Software Architecture in Practice

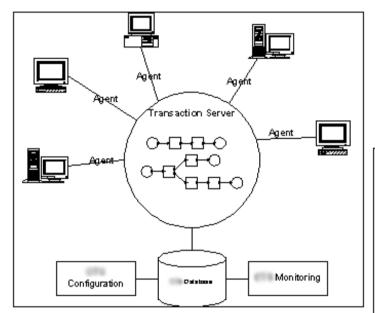
Language supports communication!

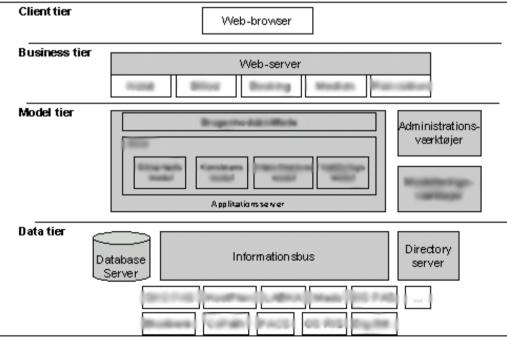
Definition

Views

Software Architecture?



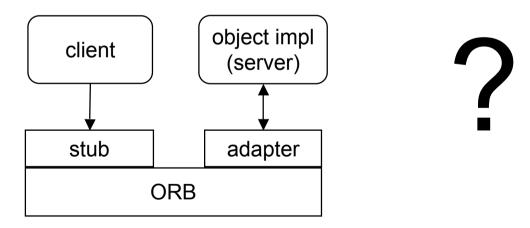




What is Software Architecture?

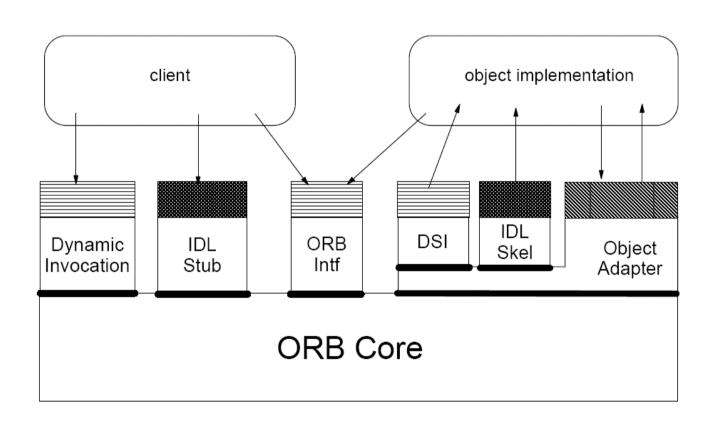


The architecture of CORBA: Common Object Request Broker Architecture



A Better Description?





Box-n-line drawings



Hence, most of the box-and-line drawings that are passed off as architectures are in fact not architectures at all. They are simply box-and-line drawings.

[Bass et al.]

What is Software Architecture?



"Architecture is the overall structure of the system"

- But what structure?
 - Classes?
 - Run-time objects?
 - Processes/threads?
 - Dataflow?

A definition



The software architecture of a computing system is the structures[†] of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. [Bass et al. 2003]

- Note: structure<u>s</u>
- externally visible properties:
 - assumptions on individual elements
 - services, performance, fault handling, shared resource usage, security, etc.

^{† &#}x27;views' is another term that is more commonly used...

Implications



architecture defines elements

- architectures are abstractions
 - components have public and private parts
 - architectures are only concerned with the public parts

elements have relationships

- decomposition require the parts/components be related
 - control flow / data flow
 - dependencies
 - and many others

Implications



architectures comprise more than one view

- what are the elements?
 - they can be "anything":
 - binary components, processes, modules, programs, libraries, hardware nodes,...

external visible properties of each elements is part of architecture

- interaction patterns, protocols, functionality
- but also: security aspects, performance, ...

