Advanced Software Engineering

Example Project:

Vienna International Airport
AODB Core System

DI Dr. techn. Mario Bernhart

About myself

- Researcher at Vienna University of Technology
- In 2012 researcher at MIT Aero/Astro
- Eclipse project lead and committer (Mylyn Reviews)
- 6 years in software engineering for air traffic
  - 2000-2005 Electronic Flight Strips (EFS) at Frequentis AG
- 3 years in software engineering for airport operations
  - 2010-2012 AODB for Vienna International Airport

- Contact: mario.bernhart@inso.tuwien.ac.at

Advanced Software Engineering (next 5 lectures)

- Example Project: Vienna International Airport AODB Core System
- Release your stuff 3 times a day
  - Dependency Management
  - Build Management and -Automation
  - Continuous Integration, Continuous Delivery
- Five challenges you solve for every project
  - Error Management
  - Transaction Management
  - Logging
  - Auditing
  - Declarative Authentication and Authorization
- Build for ten years and more
  - Layered Software Design / API Design
  - Modularization / Service Design
  - Decoupling / Event Driven Design
  - Interfacing / Integration
- From prototype to product (make it work 24/7)
  - Clustering
  - Performance
  - Monitoring
  - Automating Operational Tasks

What is *advanced* in Software Engineering?

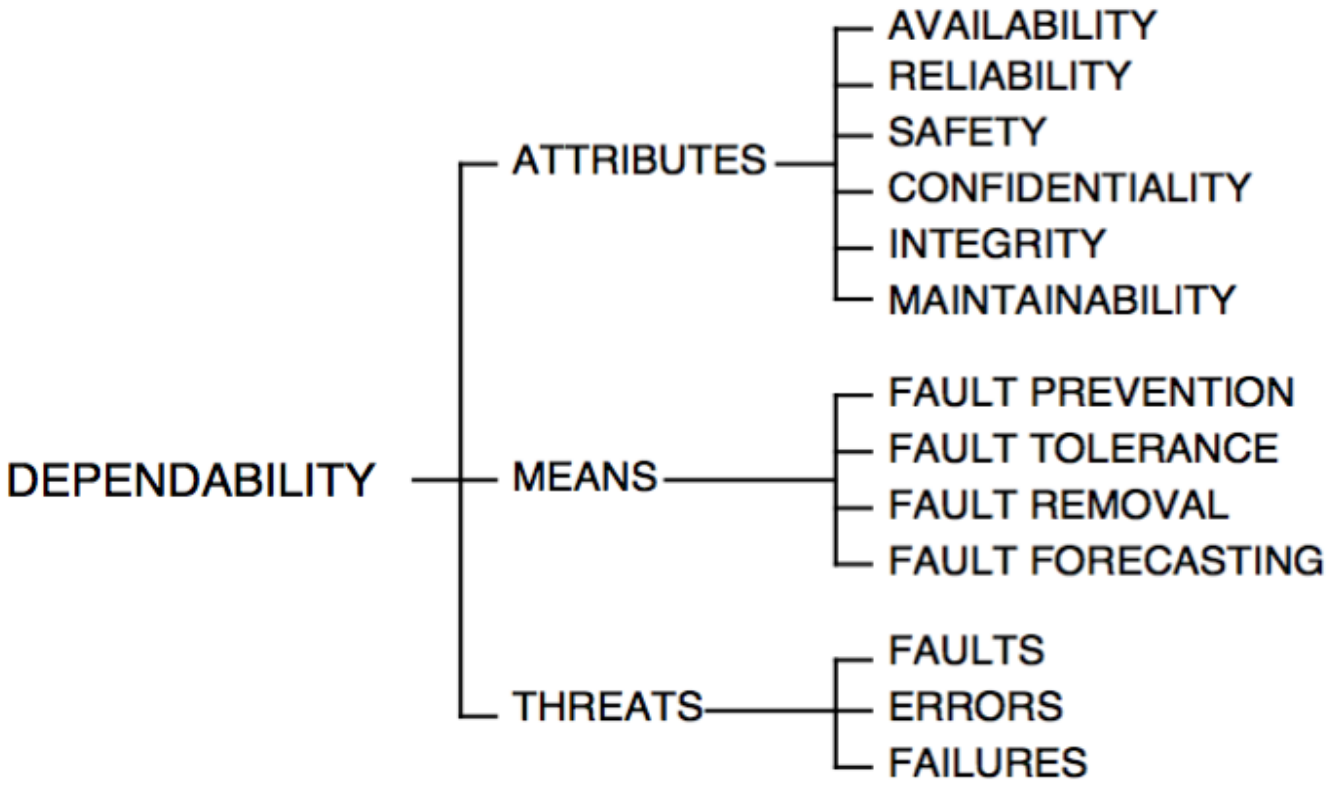
- Dependable Software
- Large, Complex, Integrated Software
- Extended Software Lifecycle

Dependable Software Basic Concepts [1]

- Definition from [1] (Fundamental Concepts of Dependability)
- Dependability is an integrative concept that encompasses the following attributes:
  - availability: readiness for correct service;
  - reliability: continuity of correct service;
  - safety: absence of catastrophic consequences on the user(s) and the environment;
  - confidentiality: absence of unauthorized disclosure of information;
  - integrity: absence of improper system state alterations;
  - maintainability; ability to undergo repairs and modifications.

Dependable Software Basic Concepts [1]

- Dependability tree from [1]



Dependable Software Basic Concepts [1]

- Fault prevention
  - Quality control
  - Software design i.e.
    - structured programming
    - information hiding
    - modularization

- Fault tolerance
  - Error detection and subsequent system recovery
  - Error handling: roll-back (checkpoint) vs. roll-forward
  - Redundancy: fault masking, voting algorithms, ...
  - Fault isolation

Dependable Software Basic Concepts [1]

- Fault removal
  - Verification (static, dynamic), diagnosis, correction
  - Fault injection (e.g. to test the error handling)
  - Corrective and preventive maintenance

- Fault forecasting
  - Qualitative
    - identify, classify, rank
  - Quantitative
    - probability model (stochastic)

Dependable Software Basic Concepts [1]

- ERROR: Discrepancy between a computed, observed or measured value or condition, and the true, specified, or theoretically correct value or condition.
  - Example: null value when not valid
- FAULT: Abnormal condition that can cause an element or an item to fail.
  - Example: Uncaught NullPointerException (NPE)
- FAILURE: Termination of the ability of an element, to perform a function as required.
  - Example: Malfunction, nonfunction or crash of a service (i.e. due to unhandled NPE)

Dependable Software Safety culture; exemplary analysis from [2] N. Leveson

- Ariane 501
  - Wrong reuse (Ariane 4 software for Ariane 5) → exploded
- Mars Climate Orbiter (MCO)
  - Metric vs. English units → navigation failure, lost
- Mars Polar Lander
  - False sensor signal → landing aborted, destroyed
- Titan / Milstar satellite
  - False constant → incorrect orbit

Dependable Software Safety culture; Flaws in safety culture [2]

- Management
  - Diffusion of Responsibility and Authority
  - Limited Communication Channels and Poor Information Flow
- Technical
  - Inadequate System and Software Engineering
    - Poor or Missing Specifications
    - Unnecessary Complexity and Software Functionality
    - Software Reuse or Changes without Appropriate Safety Analysis
    - Violation of Basic Safety Engineering Practices
  - Inadequate Review Activities
  - Ineffective System Safety Engineering
  - Flaws in the Test and Simulation Environments
  - Inadequate Human Factors Design for Software

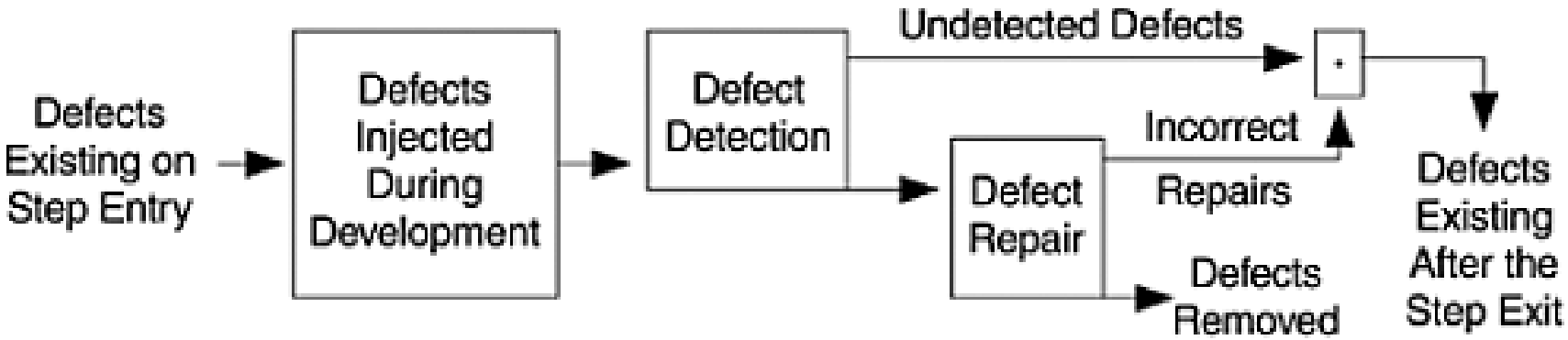
Software Aging [3]

