Experience-based Refactoring for Goal-oriented Software Quality Improvement

International Workshop on Software Quality
(SOQUA 2004)
Erfurt, Germany, September 30, 2004

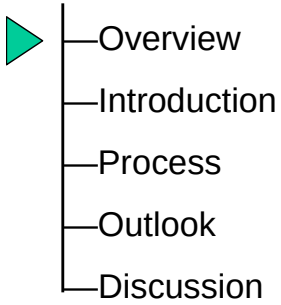
Fraunhofer Institut
Experimentelles
Software Engineering

Jörg Rech, Eric Ras,
Andreas Jedlitschka

Sauerwiesen 6
D-67661 Kaiserslautern
Germany

Overview

Outline



- Setting the scene
- Introduction
- Overview
- Details
- Outlook & Summary

Extreme Programming XP

Outline

- Overview
- Introduction
- Process
- Outlook
- Discussion

- XP as one example for agile SW development
- Essential values to be successful
  - Communication
  - Simplicity
  - Feedback
  - courage

- The 4 XP activities
  - CODING
    - → coding as learning
    - → coding as communication
    - → code as end result
    - → code as specification
  - Testing
  - Listening
  - Designing

The 12 XP practices

- The Planning Game — quickly determine scope of next release
- Small releases — put a simple system in production quickly then release new version on a short cycle
- Metaphor — guide development with a simple shared story
- Simple design — system should be as simple as possible, complexity should be removed if at all possible
- Testing — continually write unit tests, customers write functional tests
- Refactoring — restructure the system without changing behavior
- Pair programming — all code written with 2 programmer at 1 machine
- Collective ownership — anyone can change code anywhere anytime
- Continuous integration — integrate and build many times a day
- 40-hour week — work no more than 40h/wk as a rule
- On-site customer — include a real, live user on the team full time
- Coding standards — code in accord. to rules emphasizing communication

Refactoring as one key practices

Outline

- Overview
- Introduction
- Process
- Outlook
- Discussion

- Refactoring
  - before changing the program: Is there a way of modifying the program to make adding this new feature easier?
  - after changing the program: Is there a way to make the program simpler?
  - you refactor only when the systems requires you to

