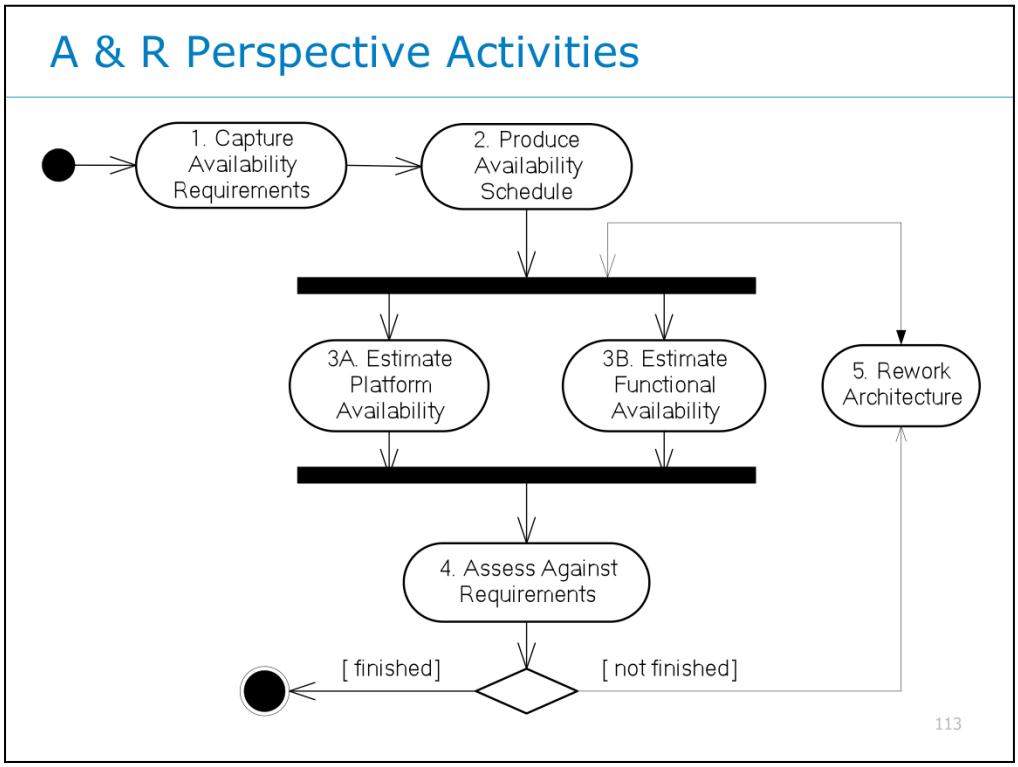
First part of Availability and Resilience perspective summary.

Availability and Resilience (ii)

Pitfalls	▪ single point of failure ▪ overambitious availability requirements ▪ ineffective error detection ▪ overlooked global availability requirements ▪ incompatible technologies

112

Second part of Availability and Resilience perspective summary.



This slide shows the set of activities suggested for applying the availability and resilience perspective to a system. The first activity is to capture and validate the requirements and then create a schedule to show the required system availability profile. Then, estimating platform and application ("functional") availability allows the overall availability to be assessed against requirements and the architecture to be reworked if required.

Evolution

Quality	ability of the system to be flexible in the face of the change that all systems experience, balanced against the costs of providing such flexibility
Concerns	▪ flexibility ▪ extensibility ▪ functional evolution ▪ deployment evolution ▪ integration evolution
Tactics	▪contain change ▪create flexible interfaces ▪apply change-oriented architectural styles ▪build variation points into the software ▪use standard extension points ▪achieve reliable change ▪preserve development environments […]

114

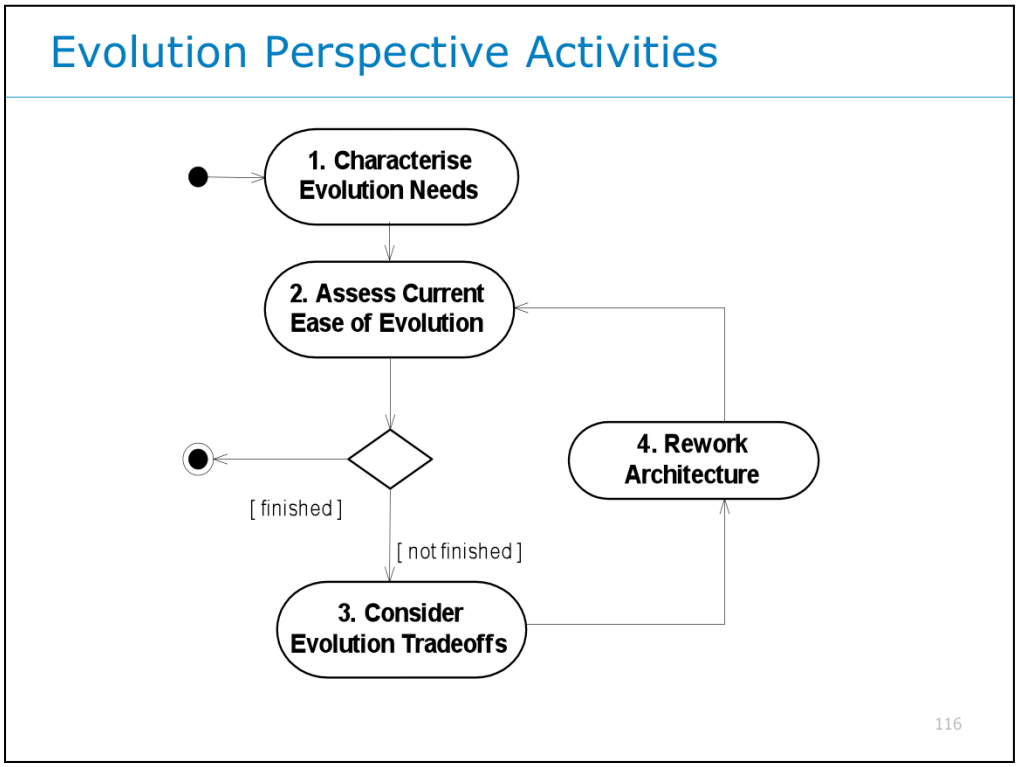
First part of Evolution perspective summary.

Evolution (ii)

Tactics (cont.)	▪ configuration management ▪ automated testing ▪ build and release management
Pitfalls	▪ prioritization of the wrong dimensions ▪ changes that never happen ▪ impact of evolution on critical quality properties ▪ lost development environments ▪ ad hoc release management

115

Second part of Evolution perspective summary.



This slide shows the set of activities suggested for applying the evolution perspective to a system. It involves understanding the evolution needs of the system, in order to characterise them by type, timeline and likelihood of occurrence. The ease of evoluation can then be assess with respect to the evolution needs and if unacceptable, considering architectural tactics that can increase the flexibility of the system and reworking the architecture appropriately.