INTEGRATED QUALITY ASSURANCE

„Why early Quality Assurance makes the difference“
About the Fraunhofer Gesellschaft

- Named after Joseph von Fraunhofer (1787-1826), a researcher and inventor and entrepreneur
- Germany's leading organization for applied research and technology transfer
- 60 institutes
- 18,000 employees
- About €1.66 billion,
  - 1/3 base funding (government)
  - 1/3 industrial projects
  - 1/3 public sector projects
Fraunhofer Locations Worldwide

USA:
- Plymouth, Michigan
- East Lansing, Michigan
- Peoria, Illinois
- Pittsburgh, Pennsylvania
- College Park, Maryland
- Boston, Massachusetts
- Newark, Delaware

Middle East:
- Dubai, United Arab Emirates

Asia:
- Beijing, China
- Bangalore, India
- Seoul, Korea
- Singapore
- Ampang, Malaysia
- Jakarta, Indonesia
- Tokyo, Japan

Fraunhofer Center for Experimental Software Engineering (CESE)
College Park, Maryland, USA

Fraunhofer Project Centers:
- Sydney, Australia
- Bahia, Brazil
Fraunhofer-Institute for Experimental Software Engineering (IESE)

Leading Institute for Software Engineering
Founded in 1996 in Kaiserslautern, Germany
200 employees, 13.5 M € turnover

Mission
- Provide innovative and value-adding customer solutions with measurable effects
- Advance the state-of-the art in software and system engineering
- Promote the importance of empirically based software and system engineering

www.iese.fraunhofer.de
IESE is looking for quantitative and qualitative evidence

\[ C_{\text{org}} + C_{\text{cab}} + \sum_{i=1}^{n_1} (C_{\text{unique}}(p_i) + C_{\text{reuse}}(p_i)) \]

“Reviews and inspections always pay off; you can expect ROI of >2.”

“Developers using SAVE live produce 60% less architectural mistakes.”

“You typically reach your product line return on invest after the third product.”

“Tracing requirements changes with our tool support improves correctness and completeness by 25%.”
Core Competencies by Fraunhofer ISE Division

Embedded Systems
- System and Software Architecture
- Variation Management
- Safety Engineering
- Model-based Testing

Process Management
- Measurement and Prediction
- Empiricism
- Process Compliance and Improvement

Information Systems
- Requirements Engineering
- User Experience for Business Applications
- Architecture-centric Engineering
- Security Engineering
- Engineering Mobile Solutions

© Fraunhofer ISE
supported phases for our customers

Assess
- Analytical baseline

Define
- Definition and design in accordance with customer goals

Innovate
- Development of improvements and innovations

Release
- Target-oriented introduction and long-term roll-out

This scheme holds for the customer’s products as well as for their development processes
As specified in the project request

As designed by the senior analyst

As installed at the user's site

What the user wanted
Communication – not as easy ...

WIZARD OF ID

BY BRANT PARKER & JOHNNY HART

I WANT A POTION THAT WILL ADD FORTY YEARS TO MY LIFE!

GLUCK
GLUCK
GLUCK

TRY THIS

...LET ME REPHRASE THAT

YOUR REQUIREMENTS DOCUMENT IS THE BIGGEST I'VE EVER SEEN.

IT'S TOO BIG TO READ, BUT I CAN GUESS FROM ITS WEIGHT WHAT MUST BE IN THERE.

YOU KNOW IT'S A MULTI-USER, GLOBAL SYSTEM, RIGHT?

NO, I'M NOT GETTING THAT.
Stakeholders

Stakeholders are all people who have an interest in the product

- they built it
- they use it
- they manage it
- they are in some way affected by the use

- Product marketing, Actual and potential customers, Old and new users
- Project leader, Technical leader, Business manager
- Architect / Designer, Developer, Tester
- Legal Department, etc.

