Case Study: Vienna International Airport

Development Strategies for Complex Software Systems

What is a Complex System

Complex Systems have

- Magnitude: Number of Elements, states, possible vs usable solutions
- Diversity: heterogeneous (many different Elements)
- Connectivity, structural Complexity: many connections between Elements

Literature defines systems as complex if

- they consist of many interacting components
- simple linear modeling insufficient to understand

Engineering in Heterogeneous Software Landscapes

Large Scale Engineering, eg car manufacturing
Cooperation of Disciplines
Heterogeneous

- Technical: Tools that do not fit seamlessly
- Semantic: Engineers, Tools and Systems use different data models
- Process: Engineers tend to extend their liberty to project mngmt level -> chaos

Integration

- Technical
- Semantic

OpenEngineeringServiceBus

Examples of Usage

- Engineering Cockpit: view on the project progress
- Engineering Object Editor: Change Management, Conflict Resolution
- Engineering Object Change Management: Change Management

Constant Changes

- Changing customer requirements
- Available Resources Change

OESB Development

- Novel combination technologies

Levels of Hierarchy: Epic > Story > Task
Tools

- Excel: Offline, good performance, Minimal learning curve
- Jira: progress monitoring, filtering, historical project analysis

Adapting Software Development Processes

Goal: Maximizing Customer Satisfaction
Feedback Loop with Customer
Use

- Sketches
- Mocking
- Prototype: Are no Mocks, show practicability of approach

Open Source Software

- Free and Open Source Software
- Redistribution, Derived Work allowed
- Source Code available, may restrict modification (only through patch files)
- License has to be distributed

Organisations

- Free Software - Free Software Foundation (FSF)
    - Free as in Freedom
    - Copyleft: All rights to the user
    - GPL
- Open Source Software - Open Source Initiative (OSI)
    - Access to Sourcecode
    - Copyright: All rights to the contributor
    - BSD

Cathedral vs Bazaar

- Cathedral: Code available with each release, access between restricted, Emacs
- Bazaar: Developed in public, Code always available. Linux Kernel

Business Models

- Sell Services: Customization, Training
- Sell Extensions and features
- Dual license models
- Sponsorship and Donations

Communities

- Communication completely open
- Code Repo, Bug Repo
- Forges provide most common infrastructure

Case Study: Eclipse

Eclipse Foundation funded by its members, responsible for

- Intellectual Property Management (EPL)
- Development Process

3 Communities

- Contributors and Committers
- Users
- Adopters

Eclipse Councils

- Planning: responsible for coordinated Release Plan
- Architecture: Monitoring and guiding the Software Architecture

Eclipse Top Level Projects

- JDT: Java Development Tools
- Mylyn: Task Integration and Connectors
- Modelling
- SOA: Service Oriented Architecture
- WTP: Web Tools Project

Top Level Project Management Committee (PMC)

- Provides leadership for projects in top level project, One or more leads, zero or more members

Project Lifecycle

- Proposal -> Incubation -> Mature -> Top-Level -> Archived

Preconditions for Project Creation Review

- Proposal document
- Well defined scope
- Two council mentors

Eclipse Public License (EPL)

- Not compatible with GPL
- Weak copyleft
- business friendly free license

Eclipse Mylyn: Task focused interface for Eclipse, Integration with Bugzilla, Trac, JIRA
TapiJI: Tools for Java Internationalization

Model Driven Development

- Development approach that uses models
- Uses automation techniques
    - code generation
    - model interpretation

- Model has to be
    - expressive enough
    - precise
    - supported by tools

Objectives

- Higher abstraction level
- Reduce development time
- Improve application quality
- Reduce maintenance costs

Model

- Representation of a part of the reality
- Captures only a small amount of the reality
- Helps to communicate ideas

Ambient Intelligence

Smart environments, Systems for controlling the environment in an intelligent way

- Tasks repeatedly performed. Example: when leaving close windows, lock door
- Automation of these task reduces user tasks, Improves Quality of Life

