Software Engineering

Prof. Dr. Joakim von Kistowski

TH Aschaffenburg
university of applied sciences

Motivation and Challenges

How to ensure Quality?

Software Test

Validation

Reviews

Disciplines in software engineering

Basic topics

Configuration management | Documentation |
Knowledge management | People in the SWE process and digital ethics | Tools

Development

Requirements
- Context analysis
- Requirements Engineering

Design
- Architecture
- Detailed design

Implementation

QualityMgt.

Quality assurance and testing
- Test, inspection, metrics

Processes and procedure models
- Improvement, process model, maturity levels

Evolution

- Roll-Out
- Operation
- Maintenance
- Further development
- Reuse
- Reengineering
- Change management

Management

- Strategy
- Economy
- Team
- Dates
- Risks
- Customer, client/contractor
- Innovation

Motivation and Challenges

How to ensure Quality?

Software Test

Validation

Reviews

Question

Which examples of Software Quality Problems come to mind?

It is very difficult to ensure quality.

We are familiar with software errors from

#failure targets

FEHLER IN DER SOFTWARE

Zehntausenden Studenten droht Bafög-Verspätung

AKTUALISIERT AM 24.08.2016 - 16:08

Seit dem ersten August gibt es höhere Bafög-Sätze und Freibeträge. Doch eine gängige Bearbeitungs-Software kennt diese Neuerungen nicht. Deshalb könnten viele Studenten zum Semesterstart erst einmal ganz ohne Geld dastehen.

https://www.faz.net/aktuell/karriere-hochschule/campus/fehler-in-der-software-zehntausenden-studenten-droht-bafoeg-verspaetung-14403775.html

Deutsche-Bank-Kunden kommen nur eingeschränkt an Geld

Nachdem bei der Deutschen Dank online fehlerhafte Doppelbuchungen aufgetreten sind, können Kunden vielfach keine Terminals nutzen. VON SARAH KRAMER

Kunden der Deutschen Bank müssen vorerst auf Kontoauszüge verzichten. FOTO: DPA

https://www.tagesspiegel.de/wirtschaft/softwarefehler-deutsche-bank-kunden-kommen-nur-eingeschraenkt-an-geld/13685606.html

It is very difficult to ensure quality.

We are familiar with software errors from

#failure targets

FREITAG, 09. SEPTEMBER 2016

Tödlicher Softwarefehler bei Airbags

GM ruft 4,3 Millionen Autos zurück

Modelle der Marken Chevrolet, Buick und Cadillac sind von dem Rückruf betroffen.

Wiedereinmal machen die Airbags einem großen Autobauer Ärger. Mehr als vier Millionen Fahrzeuge muss GM in den USA zurückrufen. Der Fehler soll bereits einem Menschen das Leben gekostet haben

https://www.faz.net/aktuell/wirtschaft/GM-ruft-4-3-Millionen-Autos-zurueck-auf-14603516.html

Roche informiert: Softwarefehler in der Diabetes-Management-App Accu-Chek Connect

VON REDAKTION · VERÖFFENTLICHT 12. JANUAR 2017 · AKTUALISIERT 12. JANUAR 2017

Accu-Chek | Diabetes | Roche Diabetes Care Deutschland GmbH

Die Roche Diabetes Care Deutschland GmbH informiert über einen Softwarefehler in der Diabetes-Management-App Accu-Chek Connect

Wie das Unternehmen mitteilt, wurde ein Softwarefehler in der Diabetes-Management-App Accu-Chek Connect der Versionen 1.2.0 und 1.2.2 (iOS und Android) entdeckt.

Es ist möglich, dass die betroffenen App-Versionen bereits gespeicherte Insulindaten nicht in die Berechnung einbeziehen und so einen falschen Insulin Bolusvorschlag ausgeben.

https://www.produktwarnung.eu/2017/01/12/roche-informiert-softwarefehler-in-der-diabetes-management-app-accu-chek-connect/4749?cookie-state-change=1622298477301

Question

Why are there errors in the software?

The primary cause of an error is an incorrect implementation

The real causes are manifold.

#failure targets

Notorious example: Ariane 501

- Ariane5 as successor to the Ariane4 family with over 100 successful launches → Lots of code reused
- Maiden flight on 4.6.1996
- After a few seconds, the rocket blew itself up
- Mission loss
  - Costs > 500 M€
  - Program held up for 3 years
- Commission of Inquiry
  - Report from 6/19/1996
  - http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html

The primary cause of an error is an incorrect implementation

The real causes are manifold.

#failure targets

Notorious example: Ariane 501

- **Primary cause:** operand error during conversion, lack of exception handling

BUT: the "root causes" are manifold, among others:

- Value ranges were assumed for the variables concerned, but not documented (distributed responsibility, lack of documentation).
- The assumptions could not be verified on the basis of the planned trajectory, as this was not part of the requirements specification (management, lack of software engineering).

The primary cause of an error is an incorrect implementation

Various quality assurance measures could have prevented the error

#failure targets

Notorious example: Ariane 501, possible quality assurance measures

- **Cost management**: Seeing the costs of a preventive measure in relation to the costs of a defect
- **Reuse**: Existing software must not be reused for a new task without inspection. It must first be checked whether its capabilities meet the requirements of the new task.
- **Specification**: The capabilities of a software and all the assumptions it makes about its environment must be clearly specified.
- **Error handling**: Every potential error situation in a software must either be handled or the reasons for not handling it must be documented in such a way that the validity of the assumptions made can be verified.

The primary cause of an error is an incorrect implementation

Various quality assurance measures could have prevented the error

#failure targets

Notorious example: Ariane 501, possible quality assurance measures

- **System test**: When testing software that consists of several components, it is not enough to test each component in isolation. Comprehensive system tests under the most realistic conditions possible are necessary.

- **Review**: In addition to a thorough test, every program must be inspected by competent experts because the fulfillment and adequacy of assumptions and results in particular often cannot be tested.

Motivation and Challenges

How to ensure Quality?

Software Test

Validation

Reviews

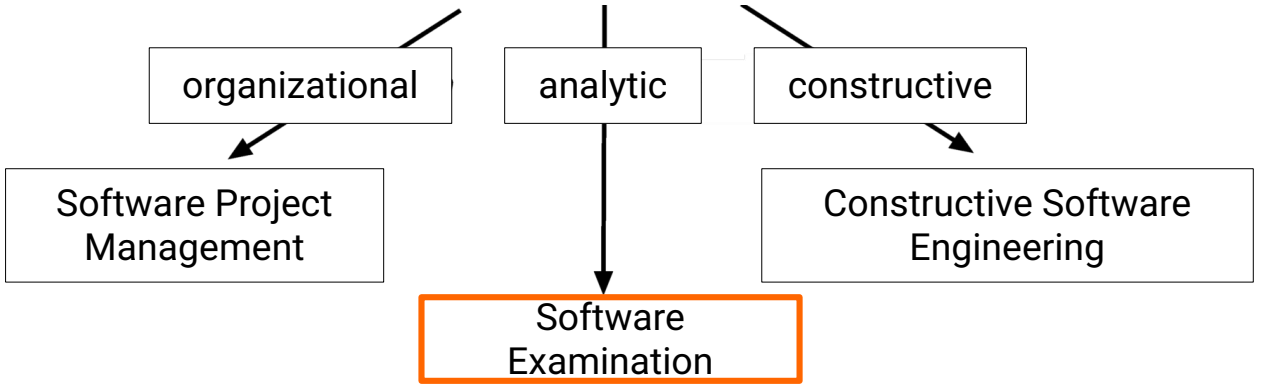
Question

How do you ensure quality?

How do you ensure quality?