- Maybe in different locations, maybe in competing divisions
Communication – A challenge for Requirements Engineers

Communication happens via speech

- Representation of experiences (perceptions)
- Communication of personal reality (presentation)

[Diagram showing the process from Customer/User to Requirements Engineer]

Objective reality → Perception → Personal reality → Presentation → Linguistic expression → Interpretation → Result
Objectives of the Elicitation Phase

Knowledge acquisition (Elicitation, Acquisition)
- About involved stakeholders and objectives
- Tasks
- Current state
- Expectations
- Domain

Negotiation
- Solving of Conflicts (between request and reality; between perspectives of different persons)

Prioritization
- Selecting objectives
Common Problems of Elicitation

- Consideration of all stakeholders

- Communication
  - Stakeholders can not describe abstractly what they are doing, why they are doing it nor what they need to be able to do things.
  - Requests are much too general

- Presentation of new possibilities and their consequences
  - Stakeholder like to stick to their old avenues of approach

- Conflicts
  - Cause of power struggles
  - Cause of opposition against changes

- Priorities/Changes
  - Stakeholders want to much
  - Stakeholders always add new ideas
Holistic Engineering of large information systems: Integration of business, end-user and technical perspective!
Product Quality (ISO 9126/DIN 66272)

- Functionality
  - Adequacy
  - Security
  - Precision of calculation
  - Interoperability
  - Conformity with standards

- Reliability
  - Maturation
  - Fault tolerance
  - Recovery

- Usability
  - Comprehensibility
  - Learnability
  - Operability

- Efficiency
  - Time response
  - Consumption

- Changeability
  - Analysability
  - Modifiability
  - Stability
  - Verifiability

- Portability
  - Adaptivity
  - Installability
  - Conformity with standards
  - Replaceability
Quality of Requirements
Typical Quality Problems of Requirements (1)

Syntactics

■ Badly structured
  ■ Risk: barely extendable, hardly readable

■ Incomprehensible
  ■ Risk: incorrect transformation, not traceable, not testable

■ Ambiguous and vague
  ■ Risk: incorrect transformation, not testable

■ Redundant
  ■ Risk: hardly changeable, barely refineable, slightly inconsistent
Typical Quality Problems of Requirements (2)

Semantics

- Inconsistent
  - Risk: incorrect transformation, hardly correctable

- Incomplete
  - Risk: solutions that do not fulfill important expectations or solutions, behave incorrectly under certain circumstances

- Incorrect
  - Risk: solutions that do not behave as they are intended to (wrong behavior)
Typical Quality Problems of Requirements (3)

**Authority**

- Superfluous (nobody asks for the requirement)
  - Risk: unnecessary developments costs or solutions that do not behave as they are intended to

- Too constricting
  - Risk: not feasible

- Inconsistent with standards, laws, previous documents
  - Risk: no approval
Characteristics of a Good Software Requirements Specification

[SRS quality attributes

- complete
- internally consistent
- correct

Externally consistent
- Electronically stored

Interambigious
- Annotated by relative importance
- not redundant

Concise
- design independent
- Annotated by version

Precise
- achievable

Organized
- modifiable

Traced
- cross referenced

Executable
- At right level of Abstraction

Annotated by relative stability

Understandable
- traceable

Precise
- reusable

At right level of Abstraction
- traceable

Reduced
- executable
Characteristics of a Good Software Requirements Specification
IEEE Std. 830-1998

- Correct (means only true requirements, implementing the “right” system)
- Complete (means all non-functional requirements, all system reactions – even for invalid inputs, explicit listing of open issues)
- Unambiguous (means only one possible interpretation)
- Traceable (means capturing the source of requirements and allocation of unique identifiers)
- Consistent (no conflicts in as-is descriptions, desired behavior, consistent terminology)
- Verifiable
- Ranked for importance and/or stability
- Modifiable (means explicit structure, good tables for overviews, no redundancy)
- Understandable
Correctness

A requirement is **correct**, if all relevant stakeholders confirm it and if the requirement has to be completely implemented in the future system. If the requirement unnecessarily increases the systems‘ range of functions, it is not correct.

- Can be (constructively) ensured by:
- Systematic derivation from superior requirements/ goals
- Avoidance of gold-plating
Completeness

A requirement is **complete**, if it is documented according to defined criteria and if there are no content-related vacancies.

Incomplete requirements leave a margin for additions.
Completeness

Example of an incomplete requirement:

- “As soon as the login window appears, customer-specific information has to be entered.”
  - Who enters the information?
    - Which information?
  - Complete: “… appears, the user has to enter his customer number, email address and password.”

