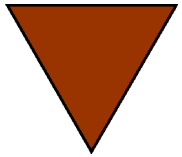


Architecting Large Business Systems

Tutorial at OOP 2001, Munich

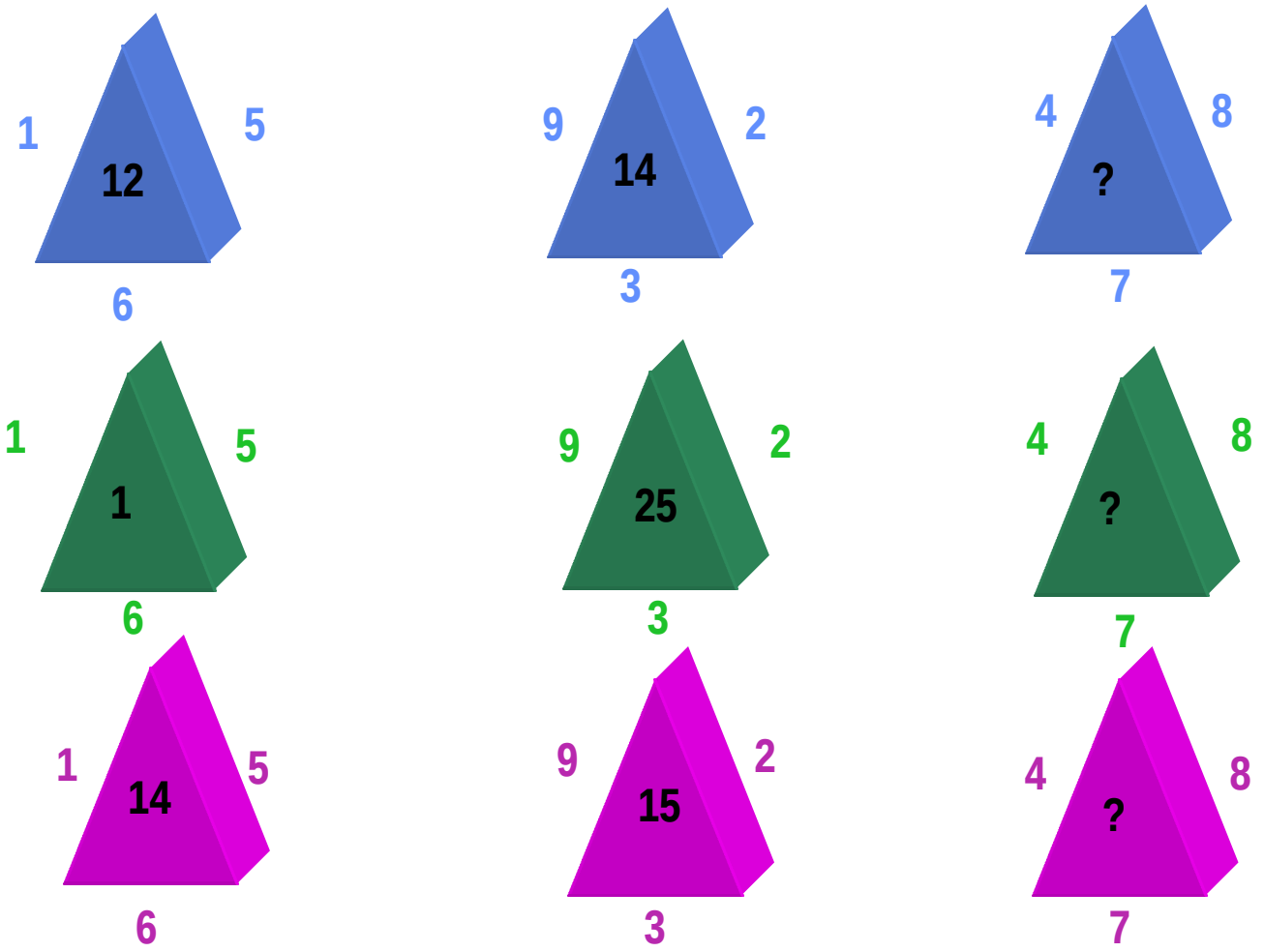
January 22nd, 2001

Alan O'Callaghan
De Montfort University
The Gateway
Leicester, LE1 9BH
United Kingdom
aoc@dmu.ac.uk