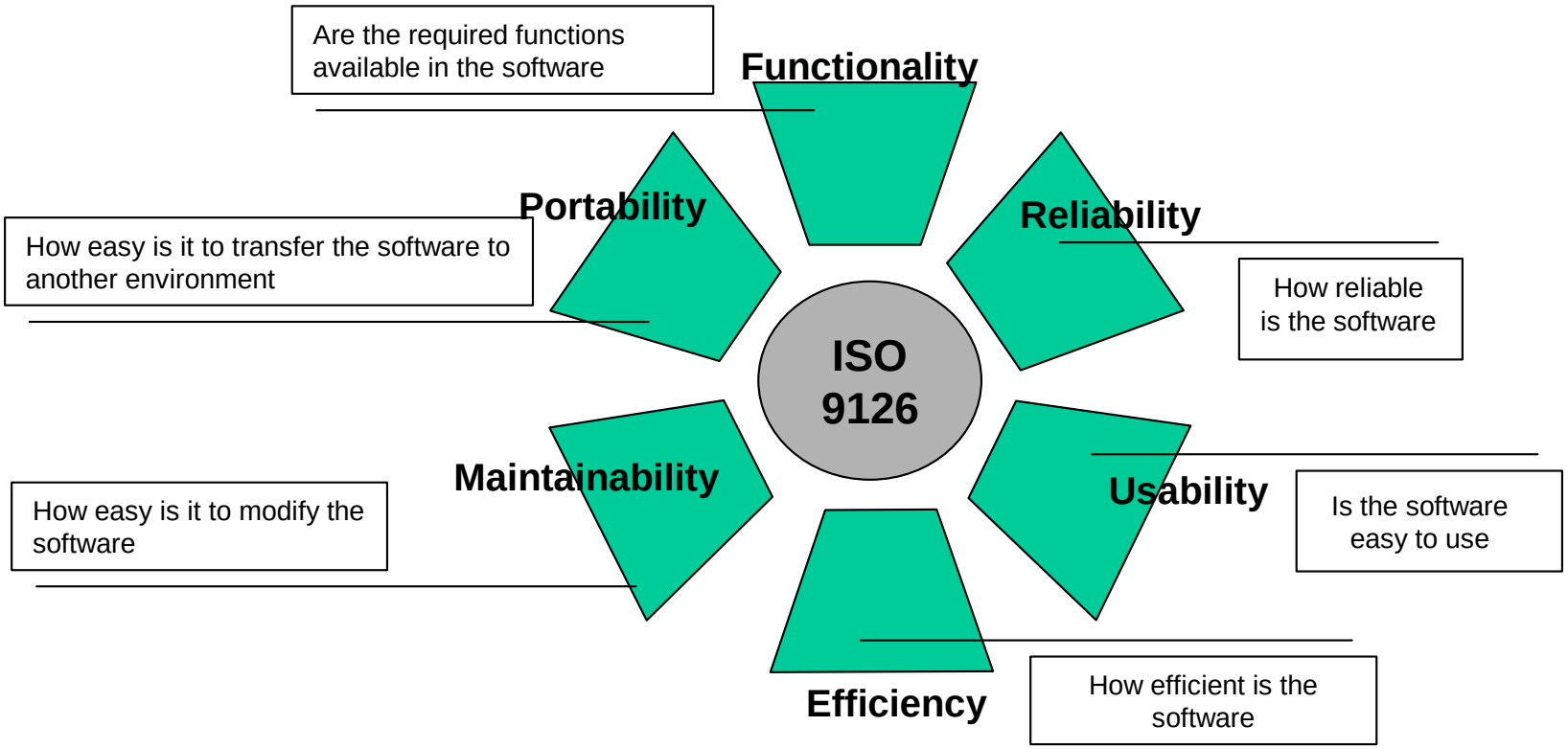
- Refactoring is context sensitive
  - *Don't refactor everything*
    (priorize and plan in order to reach specific quality-goals)
  - *Metrics help to detect quality defects*
    (but own competence for refactoring is needed)

functionality <-> quality

ISO 9126 Quality Factors

Outline

- Overview
- Introduction
- Process
- Outlook
- Discussion



Introduction

Outline

- Problems today
  - **Manual discovery of code smells** is hard, esp. in large systems
  - **Selection and planning of refactoring** activities is unclear.
  - **Application of refactorings is very subjective** (heavily based on expertise)
  - **Comprehension of experiences** is typically complex in new environments or for new users.
  - **Impact of refactoring** activities on quality aspects is unclear