Exercise



Two components –

- What are the important external visible properties

Web server

SpellChecker

Your experiences?



What 'definition' do you have of software architecture in your company?

Discussion



Every software system has an architecture

- it may, however, not be known to anyone ☺
- meaning there is a difference between architecture and architecture representation

Architecture:

- reality embedded in system

Architecture representation:

- our description of it
 - in documents, in our collective minds, on napkins, ...

Discussion



architecture may be "good" or "bad"

- raises the issue of architecture evaluation
- ... and evaluation criteria
- ... and who defines the criteria?
- … and how we measure?

How do you define 'good'?

How do you measure quality?

Software architecture is an investment

 we invest in it in order to reduce costs in the software production...

Software Architecture as Abstraction!

- omit detail in order to cope with complexity
- common overview common goal
- overview and compact documentation
- and perhaps reuse

Software Architecture as Communication!

- understanding, discussion, consensus...
- between all stakeholders

Software Architecture as **Design Plan**

- early analysis ⇒
 - risk assessment and risk management
 - optimise balance between different qualities
- bridge between requirements and implementation
 - implementation design decisions are determined by the architecture!

"Having an explicit and well-communicated architecture is the first step towards ensuring architectural conformance."

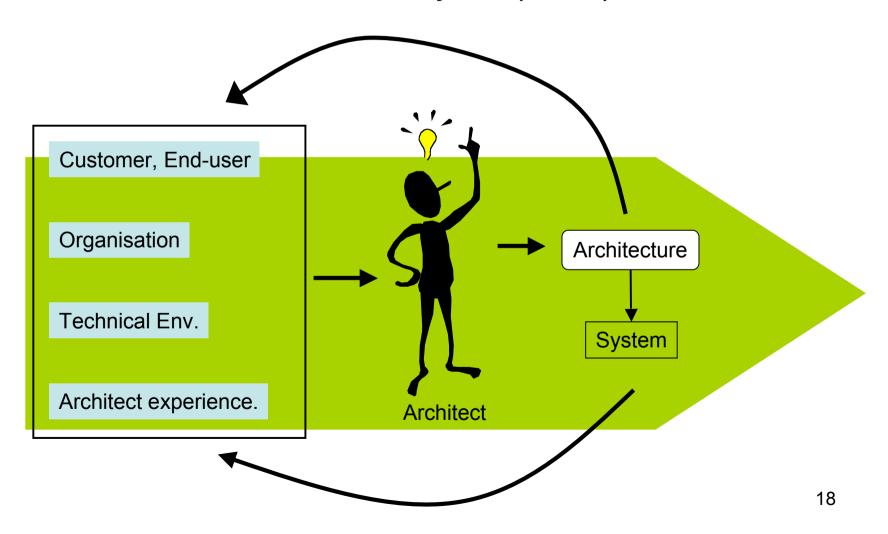
as-built corresponds to as-designed

Software Architecture and the Development Process





Architecture-Business-Cycle (ABC)



Software Architecture and the Development Process



When to software architect?

- How?
- Why?
- Who?

Three process examples

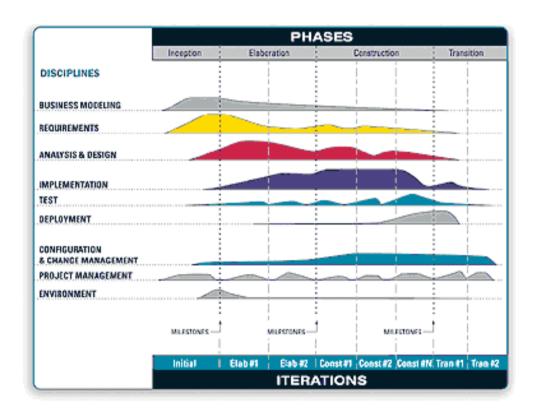
- (Rational) Unified Process
- eXtreme Programming
- Capability Maturity Model

Rational Unified Process



Use case-driven, "architecture-centric", ...