- Reasons for Software Aging from [3] D. N. Parnas
  - Lack of movement: Failure to modify the product to meet changing needs
  - Ignorant surgery: Result of the changes that are made

- Problems during lifecycle
  - Inability to keep up (growth)
  - Reduced performance (poor design)
  - Decreasing reliability (error injection)

- Preventive measures
  - Design and plan for change
  - Documentation and Reviews
  - Restructuring including partial replacement (amputation)
  - Plan for retirement and replacement

Error Injection



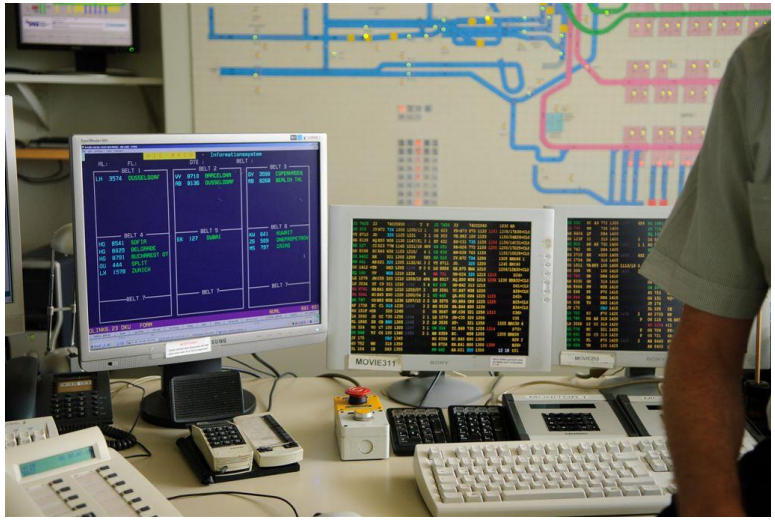
Vienna Airport facts (from 2010)

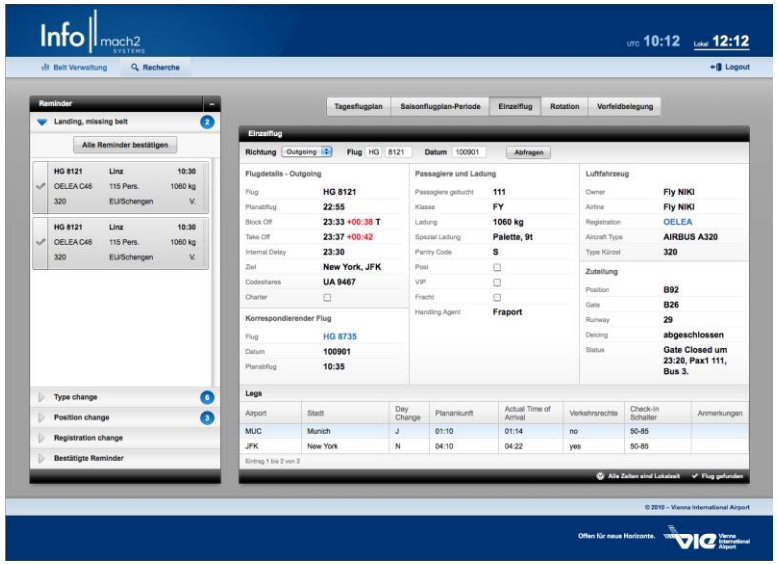
- 19,7 mio. Passengers
- 246.000 Movements
- ~230 Companies
- 70 Airlines
- 172 Destinations
- 4.266 Employees (FWAG)
- ~19.000 Employees at Vienna Airport
- 16,8 mio. Baggage Pieces
- 116 Check-In Desks
- 96 Parking Positions
- 20 Pier Positions

Vienna International Airport

Mach2info project key facts

- Core system (AODB – Airport Operational Database) for all flight operations at VIE (20 Mio. passengers / year)
- Replacement of the legacy system MACH
  - 40+ years-old BULL-GSOC8 host (about 250 KLOC Cobol source)
  - No documentation at all
  - 126 Transactions per second
- Criticality
  - 1h downtime → Delays
  - 4h downtime → Severe operational limitations
- Migration of the legacy system without significant downtime
- Incremental strategy with a strict 1:1 re-engineering policy