- **Quality management**: established process for dealing with quality
  - How to describe quality (→ Requirements Engineering)
  - How to evaluate the quality of software (SW) (→ SW testing)
  - How to achieve quality
    (→ entire lecture, professional software engineering + SW architecture)
  - How to find errors/defects (→ SW test)
  - How to avoid errors/defects
    (→ entire lecture, professional software engineering + SW architecture)
- **Quality assurance** (QA): concrete procedure for ensuring quality
  - Constructive QA
  - Analytical QA
  - Organizational QA

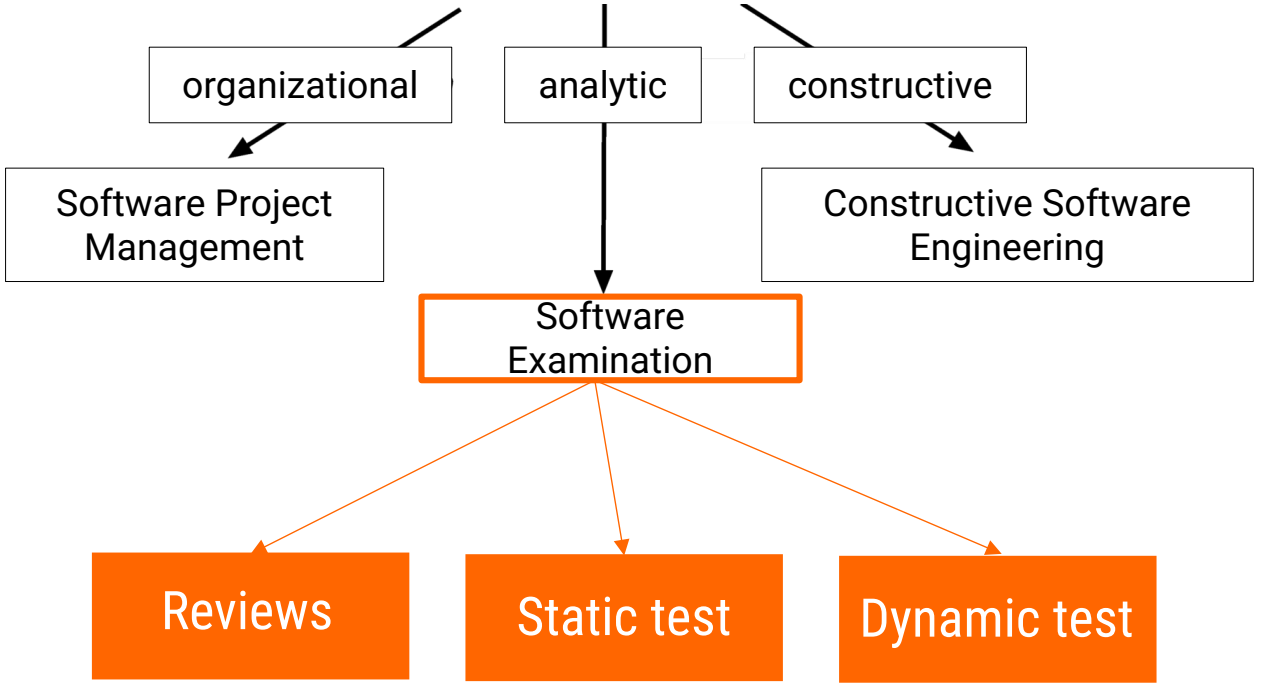
Software Quality Assurance



Quality cannot be "tested" in the product!

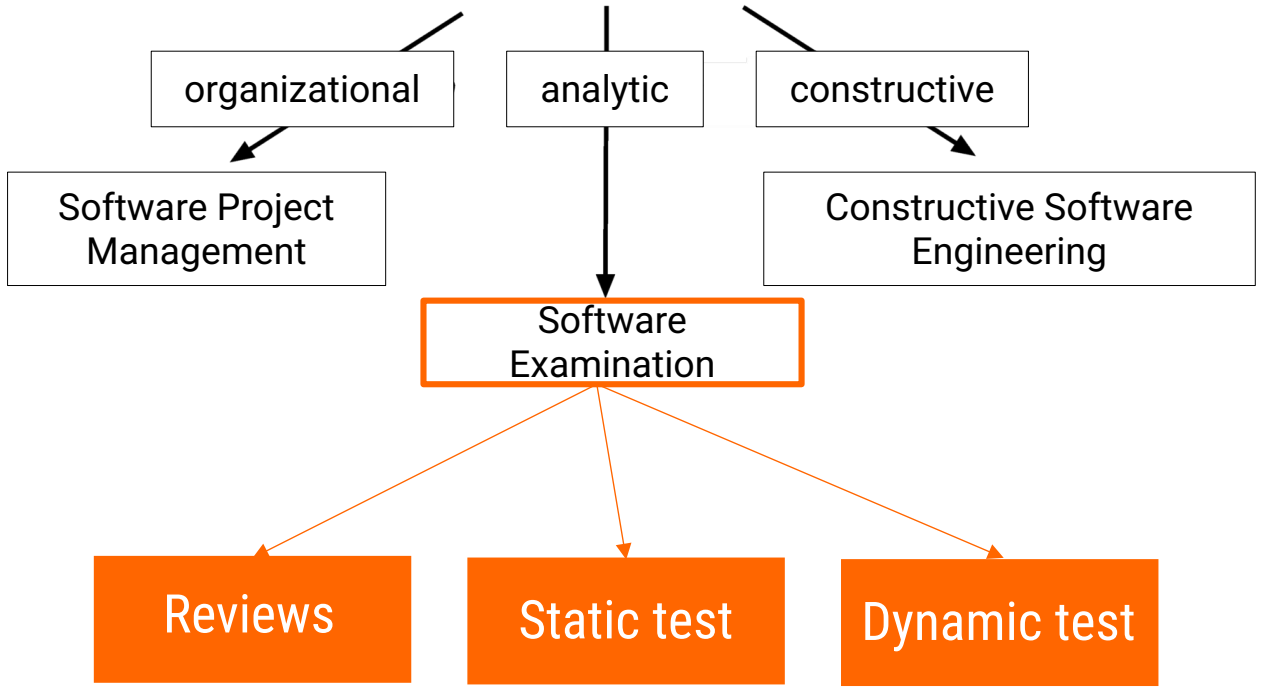
Organizational and constructive measures are much more important and are supplemented by analytical measures.

Software Quality Assurance



Software Quality Assurance

What is what?

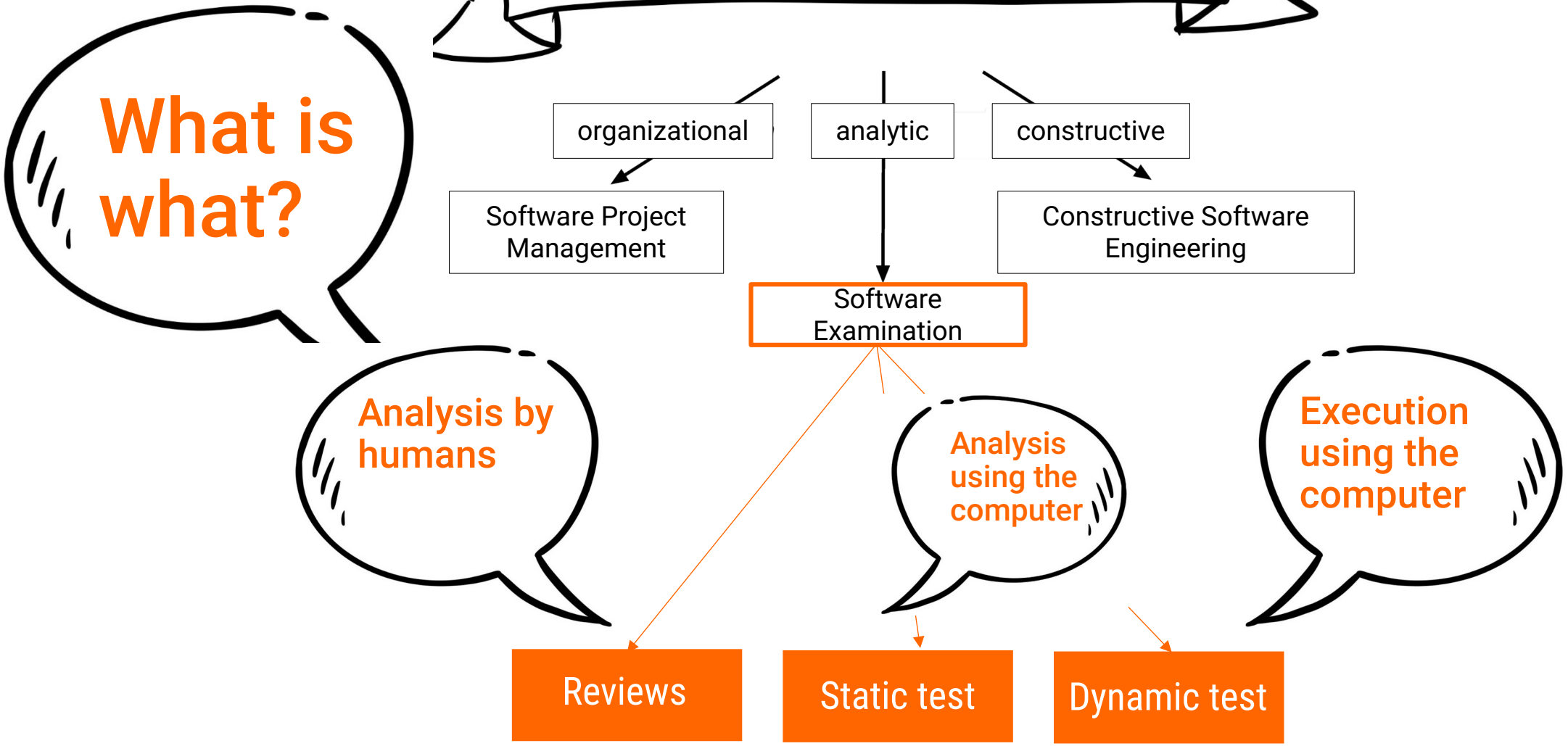


Execution using the computer

Analysis using the computer

Analysis by humans

Software quality assurance



Terms

Motivation and Challenges

How to ensure Quality?