Problems of current solutions

- Limited expressiveness
- Low level of abstraction
- Hard to understand

A Model Driven approach is

- easier to understand
- end user can participate

State of the Art

- Machine Learning: Infer from past user behaviour
- Context Aware Rules: Rules execute tasks when certain Conditions are fulfilled
- End User Oriented: Provide End User with Tools to adapt Automations by himself

The Approach

- Models
    - Example of Model: At 8:00 on working days the alarm goes off to wake up bob. He switches the lights on, raises the blinds.
- Evolution
    - models adapt execution depending on context
    - when the models changes, the changes are applied immediately
    - Graphical tool for end user

Evaluation

- based on case studies
    - Smart Homes
    - Nursing Home: ACube Project
- Models, have to be expressive enough, help to find out what users wants
- Benefits of the proposal

    - Participation of the End User
    - Rich Expressiveness
    - Evolution at Runtime

Component-based Software Engineering

- Rapid Development
- Reuse of Components
- Separation of Concerns
- Clean Architecture

A Software Component is

- Part of a System that performs a certain functionality
- Logically Cohesive, Loosely Coupled Module, Denotes Single Abstraction
- Unit of Composition with Well-Defined Interfaces and Dependencies

Component Frameworks

Examples

- Spring
- Tapestry
- Guice
- Wicket

Low Level Component Frameworks

- OSGi
- Jigsaw Java

Problems

- Startup Time
- Scalability
- Isolation
- Life-cycle management
- Native Support

OSGi

Vendor is OSGi Alliance

OSGi is

- Platform independent
- Provides Isolation and Security

Implementations

- Equinox
- Knopflerfish
- Apache Felix

OSGi Environment

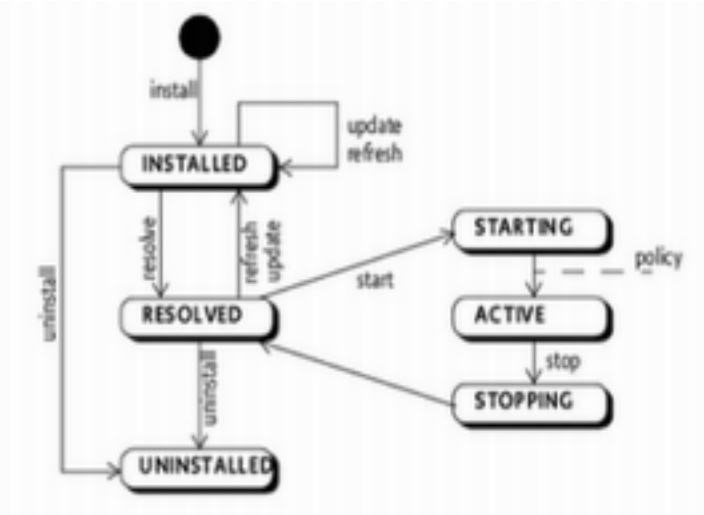
- Core Specification

- Service Compendium
- Residential Specification

Bundles

- Deliverable Application
- Registers Services, Exports and Imports Packages, Searches for Bundles
- Managed Class Path Dependencies
- Allows Long Running Applications and Dynamic Software Updates
- Each Bundle has own ClassLoader

Bundle Lifecycle



Services in OSGi

- an Object to be used by other bundles, like a java Interface, has unique id
- different bundles can implement the same service
- OSGi defines a standard set of services

Service Registry

- available to all bundles
- services have properties, query language to find service

Jigsaw

Vendor Oracle

- JDK/JRE Platform Modularization
- Modularity is a Language Construct
- Strongly Enforced Module Boundaries

Mobile Devices

Smartphone Characteristics

- Limited Resources: CPU, RAM, No Virtual Memory
- Battery
- Has to be Reliable
- Always on

Smartphone Sensors

- GPS
- Compass
- NFC, RFID
- Barometer
- Accelerometer
- Proximity Sensors
- Light Sensors