- Build skeletal system (architectural prototype)
- Fill in...



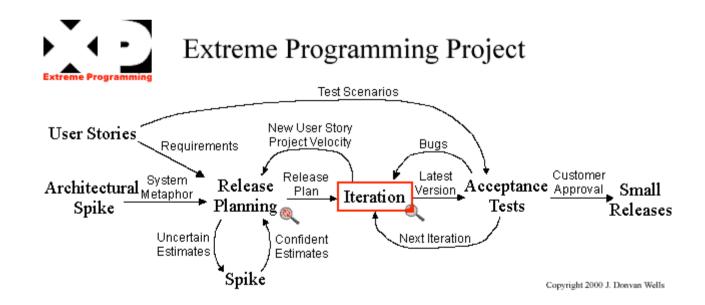
eXtreme Programming



AARHUS UNIVERSITET

Simplicity, communication, feedback, courage

- A set of practices: refactoring, unit testing, small releases
- No software architecture per se in [Beck, 1999]
 - "Metaphor" comes closest
 - ("Architectural spike" below corresponds to an exploratory architectural prototype)



Capability Maturity Model



AARHUS UNIVERSITET

Level 1: Initial

Ad hoc, person-dependent, blackbox process

Level 2: Repeatable

- Basic management processes established
- Successes can be repeated

Level 3: Defined

 Documented and standardised processes for documentation and engineering

Level 4: Managed

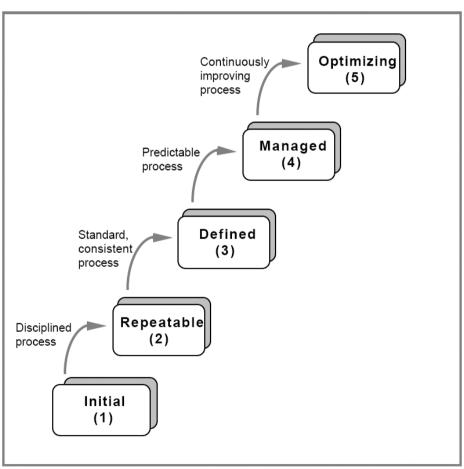
Detailed measurement of product and process quality

Level 5: Optimized

- Continuous process improvement
- Quantitative feedback and innovation

Software architecture part of Software Product Engineering

Key process area for level 3



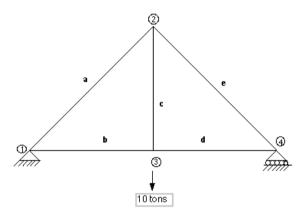
Views

Views



"A *view* represents a partial aspect of a software architecture that shows specific properties of a software system." [Buschmann]

Example: Consider a bridge





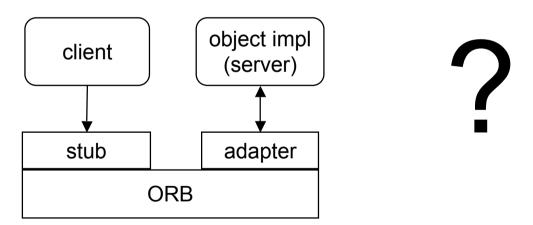
joint 2:
$$Fx = fa(-\cos 45) + fc(\cos 90) + fe(\cos 45) = 0$$
; $Fx = -cfa + cfe = 0$
joint 2: $Fy = -fa(\sin 45) + -fc(\sin 90) + -fe(\sin 45) = 0$; $Fy = -sfa - fc - sfe = 0$





Why not look at CORBA again...

Can 'views' explain CORBA better than this?