Software Test

Validation

Reviews

Question

Why do you test?

to find errors.

There are also many other test objectives. But the primary test objective is to find errors.

The primary objective of testing is to **find errors**.

Further test objectives (and many more...)

Verification

Have all requirements been met?

Create trust

Creating confidence in the level of quality

Information for stakeholders

Assessing quality risks

Validation

Check legal/contractual conformity

Question

What is the software test?

Software test

→ Definitions

Testing *is the process within the software development lifecycle that evaluates the quality of a component or system and related work products.*

ISTQB Certified Tester Glossary

Failure *is an event in which a component or system does not meet its requirements within specified limits.*

ISTQB Certified Tester Glossary

Defect *is an imperfection [...] in a work product where it does not meet its requirements [...].* → ***Cause of*** *a failure.*

ISTQB Certified Tester Glossary

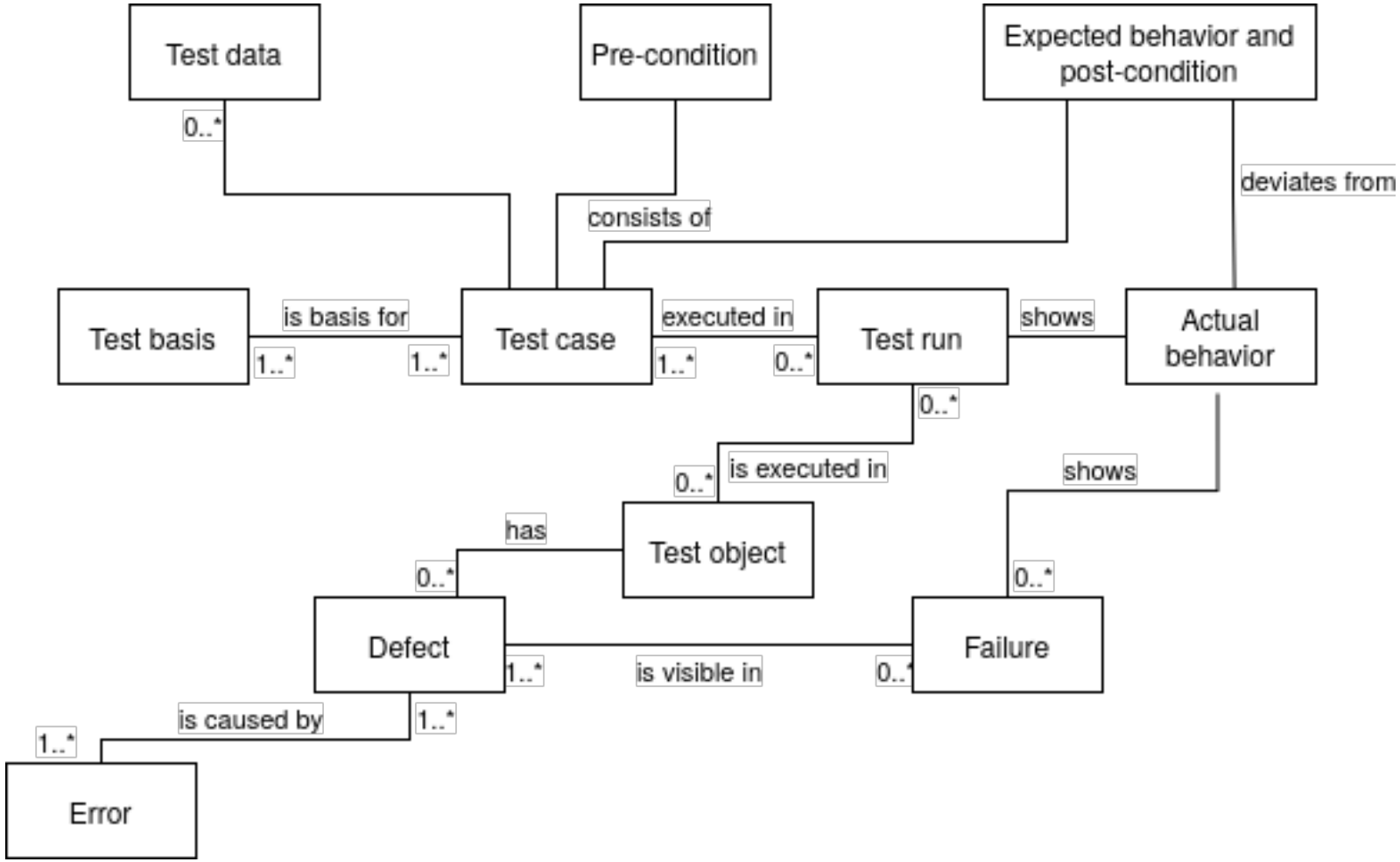
Error → *Human action that leads to an error.* *ISO 24765*

Question

When customers report errors… is that a failure or a defect?

Software test

→ Terms relating to the test



What does a test case contain?

Possible contents

- **Name** → Test objective, what is to be tested?
- **Tested request** (reference to a request or to an error)
- **Precondition** → What data/state is present?
- **Postcondition** → What data/state is available after the test case has been executed?
- **Description of the test steps** → Input, expected outputs, expected exceptions
- **Test infrastructure** → What is necessary to execute the test case?

Logical vs. concrete test case

- **Logical test case:** Value ranges for input/output
  - Empty list
  - List with one element
  - List with many elements
- **Specific test case:** Concrete input/output
  - Empty list
  - List with the element "Person x"
  - List with the elements: "Person x", "Person y", "Person z"

What does a test case involve?

→ Example

- **Name** → Inserting an item in an empty ToDo list
- **Tested requirement** (Administration ToDo list, A-102)
- **Precondition** → ToDo list is empty
- **Postcondition** → ToDo list contains added element
- **Description of the test steps** →
  - 1. user selects the option to add an item and enters the required data.
  - 2. system displays updated list
- **Test infrastructure** → Database connection, test environment X available, ...

What does a test case contain?

→ Example

Is this a concrete or a logical test case?

- **Name** → Inserting an item in an empty ToDo list
- **Tested requirement** (Administration ToDo list, A-102)
- **Precondition** → ToDo list is empty
- **Postcondition** → ToDo list contains added element
- **Description of the test steps** →
  - 1. user selects the option to add an item and enters the required data.
  - 2. system displays updated list
- **Test infrastructure** → Database connection, test environment X available, ...

What does a test case contain?

→ Example

What would a **concrete** test case look like?

- **Name** → Inserting an item in the ToDo list
- **Requirement tested** (ToDo list management, A-102)
- **Precondition** → ToDo list is empty
- **Postcondition** → ToDo list contains added element
- **Description of the test steps** →
  - 1. user selects the option to add an item and enters the required data.
  - 2. system displays updated list
- **Test infrastructure** → DB connection, ...

Positive vs. negative test case

- **Positive test case**
  - Correct input
  - Expected correct result
- **Negative test case**
  - Inadmissible entries
  - Expects exception treatment

What does a test case contain?

→ Example

What would a negative test case look like?

- **Name** → Inserting an item in an empty ToDo list
- **Tested requirement** (Administration ToDo list, A-102)
- **Precondition** → ToDo list is empty
- **Postcondition** → ToDo list contains added element
- **Description of the test steps** →
  - 1. user selects the option to add an item and enters the required data.
  - 2. system displays updated list
- **Test infrastructure** → Database connection, test environment X available, ...

Question

Who tests?

everyone.

Everyone is responsible for the quality of a product!

But: Quality cannot be tested into a product.

Question

How good are you at testing **your own results (e.g., code)**?