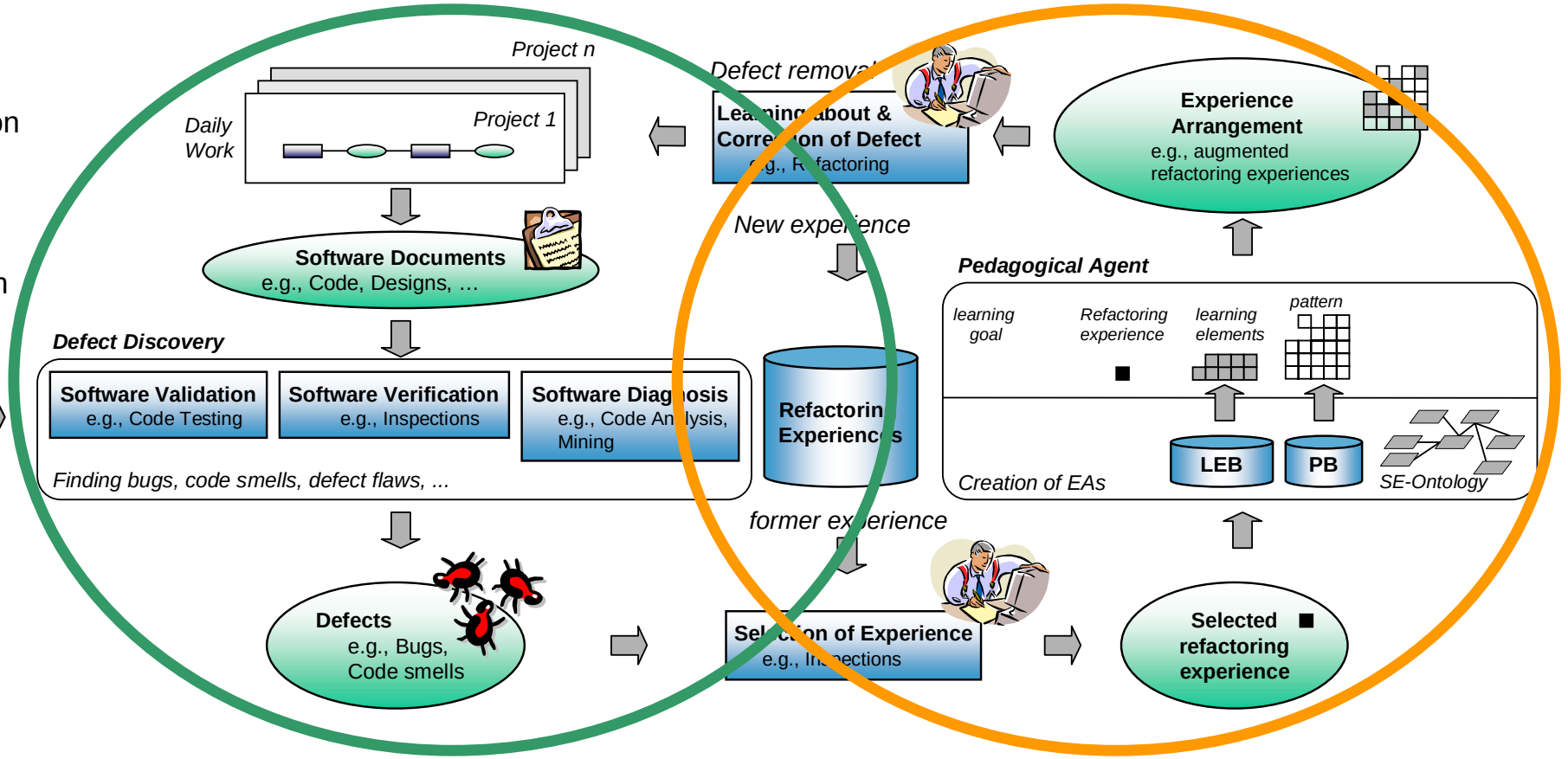
- Our approach
  - **Experience based support of the refactoring** and planning processes
  - A lightweight framework with **semi-automated support** for refactoring
    - → **Based on metrics** to detect quality defects (code smells, antipatterns, …)
    - → **Using Knowledge Discovery (KDD)** technology adapted for source code
  - **Didactical augmentation of experiences** for better comprehension