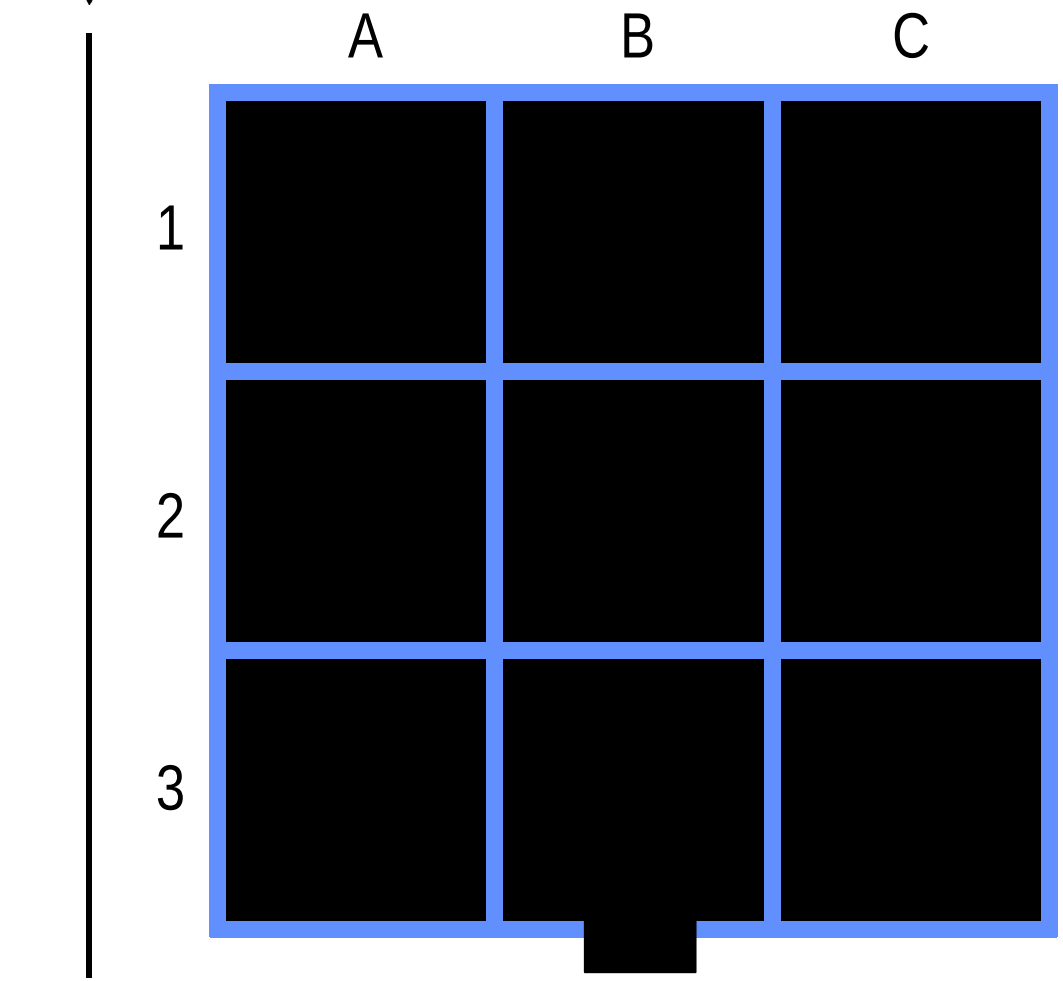
Jens Coldewey
Coldewey Consulting
Curd-Jürgens-Str. 4
D-81739 München
Germany
jens_coldewey@acm.org

Wolfgang Keller
Generali AG
Kratochwjlestr. 4
A-1220 Wien
Austria
wolfgang_keller@acm.org

Exercise 1: Configurational Knowledge



Exercise 2: Configurational Knowledge



Consider the squares in this grid to be spaces; the orange lines to be walls.

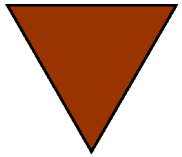
At B3 is an external entrance; use exactly 8 other *internal* entrances to connect the rooms so that every room is accessible

The Schedule for Today

10:00-10:20	Intro, Welcome Exercise
10:20-11:00	Lecture, Process explained
11:00-12:00	Exercise - Architecture: Mile Wide Inch Deep
12:00-12:40	Exercise - Architecture and Organization
12:40-13:00	Lecture - Architecture and Organization
13:00-14:00	**Lunch**