Types of Mobile Apps

- Native Client
- Java Platform
- Widgets
- Mobile Websites
- X-Platform Frameworks

Platforms

Apple iOS

- Closed, Certificate and Mac required for development, Apps reviewed by Apple
- Device Performance not Sky Rocketing
- Language: Objective C

Google Android: by the Open Handset Alliance

- Linux-Based
- Free Licensing
- Language: Java

Symbian

- Dead
- Separation of OS and GUI
- Language: C++

Java Mobile

- Cross Platform Development Solution
- Main Problem: Fragmentation
- Optional APIs defined eg Bluetooth, Location, 3D

BlackBerry

- future unclear
- J2ME with extensions, Backend System operated by RIM

- long battery life

Palm / WebOS

- Dead
- using HTML/CSS based applications

Windows Mobile Phone Edition

- WP7
- Windows CE
- often used for industrial applications

X Development Frameworks

- MONO Based: Free Implementation of .NET Runtime Environment
- HTML5, CSS, JS: Jquery Mobile, PhoneGap
- Google Applause
- Ansi C: Works on iOS, WP7, Android NDK, Useful for Algorithmns

Important Technologies

ARM

- does not build chips, only designs chips, many patents
- best watt-to-MIPS ratio, not x86
- Current Devices have serveral ARM Chips

NFC

- derived from RFID
- Range 10cm, Multiple Operation Modes (Smart Card, RW, P2P)
- Use Cases: Payment, Ticketing, Access

Java Card

- applications on Chipcard, Write Once, Run Anywhere

Oyster Card

- for Public Transportation
- Automation
- Less Queuing

How to Bill a Customer

- Sms based
- MSISDN for billing (get it through SMS or IP based lookup)
- SMS text triggers application
- Clearing by Paybox
- Mobile Network Operator provides Bank Data
- Authorization SMS “Ja”

Guidelines for Mobile App Development

Design

- Start with Sketches
- Know your Technology
- Use MVC

- Abstract Sensor and IO Data

Implementation

- Use Common Tools: More Help is Available
- Different Platforms require Different Methods
- Make Prototypes if you don't know a technology
- Develop at the Phone, Not in the Emulator
- Never consider an app or a device as Secure

Testing

- Automate Testing
- Very Time Consuming: Different devices, Carriers
- Use Acceptance Tests for UI, Tracing for the Implementation

Anti Patterns

What is a Pattern

- Solution to a Problem in a Context
- An Idea that has proven Useful and might also be in similar Contexts

Patterns help to

- provide Common Vocabulary
- Manage Complex Systems
- Facilitates Non-Functional Requirements
- Minimize Time and Costs
- Improve Documentation

Classification of Patterns

- Architectural Patterns: Structure and Relations at System Level
- Design Patterns: Structure and Relations at Class Level
    - Creational
    - Structural
    - Behavioral
- Idioms: Low-Level Details, Language Specific
- Protopatterns: Particular Case, Solution to be used in Larger Scale

Anti Pattern

- Describes Solution to a Problem in a Context which generates Negative Consequences
- Helps to Recognize common Mistakes

Key Concepts of Anti Patterns

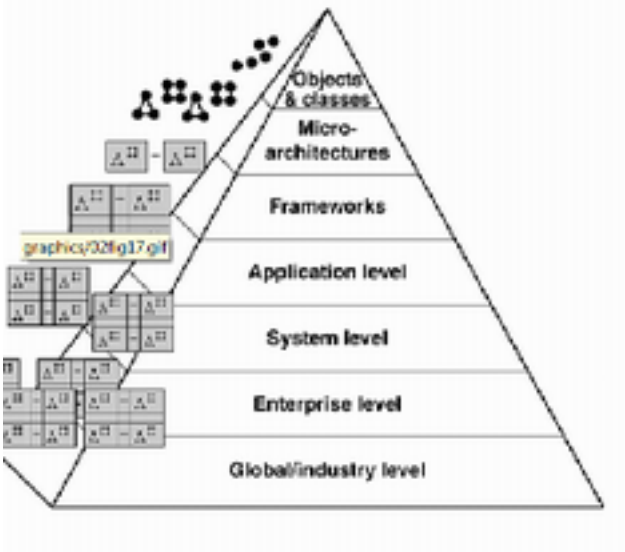
- Main Causes
    - Hasty Decisions: Schedule related stress
    - Impassivity, Sloth: Unwillingness to solve known problems