General Overview

Outline

- Overview
- Introduction
- Process
- Outlook
- Discussion

Quality-Goals



Experience - Management

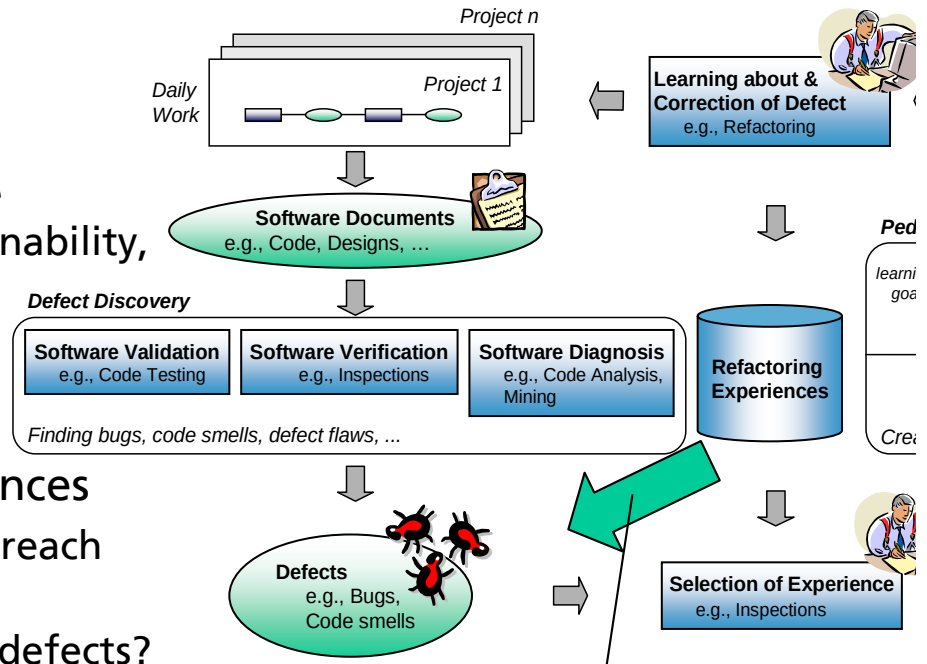
Experiential Learning

Process-Step: Refactoring

Outline

- Overview
- Introduction
- Process
- Outlook
- Discussion

- Defect Discovery
  - What might be a ***threat*** to the software quality (e.g., maintainability, reusability, or performance)?
- Selection of Defects & Experiences
  - What should be refactored to reach specific ***quality goals***?
  - What are the most ***important*** defects?
  - How do we remove these defects?
- Correction of defects
  - Based on augmented refactoring experience
  - ***Learn*** while refactoring
- Give feedback
  - Refactoring experience is collected
  - Provided experience is evaluated