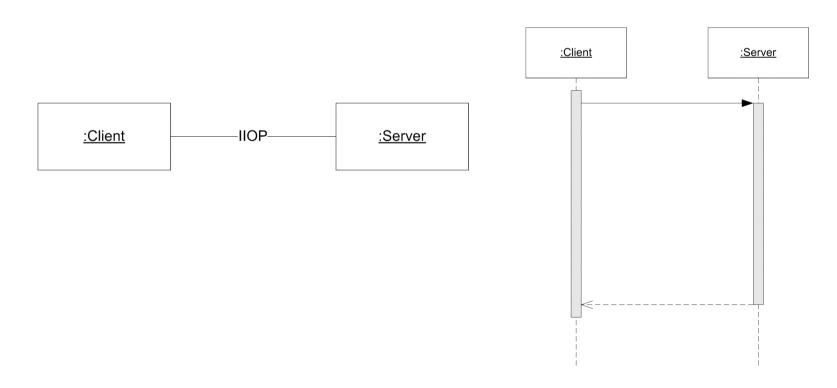


CORBA in Component Connector View (Dynamics)



CORBA is about *transparency*

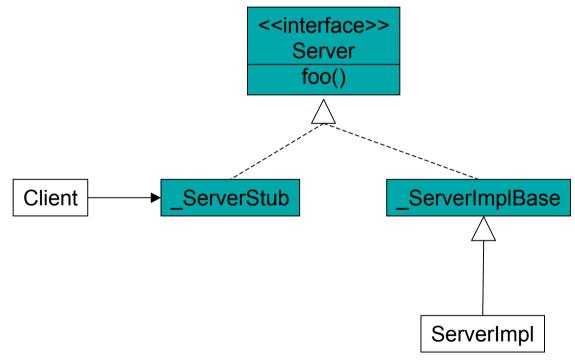
- abstract away the network
- programming model: normal method call



CORBA in Module View



From the static/class view a *proxy* pattern is used

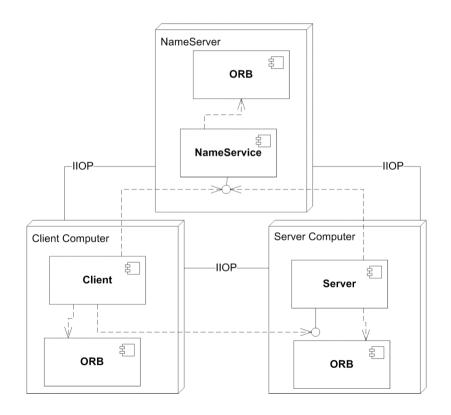


Blue classes are generated by IDL2Java compiler White classes are written by you.

CORBA in Deployment View



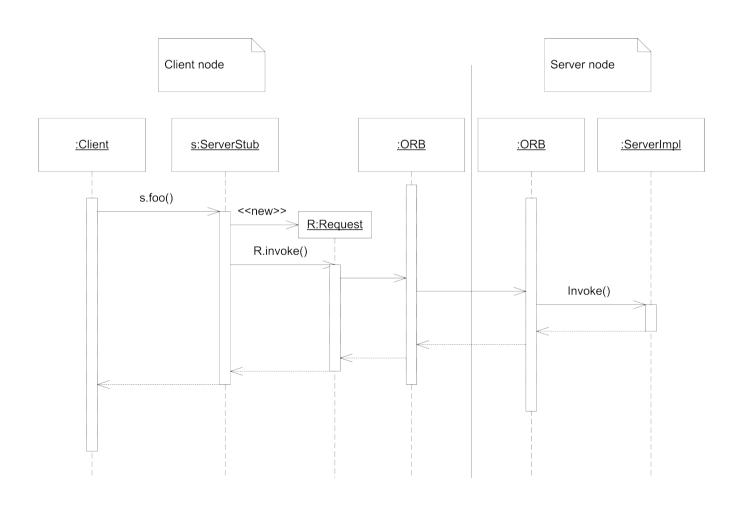
Deployment-wise, the ORB middleware supports marshalling and unmarshalling ect.



CORBA Dynamics in more detail



The invocation dynamics is pretty complex

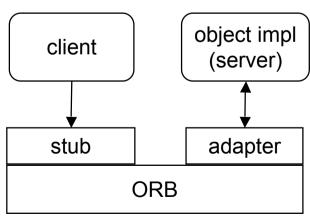


Morale

Views are important for precise communication and documentation!