✅ Can be (constructively) ensured by:

- Use of active instead of passive formulation
- Writing verifiable requirements
A requirement is unambiguous if it is verbalized in such a way that only one valid interpretation is possible. Ambiguous requirements often lead to undesired implementations of the requirement in the system.
Unambiguousness

Example of an ambiguous requirement:

“For authentication, the driver enters an electronic card and a PIN. If it is invalid, the vehicle cannot be started."

¬ Ambiguous: What is invalid? The card or the PIN, or both of them?
¬ Unambiguous: “… and a PIN. If the PIN is invalid, the vehicle cannot be started.”

✓ Can be (constructively) ensured by:
  ✓ Avoiding passive formulations
    ✓ Wrong: “The result is shown”
    ✓ Right: “The system shows the result”
  ✓ Avoiding relative clauses/ nested sentences
  ✓ Avoiding weak words
List of Weak Words

a little, about, accordingly, actual, adequate, again, all but, almost, also, analogous, another, anything, apparent, approximately, articulate, as a whole, at the most, basic, besides, best, better, big, bit, certain, certainly, circa, clear, clearly, close by, closely, common, conditionally, considerable, decided, dedicated, defined, definite, detailed, determinately, different, directly, distinct, diverse, else, enough, especially, evidently, explicit, extensive, extremely, few, grand, great, however, huge, if need, improved, in detail, in most instances, indeed, just, largely, later, like, likewise, limited, long-standing, maybe, mostly, much, nearly, necessarily, no case, no way, not exhaustive, obvious, of course, ongoing, or so, ordinary, other, particular, perhaps, persistent, possible, powerful, precise, presently, prodigious, provisory, qualified, quite, related, satisfactory, seemingly, self-evident, separately, several, shortly, similar, some day, something, sometimes, soon, special, still, sufficient, sure, sustained, to be announced, to be confirmed, to be defined, to be desired, to be detailed, to be determined, too, ultimate, uncommon, unique, unlike, usual, well, where
Consistency

A requirement is consistent,

if statements of a requirement do not contain any contradiction and are not in conflict with each other.

In the case of a software requirements specification, not only do individual requirements have to be consistent, but it is also important that no subset of individual requirements described in it conflicts.
Consistency

Examples of inconsistency:

- Ad (1): Different terminology of similar objects (e.g. “Input of travel destination”, “Input of destination place”, “Input of destination”)
- Ad (2): In at least 2 requirements, attributes of objects may conflict (e.g., one requirement may state that input is via keyboard while another may require voice input)
- Ad (3): There may be logical or temporal conflicts between two requirements (e.g., one requirement may state that “A” must always follow “B”, while another may require that “A” and “B” occur simultaneously.

✓ Can be (constructively) ensured by:
  ✓ Using a glossary
  ✓ Avoiding synonyms (use of different terms for the same object e.g., “customer number” “customer ID”)
  ✓ Avoiding homonyms (words that share the same spelling and pronunciation but have different meanings, e.g., “bank”)
A requirement is **verifiable** or **measurable**, if a person or machine can check that the software product meets the requirement. In general, any underspecified requirement is not verifiable.

Especially important for Non-functional Requirements!
Verifiability

Examples

- Not verifiable requirement: “The usual response time of the system shall not be more than 2 seconds.”

- Verifiable requirement: “Output of the program shall be produced within 20s of event E1 60% of the time; and shall be produced within 30s of event E1 100% of the time.”

  ✓ Can be (constructively) ensured by:
  ✓ Providing a specific, quantified description
  ✓ Writing testable requirements
Writing Guidelines for Requirements

Goal: Requirements are **easier to read** and therefore easier to understand

Writing guidelines consider **common problems**, project-specific additions may be useful

Guidelines should be summarized into company-specific writing guidelines
Writing Guidelines for Single Requirements (1)

- Short sentences
- Subject – Predicate - Object
- Describe just one single requirement per sentence
- Avoid “and”
- Avoid colloquial language
- Economical use of abbreviations
- Short paragraphs – 7 sentences maximum
- Bullet lists instead of long sentences
- Consistent use of notations and terminology
- Avoid nested sentences about logical connections
- Allocate requirements to unique identifier
- Avoid gold-plating (embellishments, additional functions)
Writing Guidelines for Single Requirements (2)