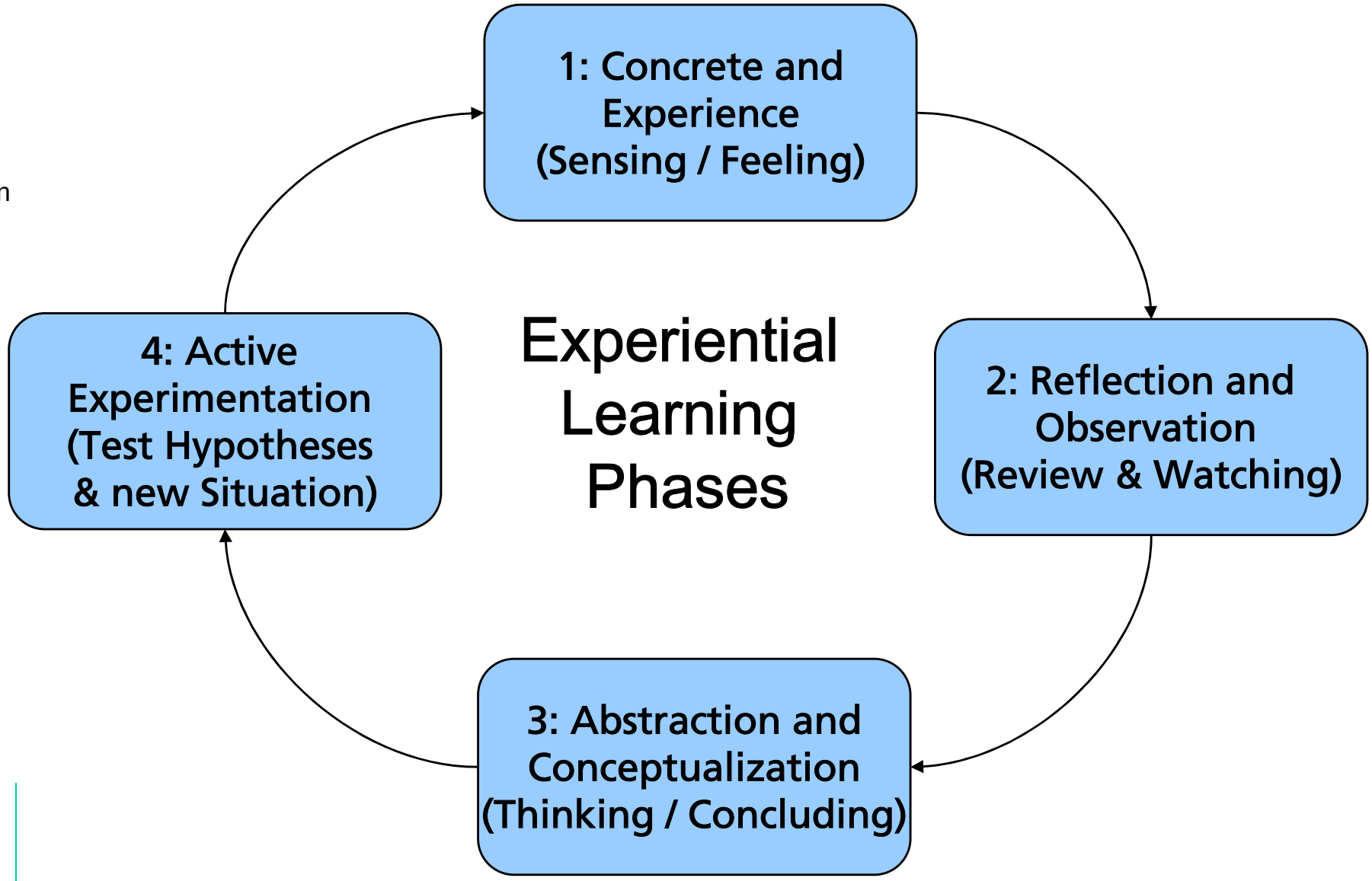


New: Use experience to categorize defects

Kolb's Experiential Learning Phases

Outline

- Overview
- Introduction
- Process
- Outlook
- Discussion



e.g., Lessons Learned

Requirements for Micro-didactical Arrangements

Requirements for the creation process:

- Consider different instructional design rules (ID-rules)
  (from ID Theories: e.g., elaboration theory, problem-based learning etc.)
- Consider different learning goals (according to Bloom taxonomy, [Bloom, 1956])

An ideal arrangement should:

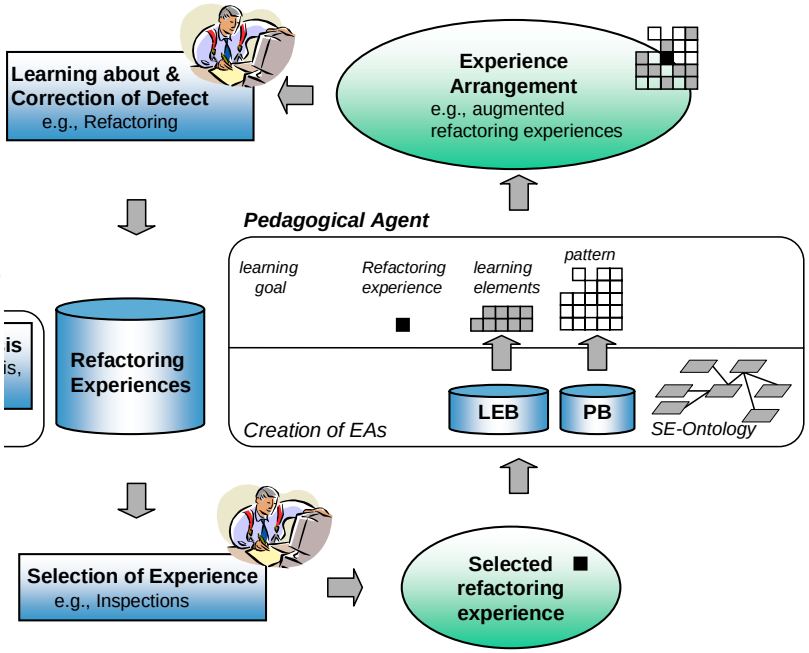
- include each of the four learning phases
- anchor contextual knowledge (e.g., cases, experiences) with declarative (e.g., facts, definitions, process descriptions) and procedural knowledge (e.g., conditions and actions)
- facilitate self-directed learning
- support individual and social constructivism (e.g., by integrating Communities of Practice)