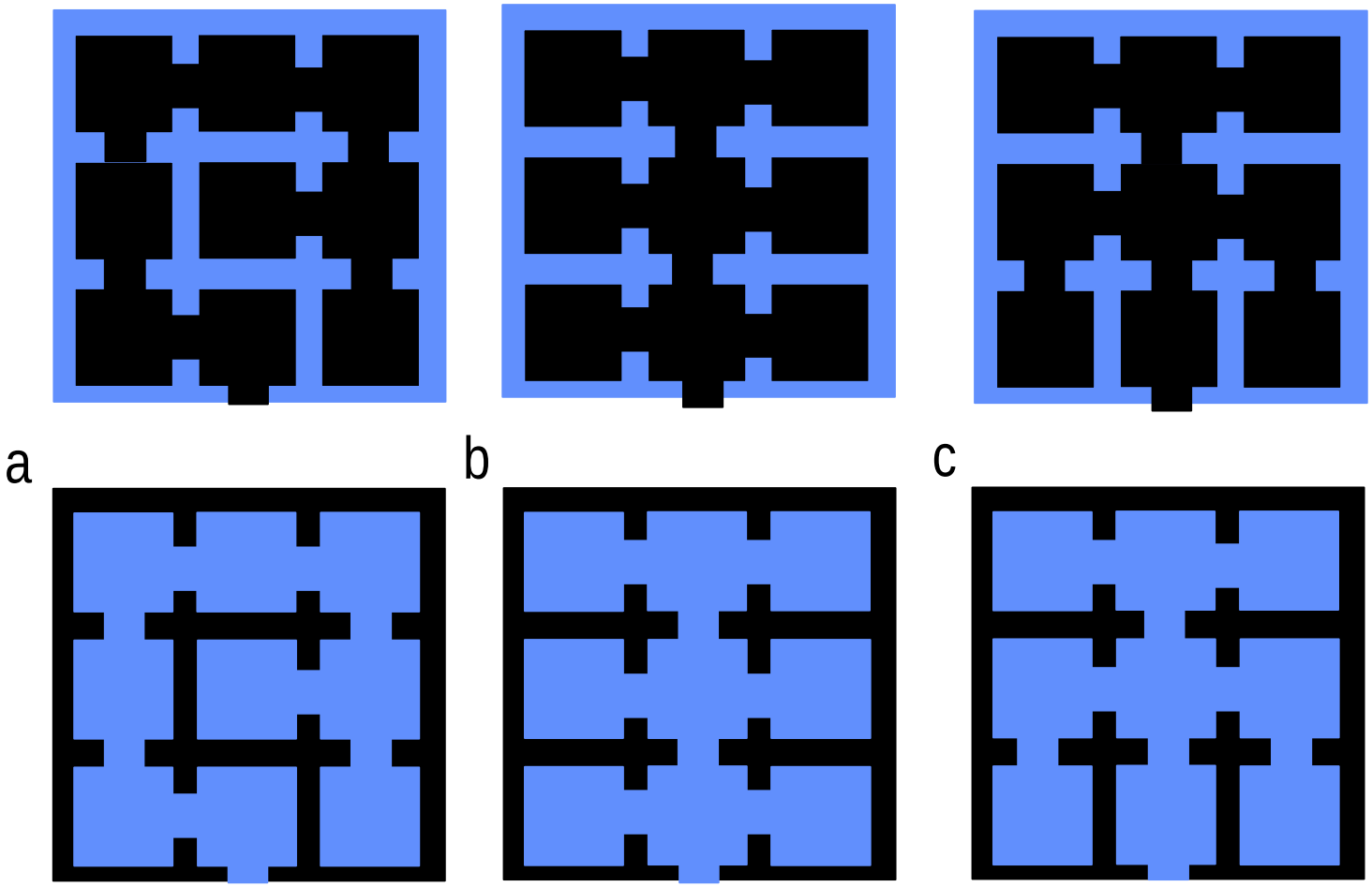
The Schedule for Today

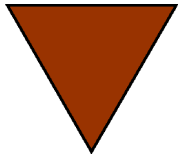
13:00-14:00	**Lunch**
14:00-14:30	Lecture - Detailed Design
14:30- 16:00	Exercise - Detailed Design
16:00-16:15	**Coffee Break**
16:15-16:45	Group Work Results Presented
16:45-17:00	Wrap-Up



Introduction
Lecture 10:20-10:50 (Alan)

Architecture is Configuration of Space



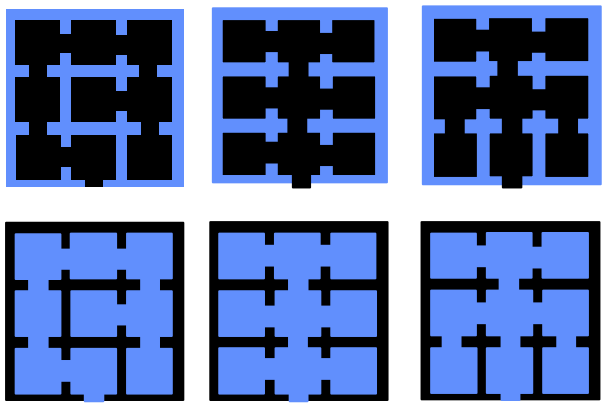


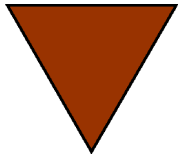
In the Beginning there was an Idea: Where this Tutorial Comes From

- Basis are the ADAPTOR and Janus pattern languages, both being results of Alan's research and consulting on reengineering and software architecture
- Partly ADAPTOR builds on James Coplien's work on organisational research
- Additional insights came from Wolfgang's work as Generali's platform manager for middle and east Europe and Jens' consulting work on architecture with several large insurance and bank companies

Rooms and Their Connections Configure Space

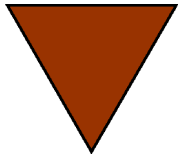
- Three notional courtyard buildings
  - Same basic physical structures and cell division
  - Same number of internal, external openings
  - Lower figure highlights space as against normal view of ‘structure’ above
- ‘Only’ difference is the location of cell entrances
  - But this radically changes the patterns of movement through the buildings
  - Which offers more opportunities for “private” space?





Software Architecture Also Deals With Spatial Configuration

- Software does not deal with physical spaces
- But space is not merely a *physical* construct in the architecture of the built environment
  - It also embodies notions of *logical* and *social* spaces
- We can consider modules, packages, components etc., to occupy virtual spaces in software
  - And connectors to be access paths to these spaces which make them interdependent
- Therefore the knowledge of how to put modules and connectors together appropriately is architecture



How To Develop an Architecture (I): Mile Wide Inch Deep

- How do you develop an architectural vision of a system without over-constraining later design decisions?
  - Interaction between high-level and low-level design vs. maintaining conceptual integrity
  - Design decisions almost always have additional, unanticipated effects
- Develop the software as growing. Living structure with the first iteration forming an outer shell

Find the complete text in Alan O'Callaghan: "Patterns for Architectural Praxis", EuroPLoP 2000, http://www.coldewey.com/europlop2000/

How To Develop an Architecture (II): Archetype

- Where do you start to find the building blocks of a first-cut software architecture?
  - The client's conceptual model of the problem is shaped by "real world" forces *but* the team's model is driven by design trade-offs in the software
  - Traceability between design and business needs forms a good solution *but* the mapping between client's and team's models is rarely straight forward
  - Specification and implementations change
- Build a model based on the client's vocabulary by capturing the key abstractions as object types

How to turn the object network into components (I): Time-Ordered Coupling

- How is the high-level structure of a system best organised for adaptation in the long-term?
- Organise the system into partitions, so that the components of each of the partitions have similar lifespans and/or change-rates.

Examples:

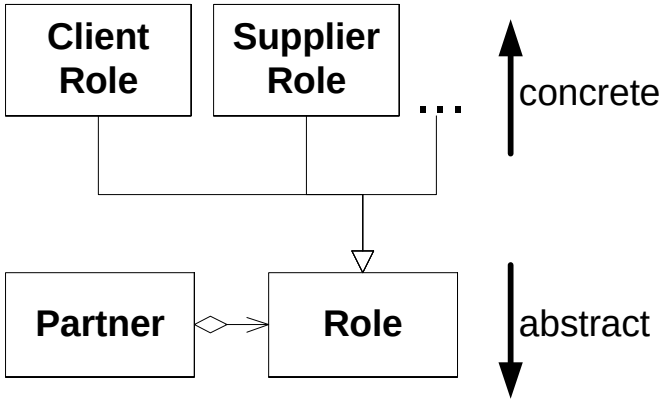
- An own subsystem for tax-related issues
- Subsystems organised according to customising levels
- Product-driven architectures

Counter-example:

- The overall rule-system

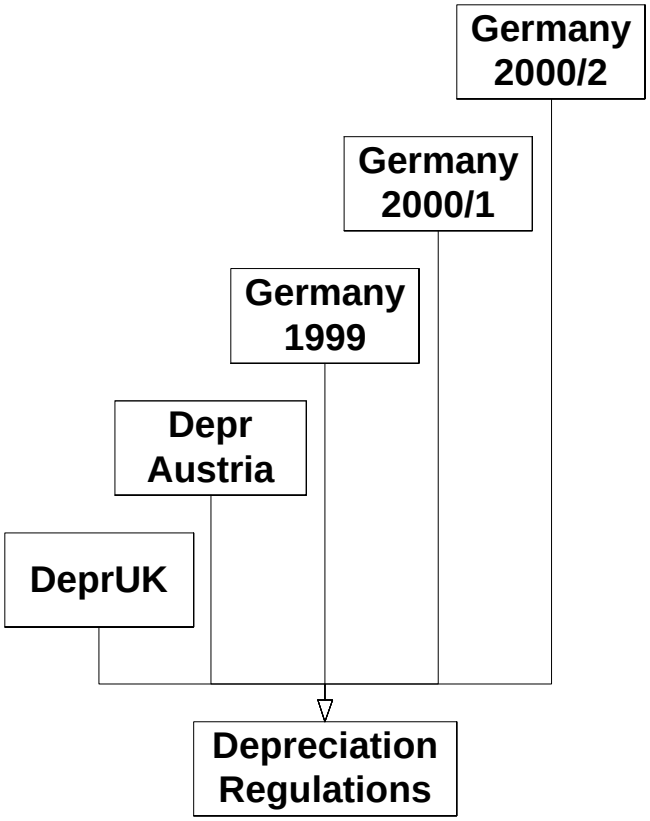
How to turn the object network into components (II): Abstract Foundation

- How do you position the least changeable abstractions in a potentially long-lived system?
- Push them towards the root in a layered hierarchy and represent them as abstract classes or interfaces



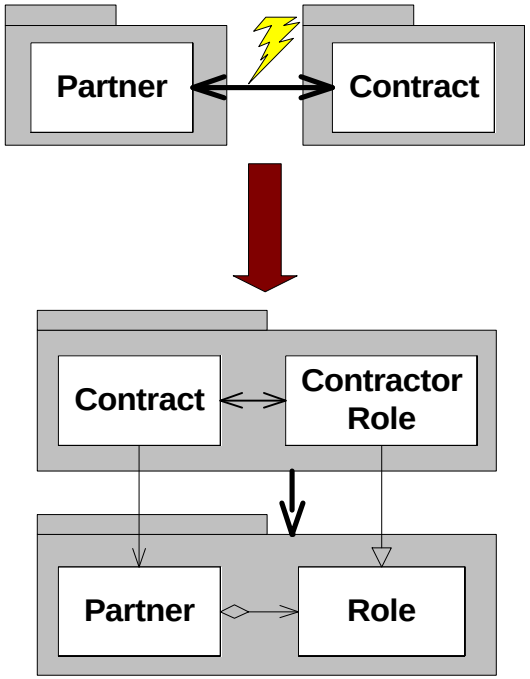
How to turn the object network into components (III): Volatile Top

- How do you position the most changeable abstractions in a potentially long-lived system?
- Push them ‘up’ towards the leafs of the hierarchy and represent them as concrete classes



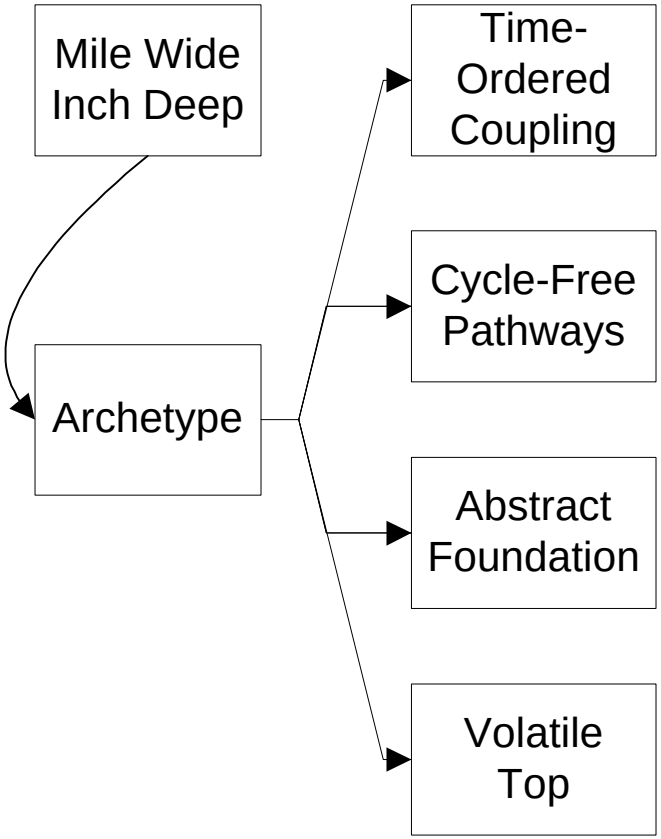
How to turn the object network into components (IV): Cycle-Free Pathways

- How do you manage the dependencies between packages and components?
- Seek an overall scheme that is free of direct or indirect cycles. Use Abstract Foundation, Volatile Top, or lower-level design to break “domain cycles”.



These Five Patterns Help to Avoid the Most Common Mistakes

- You cannot do no architecture
- *Time-Ordered Coupling* helps to find good domain-level partitions
- *Abstract Foundation* fosters reusability
- *Volatile Top* fosters maintainability
- *Cycle-Free Pathways* also avoid “Blobs”