I had to reverse engineer a Java CORBA example to really understand what was going on.

This diagram is nonsense...



Discussion

Do you use 'views' in your documentation?

What views do you use?

Are you strict about separating the deployment, class, dynamic, etc. structure?

The "View Zoo"

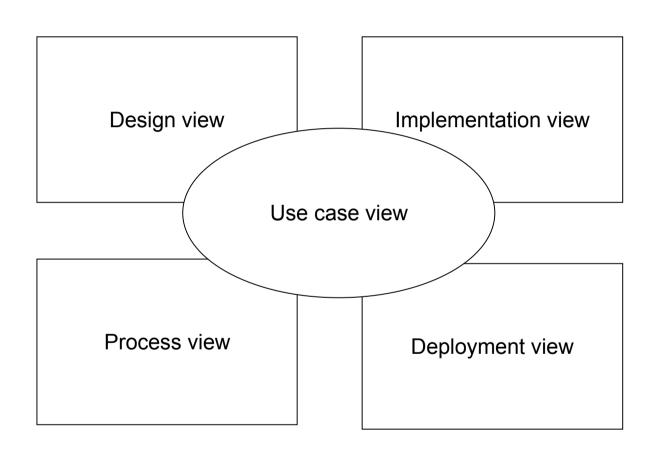


Software architecture is a young research field.

Therefore different authors stress different views and what goes into them...

The Start: Kruchten 4+1 model





4+1 model [Rational]



Design View

- vocabulary of problem and its solution
- primarily supports functional requirements

Process view

- threads and processes / concurrency & synchronization
- addresses performance, scalability and throughput

4+1 model [Rational]



Implementation view

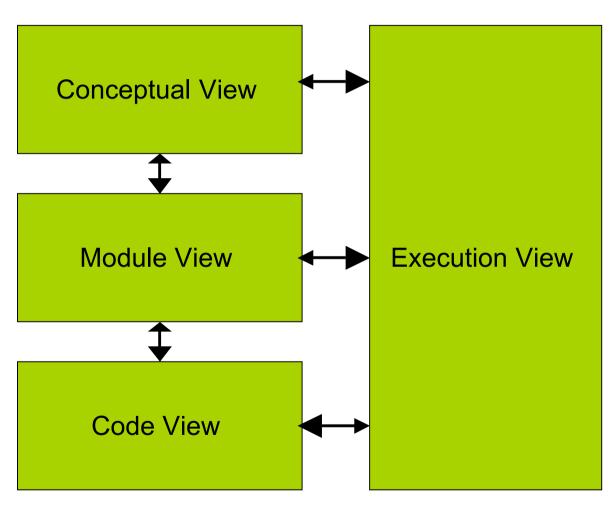
- components & files used to assemble and release the physical system
- addresses configuration management

Deployment view

- encompasses the nodes / hardware topology
- addresses distribution, delivery, installation of part that make up the physical system.







Hofmeister views



AARHUS UNIVERSITET

Logical/conceptual view

deals with functionality

Module/implementation view

 deals with team structure, work-break-down, software configuration management and source control

Process/execution view

deals with dynamics: performance, concurrency, synchronization

Deployment view

deals with distribution, network performance, deployment at customer site

Enterprise Architecture (EA)



The architecture of an enterprise

IT architecture and business architecture

Consider the Bass et al. definition of software architecture

The software architecture of a computing system is the structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass et al., 2003]

– How does this relate to Enterprise Architecture?