Process-Step Experiential Learning

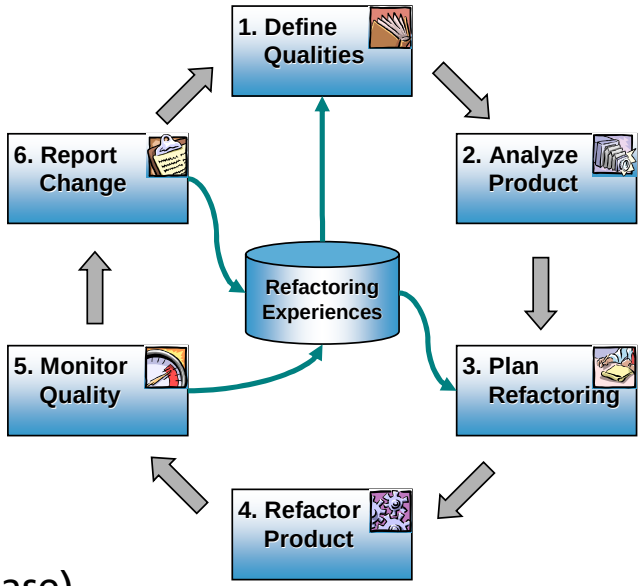
Outline

- Overview
- Introduction
- Process
- Outlook
- Discussion

- *"Learning is considered to be a fundamental part of KM since employees must internalize (learn) shared knowledge before they can use it to perform specific tasks"* **[Rus and Lindvall, 2002, IEEE Software]**
- Learning goals: e.g., "application" but comprehension is a prerequisite
- Most of us learn through reflection upon every day experience
- learning = Experience plus reflection **[Dewey, 1938]**
- Synonym (but less popular): experience-based learning



Simple Example: "Long Method" smell



Long Method

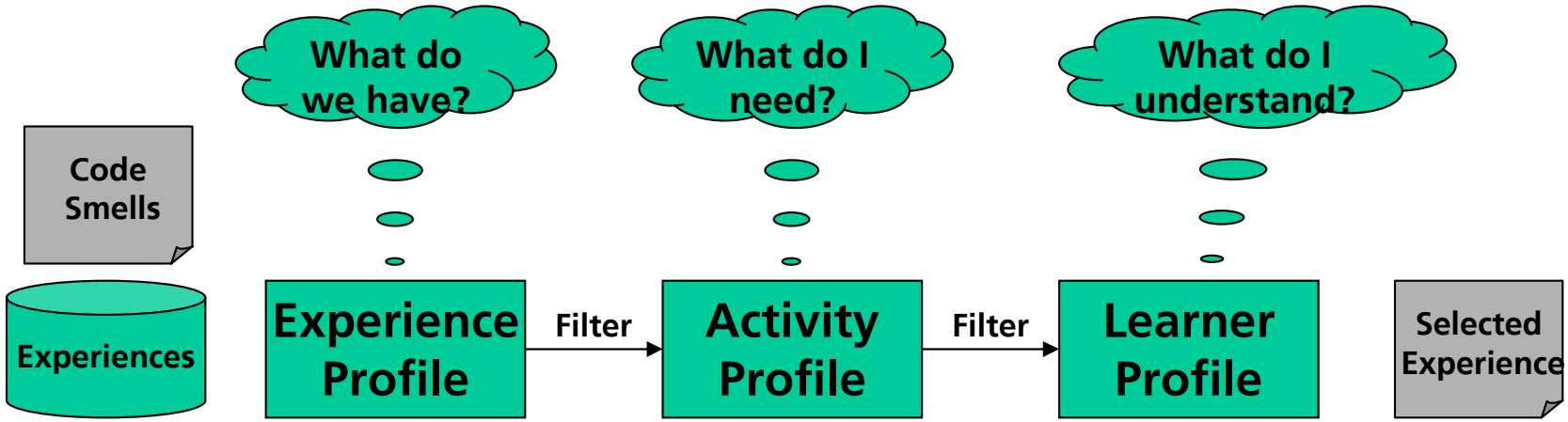
- Long methods are (typically) harder to understand than short ones and therefore make maintenance difficult.