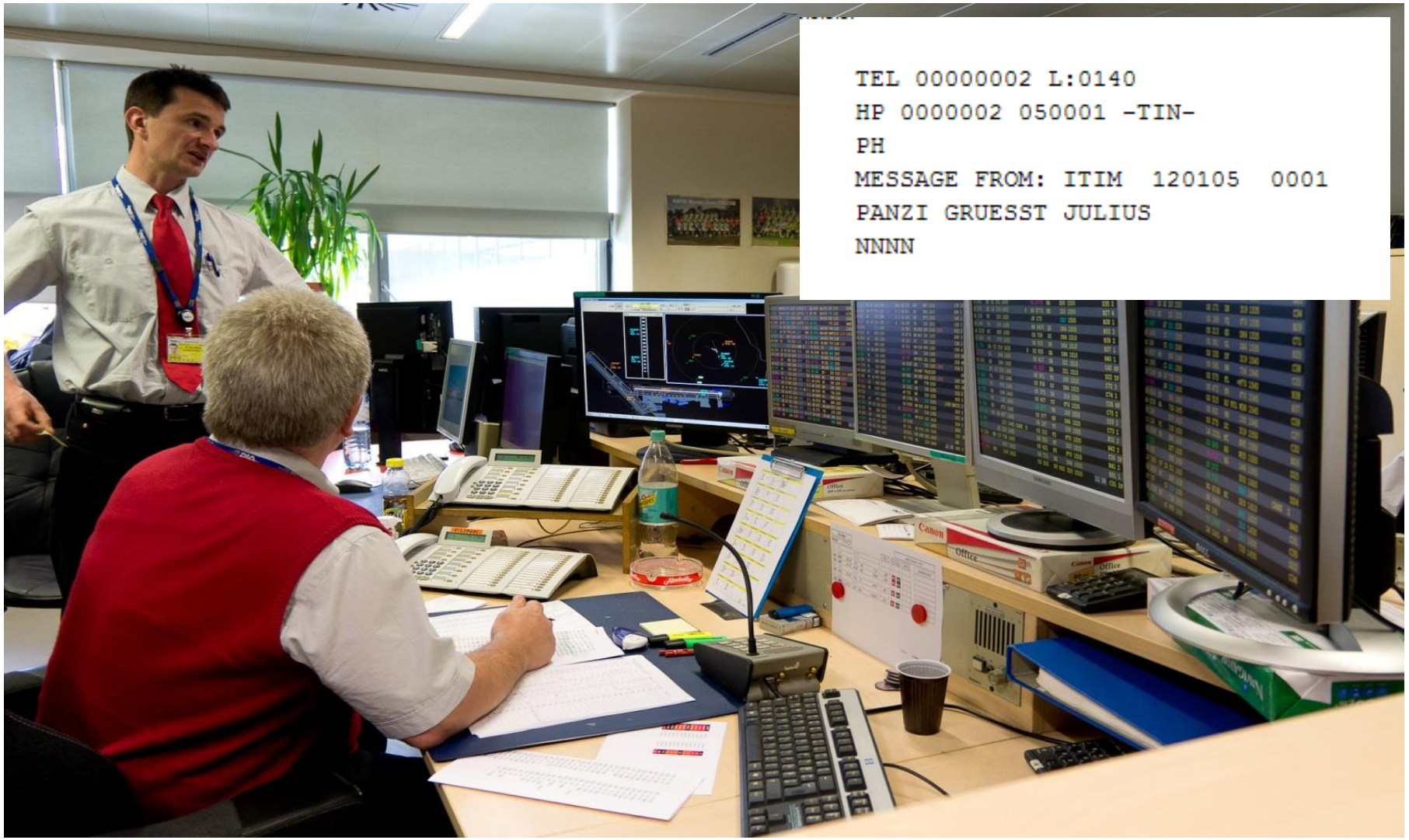
Baggage office

Movement Control

Operations center

Ground handling

Operations


TEL 00000002 L:0140
HP 0000002 050001 -TIN-
PH
MESSAGE FROM: ITIM 120105 0001
PANZI GRUESST JULIUS
NNNN

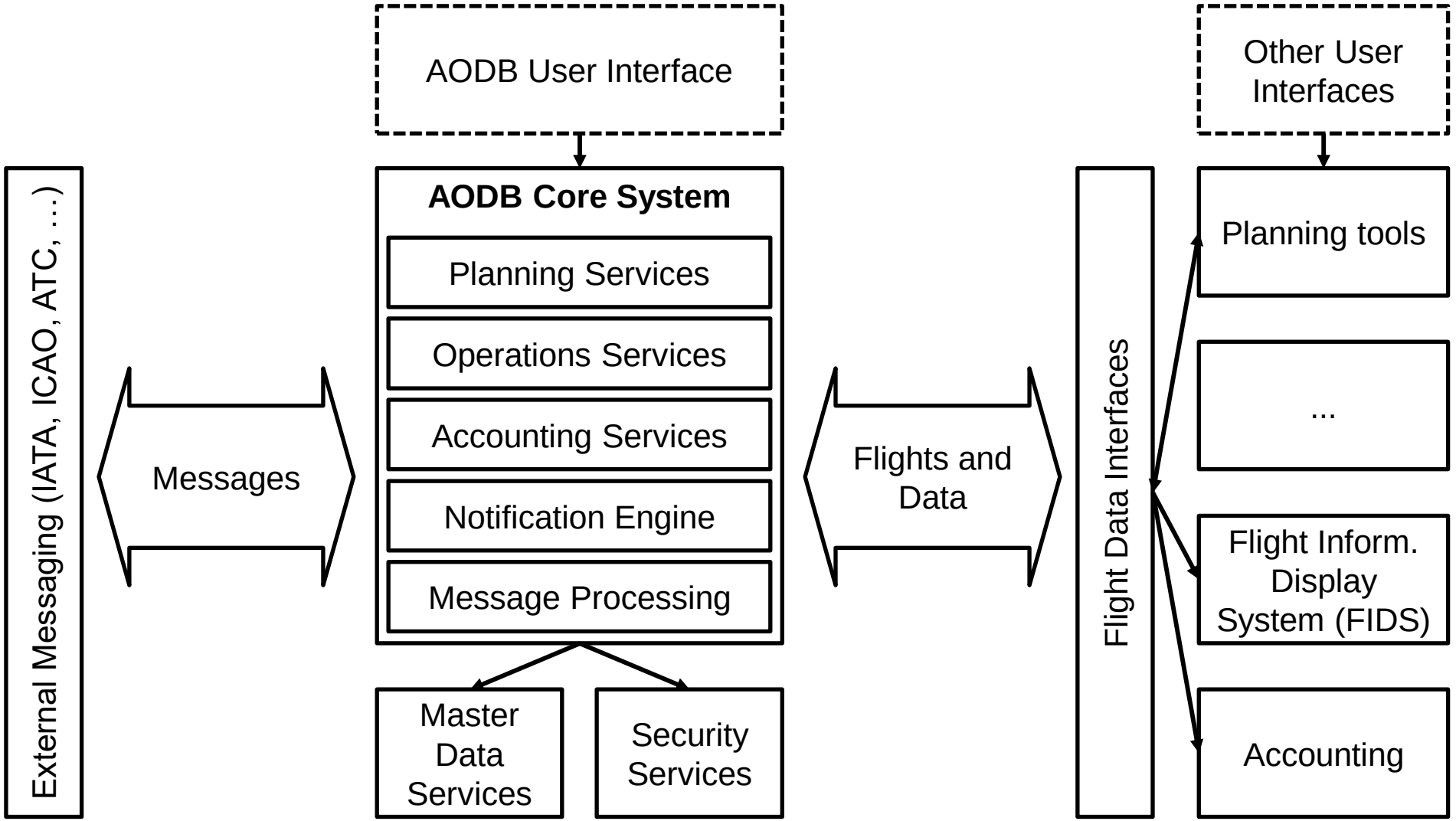
Project in numbers

- 31 team members
- 106 Features (e.g. core flight process, turnaround management, messaging, gate and stand planning, notifications, deicing...)
- About 1000 users and 16 individual user groups on
  - Standard PCs
  - IATA Cute Terminals (no mouse)
  - Ruggedized touchscreen laptops
- 5 releases with incremental operations of mach2info and incremental shutdown of legacy system
- 2 years duration; now successfully in full operations

Mach2info features

- Inbound / Outbound flight processing
- Turnaround management
- Gate and stand allocation
- Deicing management
- Notification system
  - Application integrated (with drag and drop support)
  - Email
  - System2system
- Flight planning services (message-based and manual)
- IATA message processing (receiving and sending)
- Accounting data export
- Interfaces for e.g. public displays (FIDS), load planning, billing, ground radar, resource planning tools …