Psychology of testing

- **Task: Which of the following 4 words does not fit in the row?**

SKYSCRAPER CATHEDRAL TEMPLE PRAYER

Psychology of testing

- **Task: Which of the following 4 words does not fit in the row?**

SKYSCRAPER CATHEDRAL TEMPLE PRAYER

CATHEDRAL PRAYER TEMPLE SKYSCRAPER

Psychology of testing

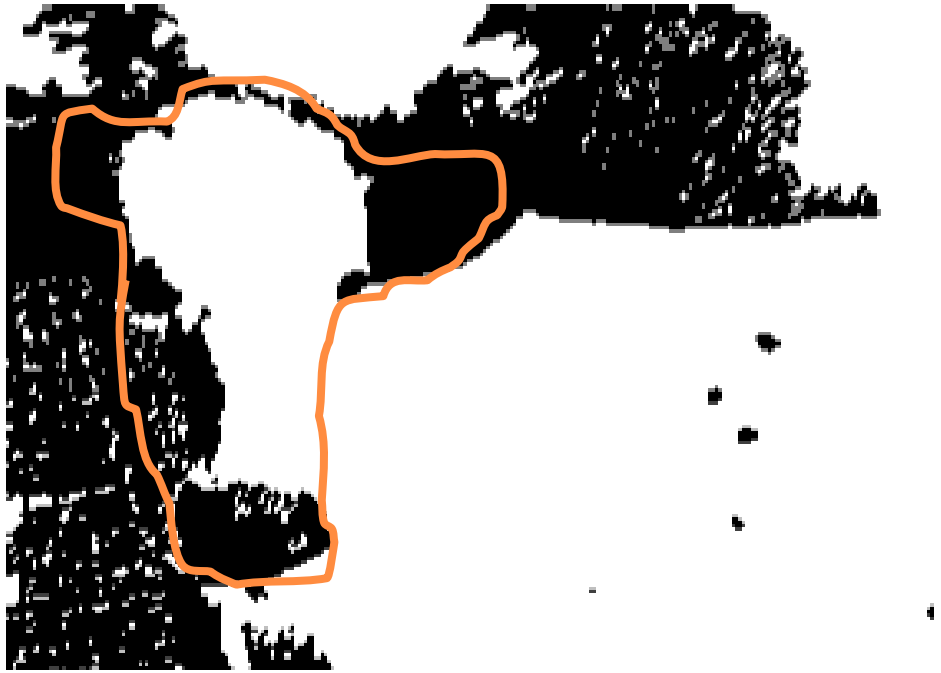
- **Task: Which of the following 4 words does not fit in the row?**
- **Left half:**

  SKYSCRAPER CATHEDRAL TEMPLE PRAYER
- **Right half:**

  CATHEDRAL PRAYER TEMPLE SKYSCRAPER
- In both cases, the word on the right is usually used because the first three fit together
- It is read from the left and a hypothesis is generated in this way.

Psychology of testing

Psychology of testing



Psychology of testing

- Test success is influenced by psychological factors.
  - Blindness to your own mistakes
  - Difficulty in leaving the path already taken
  - Human "weaknesses" in the interpretation of information

Techniques to minimize your own bias

- **Pair *** (Programming, Testing)
- **Test Driven Development** (first test, then write code)
- **Test/review** by another person

Question

When do you test?

always.

Untested code is not code!

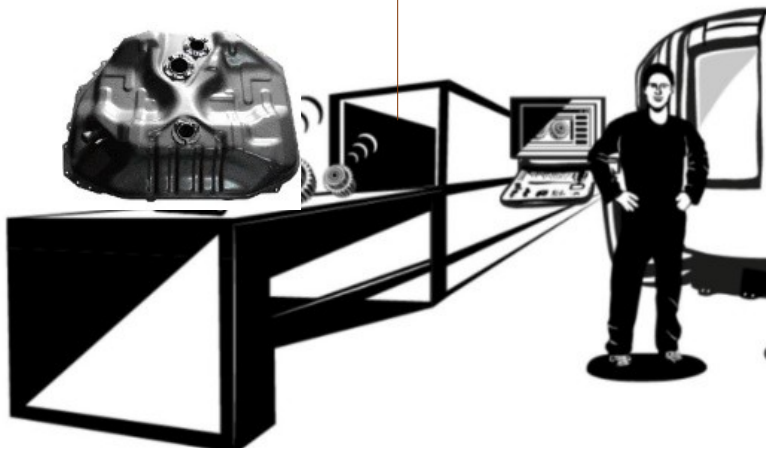
Continuous Testing

The earlier a defect is found, the cheaper it is to fix.

How do others test?

Fuel tank

Scanner

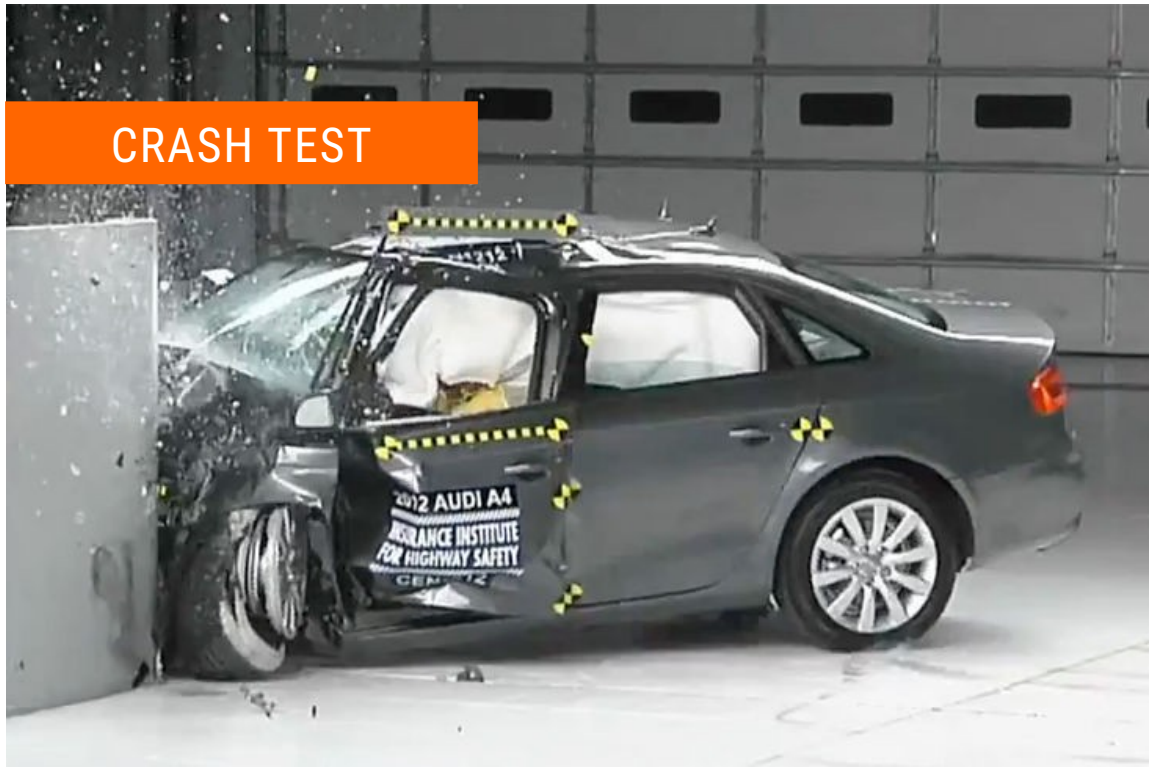


01 Scanner checks for completeness and correctness of the installed parts

How do others test?

02 TANK CRASH-TEST

Tank can withstand certain impact forces.



"Only if the tank passes the test will it be delivered."

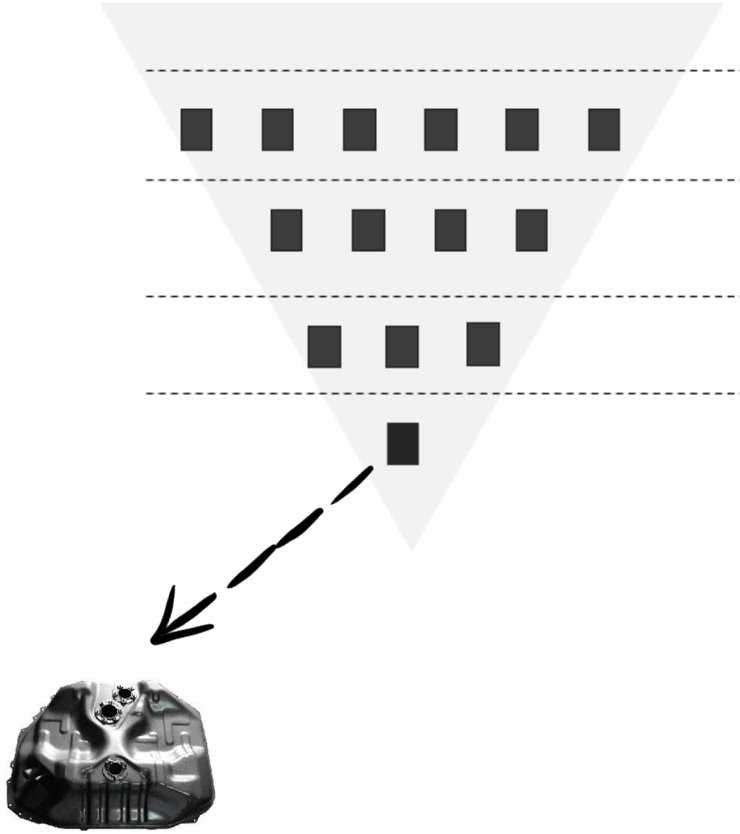
How do others test?

03 TANK TEMPERATURE TEST

Tank is functional down to -40 degrees.

"Only if the tank passes the test will it be delivered."