1. Define Qualities: E.g.,
   - **Maintainability** and **Reusability** is relevant
   - **Performance** is irrelevant
2. Analyze: Measuring LOC
   - Method A: 300 LOC (e.g., write data to database)
   - Method B: 400 LOC (e.g., standard RSA encryption algorithm)
   - Method C: 500 LOC (e.g., sort algorithm)
3. Plan:
   - Refactor method A and C but not method B (due to Reusability)
4. Refactor: Apply the "Extract Method" refactoring
   - Reuse experiences from previous refactorings
5. Monitor: effects of refactorings on product quality
6. Report & Adapt: quality model (learn about refactoring effects)

Retrieving & Preprocessing Experience

Outline

- Overview
- Introduction
- Process
- Outlook
- Discussion

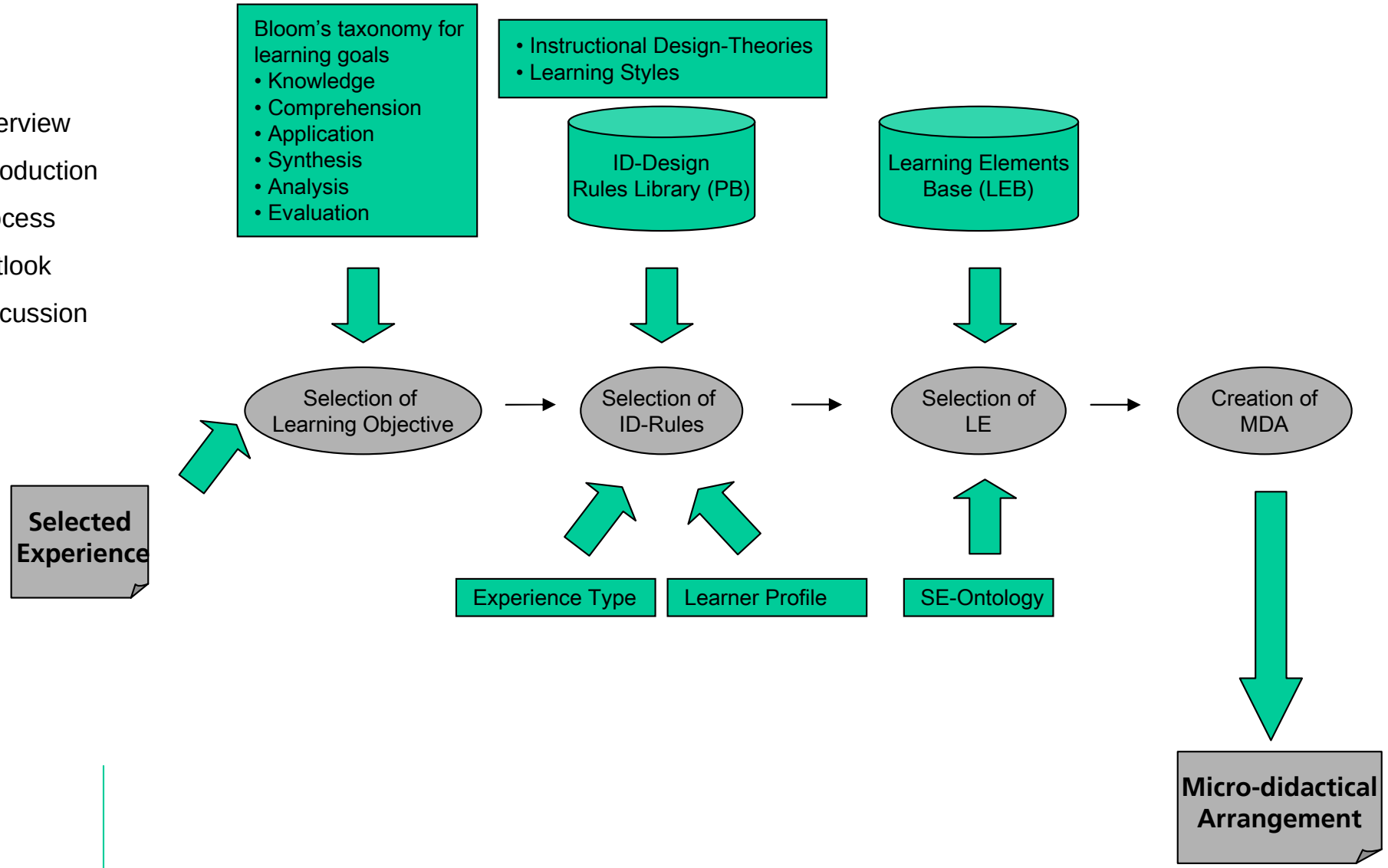


- Filtering of Experiences to reduce number of hits
  - **Experience Profile** is matched against the query to retrieve cases of interest.
  - **Activity profile** is used to remove experiences *irrelevant* to activity / goal by using metadata from the experience profile and the query (or Workflow system).
  - **Learner Profile** is used to remove experiences *incomprehensible* by using metadata from the experience profile and the user profile (from a Skill Management system).

Creation of Micro-didactical Learning Arrangement

Outline

- Overview
- Introduction
- Process
- Outlook
- Discussion



Tool Basis: MASE

- MASE (a WIKI for Agile Software Engineering)
  - Standard JSP-WIKI (Pages, Blogs, News, …)
  - Planning of Iterations
  - Definitions of Tasks
  - User Stories
- Application
  - Freeform tool to note experiences
  - Metadata in pages used as experience profile
  - Basis for communication about refactoring