    - Ignorance and Narrow-Mindedness
    - Avarice: High Complexity, Insufficient Abstraction
    - Pride: Reinventing the Wheel
- Forces
    - Vertical Forces = Domain Specific forces, unique to particular software
    - Horizontal Forces: applicable to Multiple Domains
- Primal Forces: Management of
    - Functionality
    - Performance
    - Complexity
    - Change
    - Resources
    - Technology Transfer

Software Design-Level Model

Separate Concerns based on scale of software
Micro Component Levels: solve recurring software problems
Macro Component Levels: organization and development of application frameworks
Design Levels

- Object & Classes: Reusable code
- Micro Architectures: Patterns combining several classes
- Frameworks: Development of Design Patterns involving micro architectures
- Application Level: Application development meeting user requirements
- System Level: Communication and coordination across applications
- Enterprise Level: Communication and coordination in a single organization
- Global/Industry Level: Cross-organizational communication, information sharing



Types of Anti Patterns

- Software Development Anti Patterns: Problems encountered by Developers

- Software Architecture Anti Patterns: System Structure Problems
- Software Project Management Anti Patterns: Human Communication

Software Project Management Anti Patterns

Analysis Paralysis

- Try to Achieve perfection in early phase
- Over-analyzing without deciding and taking actions
- Long phases of project planning, Little value created
- assuming waterfall progression, insisting on finishing analysis before design

Solution

- Incremental development
- Learn Details of problem and solution in the process

Viewgraph Engineering

- Producing Viewgraphs for the management
- No development

Solution

- Prototyping: Mockups, Engineering Prototypes

Death by Planning

- Spending more time on planning than development, Focus lies on Costs
- Failure to deliver, Project plan not describing deliverables

Solution

- Define deliverables
- Update development plan continuously, Validation milestones

Smoke and Mirrors

- Management guarantees something the developers cannot hold
- Can not deliver, Loss of credibility

Solution

- Manage end user expectations

Project Mismanagement

- No Attention to Software Development (planning, quality assurance)
- Inconsistent view of development
- Insufficient test suites, infrequent code reviews

Solution

- Proper risk management

Fire Drill

- Management prevents development by giving conflicting directions

- Then suddenly makes unrealistic demands for sw delivery

Solutions

- Internal environment (=Developers) focus on long-term goals
- External environment (=Management) acts as an interface to outside world

E-Mail is Dangerous

- Inappropriate communication medium for sensitive topics
- Communication problems

Solutions

- Treat every mail as if it goes public
- Consider alternative ways of communication

Software Development Anti Patterns

The Blob

- Single class with high complexity, Unrelated elements together
- Lack of oop architecture

Solution

- Move behaviour away from blob
- Identify related elements

Lava Flow

- Undocumented complex functionality
- Blobs of code stuck in system
- Whole blocks of commented- out code
- R&D code placed into production
- Unclear project goals

Solution

- Proper architecture specification
- Quality-driven development

Golden Hammer (Time)

- Technology that “Solves” all problems
- Same tools are used for many different things
- Development team is commited to the technology they know

Solution

- Always choose your tools depending on use case

Spaghetti Code

- System with no structure, Minimal relationship between objects
- Methods are process oriented, have no parameters
- Difficult to maintain
- Inexperience with oo, Lack of code review and guidance

Solution

- Continuous refactoring

Copy & Paste Programming

- Copy and modify existing code
- Difficult to locate and fix bugs, Same bug recurs