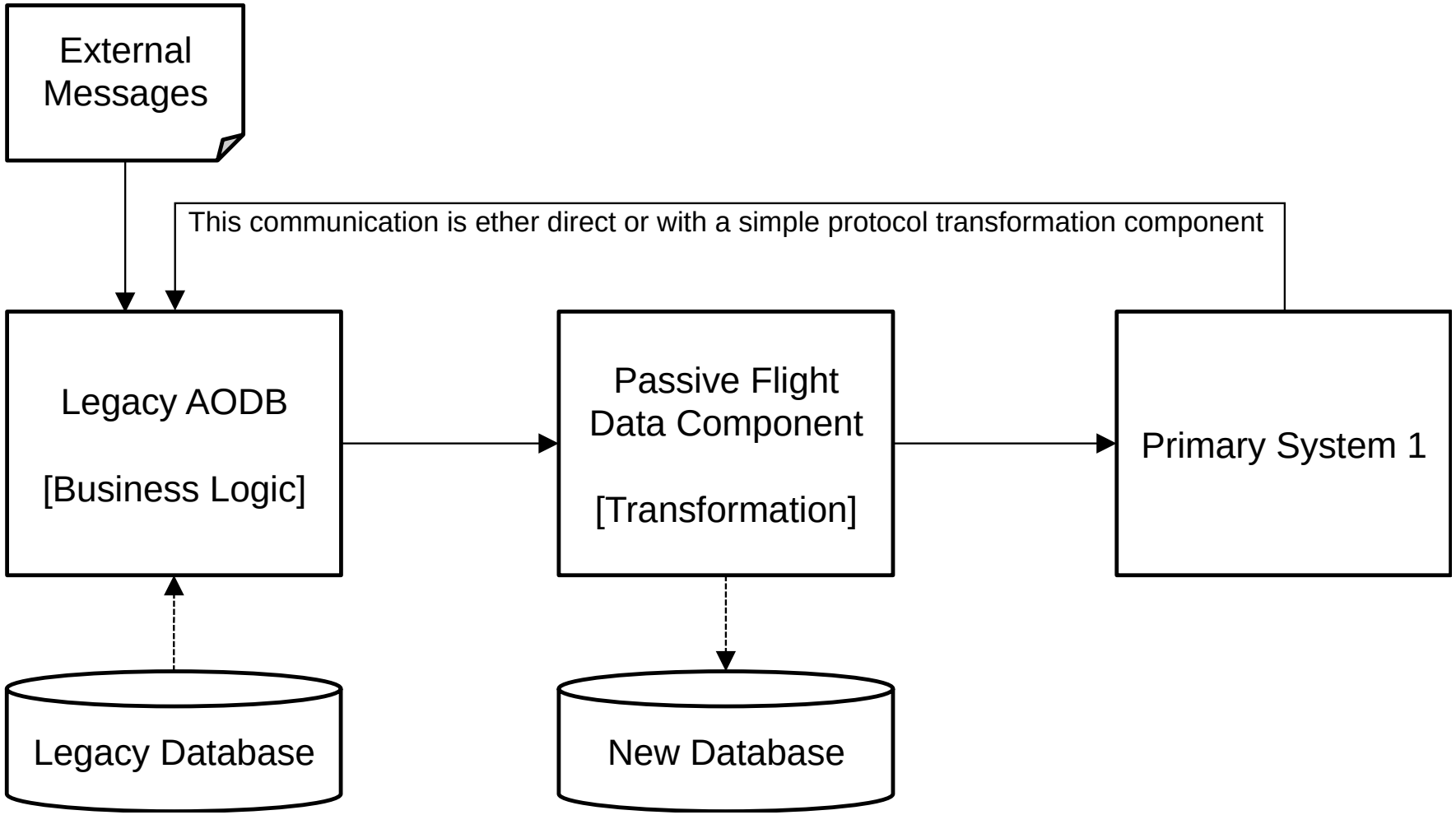
AODB example archticture



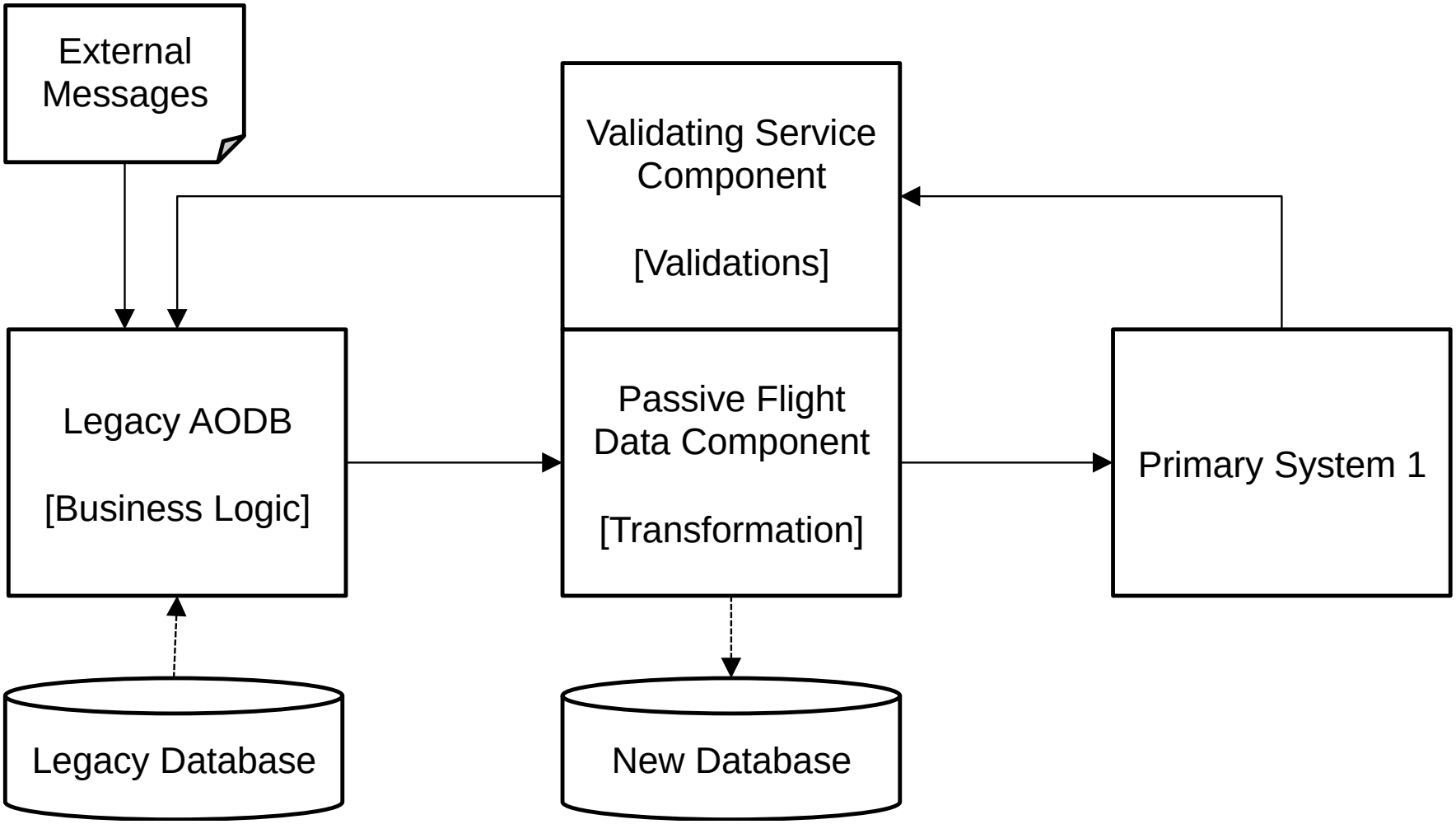
Technical strategy

- **1:1 migration** (feature-wise, not technical)
- Minimize the changes in the legacy system (high risk!)
- Incremental transfer of user groups to the new system
  - Try to evenly distribute user size through a number releases
- Technical „little big bangs“
  - Migrate smallest possible size, but coherent parts
- Parallel operations of both, the legacy and the new system
  - Parallel input: Interfaces and messages
  - Serial input: User input

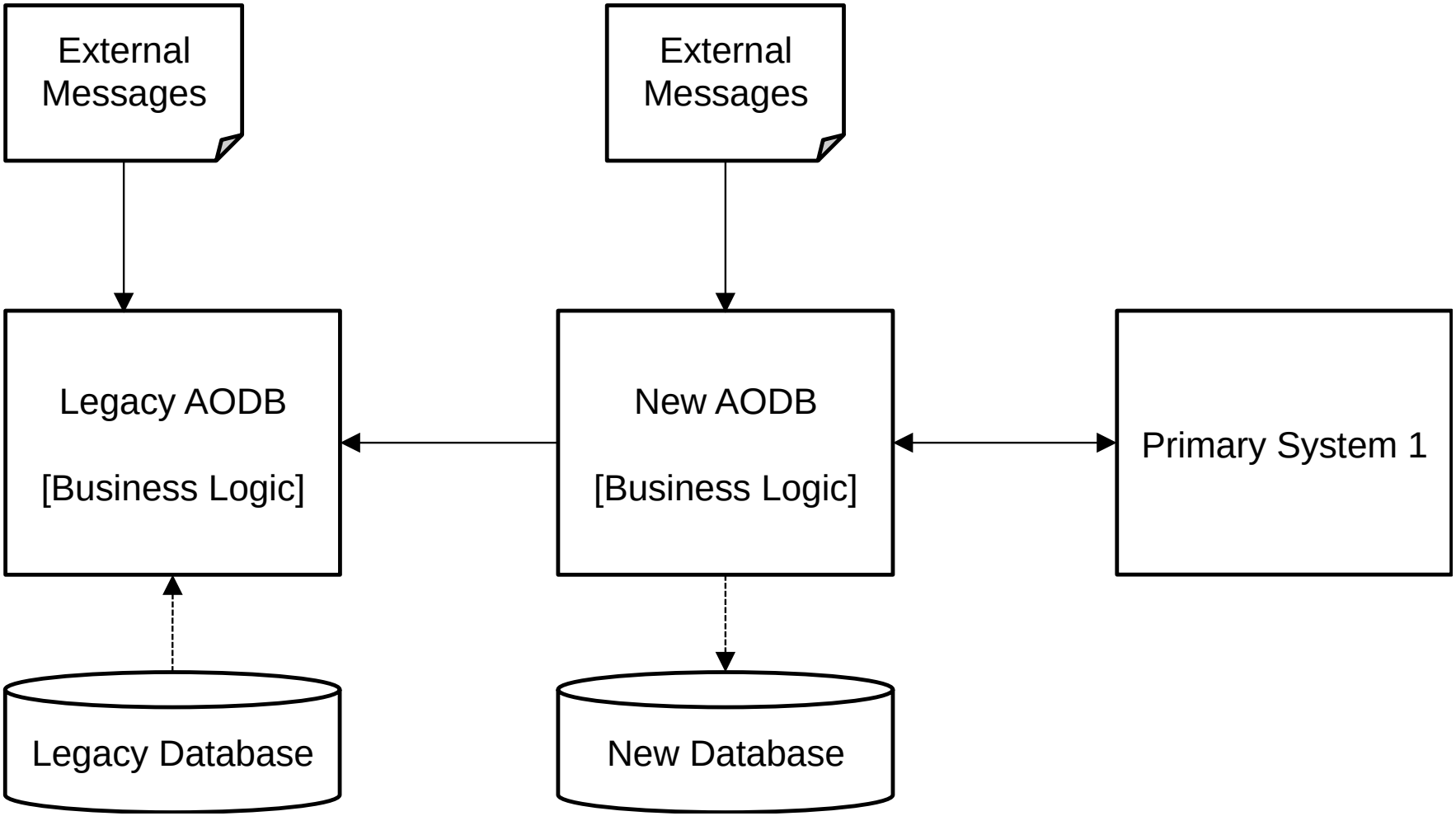
Step 1 of the incremental migration strategy. The main element is the Flight Data Component that listens on the legacy interface for flight data updates and transforms that to a new model before forwarding it to the primary systems.