Test levels

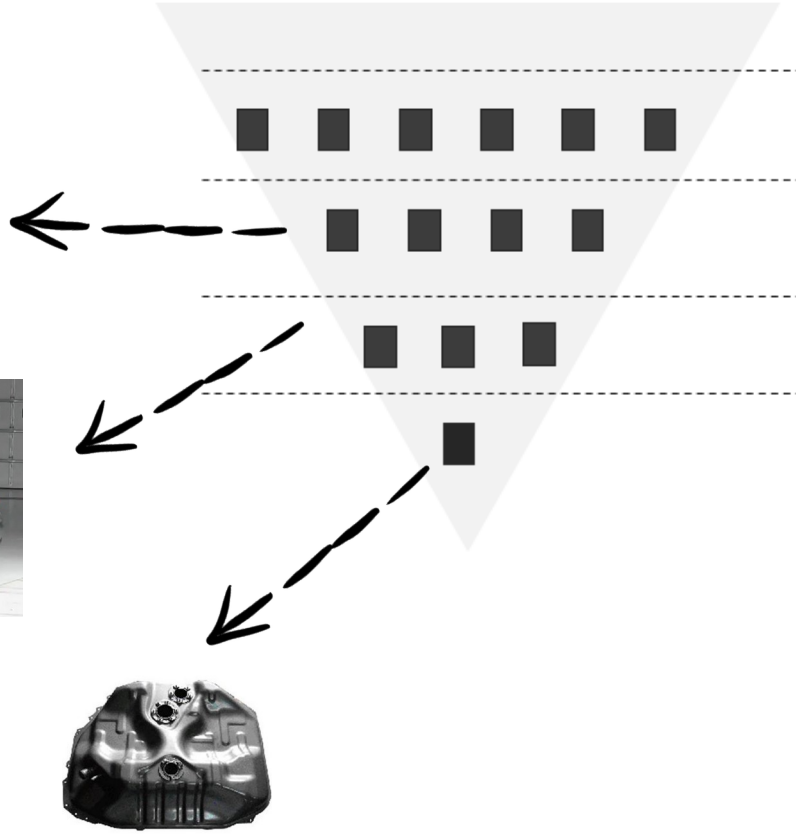


Test level	Test objective
Unit tests, Component tests	Uncover errors in components

Test levels

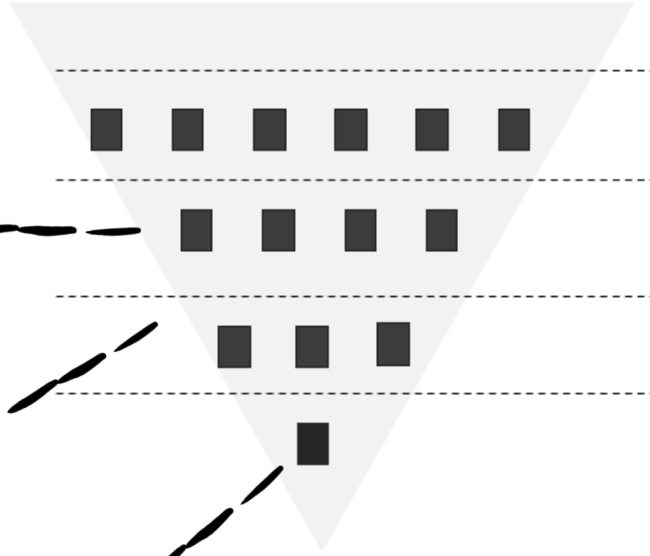
Test level	Test objective
Integration test	Uncover errors in interaction of multiple components
Unit tests, Component tests	Uncover errors in components

Test levels



Test level	Test objective
System test	Uncover errors in end-to-end application workflows
Integration test	Uncover errors in interaction of multiple components
Unit tests, Component tests	Uncover errors in components

Test levels



Test level	Test objective
Acceptance test	Generate confidence in product Check applicability for intended use
System test	Uncover errors in end-to-end application workflows
Integration test	Uncover errors in interaction of multiple components
Unit tests, Component tests	Uncover errors in components

Question

How do you test?

systematically.

And based on experience. But never without a goal and a plan.

Check-Out

Derived without knowledge of the program logic

Photo: Fotolia / Dan Race

Derived with knowledge of the program logic

Source: http://www.operation.de/knie/

Check-Out

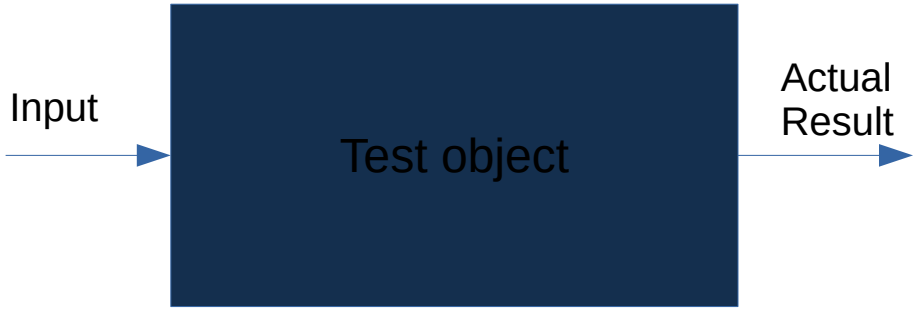
Derived without knowledge of the program logic

Photo: Fotolia / Dan Race

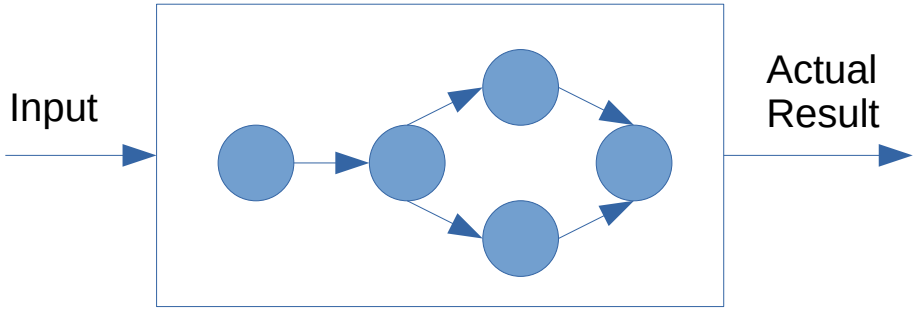
Derived with knowledge of the program logic

Source: http://www.operation.de/knie/

Black-Box Test



White-Box Test



Disciplines in software engineering

Basic topics

Configuration management | Documentation |
Knowledge management | People in the SWE process and digital ethics | Tools

Development

Requirements
- Context analysis
- Requirements Engineering

Design
- Architecture
- Detailed design

Implementation

QualityMgt.

Quality assurance and testing
- Test, inspection, metrics

Processes and procedure models
- Improvement, process model, maturity levels

Evolution

- Roll-Out
- Operation
- Maintenance
- Further development
- Reuse
- Reengineering
- Change management

Management

- Strategy
- Economy
- Team
- Dates
- Risks
- Customer, client/contractor
- Innovation

- What is software testing? → Execution of the program with the objective of finding errors.
- How do you ensure quality? → Through constructive, analytical and organizational measures
- When to test? → As early as possible, always
- Why do you test? → To find errors
- Who tests? → Everyone, all team members/project members are responsible for the quality of the software.

- How much testing do you do? → Software can be delivered with acceptable risks.
- How do you test? Systematically. Black box, white box, equivalence scale analysis (we will deal with this in the next semester ;-)
- What is a failure? → Visible occurrence of a fault.
- What is a defect? → Cause of a failure
- Can you test your own results well? → There are psychological aspects that need to be taken into account.

Test Documentation

Motivation and Challenges

How to ensure Quality?

Software Test

Validation

Reviews

Question

What does test documentation look like?

Test specification

- Specifies the test results.
- Logical test steps are supplemented by specific test data and test steps before or during execution.
- Specific test data is often only logged in the event of an error.

Test case no	Test objective	Precondition	(log.)Test steps	Expected result	Postcondition	Test result
Test case no	Test objective	Precondition	Test steps	Postcondition	Postcondition	fail
Test case no	Test objective	Precondition	Test steps	Postcondition	Postcondition	pass
Test case no	Test objective	Precondition	Test steps	Postcondition	Postcondition	pass
Test case no	Test objective	Precondition	Test steps	Postcondition	Postcondition	pass
Test case no	Test objective	Precondition	Test steps	Postcondition	Postcondition	pass
Test case no	Test objective	Precondition	Test steps	Postcondition	Postcondition	pass

Error logging

→ Spillner, Linz

- Error log contains
  - The reference to the test case that led to the error
  - All information required to **reproduce** the test case**, in particular the specific test** data
  - Classification of the error

Class	Description
1	System crash with potential data loss; test object can not be put into use
2	Critical function exhibits errors; requirement not considered or implemented wrong; test object can only be used with significant restrictions
3	Functional deviation („normal error“); requirement wrong or only implemented partially; test object can be used with restrictions
4	Small deviation; Test object can be used without restriction
5	Cosmetic error; Test object can be used without restriction

Test protocol

→ Spillner, Linz

- Summary of the executed test cases: What was tested?
  - Typically in tabular form and in natural language.
- Summary of the test results: What is the test result?
  - Typically in tabular form (list of errors found and categorization) and in natural language.
  - At system test level: Basis for discussion with the customer in preparation for acceptance.
  - Discussion of the extent to which defects known prior to acceptance prevent acceptance.