Frameworks for EA



Zachman

Department of Defense Architecture Framework (DODAF)

UK Ministry of Defence Architectural Framework (MODAF)

The Open Group Architecture Framework (TOGAF)

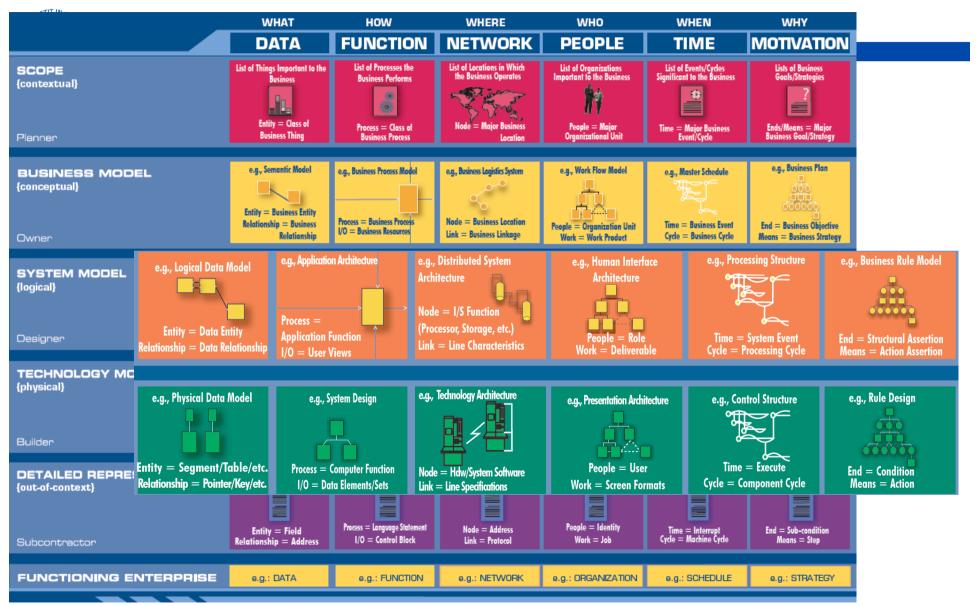
Offentlig Information Online (OIO) EA Framework

Zachman



	WHAT	HOW	WHERE	WHO	WHEN	WHY
	DATA	FUNCTION	NETWORK	PEOPLE	TIME	MOTTVATION
SCOPE {contextual}	List of Things Important to the Business	List of Processes the Business Performs	List of Locations in Which the Business Operates	List of Organizations Important to the Business	List of Events/Cycles Significant to the Business	Lists of Business Goals/Strategies
Planner	Entity = Class of Business Thing	Process = Class of Business Process	Node = Major Business Location	People = Major Organizational Unit	Time = Major Business Event/Cycle	Ends/Means = Major Business Goal/Strategy
BUSINESS MODEL {conceptual}	e.g., Semantic Model Entity = Business Entity Relationship = Business	e.g., Business Process Model Process = Business Process	e.g., Business Logistics System Node = Business Location	e.g., Work Flow Model	e.g., Master Schedule Time = Business Event	e.g., Business Plan U OOO IIII End = Business Objective
Owner	Relationship	I/O = Business Resources	Link = Business Linkage	Work = Work Product	Cycle = Business Cycle	Means = Business Strategy
SYSTEM MODEL {logical} Designer	e.g., Logical Data Model Entity = Data Entity Relationship = Data Relationship	e.g., Application Architecture Process = Application Function I/O = User Views	e.g., Distributed System Architecture Node = 1/S Function (Processor, Storage, etc.) Link = Line Characteristics	e.g., Human Interface Architecture People = Role Work = Deliverable	e.g., Processing Structure Time = System Event Cycle = Processing Cycle	e.g., Business Rule Model End = Structural Assertion Means = Action Assertion
		I/O — Oser Flews			June Historia	means — activit assertivit
TECHNOLOGY MODEL {physical}	e.g., Physical Data Model	e.g., System Design	e.g., Technology Architecture Node = Hdw/System Software	e.g., Presentation Architecture	e.g., Control Structure	e.g., Rule Design
Builder	Relationship = Pointer/Key/etc.	I/O = Data Elements/Sets	Link = Line Specifications	Work = Screen Formats	Cycle = Component Cycle	Means = Action
DETAILED REPRESENTATIONS {out-of-context}	e.g., Data Definition	e.g., Program	e.g., Network Architecture	e.g., Security Architecture	e.g., Timing Definition	e.g., Rule Specification
Subcontractor	Entity = Field Relationship = Address	Process = Language Statement I/O = Control Block	Node = Address Link = Protocol	People = Identity Work = Job	Time = Interrupt Cycle = Machine Cycle	End = Sub-condition Means = Step
FUNCTIONING ENTERPRISE	e.g.: DATA	e.g.: FUNCTION	e.g.: NETWORK	e.g.: ORGANIZATION	e.g.: SCHEDULE	e.g.: STRATEGY