- Similar tools:
  - SnipSnap [http://snipsnap.org/]
  - xpWeb [http://xpweb.sourceforge.net/]

Outlook

- Discovery of quality defects
  - Investigation of **metrics** and rules for defect discovery
    → Development of a **Metric-Defect Model**
  - Investigation of the effect of defects on **qualities** (e.g., reusability)
    → Development of a **Quality-Defect Model**

- Knowledge presentation
  - Definition of **ontologies** for defects and correction experiences
    → Can we use clustering or information **retrieval** techniques for experiences? How effective are they?
    → What **context information** is required?
  - Development of techniques for the creation of **experience arrangements**

- **Empirical evaluation**

Summary

- Our approach
  - A lightweight framework with **automated support** for refactoring
    - → **Based on metrics** to detect quality defects (code smells, antipatterns, …)
    - → **Using Knowledge Discovery (KDD)** technology adapted for source code
  - **Experience based support of the refactoring** and planning processes
  - **Didactical augmentation of experiences** for better comprehension

- Benefits
  - Improved internalization of experience through experiential learning combined with experience management
  - Active support for the task at hand
  - Improved transferability of experience

Outline

- Overview
- Introduction
- Process
- Outlook
- Discussion

Thank you for listening!
Any questions or comments?

Contact information:

Jörg Rech (Refactoring, Code Retrieval, Code Mining)
Eric Ras (Learning supported Software Engineering)
Andreas Jedlischka (Decision Support)

Address:

Fraunhofer IESE
Sauerwiesen 6
D-67661 Kaiserslautern
Germany