Motivation and Challenges

How to ensure Quality?

Software Test

Validation

Reviews

Verification vs. Validation

Testing "against requirements"

Check against "application context"

Verification vs. Validation

- Requirements:

Made of plastic, blue

Container that can be filled with water

Capacity 10 L

With handle

- Fulfilled?

Verification

Verification vs. Validation

- Requirements:

Made of plastic, blue

Container that can be filled with water

Capacity 10 L

With handle

- Useful?

Validation

Validation using the example of usability: Not everything that is functionally ok is also easy to use.

10 heuristics for the design of user interfaces

Jakob Nielsen

Principle 1: Visibility of system status - **FEEDBACK**

The system should always keep users informed of the current status by providing appropriate feedback within a reasonable period of time.

Example

10 heuristics for the design of user interfaces

Jakob Nielsen

Principle 1: Visibility of system status - **FEEDBACK**

The system should always keep users informed of the current status by providing appropriate feedback within a reasonable period of time.

Example

Sound during collection

Haptic feedback

Animation after visible action

Visual feedback when level is completed

10 heuristics for the design of user interfaces

Jakob Nielsen

Principle 2: Correspondence between system and real world - **METAPHOR**

The system should speak the language of the user, using words, phrases and concepts that are familiar to the user, rather than system-oriented terms. It should follow the conventions of the real world so that the information appears in a natural and logical order.

10 heuristics for the design of user interfaces

Jakob Nielsen

Principle 2: Correspondence between system and real world - **METAPHOR**

The system should speak the language of the user, using words, phrases and concepts that are familiar to the user, rather than system-oriented terms. It should follow the conventions of the real world so that the information appears in a natural and logical order.

Language: Backpack is full, go to the trash can. Vs. mistake, go to the garbage can

Language: Cryptic error code.

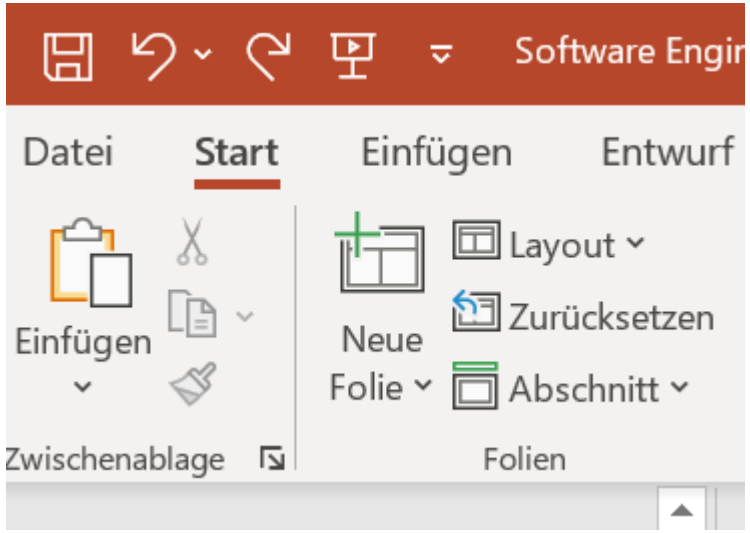
Language: Clear visual language

10 heuristics for the design of user interfaces

Jakob Nielsen

Principle 3: User control and freedom - **NAVIGATION**

Users often accidentally select system functions and need a clearly marked "emergency exit" to leave the unwanted state without having to go through a lengthy dialog. Supports **undo** and **redo** and clear navigation.

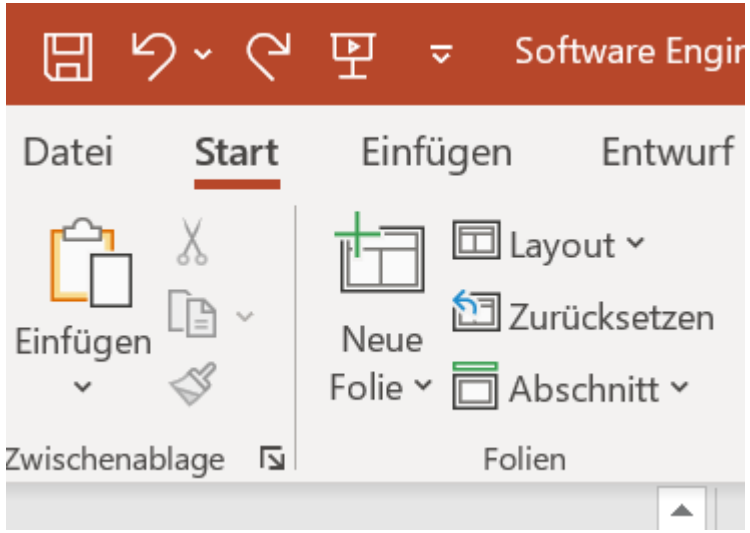


10 heuristics for the design of user interfaces

Jakob Nielsen

Principle 3: User control and freedom - **NAVIGATION**

Users often accidentally select system functions and need a clearly marked "emergency exit" to leave the unwanted state without having to go through a lengthy dialog. Supports **undo** and **redo** and clear navigation.



Undoing steps in the construction process

Pause: Pause game

Break: Abort

Change character

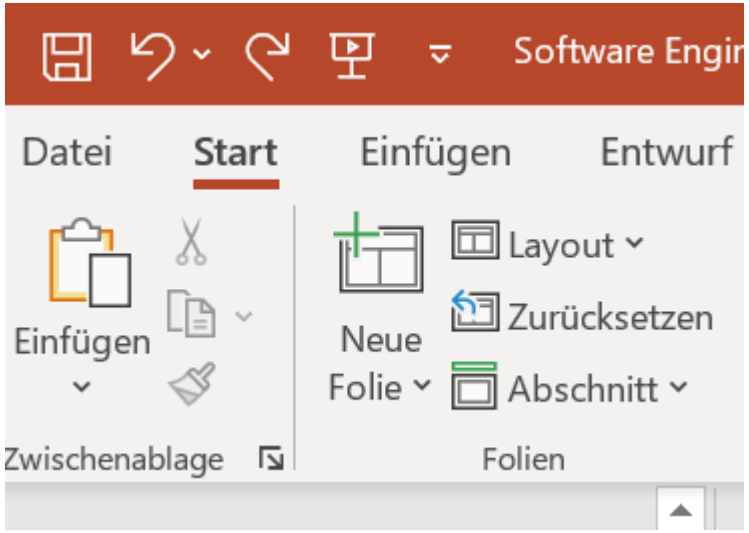
Pause: Different control options WASD, voice

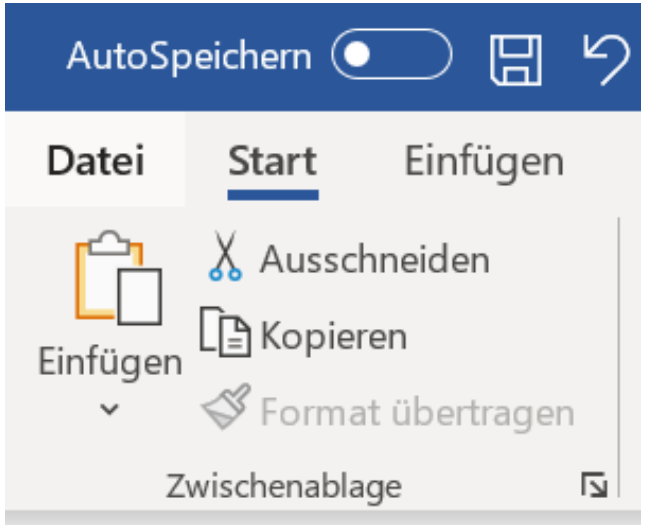
10 heuristics for the design of user interfaces

Jakob Nielsen

Principle 4: Consistency and standards - **CONSISTENCY**

Users should not have to ask themselves whether different words, situations or actions mean the same thing. Follow the conventions of the platform.