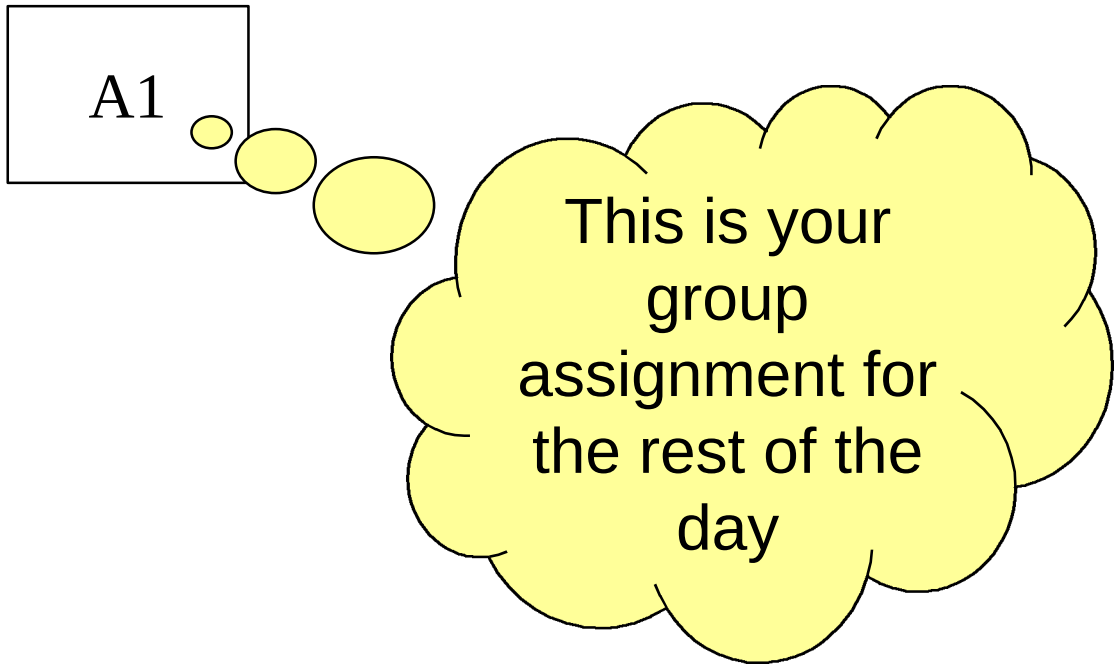
The Open Space Process Explained
Process Intro 10:50-11:00
(Wolfgang)

Process – Teams and Tutors
Why is this limited to 48 people?

Alan	A1	A2	A3	A4
Jens	J1	J2	J3	J4
Wolfgang	W1	W2	W3	W4

48 = 3 tutors * 16 people
= 3 tutors * (4 groups * of 4 people)

Process
Please pick up a group ticket



At the end of the day, this group will have one design and present it 10 mins from 16:25-16:35

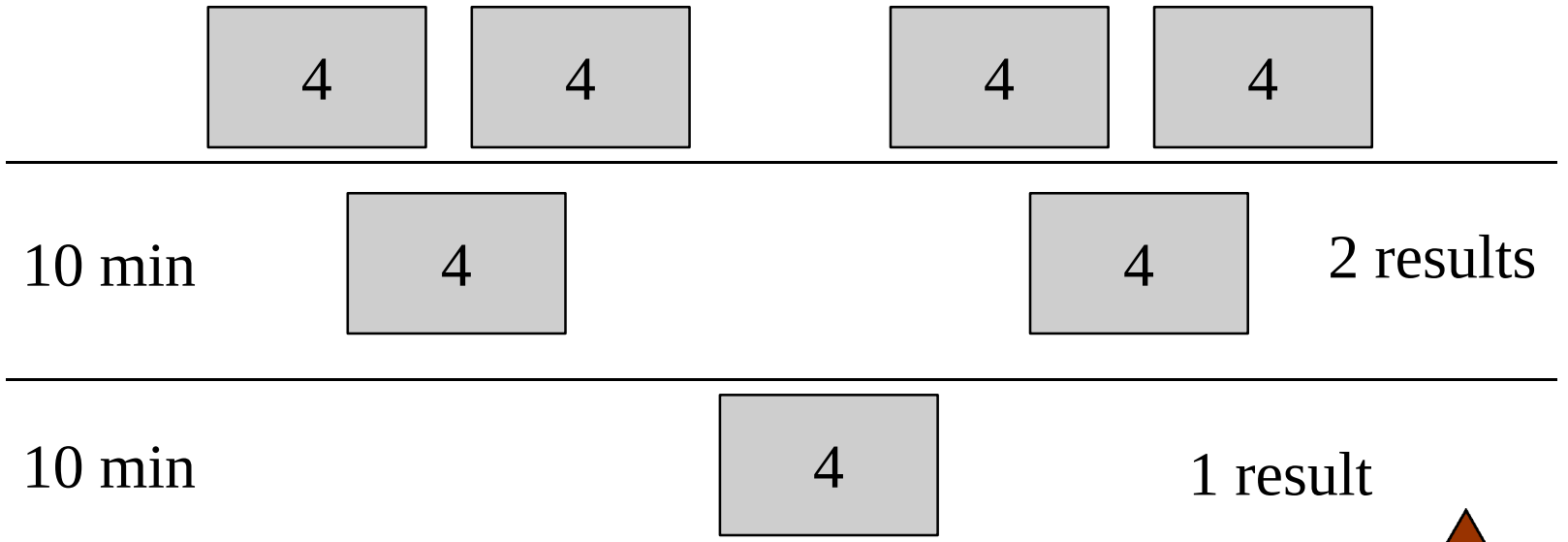
Jens	J1	J2	J3	J4

Process
The Ideas behind the Process

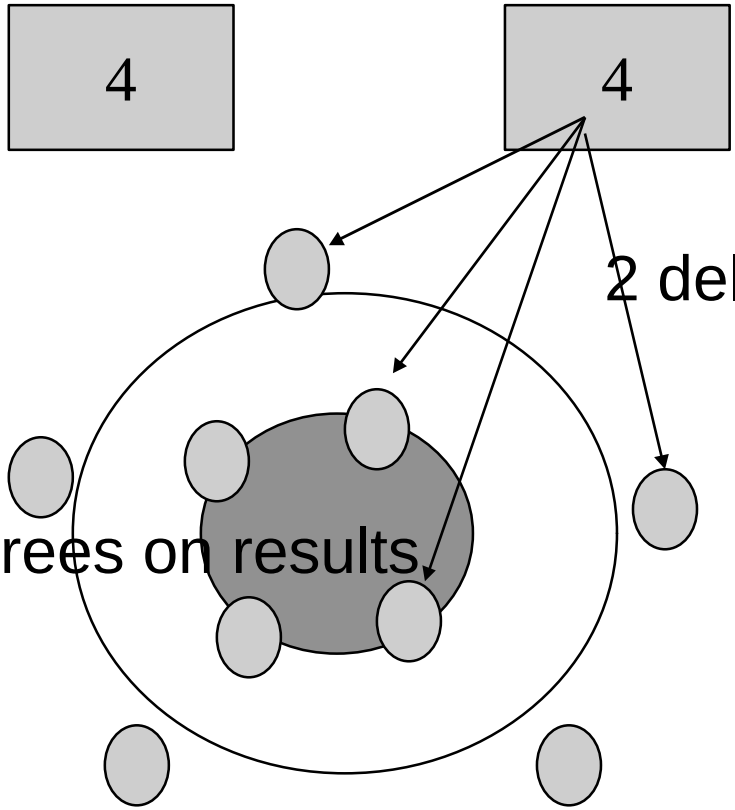
- Case Study consists of 3 Exercises
  - Mile Wide Inch Deep (with shoot out)
  - Project Organization (with shoot out)
  - Detailed Design Work in Groups
- 48 people work in 3 groups of 16
- Ideas from open space conferences – make large groups produce results under time pressure –pressure helps