- Illustrate the emphasis of requirements
- Use words like “have to” “should” or “can” only if their meaning has been defined before (e.g., “have to” or “should” means that the requirement is obligatory)
- Separate optional from obligatory requirements by using a definition, a suitable attribute, or a heading
- Justify requirements
- Use active instead of passive sentences
- Wrong: The result is shown
- Right: The system shows the result (clearly defines who is the actor)
- Illustrate complex circumstances/dependencies with the help of graphics
- Use correct and precise references
- Use automated spell check
Writing Guidelines for Single Requirements (3)

✓ Write testable requirements for verifying whether the system can fulfill the requirements
✓ Is it possible to create a test case from requirement X?
✓ Avoid generalization
✓ Leads to ambiguity
✓ Document the origin of each requirement
✓ Having a huge set of requirements, it will be difficult to remember the origin of every requirement if a requirement has to be changed (e.g., in case of a conflict)
✓ Explanations in requirements are confusing
✓ Bad example: “To work as efficiently as possible, the experienced user’s access rights are verified by a double click on a list entry and if his confirmation is valid, customer-specific information is shown in the field ‘Access’. If an exception is thrown from the SQL request, then …”
✓ Better solution: Explicit labeling of explanations and rationale
Writing Guidelines for Software Requirements Specifications

- Define an obvious document structure
- Add a glossary
- Avoid synonyms/ homonyms
- Ensure traceability of requirements
  - e.g., by using matrices, graphs
# Costs

# found defects vs. costs per defect
Zone of chaos
A typical Test focus...

Not all defects can be found (efficiently) by testing!

Testing usually observes failures → debugging necessary

Testing focuses on failures according to the IEEE Standard 1044-2009 [2]
@Override
public void onClick(View v)
{
    Event milestone = new Event("MilestoneEvent");
milestone.addAttribute("milestoneName", "TOA");
milestone.addAttribute("shipmentID", currentShipment.orderName);
milestone.addAttribute("timeReached", "" + System.currentTimeMillis);
EventbusManager.getManagerInstance().publishEvent(milestone);
db.insert(currentShipment.orderName, 1);
currentShipment.isAccepted = true;
}

rejectButton = (Button) findViewById(R.id.btnReject);
rejectButton.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View v) {
        currentShipment.isAccepted = false;
        clicked();
    }
});
What can be inspected?
Goals of inspections
Finding defects

```java
@Override
public void onClick(View v) {
    Event milestone = new Event("MilestoneEvent");
    milestone.addAttribute("milestoneName", "TOA");
    milestone.addAttribute("shipmentID", currentShipment.orderName);
    milestone.addAttribute("timeReached", "" + System.currentTimeMillis() / 1000);
    EventbusManager.getInstance().publishEvent(milestone);
    db.insert(currentShipment.orderName, 1);
    currentShipment.isAccepted = true;
    clicked();
}
}
rejectButton = (Button) findViewById(R.id.btnReject);
rejectButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        currentShipment.isAccepted = false;
        clicked();
    }
});
```

© Fraunhofer IESE
Better quality
Gathering metrics
Finding solutions
Aufbau von gemeinsamem Wissen

Coaching
Distributing knowledge
Team building
Reading Techniques – Overview

- Ad-hoc
  - No support, experience-based

- Checklist-based reading
  - Checklist-based reading (CBR): “one checklist for all”
  - Focused checklist: Inspectors receive different checklists containing questions that focus on the needs of different perspectives
  - Guided checklist: Inspectors receive different checklists containing questions that focus on different quality aspects

- Scenario-based reading
  - Perspective-based reading (PBR)
  - Defect-based reading (DBR)
  - Usage-based reading (UBR)

- Others
  - Stepwise abstraction
Example of a “tester” scenario for requirements inspection:

Imagine you are a tester of the requirements. One of your main tasks is to derive acceptance test cases for the various requirements specified in the functional specification and the related user scenarios. To successfully perform your task, it is most important for you that the test cases are easy to derive from the requirements.

Instructions: For each assigned requirement of the functional specification in the document and for each related functional user scenario generate a test case or set of test cases that allow you to ensure that an implementation of the system satisfies the functional specification (set of acceptance cases)...

Document all issues you identify while performing the task in the issue list.