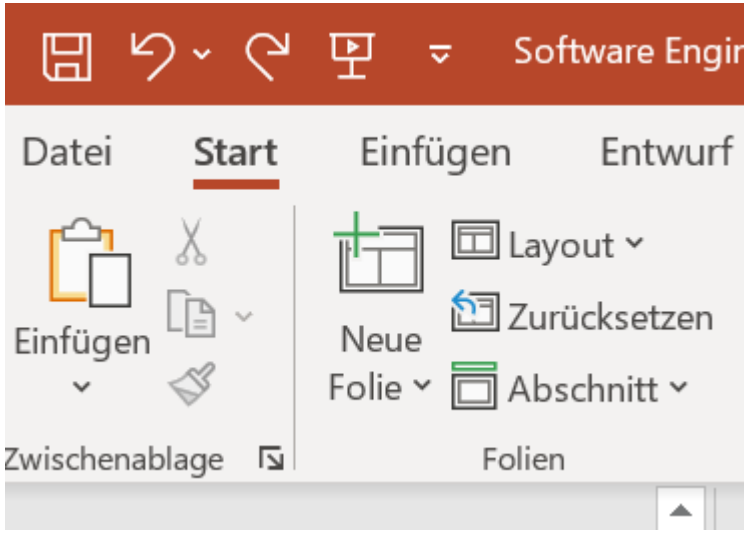


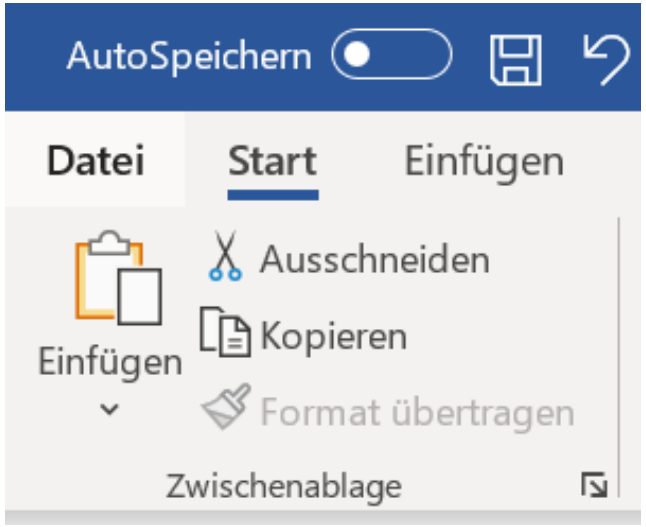
10 heuristics for the design of user interfaces

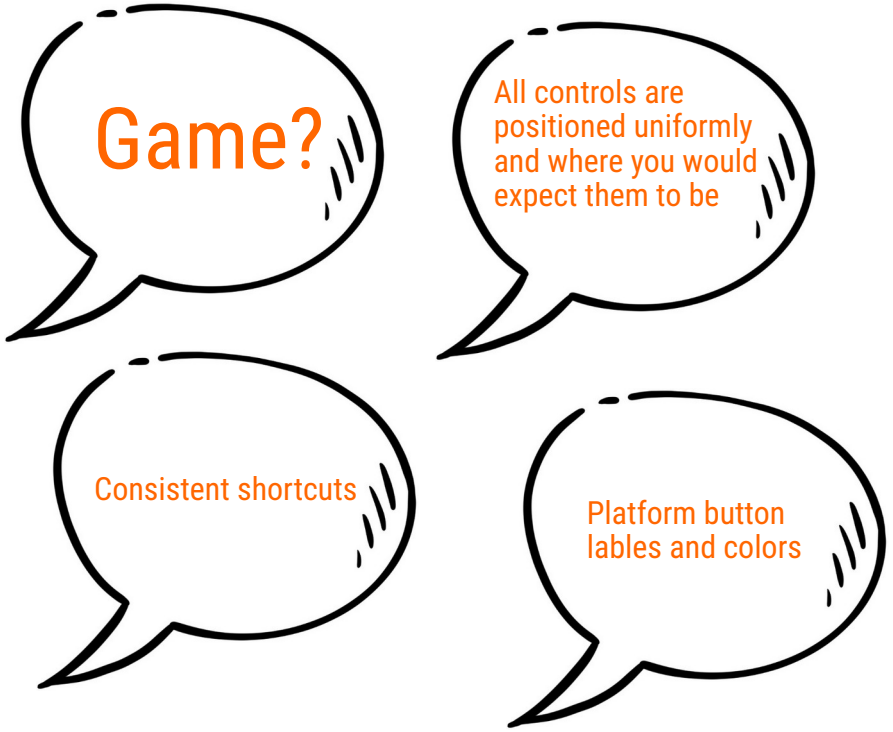
Jakob Nielsen

Principle 4: Consistency and standards - **CONSISTENCY**

Users should not have to ask themselves whether different words, situations or actions mean the same thing. Follow the conventions of the platform.





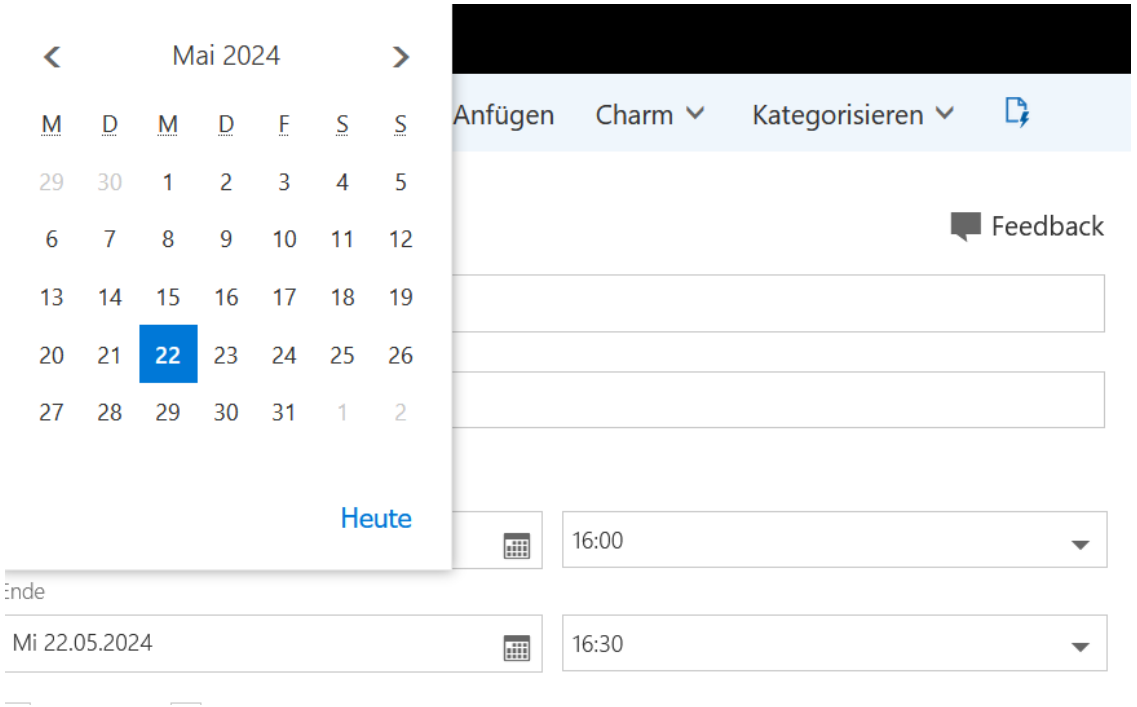


10 heuristics for the design of user interfaces

Jakob Nielsen

Principle 5: Error prevention - **PREVENTION**

Even better than good error messages is a careful design that prevents a problem from occurring in the first place. Either remove error-prone conditions or check them and offer users a confirmation option before they perform the action.

