Software Design

Jesper Andersson

Linnæus University

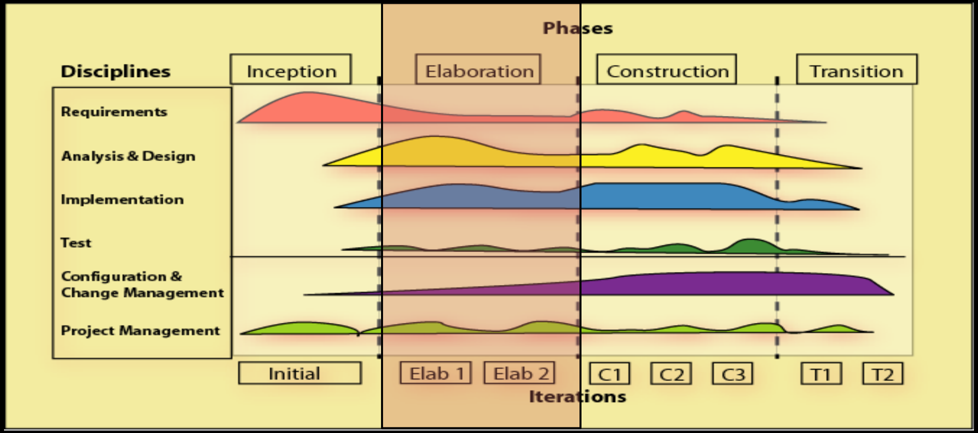
What is Software Design?

- ✓ Purpose:
  - – Describe and Understand the solution
  - – Create a model of the solution using the **language and technology** of the implementers. Close to implementation language and APIs, frameworks, middleware, etc.
- ✓ Models a system of communicating objects performing different tasks
- ✓ Decisions regarding Implementation Technology!
  - – Language
  - – Database
  - – etc.

From Use Case to Code

- ✓ Requirements
  - Use case model
  - System wide requirements
- ✓ Implementation technology
  - Languages
  - Frameworks (reuse)
  - Middle-wares (reuse)
- ✓ Design
  - Static model of you user defined types (classes)
  - Dynamic model of collaborating objects

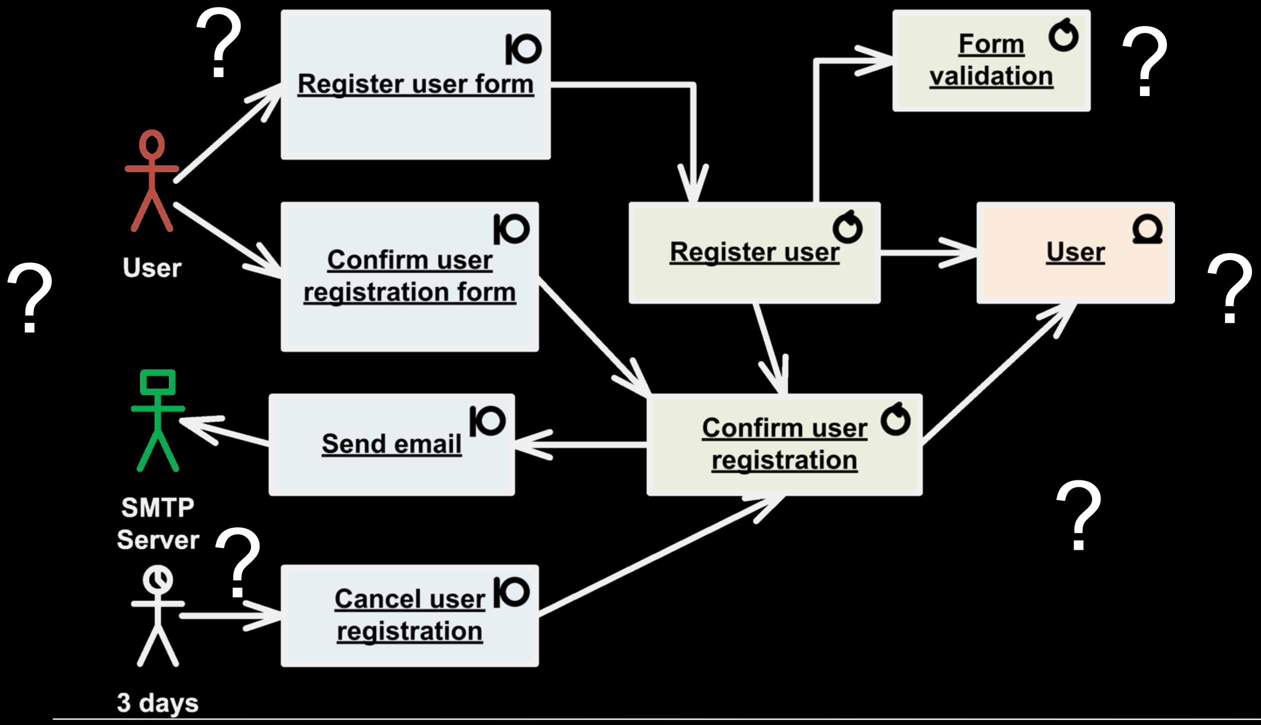
UP – Elaboration



Elaboration

- ✓ **Purpose:** *Establish the system's baseline architecture*. This will be the foundation for the the continued development in the next phase
- ✓ Requirements – more details and better understand requirements
  - More insights regarding critical requirements at the architecture level➔ for example understand how the qualities interact
- ✓ Design, implement, and validate ➔ a baseline architecture.
- ✓ The **Baseline architecture**: A skeleton with "critical functionality"
- ✓ Risk management
  - Mitigate the more critical risks
  - At the architecture level