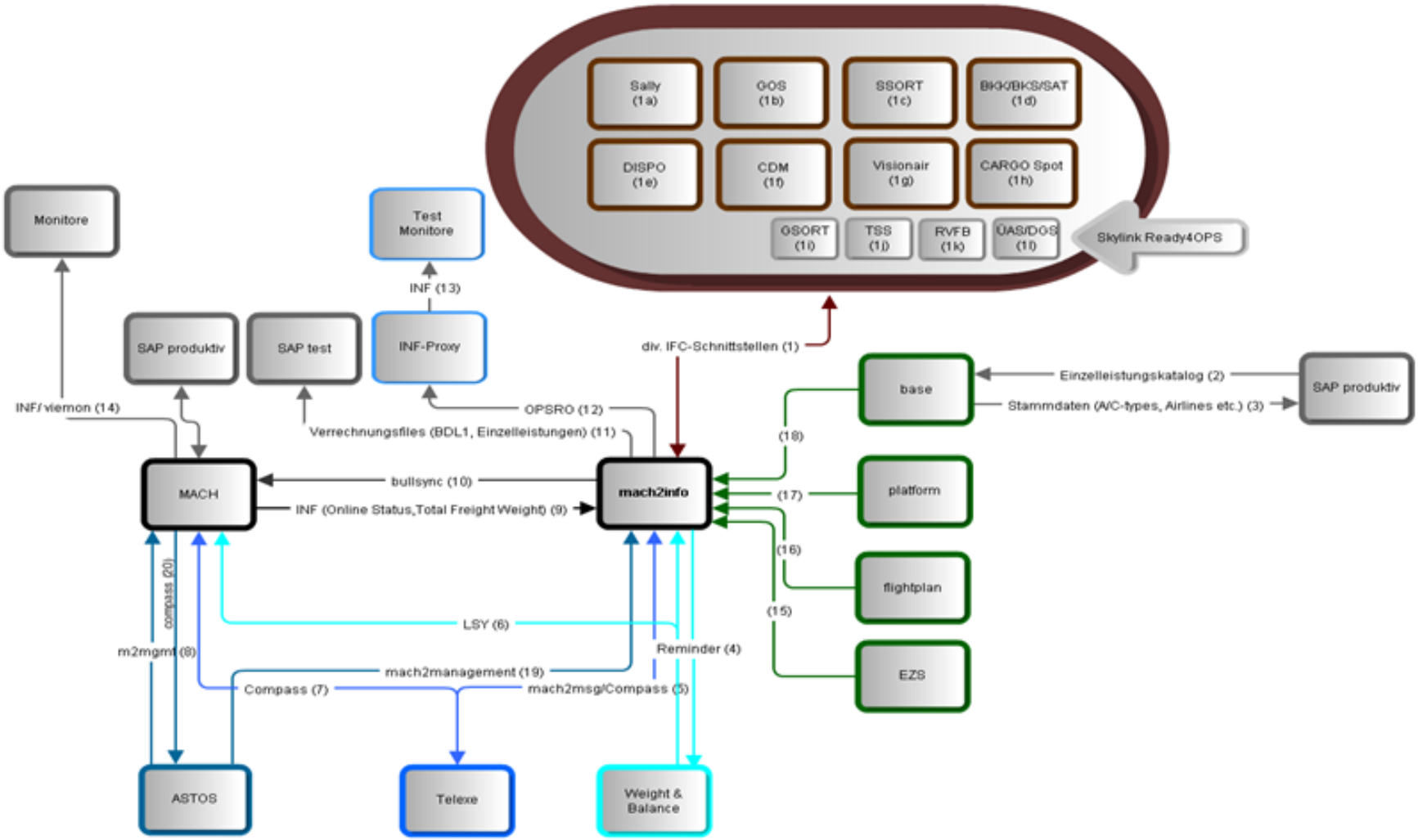
Step 2 of the incremental migration strategy. The main element is the Service Component that provides service interfaces with all relevant validations, but not the business logic.



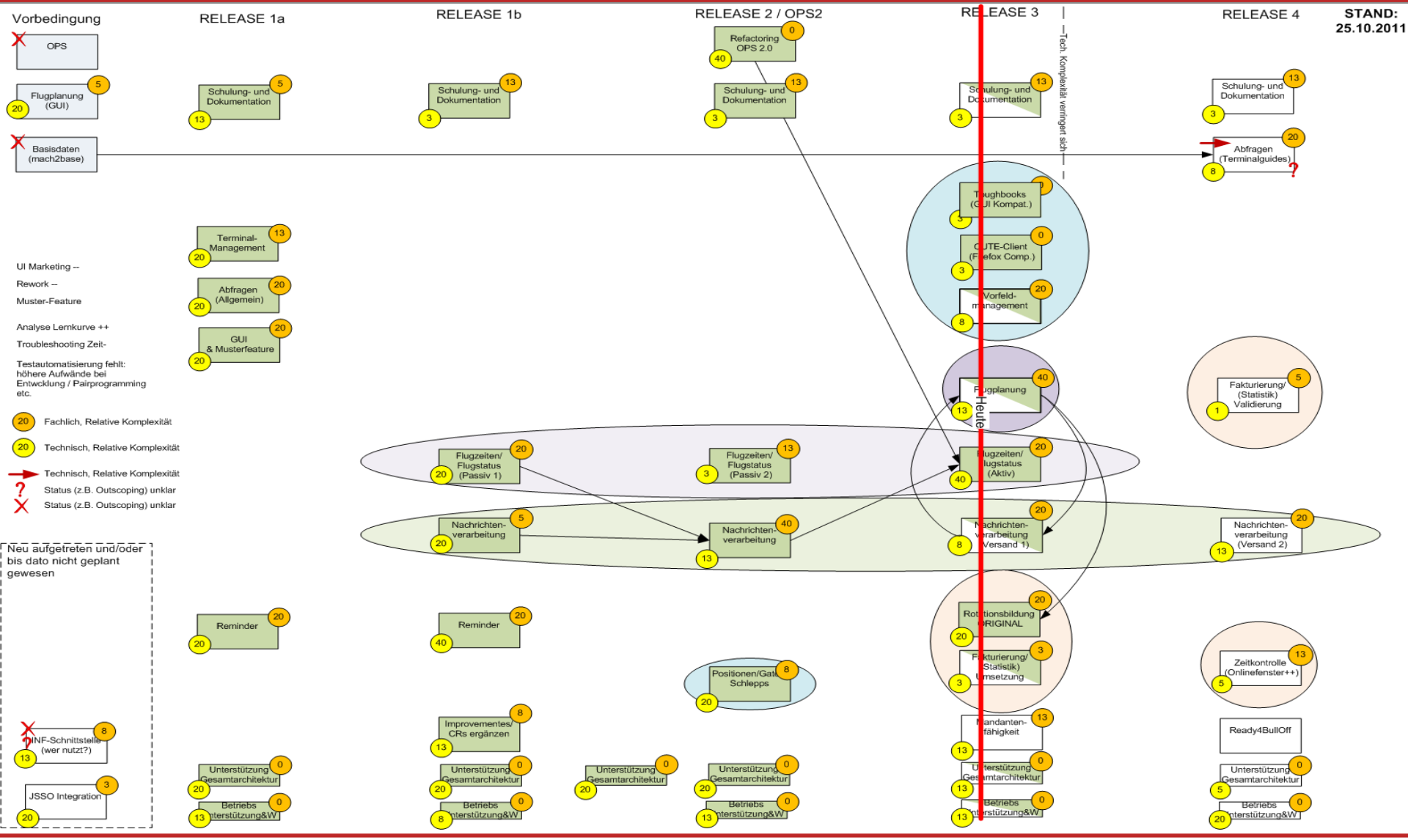
Step 3 of the incremental migration strategy. The two components of Step 2 are now merged together and provide a fully functional AODB. The legacy system is updated asynchronously to keep in sync.



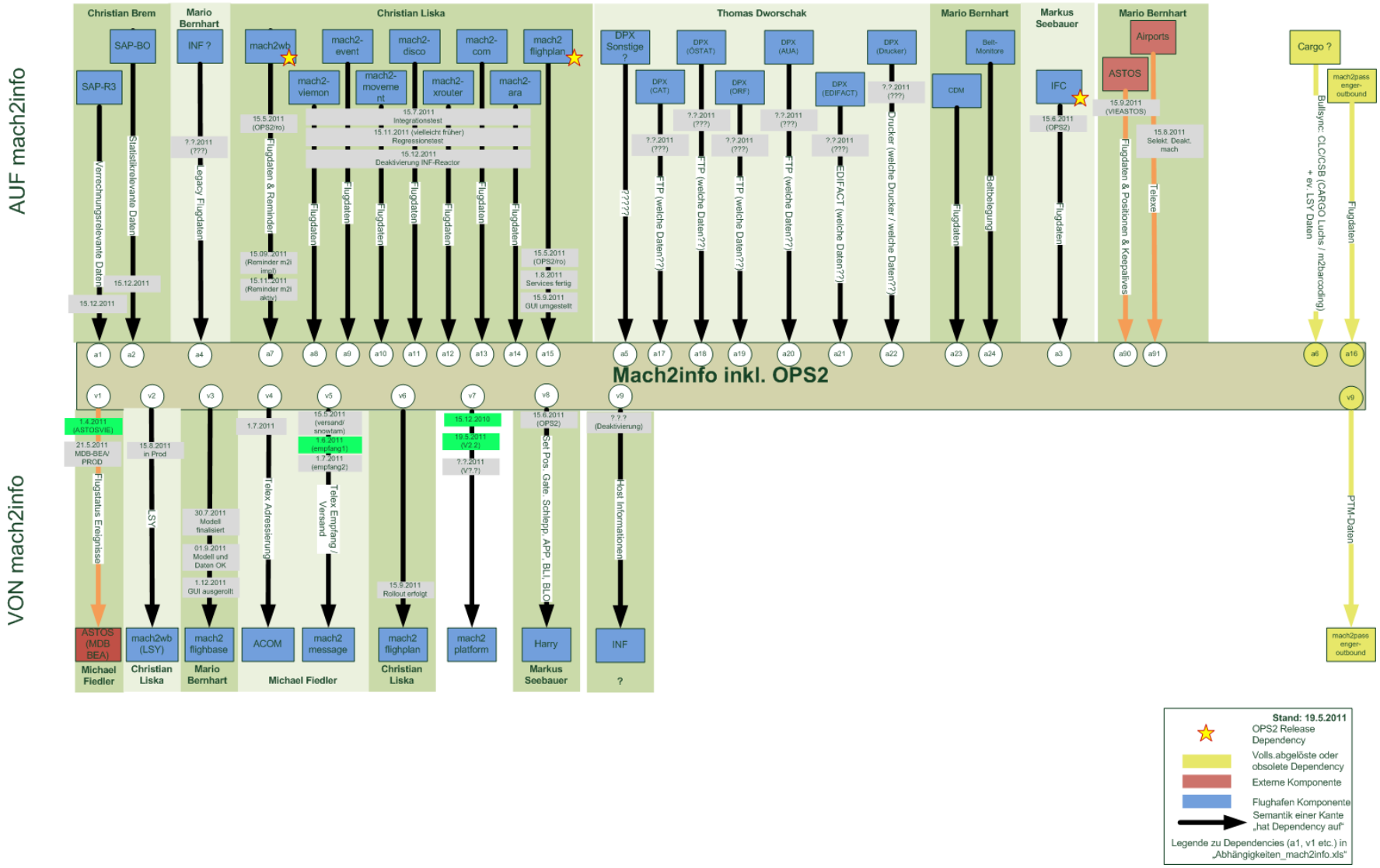
Integration architecture (at release 3)