Solution

- Refactoring into reusable components

Enterprise Architectures

What is Architecture

- Composition and Interaction of different components
- IEEE Standard 1471: the fundamental organization of a system, and their relationships
- There isn't just one architecture

Enterprise Applications

- Complex data structures
- Lots of data, Data often outlast the system
- Business logic
- Involve a lot of people
- Have to integrate with other enterprise applications

Examples of Enterprise Applications

- Payroll
- Patient Records
- Shipping
- Customer Service

Boyd's Law of Iteration

- Fast iteration analysis produces better results than in-depth analysis

Partitioned Iteration

- Design small and simple systems, build one and after that start on the next
- Decreases overall complexity (Divide and Conquer)
- Higher Success Rates!

Phases of Enterprise Application Development

1. Business architecture design
2. Technical architecture design

3. Implementation
4. Testing
5. Deployment

Traditional approach: do every Phase one time

Better use Iterative: do every Phase, then start at the first again

- much better success rate
- cost reductions, better control of costs
- faster delivery, better prediction of delivery dates

Rules of Success

- Start with low-hanging fruits
- Use Agile Processes -> faster iteration
- Reduce Complexity
- Rely on proven concepts
- Divide and Conquer
- Low Coupling, High Cohesion

Fundamentals of Software Architectures

- Strategical View
    - Architecture in the large, Focus on Systems
    - Non-Functional Requirements
- Tactical View
    - Architecture in the small, Design of Classes
    - Functional Requirements

Patterns

- Pattern Language: Collection of Design-Patterns
- Model-View-Controller
    - Separate Model, View and Controller Code
- Pipes and Filters
    - Pipes, Filters which process and transform information
    - Can be ordered at will
- Layered Architecture
    - communicates only with lower layers
    - no dependency with higher layers
    - communication through well defined interfaces
- There are a lot of more patterns:
    - Domain Logic Patterns
    - Offline Concurrency Patterns
    - Distribution Patterns

Domain Logic Patterns

- Conceptual model of a domain of interest, Describes the problem domain
- Represents vocabulary and key concepts, good Communication tool

Architectural Types and Domains

- Business architecture
- Information architecture
- Technical architecture

Case Study: Aircraft Operational Database

- Management of Resources, Litte Downtime
- High number of interfaces to external Systems
- Incremental replacement of legacy systems
- Initial Strategy failed because of wrong partitioning and big bang
- New Strategy succeeded: Complexity Reduction, Grouping, Decoupling of GUI

Themen die oft in Prüfungen vorkamen

- OSGi
- Enterprise Applications
- ESB
- Begriffe Patterns, Antipatterns
- Patterns
- CAP
- Sensoren von Mobiltelefonen und konkrete Use Cases

Fragen:

- Warum ist Automatisierung wichtig? Was sollte automatisiert werden?
- Nennen Sie die Charakteristika von REST Services! Zeigen Sie den Zugriff auf eine Resource exemplarisch!
- Was ist der Unterschied zwischen dem Cathedral und Bazaar Modell?
- Nennen Sie 3 Business Models von FOSS!
- Erklären Sie die Vorteile von Partitioned Iteration!
- Nennen Sie 4 Gründe, warum Antipatterns entstehen!
- Was versteht man unter OSGi und woraus setzt es sich zusammen? Nennen Sie prominente Implementierungen von OSGi!
- Nennen Sie 4 Sensoren in einem Mobiltelefon und geben Sie zu jedem einen konkreten Use Case an!

- Was ist ein Pattern?
- Was sind Enterprise Applications?
    - + 4 Charateristika
- Was unterscheidet Desktop- und Mobile-Applikationen bzgl. Development?
- Unterschiede zwischen OSGi und Jigsaw erklären + in welchem Szenario würde man OSGi einsetzen?
- Was sind die Nachteile von Design Patterns?
- ??? Drei Varianten von Integration Patterns erklären
- Was ist der Enterprise Service Bus? + 2 bekannte Implementierungen
- Vorbedingungen fürs Refactoring
-