Example Chat server



Linnæus University
Sweden

More Questions to Answer

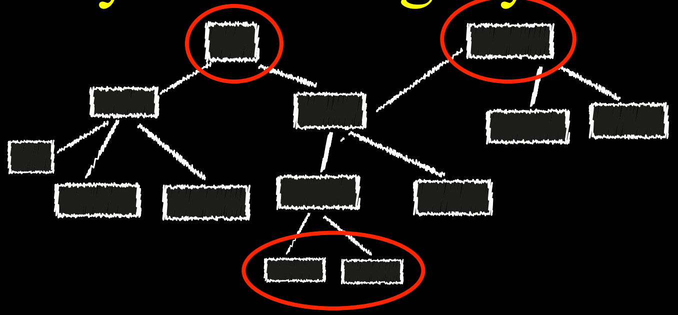
- ✓ What will the system structure look like?
- ✓ How should architecturally significant requirements be realized?
- ✓ What technologies will be used to implement it?
  - Software Reuse, COTS, Open Source
  - Develop.

- ✓ Iterative and Incremental
  - Each iteration will add more **capability** and
  - **provide more detail**

Software Architecture

- ✓ System decomposition
  - – How is the divided into subsystems?
  - – Do we have all the parts?
  - – Will the parts fit together?
- ✓ System concerns
  - – Responsibilities
  - – System characteristics and qualities
  - – System quality trade-offs
- ✓ **System integrity**

The Design Decision and System Integrity



- ✓ Important decisions first
- ✓ Details wait
- ✓ System Integrity
  - Decomposition
  - Some decisions must be made before decomposition!

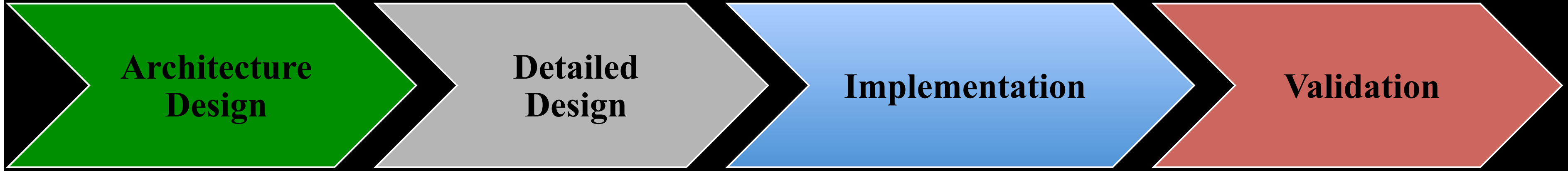
The Important Design Decisions - Architecture

- ✓ Goal: Architecture that "enables" (make it possible to achieve) qualities
- ✓ Effect: System Global
- ✓ Example: Implementation technology
  - – Frameworks
  - – Middleware
- ✓ Example: Build new or Reuse
- ✓ Example: Quality realization – Authentication strategy
- ✓ Example: Allocate responsibilities – Component model

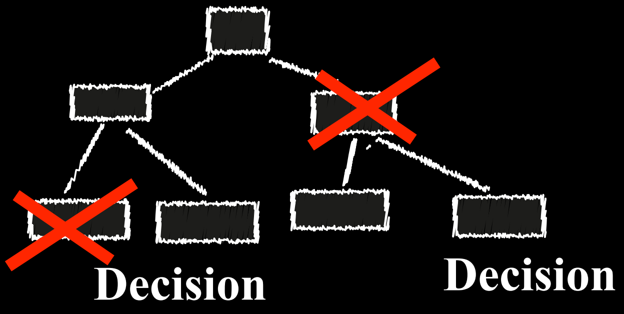
The Important Design Decisions – Detailed Design

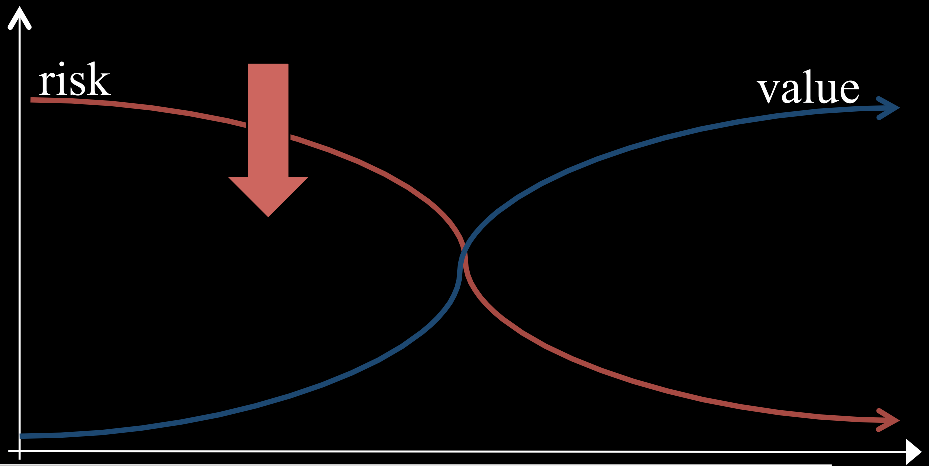
- ✓ Goal: Optimized solution – Effect: Local
- ✓ Use case based
  - Implement boundaries (communication, IO, GUI, event handlers)
  - Implement entities (data structures, storage, persistent/transient)
  - Implement control (algorithm)
- ✓ Example: Type Hierarchy
  - Inheritance structure
  - Design patterns
- ✓ Example: Type implementation Specifics
  - Algorithm
  - Data structure