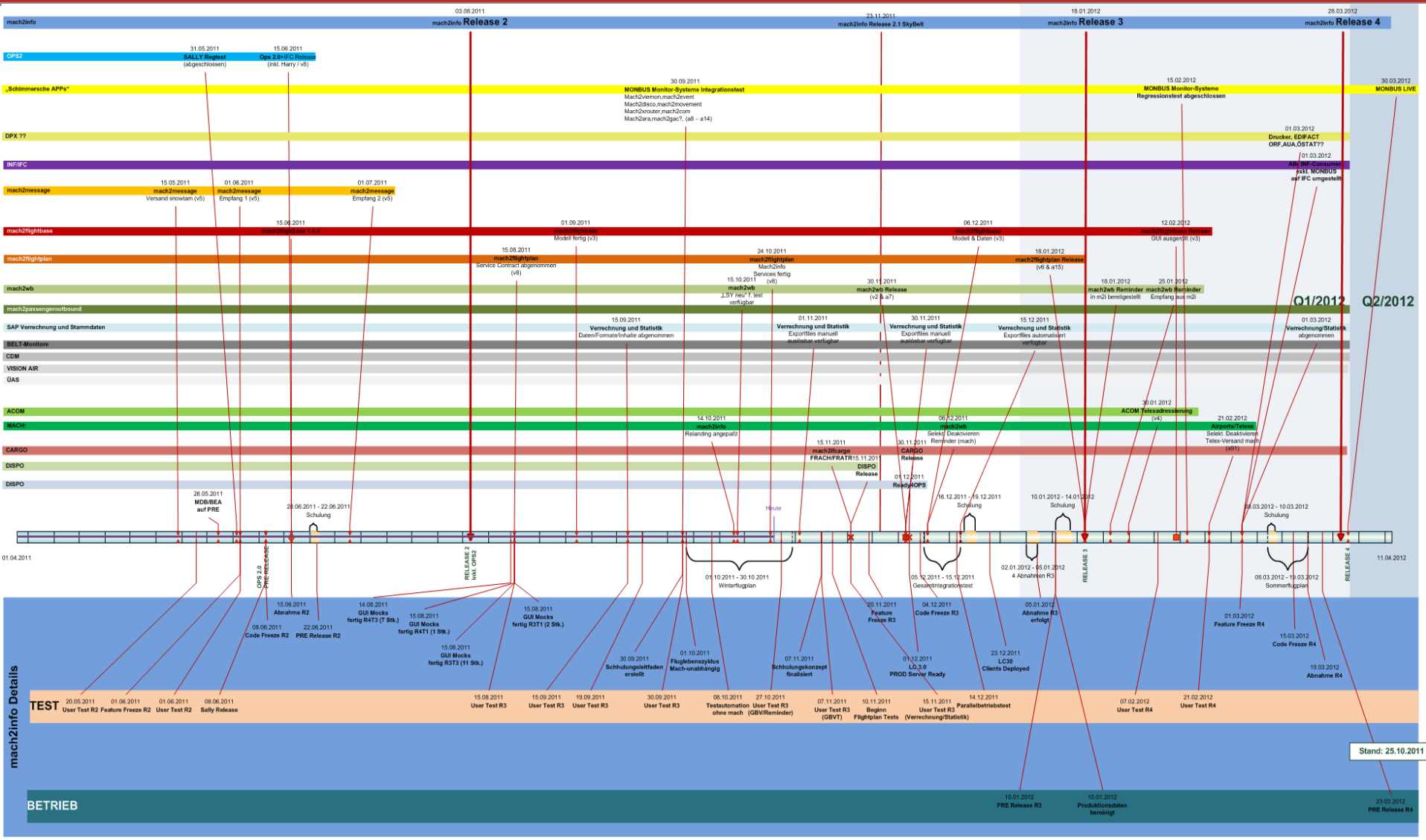
Project planning for 5 mach2info releases



„Anatomy“ of a core system (dependency graph)



Integrating a core system



Requirements Engineering

- Integration and intensive cooperation with 3 core business units
- Coordination with 3 business units representatives
- Identification of user groups
  - About 1000 users and 16 user profiles
- Tailoring the user interface engineering process to meet the demands of the respective business units
- Reading legacy code to extract requirements in human readable language
  - Separation of functional and technical aspects of the legacy source

Cobol code analysis

- 114 Transaction Programs (TPRs) with 124k LOC
- (665k LOC total in MACH)
- 819 Functions/procedures (called by TPRs) with116k LOC
- 109 DB-Configurations, with 3.5k LOC

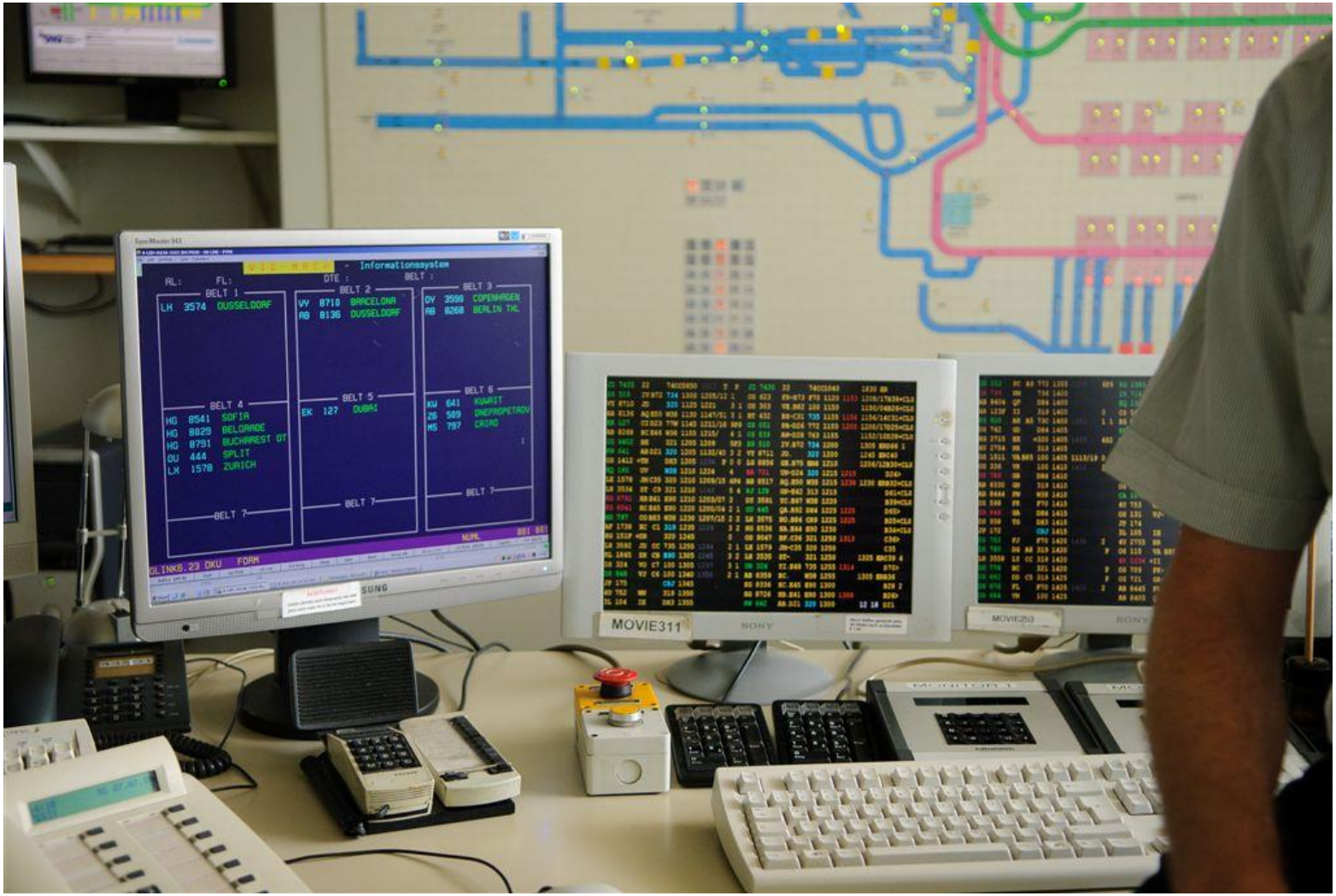
```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
      DECIMAL-POINT IS COMMA.
DATA DIVISION.
WORKING-STORAGE SECTION.

01 DATUM.
     05 JAHR    PIC 9(4).
     05 MONAT   PIC 9(2).
     05 TAG     PIC 9(2).
     05 REST    PIC X(13).

01 GESEHEN PIC X.

PROCEDURE DIVISION.
       MOVE FUNCTION CURRENT-DATE TO DATUM.
       DISPLAY SPACES UPON CRT.
       DISPLAY TAG      AT 0101.
       DISPLAY MONAT    AT 0103.
       DISPLAY JAHR     AT 0105.
       ACCEPT GESEHEN.
STOP RUN.
```