Process
Knock Out Process

The first two exercises require a an agreed upon result for the third exercise – therefore there is a fomalized discussion (knock out) process ...



Process
Knock out process



2 delegates per round

Inner circle: agrees on results

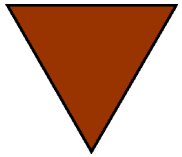
Outer circle: observes

Process
Mile Wide Inch Deep (with Shoot Out)

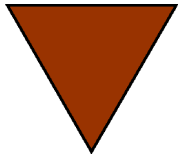
- In this part of the case study you will develop 1 overall design (mile wide – inch deep) for your case study‘s system
- You work in groups of 4
- But you work for your tutor‘s group of 16

Process
Mile Wide – Inch deep

- You work on the case study in groups of 4 people – 45 mins, starting 11:00h
- There will be a shoot out process starting 11:45h
- At 11:52h each tutors group of 16 will have 2 results
- At 12:00h each tutors group of 16 will have 1 result



Exercise
Mile Wide Inch Deep
11:00-11:45

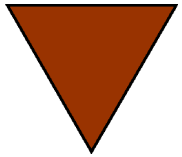


Process
Architecture and Organization Exercise (with Shoot Out)

- In this part of the case study you will propose a team organization for all 16 people for the design phase to follow (exercise 3)
- You work in groups of 4
- But you work for your tutor‘s group of 16

Process Architecture and Organization Exercise

- You will work on the case study in groups of 4 people – 25 mins, starting 12:00h
- There will be a shoot out process starting 12:25h
- At 12:32h each tutor‘s group of 16 will have 2 results
- At 12:40h each tutor‘s group of 16 will have 1 result
- There is a fallback solution if a group has trouble producing a result.



Organisation shapes Architecture - Architecture shapes Organisation Lecture 12:40-13:00 (Wolfgang)

Architecting a Compatible Organisation or How to Help the Architecture to Become Alive

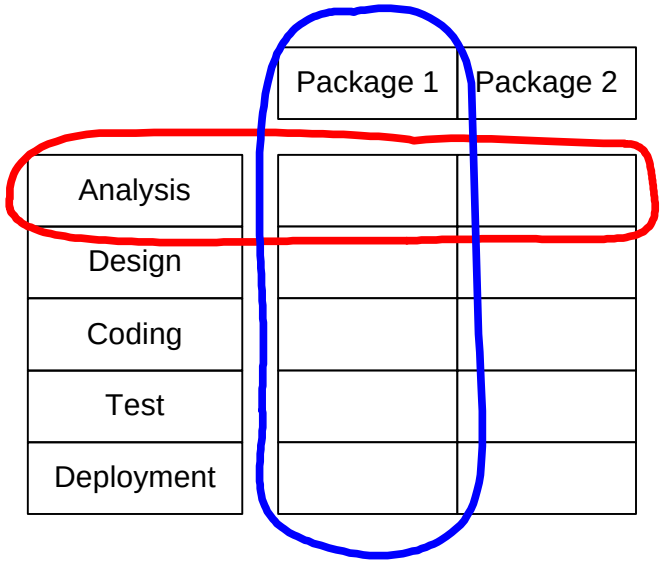
How to Enable Efficient Communication Paths: Convey's Law

- How do you form your roles into an organisation?
  - The team has to communicate along the paths of the architecture
  - Organisation establishes or hinders communication paths
- Make sure the organisation is compatible with the architecture

! The architecture will follow the organisation rather than your ideas if you fail to control the organisation !

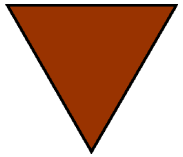
How to Find the Roles You Have to Cover: Form Follows Function

- What roles do you have to cover?
  - Activities are too small to be useful as roles
  - Activities often cluster together domain relationships
- Group closely related activities into roles
- Heavyweight leads to “horizontal” clusters, lightweight leads to “vertical” clusters



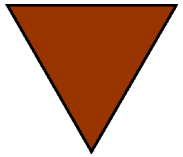
How to Support Architectural Consistency: Keeper of the Flame

- How do you enable the architecture to adapt and grow consistently?
  - A product designed by many individuals lacks elegance and cohesiveness
  - Totalitarian control is not appreciated
  - The right information must flow through the right roles
- Create an architect role that advises and controls the developers while being in close touch with the customer



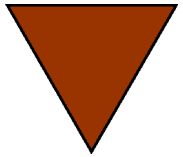
How to Capture Functional Requirements: Scenarios Define Problems

- You need an effective way to communicate between domain experts and technical experts
  - Design documents and figures are often hard to understand for domain experts
  - Domain descriptions are often hard to understand for technicians
  - Design is focussed on abstraction rather than domain issues
- Capture requirements as use cases [Coc00]



How to Make Sure You Are Doing the Right Project: Engage Customers

- How do you best maintain customer satisfaction?
  - Requirements may change after coding has begun
  - Missing customer requirements may lead to building the wrong system
  - Customers often have a different view of the world
- Engage a customer to work in the team. Couple the customer role to developer and architect roles, not only to testing