Design process – Iteratively

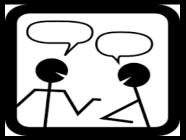


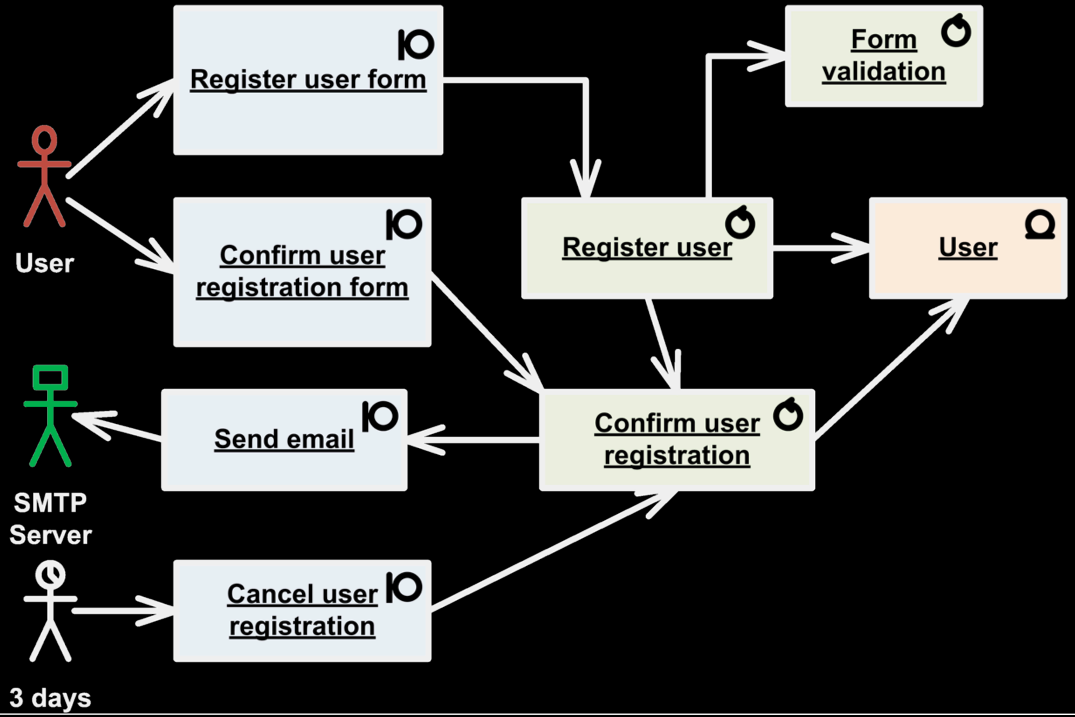
1. Create something stable
2. Iterate an improve incrementally





The Chat server revisited



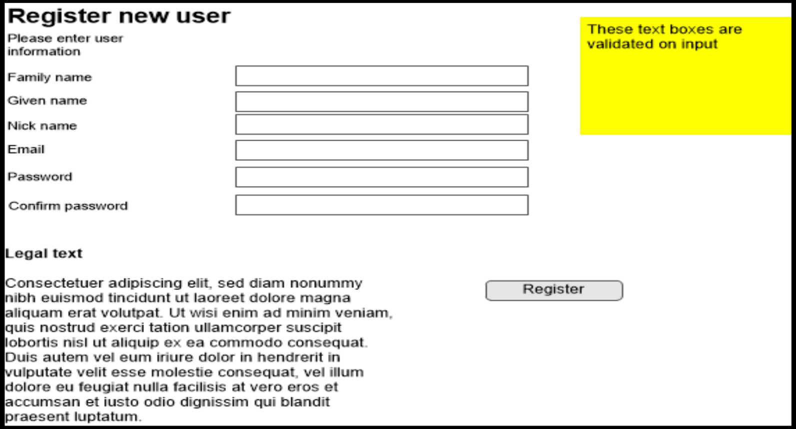


What would this Form look like?

Register user form

- ✓ Architectural level
  - – Implementation technology
    - GUI
    - Web/Client/Mobile (Cross platform development)
  - – Will impact Components and allocation of Responsibilities
  - – UI Design Patterns for the project.
- ✓ Design Level - Design UI
    - UI Patterns
    - UI Static
    - UI Behavior

From Mockup to Implementation



Register new user

Please enter user information

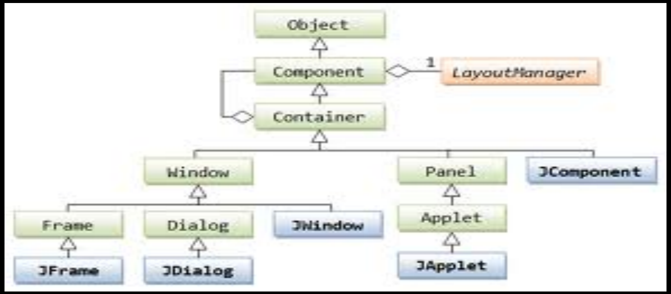
Family name	
Given name	
Nick name	
Email	
Password	
Confirm password	

These text boxes are validated on input

Legal text

Consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum.

?



Linnæus University
Sweden

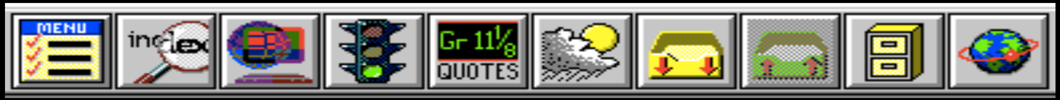
GUI Design

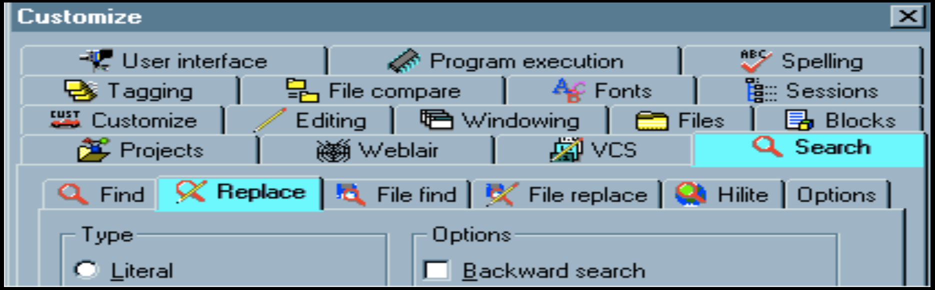
8 Golden Rules of User Interface Design

1. Strive for consistency
2. Enable short-cuts for frequent users
3. Informative feedback
4. Design dialogs to yield closure
5. Offer simple error handling
6. Permit easy reversal of actions
7. Support internal locus of control
8. Reduce short-term me