Usability Engineering

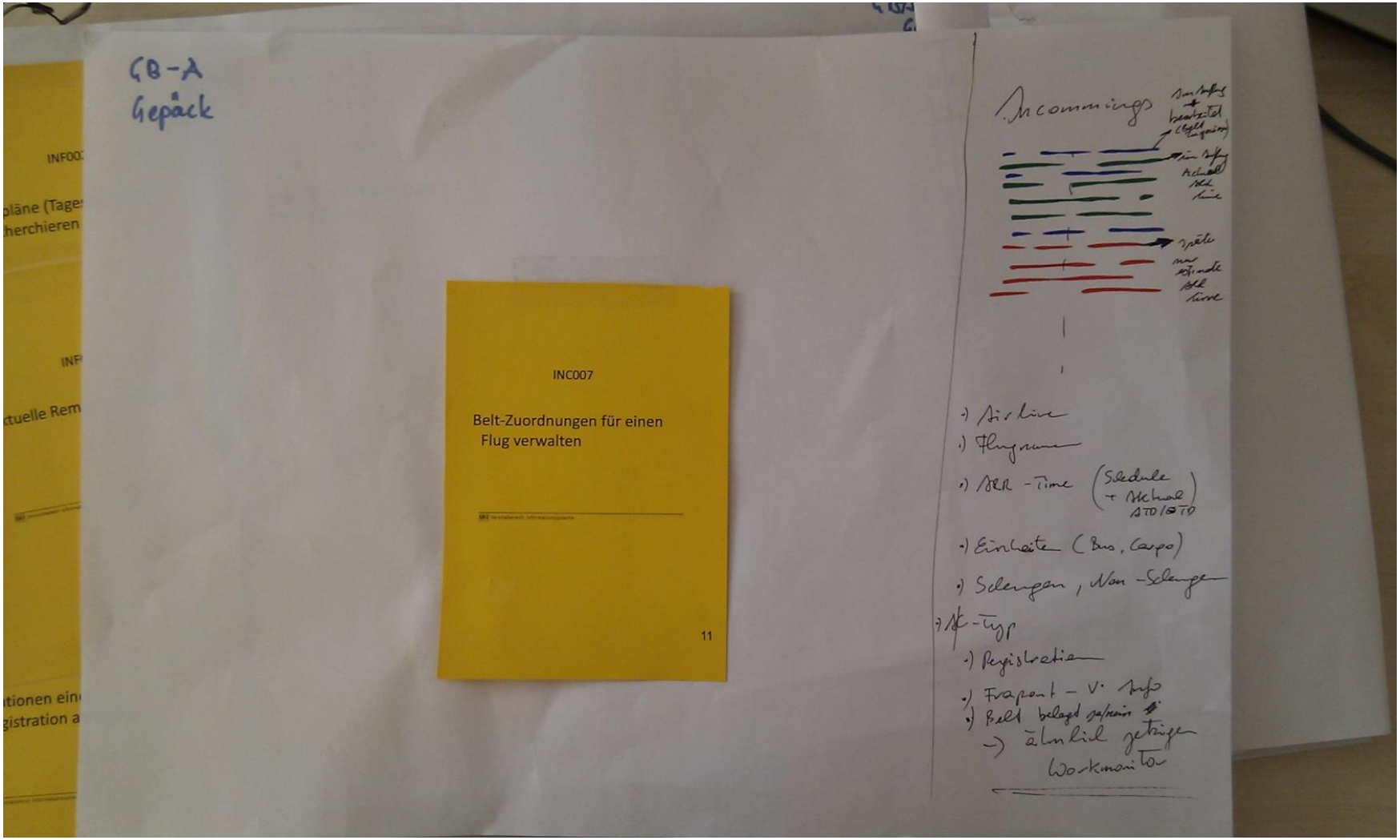
- Contextual enquiry of the working environment
  - „How is the operative environment for each user set up?“
- Individual design of user interface for each user group
  - „1:1 functional replacement, but optimized user interfaces“
  - design for the technical user environment (PC, mobile etc.)
- Analysis of usage statistics of legacy system
  - „How frequently is a function used by a user and what is the workflow?“
- Mockups of user interfaces before implementation