Zachman





OIO EA Framework

AARHUS UNIVERSITET

TAS ARHO					
	Strategi	Forretning	Information	Applikation	Teknologi
Konceptuelt	Strategier Alle de dokumenter der overordnet definerer forretningen: dens mål, strategier, forretningsomgivelser, forretningsdrivere mv. Finansielle tall er ikke en del af dette.	Forretningsstruktur Dokumenter der fastlægger forretningens opbygning: organisation, ansvarsområder, aktører og geografisk struktur. Dokumenter der specificerer forretningsservices.	Objektmodeller Dokumenter der identificerer og beskriver forretningens kemebegreber og (forretningsobjekter) og de forretningsspørgsmål man skal kunne besvare.	Applikationsstrategi Dokumenter der beskriver strategiske valg af applikations- og integrations-tilgange.	Teknologistrategi Dokumenter der fastlægger de overordnede teknologivalg, herunder strategiske standarder og en plan for brug af teknologi fremadrettet.
Logisk	Principper Dokumenter der – afledt af forretningens strategier - fastlægger principper og politikker for brug af it i forretningen, med rationale og implikationer for disse.	Processer Dokumenter der definerer forretningens procesmodel(ler) - på højt niveau og på aktivitetsniveau; evaluering af processerne; processer relateret til forretningsobjekter, lokationer, etc.	Logiske datamodeller Dokumenter der specificerer forretningsobjekter ud til en logisk datamodel. Dokumenter der specificerer databaselandskabet.	Applikationsstruktur Dokumenter der beskriver applikationer og deres integrationer. Dokumenter der specificerer forretningens brug af applikationer, i eksempelvis Use Cases. Dokumenter der specificerer services fra en mere teknisk vinkel.	Teknologistruktur Dokumenter der beskriver teknologivalg, i form af standard-konfigurationer, foretrukne systemtopologier, og tekniske tilgange til tværgående områder såsom sikkerhed.
Fysisk	Forretningsregler Dokumenter der konkretiserer strategier og principper i relativt detaljerede forretningsregler.	Workflows Dokumenter der konkretiserer processer og aktiviteter til mere detaljerede workflows, og beskriver anvendelsen af forretningensregler i disse workflows.	Fysiske datamodeller Dokumenter der specificerer fysiske datamodeller, herunder detailspecifikation af de enkelte objekter, eksempelvis via OIOdatastandarder.	Applikationsdesign Dokumenter der specificerer strukturen af applikationer - deres komponentopdeling og brug af services, samt hvordan tekniske services bruges til at levere forretningsservices.	Teknologilandskab Dokumenter der helt konkret specificer de enkelte systemer og deres konfigurationer.

Styringsrammer

Trends og projektgrundlag

Dokumenter der opsamler de trends og udfordringer der skal tages højde for i enterprise arkitekturen.

Dokumenter der på baggrund af dette tilpasser OIO EA metoden til behovet, og beskriver EA projektet.

Gap analyse

Dokumenter der analyser restriktioner, risici og muligheder ved den fremtidige arkitektur. Dokumenter der analyser hullerne imellem den nuværende og fremtidige arkitektur.

Forandring

Dokumenter der beskriver tilgange til migrering, og konkrete planer for migrering, samt konsekvenserne af migreringsplanerne.