From Designing the User Interface by Ben Shneiderman

User Interface Hall of Shame



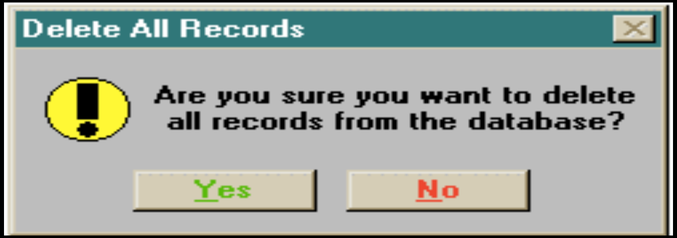

Customize

User interface | Program execution | Spelling
Tagging | File compare | Fonts | Sessions
Customize | Editing | Windowing | Files | Blocks
Projects | Weblair | VCS | Search

Find | Replace | File find | File replace | Hilite | Options

Type
Literal

Options
Backward search


Delete All Records

Are you sure you want to delete all records from the database?

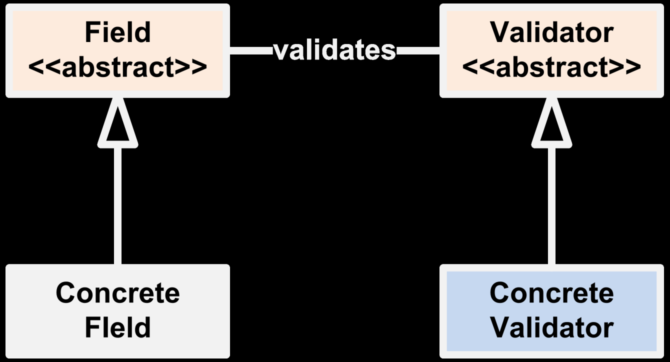
Yes | No

http://ui-patterns.com/

Linnæus University
Sweden

What are the validation rules for this?

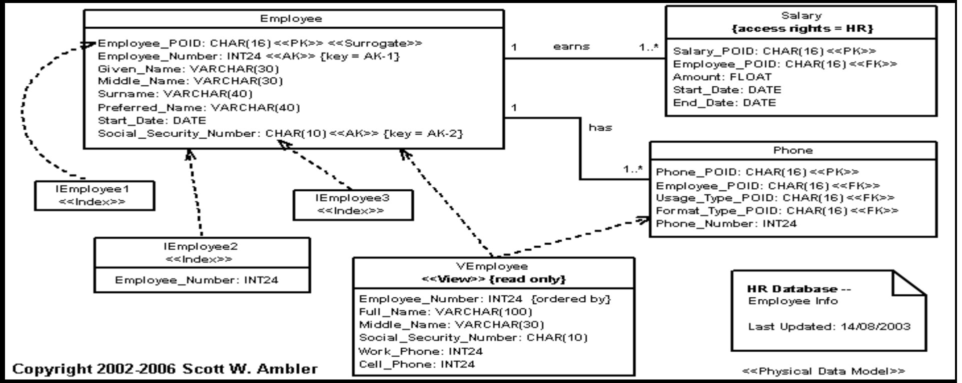
- ✓ Architecture level
  - – Common validation rules and mechanism for Usability Quality
  - – Interface to the validation and possible error messages.
- ✓ Design level
  - – Define and Describe the validation algorithm for this particular validation
  - – Describe what it should return
  - – A design pattern? How many input forms do we have?



Linnæus University
Sweden

How is data managed and stored?

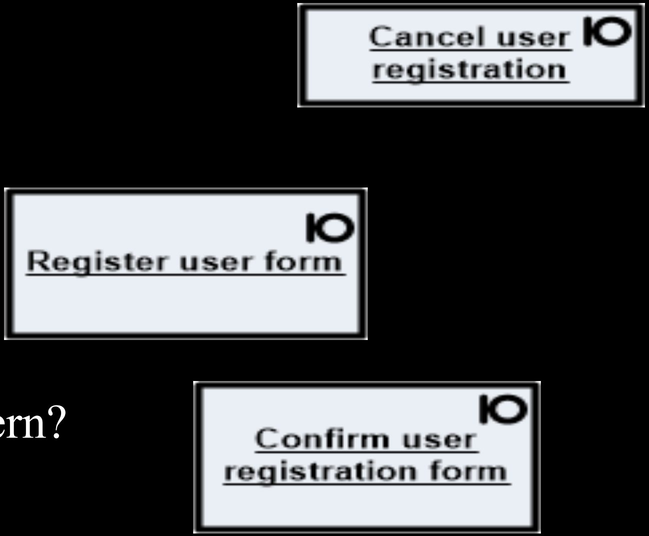
- ✓ Goal: Software Quality – Persistency
- ✓ Architecture level
  - – Implementation technology
    - • DBMS, Caching
    - • Transactions
  - – Backup?
- ✓ Design level
  - – “Database design”, Schema design RDMS
  - – “Connection” to data
  - – Design patterns (e.g. ,DAO)