Baggage management example

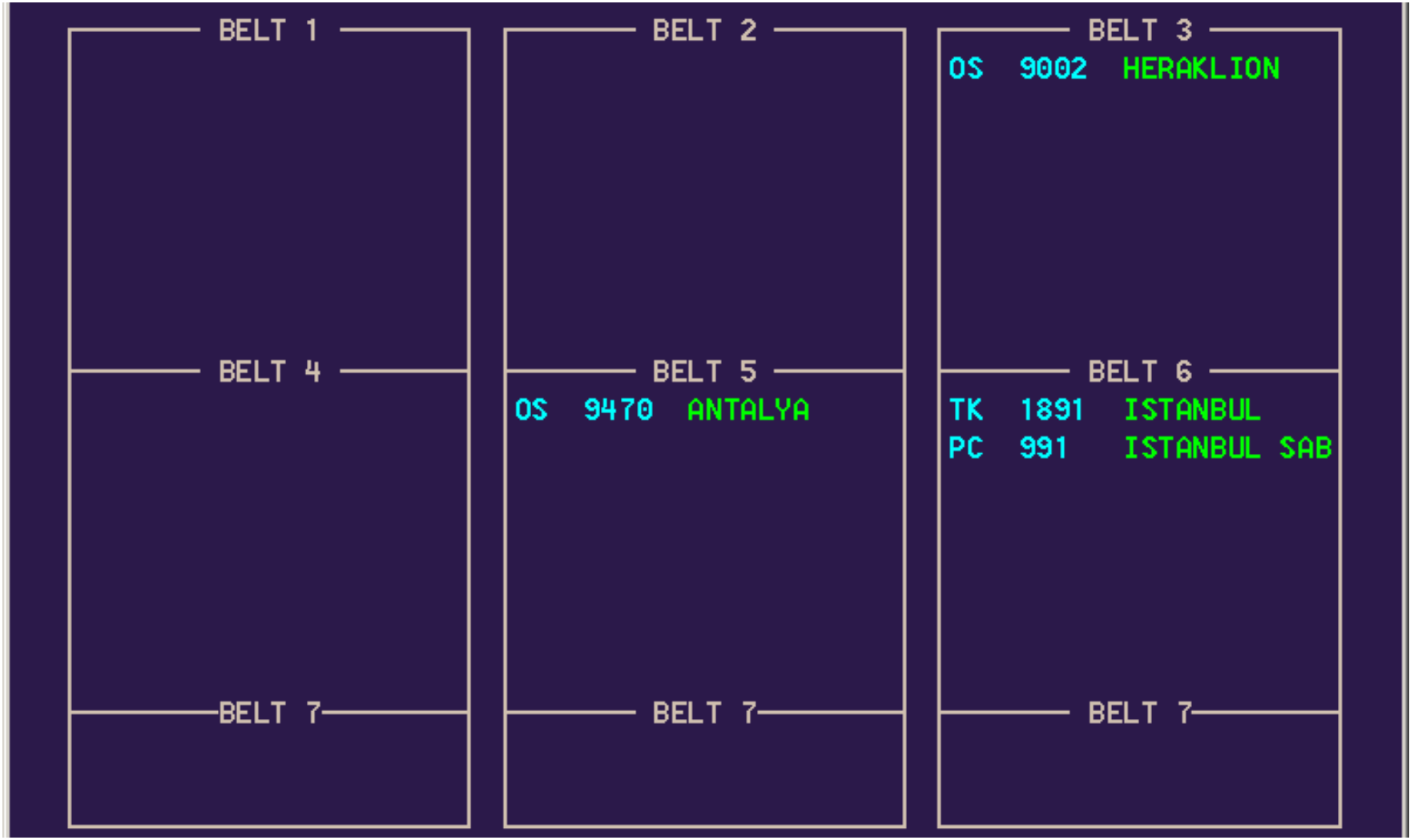
Contextual inquiry of work environment



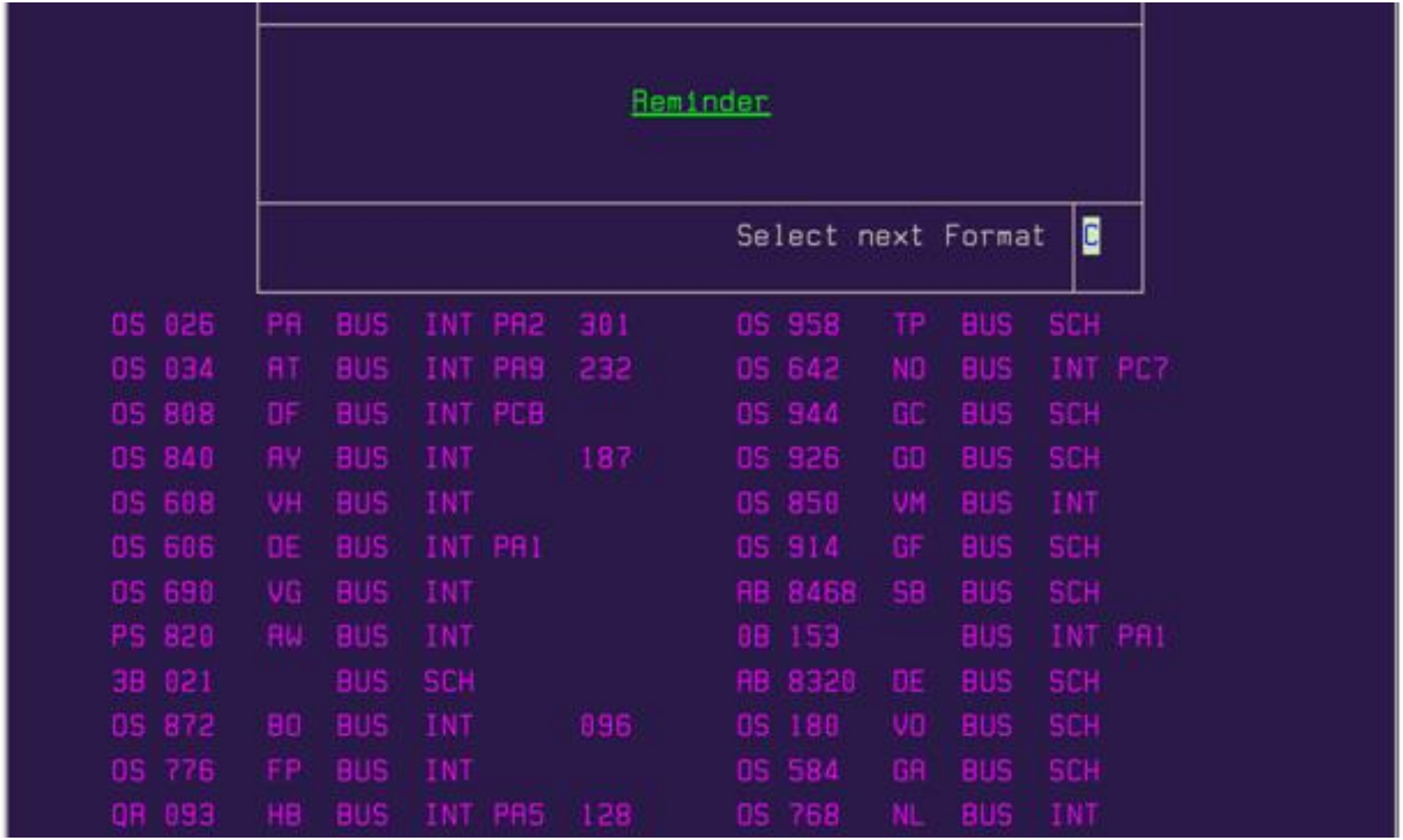
Early design draft for baggage system



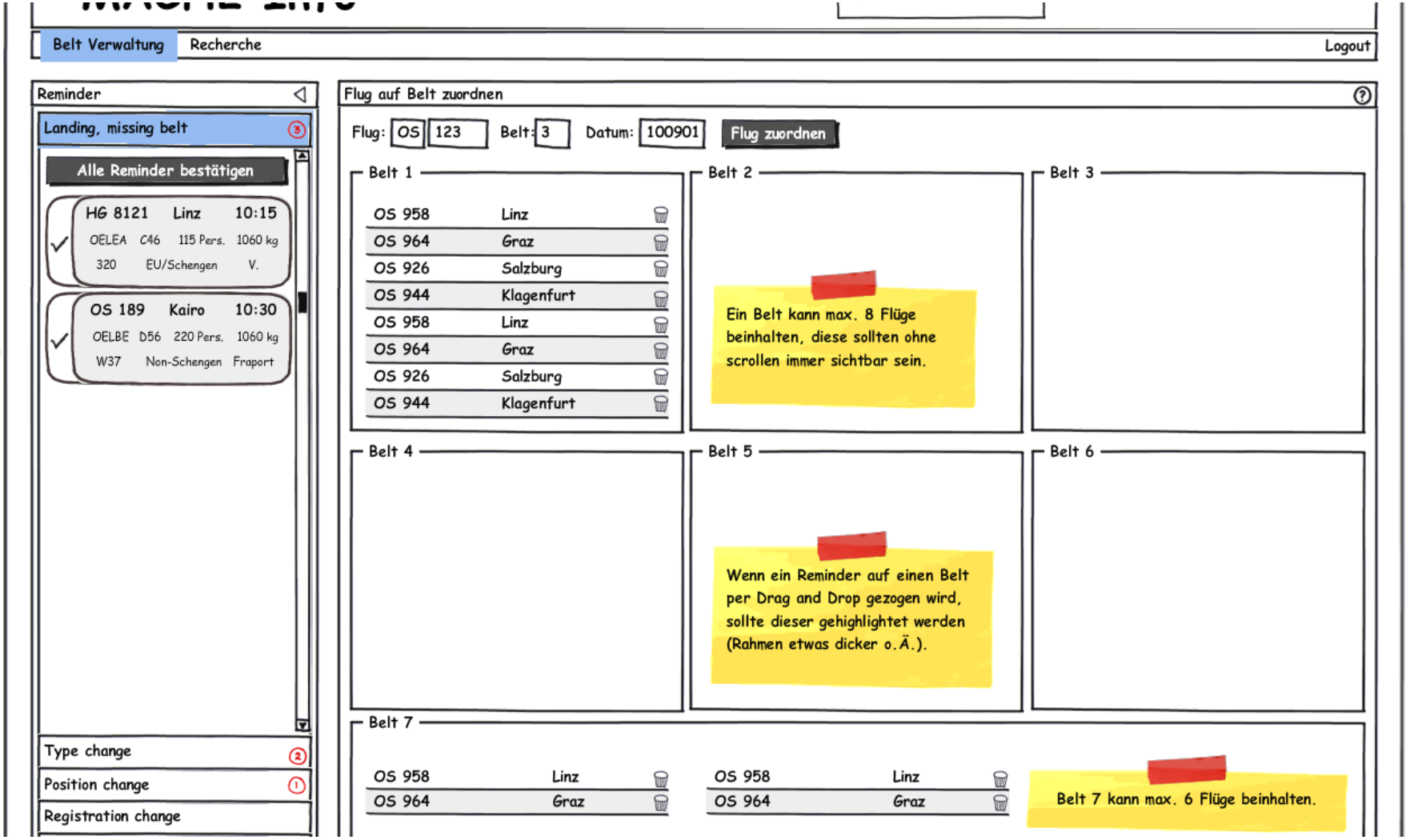
Legacy system for baggage system



Legacy system for notifcation system



Mock-up for baggage (right) and notfications (left)



Reminder

Landing, missing belt 2

Alle Reminder bestätigen

HG 8121	**Linz**	**10:30**
OELEA C46	115 Pers.	1060 kg
320	EU/Schengen	V.

EZY 5357	**London**	**10:40**
GEZIC D24	146 Pers.	0 kg
319	EU/Schengen	Fraport

Type change 6

Position change 3

Registration change

Bestätigte Reminder

Flug auf Belt zuordnen

Flug OS 513 Belt 3 Datum 100901 Flug zuordnen

Belt 1		Belt 2		Belt 3	
OS 985	Paris	**UA 2912**	Linz		
		OS 271	Boston		
		UA 2912	New York City		
		HG 2714	Atlanta		

Belt 4		Belt 5		Belt 6	
HG 2714	Atlanta	**OS 271**	Atlanta	**UA 2912**	New York City
UA 2912	Atlanta	**OS 985**	Boston	**UA 2912**	Linz
UA 5447	Boston	**HG 2714**	Boston	**HG 2714**	New York City

Belt 7			
OS 271	Linz	**OS 271**	Boston

Tracking software development

Most important: motivation

References

1. A. Avizienis, J.-C. Laprie and B. Randell: *Fundamental Concepts of Dependability*. Research Report No 1145, LAAS-CNRS, April 2001
2. *The Role of Software in Spacecraft Accidents* by Nancy Leveson. AIAA Journal of Spacecraft and Rockets, Vol. 41, No. 4, July 2004
3. David Lorge Parnas: *Software Aging*. In: International Conference on Software Engineering. IEEE Computer Society Press, Sorrento, Italy 1994
4. Bernhart, M.; Mauczka, A.; Fiedler, M.; Strobl, S.; Grechenig, T., *Incremental reengineering and migration of a 40 year old airport operations system*, 28th IEEE International Conference on Software Maintenance (ICSM), pp.503,510, Sept. 2012