Questions: While following these instructions answer the following questions:

• Which parts of the requirements are difficult to understand and hard to derive the test cases from, i.e. where is it unclear what the correct behaviour should be?
• Which necessary information is missing to identify the item being tested and the test criteria (e.g. where are valid inputs or outputs missing, are acceptance criteria missing)?
Comparison of Inspections and Testing

<table>
<thead>
<tr>
<th>Inspections</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can be used in early development stages</td>
<td>Is usually used in late development stages</td>
</tr>
<tr>
<td>Can address different aspects of quality</td>
<td>Requires an executable system/model</td>
</tr>
<tr>
<td>Usually manual performance</td>
<td>Identification of “wrong behavior” → debugging is necessary</td>
</tr>
<tr>
<td>Identification of potential defects</td>
<td>Very useful for dynamic aspects</td>
</tr>
</tbody>
</table>

For effective QA, joint use of inspections and testing is required
Inspections and testing should be applied both!
In2Test – Application

1. Inspection
- Defect profile
- Product data
- Historical data

2. Quality monitoring
- Scope of validity
- Empirical knowledge / evaluated assumptions

3. Prioritization
- Selection rules
- System parts
- Defect types
- System parts & defect types

4. Focused test (based on QA strategy)

Key:
- Product
- Process
- Database
- EDB

Scope of validity

Corrected Artifact

1-stage approach

2-stage approach
Finally… some words on automation!
Solution Idea

- Usage of “continuously collected Requirements based metrics” to...
  - support automatic visualization of the quality and status of the project requirements artifacts,
  - supporting early evaluation of requirements information, e.g. consistency, completeness and complexity,
  - supporting decision making

- We use a combined approach of requirements quality metrics and traceability information
Requirements Engineering metrics

Definition: Metric

“A measurement derived from a software product, process, or resources. Its purpose is to provide a quantitative assessment of the extent to which the product, process, or resources possess certain attributes.” [Costello]

Required properties of a metric

- Clear definition of calculation rule and data collection (How?)
- Interpretation of the metric / significance (What?)
- Purpose of the metric in the project / organization (Why?)

[Costello, R.J.: Metrics for requirements engineering. in Sixth annual Oregon workshop on Software metrics. 1995. Silver Falls, Oregon, US]
Influencing Metrics for Requirements Documents

Purpose of some special metrics

- Assessment of the **quality of the requirements documents**
  - Language aspects (mostly natural language / Use Cases)
    - Like ambiguity
  - Content-related aspects
    - Like complexity
  - Structural / documentation aspects
    - Like completeness and consistency

- Assessment of the **Requirements Engineering (development) processes**
  - Quality assurance measures (testing, inspection, etc.)
  - Status information about the project (progress)
Capturing Requirements metrics by applying requirements models

Goal
- Measure the complexity and consistency of the current SRS

Question
- How many relationships (Traces) exist in the requirements specification among the documentation artifacts?
- Are the relationships consistent to the Trace model?

Metric
- Traceability (Linkage statistics)

\[ LStat = \sum \text{of \_higher \& lower \_level \_Req.to \_which \_each \_Req \_is \_traced} \]
Example for Complexity and Consistency

Consistency Error!
No Activity diagram for Task 2

1 Use Cases with more than 10 Functions
→ High complexity of the Use Case?
→ Use Case over specified?

8 Use Cases with only 9 Functions
→ Use Cases to simple?
→ Functionality Missing?

Example: Complexity and Consistency

- Consistency Error!
  - No Activity diagram for Task 2

- Metrics

Graph showing:
- Tracelinks vs. # of artefacts
- 8 Use Cases with only 9 Functions
- 1 Use Cases with more than 10 Functions

Questions:
- High complexity of the Use Case?
- Use Case over specified?
Example: Requirements Volatility

Goal
- Identify the current work in progress and working areas of the requirements specification

Question
- Which parts of the current specification are currently under development?
- Which requirements have been changed very often?

Metric
- Count the number of requirements changes during period t
## Example for Volatility

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Source ID</th>
<th>0</th>
<th>1</th>
<th>2-4</th>
<th>&gt;=5</th>
<th># Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A0</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A1</td>
<td></td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A2</td>
<td></td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A4</td>
<td></td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B1</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B2</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>B3</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>B4</td>
<td></td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C1</td>
<td></td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>C4</td>
<td></td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>C5</td>
<td></td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

![Graph](image)

Legend:
- NA
- 0
- 1
- 2-4
- >=5

### Period
- January
- February
- March
- April
- May
- June
Example: Evaluation for Test Coverage

Goal: “Identification of when testing becomes borderline.”