Employee
Employee_POID: CHAR(16) <<PK>> <<Surrogate>>
Employee_Number: INT24 <<AK>> {key = AK-1}
Given_Name: VARCHAR(30)
Middle_Name: VARCHAR(30)
Surname: VARCHAR(40)
Preferred_Name: VARCHAR(40)
Start_Date: DATE
Social_Security_Number: CHAR(10) <<AK>> {key = AK-2}
1
earns
1..*
Salary
{access rights = HR}
Salary_POID: CHAR(16) <<PK>>
Employee_POID: CHAR(16) <<FK>>
Amount: FLOAT
Start_Date: DATE
End_Date: DATE
1
has
1..*
Phone
Phone_POID: CHAR(16) <<PK>>
Employee_POID: CHAR(16) <<FK>>
Usage_Type_POID: CHAR(16) <<FK>>
Format_Type_POID: CHAR(16) <<FK>>
Phone_Number: INT24
IEmployee1
<<Index>>
IEmployee3
<<Index>>
IEmployee2
<<Index>>
Employee_Number: INT24
VEmployee
<<View>> {read only}
Employee_Number: INT24 {ordered by}
Full_Name: VARCHAR(100)
Middle_Name: VARCHAR(30)
Social_Security_Number: CHAR(10)
Work_Phone: INT24
Cell_Phone: INT24
HR Database -- Employee Info
Last Updated: 14/08/2003
Copyright 2002-2006 Scott W. Ambler
<<Physical Data Model>>

Linnæus University
Sweden

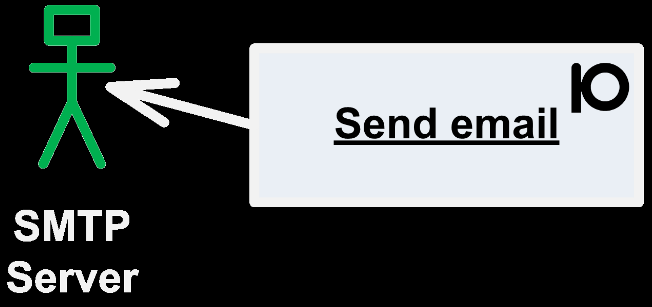
Which technique will we use for the user interface?

- ✓ Architectural level
  - – Client types
    - GUI
    - WEB?
    - Mobile
  - – Architecture pattern?
- ✓ Design level
  - – Cross platform design?
  - – Separation of concerns

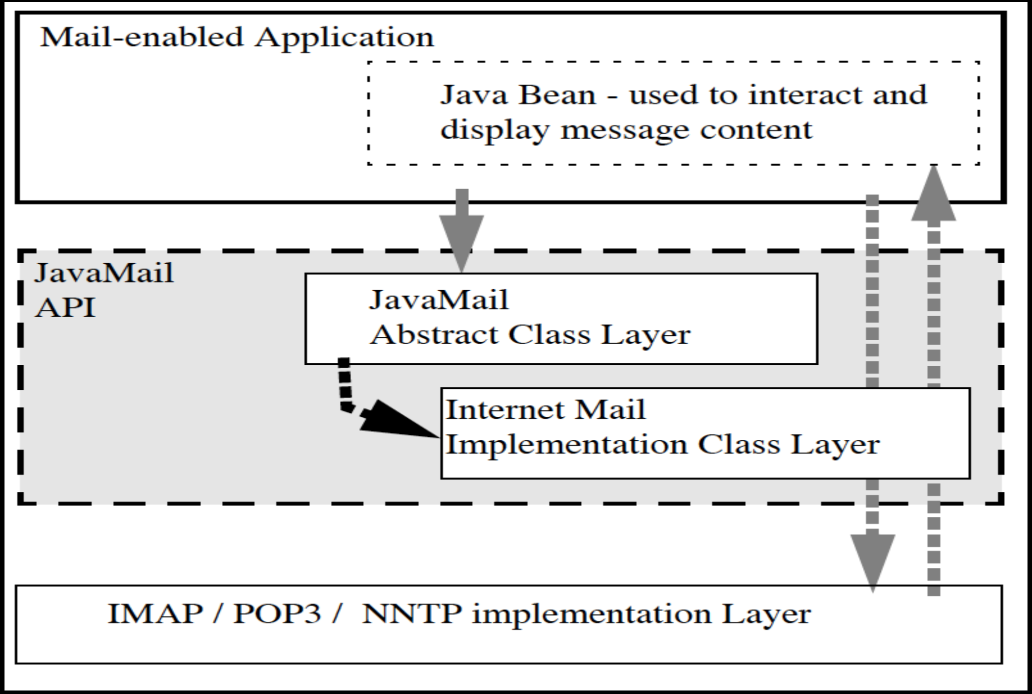


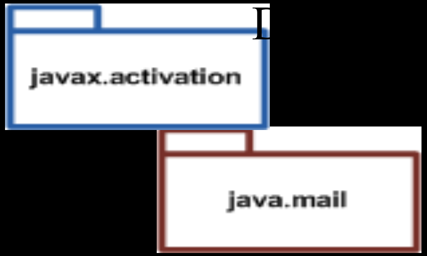
How do I communicate with the Mail server.