How to Prevent the Architect From Loosing Ground: Architect also Implements

- How do you preserve the vision through to implementation
  - Architecture rarely is right or complete right from the start
  - Easy-looking ideas may turn out being nightmares
  - Every architecture runs on the virtual Powerpoint machine
  - The architect has to control without being in the management line
- Architects should also implement beyond advising and communicating

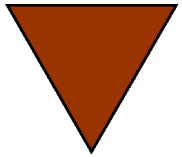
How to Find the Right People to Work on the Project: Self-Selecting Team

- Who should work in your team?
  - Empowerment depends on competency
  - The worst team dynamics can be found in appointed teams
  - Broad interests seem to indicate successful team players
- Build self-selecting teams, doing limited screening on the basis of track records and broad interests

Find the complete text in Jim Coplien: “A Generative Development-Process Pattern Language” in: Coplien, Schmidt: “Pattern Languages of Program Design”, Addison-Wesley 1995

How to Ensure Product Quality: Group Validation

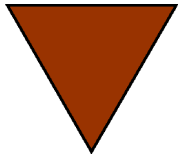
- How do you ensure product quality?
  - QA usually checks the end product only
  - Group sittings may bring additional insights and perspectives
  - Individuals may not have the overall knowledge necessary to discover a bug
  - Some developers don’t like others to see their work
- Engage every team member in finding potential for improvement in the work. Create an environment that accepts and welcomes mistakes



For more Org Patterns see

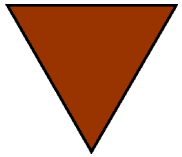
Jim Coplien: “A Generative Development-Process Pattern Language” in: Coplien, Schmidt: “Pattern Languages of Program Design”, Addison-Wesley 1995

Plus Cope’s web site



The Card Thing

14:00 – 14:05 (Alan)



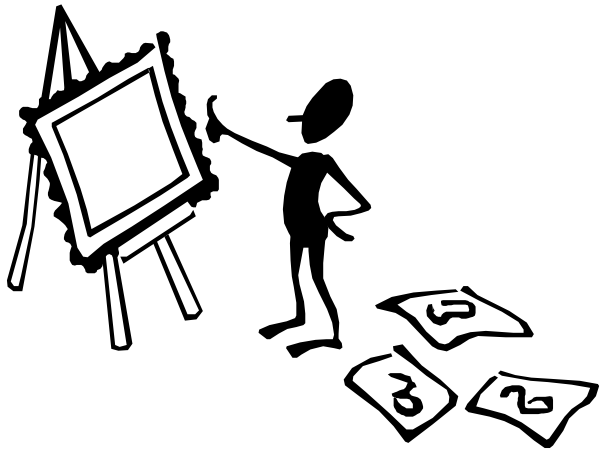
Design - Putting Ideas Into Reality
Lecture 14:05-14:30 (Jens)

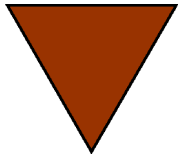
"… Recognize that you are not assembling a building from components like an erector set, but that you are instead weaving a structure which starts out globally complete, but flimsy; then gradually making it stiffer but still rather flimsy; and only finally making it completely stiff and strong…."

Christopher Alexander [Ale77] (pp963-968)

Design and Code Put Architecture into Reality

- Architecture is about separating and connecting parts of the system
- Design is about ensuring the separation and making the connections work - among others
- Without the proper design techniques to implement an architecture, it is not more than an intellectual exercise





Therefore, We Discuss Some Designs that Foster Separation of Concerns

Building what you don't know

- Factory Method
- Prototype

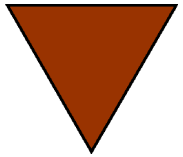
Using what you don't know

- Strategy
- Adapter
- Facade
- Proxy

Breaking Cycles

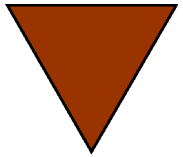
- Inverted Association
- Observer

Find the complete text in Erich Gamma, Ralph Johnson, Richard Helm, John Vlissides: "Design Patterns - Elements of Reusable Object-Oriented Software", Addison-Wesley 1994



Building What You Don't Know: Creating Objects in Other Subsystems

- Despite all polymorphism there is a moment of truth when you have to know the exact class of an object: Creation
- To keep subsystems separated from each other, you need creation techniques that free you from knowing the class to create
- We just show the two most important examples of how to do that



How to Provide an Interface for Object Creation: Factory Method

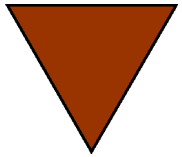
- How do you provide an Interface for Object Creation without exposing your internal class hierarchy?
- Abstract the process of creating objects. Instead of

```
o = new AClass ();
```

put

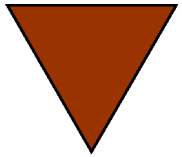
```
o = aFactory.createAClass();
```

- This gives you complete freedom of what to actually create in your subsystem



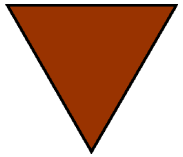
How to Create Objects if the Class Hierarchy Grows Dynamically: Prototype

- How do you create objects if the class hierarchy may change dynamically?
  - This may be due to run-time extensions or because the controlling subsystem doesn't own the subclasses (remember Roles!)
- Register an prototypical instance and create new objects by copying this prototype



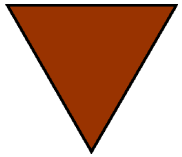
Using What You Don’t Know

- Every exported information limits future changes
- Hence, a package should present as few information to other packages as possible (information hiding)
- Sometimes polymorphism just isn’t enough to ensure encapsulation



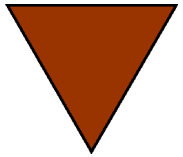
How to let Subsystems Modify the Behaviour of other Packages: Strategy

- How do you design the behaviour of a package so that other packages can modify and extend it?
- Encapsulate the behaviour in subclasses with a common interface. Specify the interface as part of the packages interface so that other packages can add different behaviours
- Alternative: Template Method
- Beware of object creation!