Governance

Dokumenter der beskriver den generelle tilgang til EA governance, sävel som de aktuelle governance processer og organisationsstrukturer.

Styring

Dokumenter der beskriver styring. Indadtil ved at definere tilgang til drift og budget- og ressourceplanerne. Udadtil ved at fastlægge lovmæssige bindinger og kontraktforhold.



OIO EA Framework

AARHUS UNIVERSITET

TAS ARHUS							
	Strategi	Forretning	Information	Applikation	Teknologi		
Konceptuelt	Strategier (Ingen standarder identificeret)	Forretningsstruktur (Ingen standarder identificeret)	Objektmodeller UML Class	Applikationsstrategi (FESD)	Teknologistrategi OIO-kataloget (refererer de relevante standarder)		
Logisk	Principper (Ingen standarder identificeret)	Processer BPMN UML Activity	Logiske datamodeller UML Class UML State (UML Sequence, UML Collaboration)	Applikationsstruktur UML Use Case UDDI	Teknologistruktur (OIO-kataloget)		
Fysisk	Forretningsregler (Ingen standarder identificeret)	Workflows BPMN UML Activity UML State BPEL	Fysiske datamodeller OIOXML, WSDL, ESDH,	Applikationsdesign OWSA SOA, DCOM UML Component, UML Deployment (UML Sequence, UML Collaboration)	Teknologilandskab (Ingen standarder identificeret)		
Styrings- rammer	Trends og projektgrundlag (Ingen standarder identificeret)	Gap analyse (Ingen standarder identificeret)	Forandring (Ingen standarder identificeret)	Governance ITIL COBIT	Styring (Ingen standarder identificeret)		

OIO EA Framework AARHUS UNIVERSITET Information Applikation Strategi Forretning knologi Objektmodeller Applikationsstrategi Strategier Forretningsstrukt ologistrategi (Ingen standarder identificeret) (Ingen standarder identificen et (refererer de (FESD) UML Class Konceptuelt andarder) Processer Principper ologistruktur **BPMN** (Ingen standarder identificeret) Logiske datamodeller Applikationsstruktur Logisk UML Activity **UML Class** UML Use Case UML State UDDI (UML Sequence, UML Collaboration) Workflows Forretningsregler ologilandskab (Ingen standarder identificeret) BPMN larder identificeret) Fysisk **UML Activity** UML State **BPEL** Fysiske datamodeller Applikationsdesign OIOXML, WSDL, ESDH, OWSA SOA, DCOM UML Component, UML Deployment (UML Sequence, UML Trends og projektgrundlag Gap analyse Styring Collaboration) (Ingen standarder identificeret) (Ingen standarder identificer larder identificeret) Styringsrammer

Views



Brooks in his "No Silver Bullet"-paper states:

Software is invisible and unvisualizable... In spite of progress in restricting and simplifying the structures of software, they remain inherently unvisualizable, thus depriving the mind of some of its most powerful conceptual tools.

Benefits of views



The most important benefit:

The recognition that there is no single view on a software system!

Clear, precise, focus in diagrams, documentation and discussions.

Benefits of views



Handling complexity

- separation of concern: the 'big mudpile' is partitioned into distinct views = aspects of the architecture
- compact notation with precise (?) meaning

Form checklist over aspects that you ought to consider relevant

Documentation is decoupled ⇒ more robust to changes...





We suggest a *minimalist* approach:

- C&C view: Focus on run-time behavior
- Module view: Focus on compile-time elements
- Deployment view: Focus on computing nodes

• Inspired by Clements et al. 2002

Your experience



Are you using any of these views?

Or others?

What are the most appropriate for your domain?

Discussion



AARHUS UNIVERSITET

But...

- what views are important?
- what aspects are part of what views?
- what to call the views?
- what notation to use?
- can we use UML to describe architecture?

The central point is

- important to consider the relevance of central views
- important to split the documentation
- important to have compact notation that central (all?) stakeholders understand