- ✓ How do create and send emails?
- ✓ Architectural Decision
  - – Implementation technology
    - External systems
    - Frameworks
- ✓ Design level
  - – Interfaces
  - – Specializations



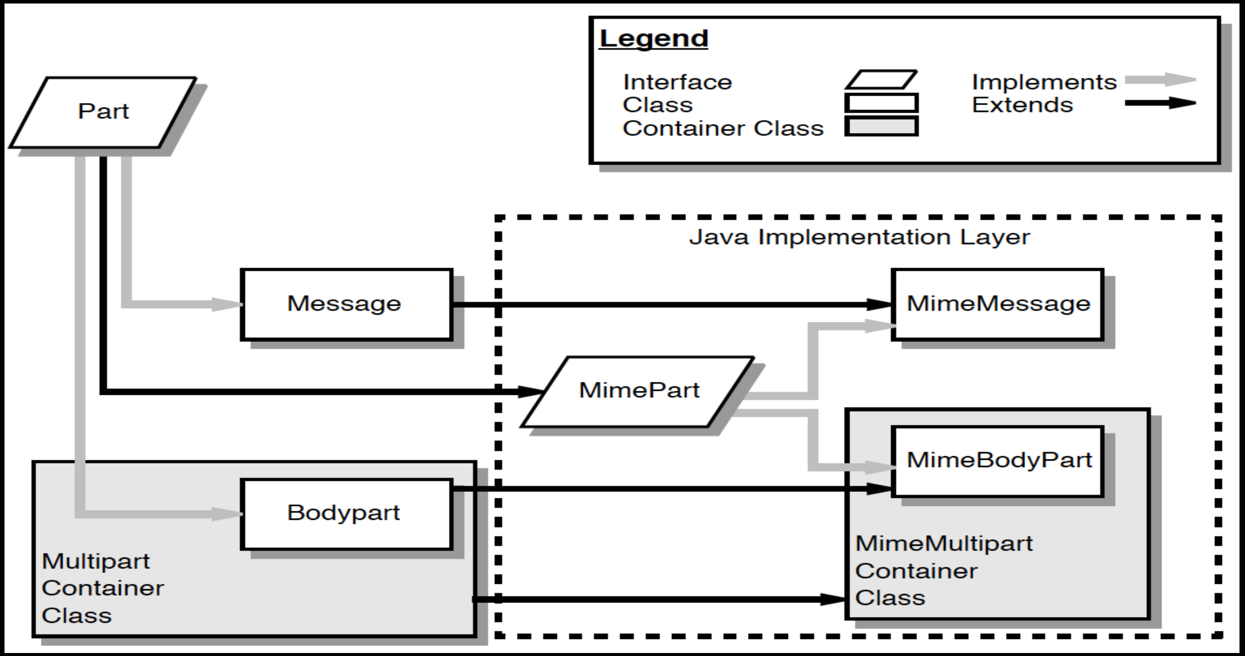
Example: Java Mail API – Decisions are connected


Mail-enabled Application
Java Bean - used to interact and display message content
JavaMail API
JavaMail Abstract Class Layer
Internet Mail Implementation Class Layer
IMAP / POP3 / NNTP implementation Layer



Linnæus University
Sweden

Example: Java Mail API – Class Hierarchies (part-of)



What are the Characteristics of this server?

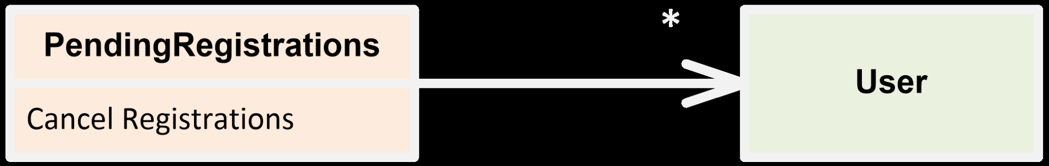
- ✓ Operating system version etc.
- ✓ Installed software
- ✓ Networking

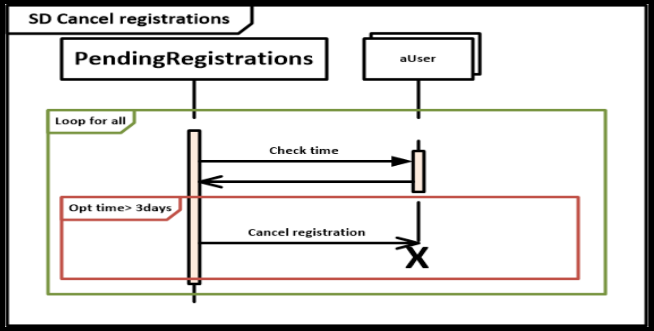
- ✓ Why is that important?
  - Compatibility
  - Testing

- ✓ Use UML component diagrams

How will I cancel registrations after 3 days?

- ✓ Architecture level?
  - Not much
- ✓ Design level
  - Design for this collaboration
  - Trigger it daily or more frequent
- ✓ Example of designing the DOMAIN



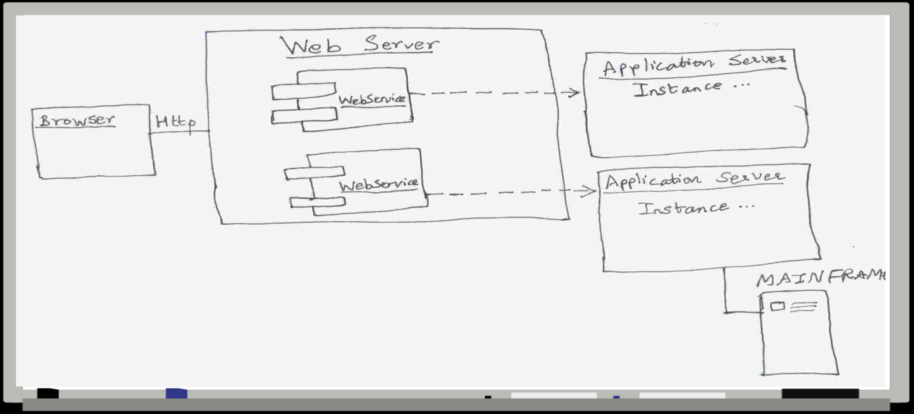


Linnæus University
Sweden

System level structure – Decomposes the Solution

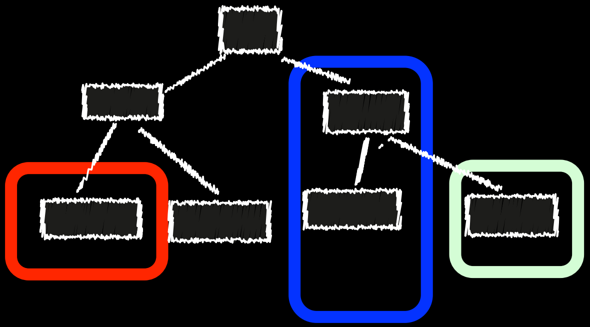
Example Restful API

- ✓ Each request from any client contains all the information necessary to service the request, and session state is held in the client.
- ✓ HTTP
  - – GET
  - – POST
  - – PUT
  - – DELETE
- ✓ HTTPS?
- ✓ JSON/XML for data