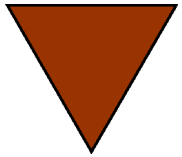
How to Narrow the Information a Package Exports: Facade

- How do you limit the number of classes exported by a package if a package has a fine-grained internal model?
- Define a Facade class that acts as a single interface but delegates all behaviour to other objects
- This pattern is often misused, because the architects did not care about object identity: What does an instance of a Facade mean?



How to Control Access to an Object: Proxy

- How do you hide additional tasks a package has to fulfil if it is accessed?
- Provide a surrogate for the real object at the interface of your subsystem. Each time another package sends a message to this surrogate it can trigger the additional action before delegating the message to the “real” object.



Making a Cycle-less Architecture possible: Breaking Cycles

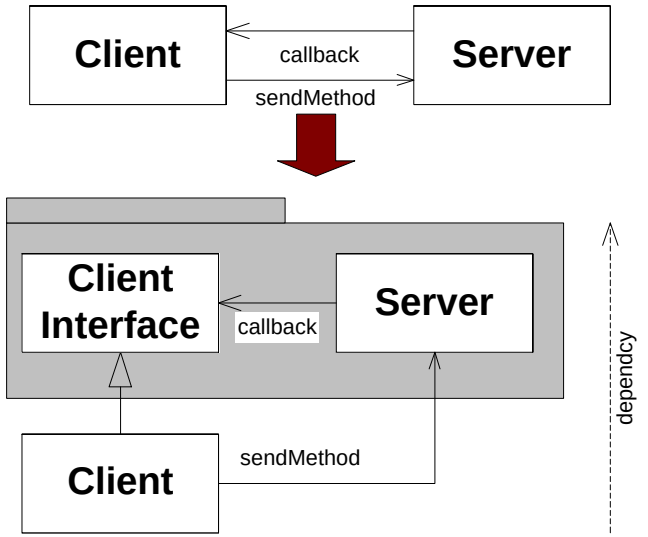
- Cycles between packages are evil!
- Sometimes the domain contains cycles: You want to navigate between objects in different packages in both directions
- If you fail to break these cycles in your design the architecture doesn't work

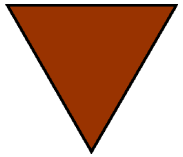
Turning a Cycle Into Two One-Way Streets: Inverted Association

How do you break a cycle caused by messages sent in both directions?

Put one direction on a more abstract level inside of the subsystem. Use interface inheritance to enable external clients to be called back

This is the basic pattern of many decoupling techniques





The Classic Cycle Breaker: Observer

- How do you enable other subsystems to react to changes in your package without having to know the other packages in advance?
- Let clients register as “Observers” of an object. Every time the object changes it sends a “changed” message to all registered observers. It is their responsibility to react accordingly.

Process
Detailed Design Exercise
(with Presentation)

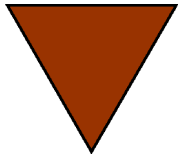
- In this part of the case study you will work in the roles assigned to you by the outcome of the „organization“ part of the case study
- You work in groups of 4
- But you work for your tutor‘s group of 16
- 4 delegates from the four subgroups will prepare a presentation from 15:45 – 16:00

Process
Detailed Design Exercise (with Presentation)

- You will work on the case study in the roles or groups assigned to you, starting 14:30h
- Four delegates will prepare the presentation starting at 15:45h
- At 16:00h each tutor‘s group of 16 will have one presentation and one presenter
- **16:15h: Presentation Alan‘s group**
- 16:25h: Presentation Jens‘s group
- 16:35h: Presentation Wolfgang‘s group

Process
Wrap Up, 16:45

- Please DO NOT FORGET to fill out your feedback form and return it
- Please feel free to send us any suggestions for improvement that might have occured to you – you can use email or make notes on the feedback form



This Presentation is build on the following papers and publications

- Alan O'Callaghan and Kevlin Henney: "Software Architecture with Patterns", OOPSLA 2000
- Alan O'Callaghan: "Patterns for Architectural Praxis", EuroPLoP 2000, http://www.coldewey.com/europlop2000
- Jim Coplien: "A Generative Development-Process Pattern Language" in: Coplien, Schmidt: "Pattern Languages of Program Design", Addison-Wesley 1995
- Jens Coldewey: "Decoupling of Object-Oriented Systems", Coldewey Consulting 2000, http://www.coldewey.com

References

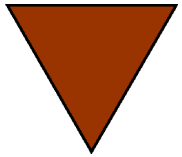
[Ale77]: Christopher Alexander, Sara Ishikawa and Murray Silverstein with Max Jacobson, Ingrid Fiksdahl-King and Shlomo Angel, *A Pattern Language: Towns, Buildings, Construction*, Oxford University Press, 1977.

[Coc00]: Alistair Cockburn: “Writing Effective Use Cases”, Addison-Wesley 2000

[GOF95]: Erich Gamma, Ralph Johnson, Richard Helm, John Vlissides: “Design Patterns - Elements of Reusable Object-Oriented Software”, Addison-Wesley 1994

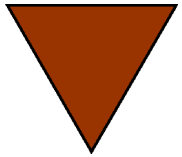
A few Process Observations
Final Remarks from Wolfgang

- Subteams work – Architect communicates: Have you observed a problem?
- If vision not clear enough => double work
- What it needs is a clear common picture – a basic idea
- Beware of the technical infrastructure team (the framework guys)
- The glossary problem ☺



A few Process Observations
Final Remarks from Wolfgang

- There was „unemployment“ like in real life, due to ???
- Responsibility driven design seems to help – people were looking for the glossary but discussing responsibilities



Some final remarks from Jens

- Archtecture is about communication!
- Steps to a successful architecture:
  - Build up a communication culture
  - Understand the problem
  - Do a good package design based on abstractions
  - Make sure everybody was involved
  - Nothing is written in stone - Be ready to change everything later

Alan – Final Remarks based on Feedback Cards

- Will be added later
- Wolfgang has the cards and will document them ...