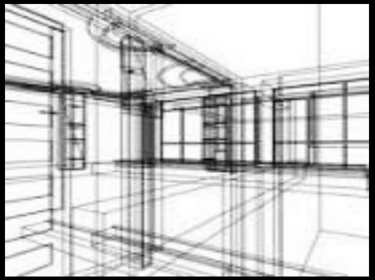


Design and Software Reuse

- ✓ Important aspect of elaboration
- ✓ Identify how you will work with reusable assets in your project
  - Build them!
  - Reuse them!
  - Buy them!



Reuse and Architecture



- ✓ Reuse is important when you have to build up your baseline architecture quickly, with quality
  - Frameworks
  - Platforms
- ✓ The architecture divides the system up into subsystems and defines the subsystems’ interfaces
- ✓ Excellent starting point for start looking for components
- ✓ The architecture in a system is often determined by the reused components in the system.

Types of Reuse

- ✓ Applications
  - – Reuse and adapt applications (COTS), for example MS-Excel
  - – Development of application families, for example MS-Office
- ✓ Reuse of components
  - – Components, subsystems or classes are reused in another system
- ✓ Reuse of individual functions
  - – Small components that implement some well-defined functionality.
  - – For example a method that sorts integers

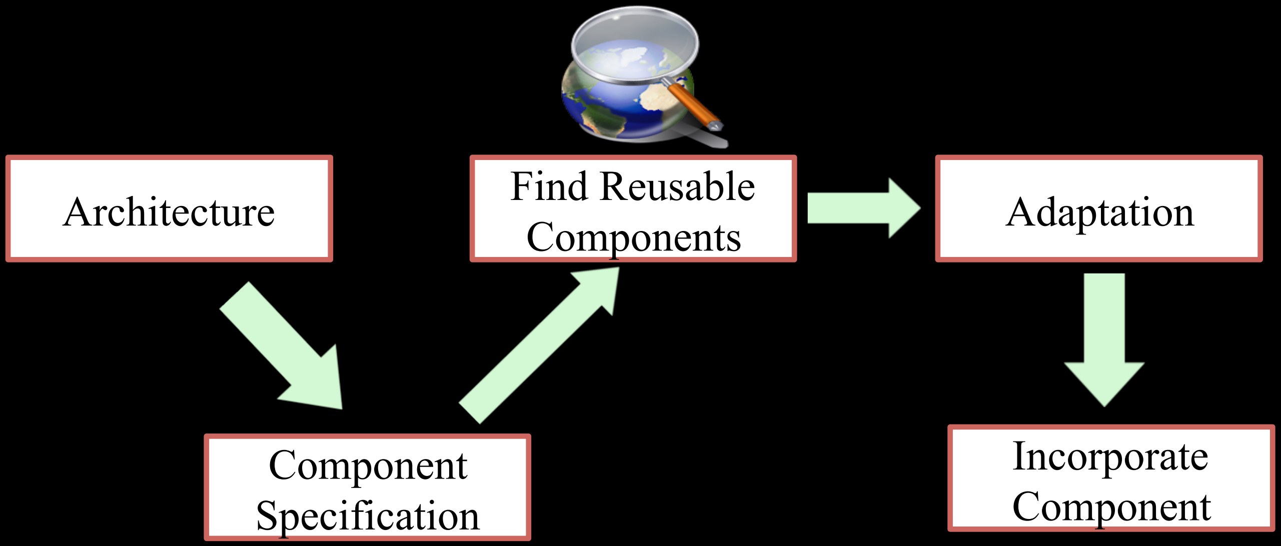
Opportunistic vs. Systematic

- ✓ The opportunist, as soon as there is something can win/save you reuse
  - – Copy - paste
- ✓ Systematic reuse, something different…
- ✓ If reuse should be efficient it must be systematic!!

How to “use” Reuse

- ✓ Design with reuse
- ✓ Design for reuse
- ✓ Generator-based reuse
- ✓ Application system reuse

Development with Reuse



Linnæus University
Sweden

Development with Reuse

An "algorithm"

- ✓ **Find** right component
  - – Difficult, should match requirements
  - – Solution ➔ Develop a reuse organization
- ✓ **Incorporate** the component
  - – Adapt the component
    - Interface adaptation (wrapping)
    - Invasive adaptation
  - – Integrate into system

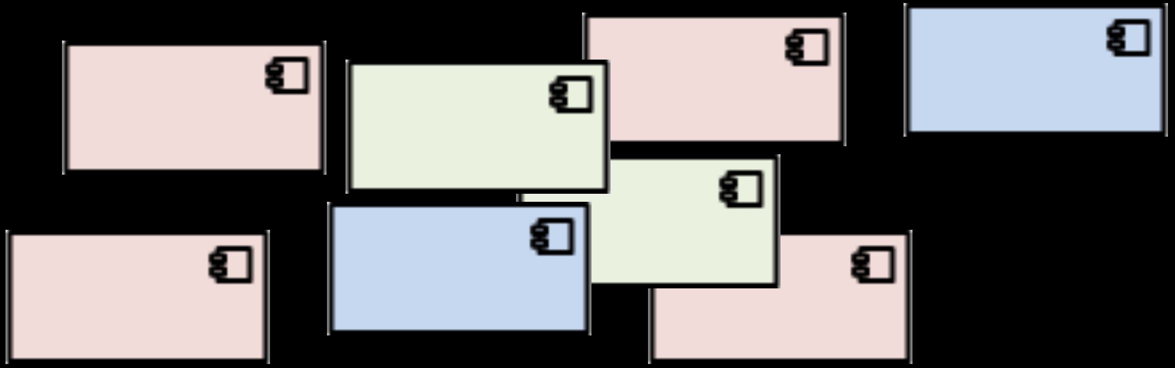
Problems with Reuse

- ✓ Difficult **to find** good and reusable components
- ✓ Components could be difficult to **understand**
  - – Methods - ok!
  - – Classes - …yes!
  - – Sub-systems - not that easy!
- ✓ “**Not invented here**” - system developers relucta work

Linnæus University
Sweden

Development for Reuse

- ✓ Goal is to develop a base of reusable assets
- ✓ More difficult than traditional development
  - Don’t know how the component will be used!
  - Making a component reusable is difficult!



Linnæus University
Sweden

Development for Reuse – How?

- ✓ Oracle
  - What should be included?
  - Which variability should be supported?
- ✓ Organization/Structure
  - Patterns
  - Naming
  - Methods, constructors, operators etc.
  - Exception handling
  - Generics/templates
  - Life-cycle aspects for reusable assets!

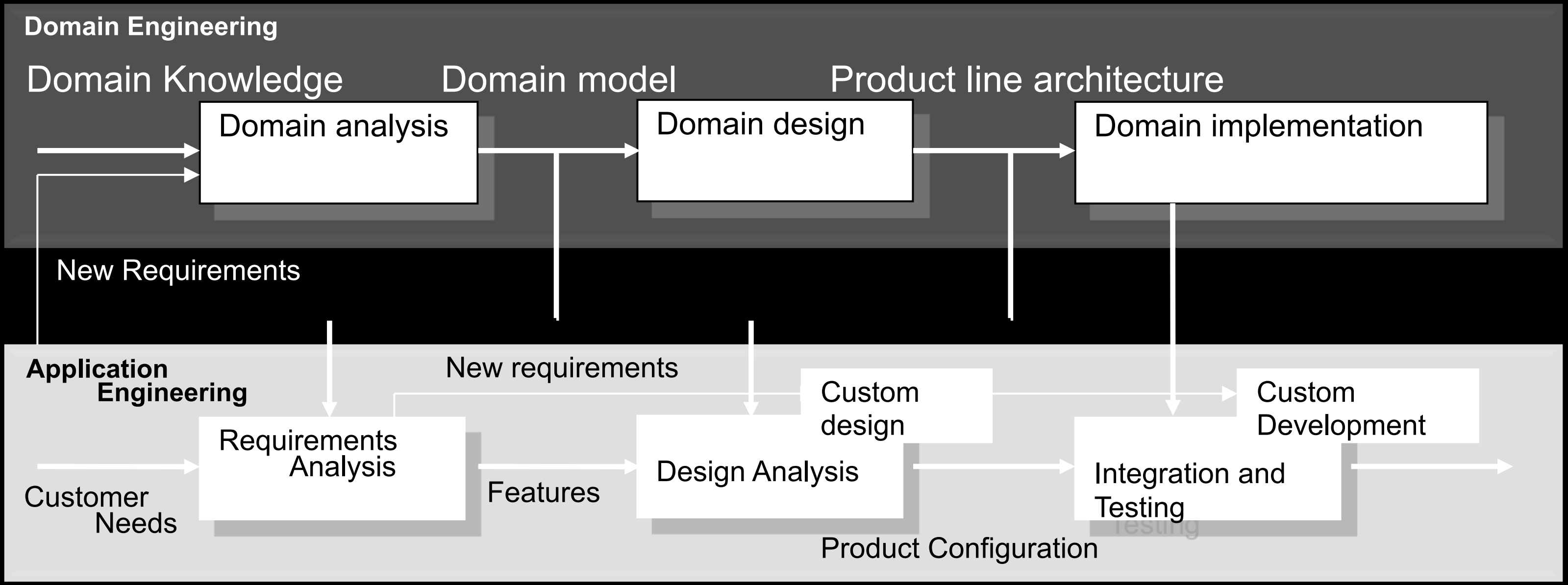
```
<html>
    <head>
        [illegible]
    </head>
    <body>
```

Linnæus University
Sweden

Software Product lines

- ✓ Creating a product line of software applications is not simply just a matter of finding useful pieces of code and bolting them together in an ad-hoc fashion.
- ✓ **Domain engineering** has emerged as a means to define families and assemble the required assets.
- ✓ **Commonality and variability** analysis is a process of identifying common and variable parts in the domain.
- ✓ Combines *Development for Reuse* and *Development with Reuse*

A Process model with Domain Engineering



Knowledge Reuse – Software Patterns

- ✓ Recurring solutions to common software design problems in the Object-oriented paradigm
- ✓ Make OO designs more **flexible**, **elegant** and **reusable**. Improve documentation and maintenance by a **shared vocabulary**
- ✓ GoF Design Patterns
  - Creational – these abstract how objects are instantiated
  - Structural – abstracts how classes/objects may be organized
  - Behavioral – abstracts object-to-object communications

Takeaways

- ✓ Design decisions
  - – Architectural
  - – Detailed
- ✓ Approach is iterative and incremental
- ✓ Create something stable and evolve!
- ✓ Reuse when you can
- ✓ Develop for reuse when you can afford it