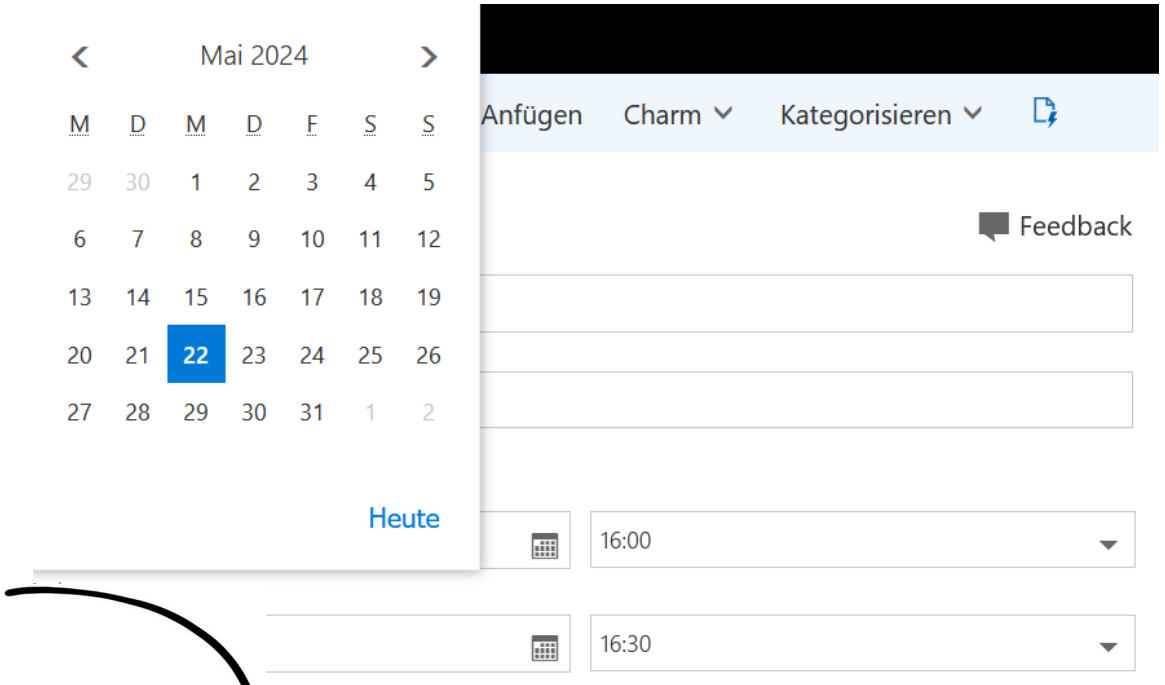


10 heuristics for the design of user interfaces

Jakob Nielsen

Principle 5: Error prevention - **PREVENTION**

Even better than good error messages is a careful design that prevents a problem from occurring in the first place. Either remove error-prone conditions or check them and offer users a confirmation option before they perform the action.



Lock levels that are not yet unlocked.

Placement of objects only in potentially permitted locations possible

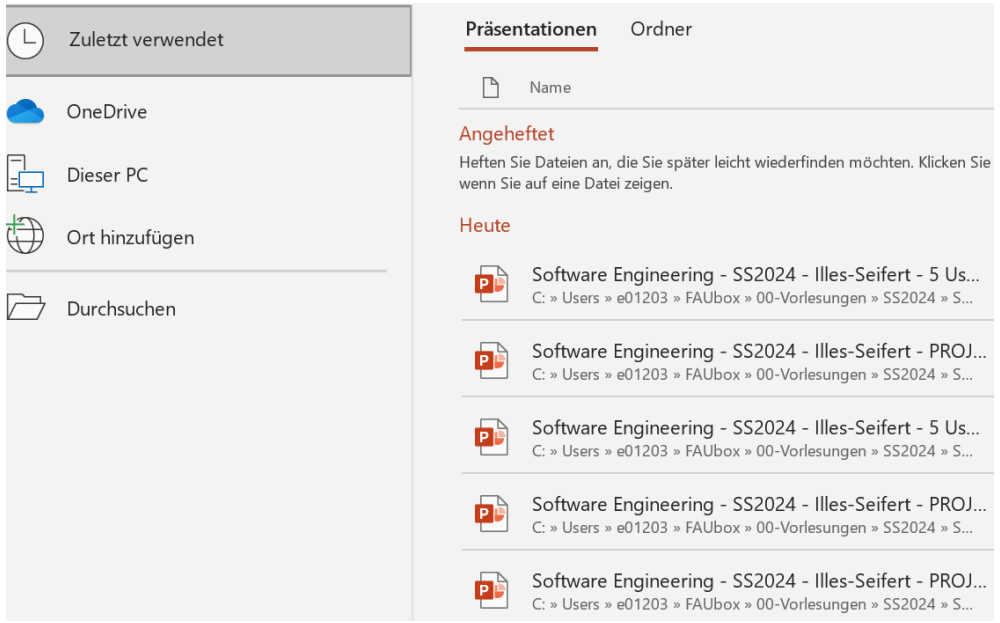
10 heuristics for the design of user interfaces

Jakob Nielsen

Principle 6: Recognize instead of remember - **MEMORY**

Minimize the user's memory load by making objects, actions and options visible.

The user should not have to remember information from one part of the dialog to another. The instructions for using the system should be visible or easily retrievable whenever appropriate.



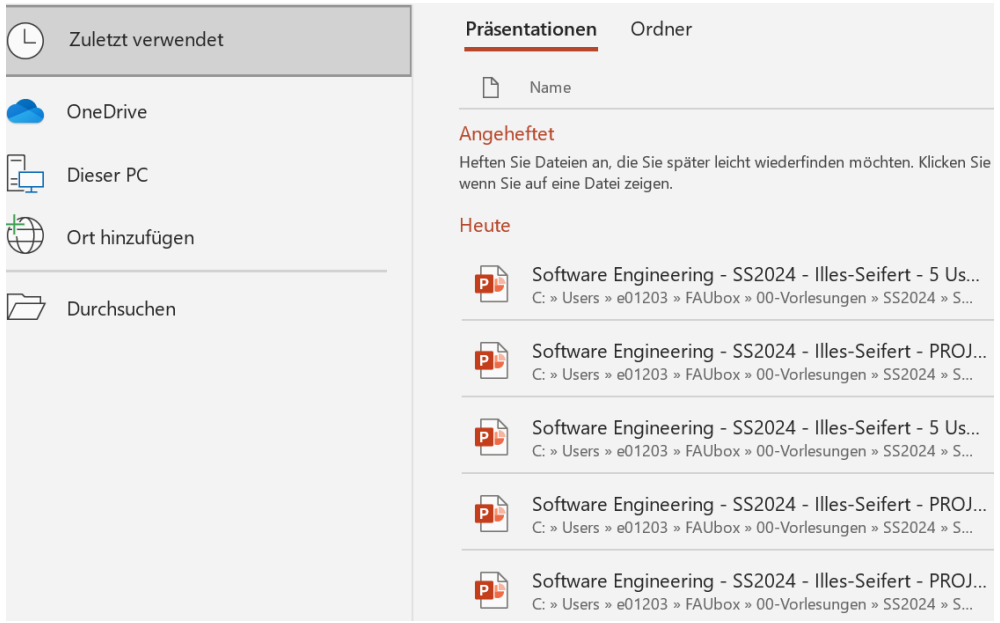
10 heuristics for the design of user interfaces

Jakob Nielsen

Principle 6: Recognize instead of remember - **MEMORY**

Minimize the user's memory load by making objects, actions and options visible.

The user should not have to remember information from one part of the dialog to another. The instructions for using the system should be visible or easily retrievable whenever appropriate.



Display of input that must be pressed to trigger a specific action

Display how many levels you have completed.

Various memory statuses are displayed.

10 heuristics for the design of user interfaces

Jakob Nielsen

Principle 7: Flexibility and efficiency of use - **EFFICIENCY**

Accelerators that the inexperienced user does not see can often speed up the interaction for the experienced user, making the system suitable for both inexperienced and experienced users. Allow users to customize **frequent actions**.

Visual Studio Code

Keyboard shortcuts for Windows

General

Ctrl+Shift+P, F1	Show Command Palette
Ctrl+P	Quick Open, Go to File...
Ctrl+Shift+N	New window/instance
Ctrl+Shift+W	Close window/instance
Ctrl+,	User Settings
Ctrl+K Ctrl+S	Keyboard Shortcuts

Basic editing

Ctrl+X	Cut line (empty selection)
Ctrl+C	Copy line (empty selection)
Alt+ ↑ / ↓	Move line up/down
Shift+Alt + ↓ / ↑	Copy line up/down
Ctrl+Shift+K	Delete line
Ctrl+Enter	Insert line below
Ctrl+Shift+Enter	Insert line above
Ctrl+Shift+\	Jump to matching bracket
Ctrl+] / [	Indent/outdent line
Home / End	Go to beginning/end of line
Ctrl+Home	Go to beginning of file
Ctrl+End	Go to end of file
Ctrl+↑ / ↓	Scroll line up/down
Alt+PgUp / PgDn	Scroll page up/down
Ctrl+Shift+[	Fold (collapse) region
Ctrl+Shift+]	Unfold (uncollapse) region
Ctrl+K Ctrl+[	Fold (collapse) all subregions
Ctrl+K Ctrl+]	Unfold (uncollapse) all subregions
Ctrl+K Ctrl+0	Fold (collapse) all regions

Ctrl+M	Toggle Tab moves focus

Search and replace

Ctrl+F	Find
Ctrl+H	Replace
F3 / Shift+F3	Find next/previous
Alt+Enter	Select all occurences of Find match
Ctrl+D	Add selection to next Find match
Ctrl+K Ctrl+D	Move last selection to next Find match
Alt+C / R / W	Toggle case-sensitive / regex / whole word

Multi-cursor and selection

Alt+Click	Insert cursor
Ctrl+Alt+ ↑ / ↓	Insert cursor above / below
Ctrl+U	Undo last cursor operation
Shift+Alt+I	Insert cursor at end of each line selected
Ctrl+L	Select current line
Ctrl+Shift+L	Select all occurrences of current selection
Ctrl+F2	Select all occurrences of current word
Shift+Alt+→	Expand selection
Shift+Alt+←	Shrink selection
Shift+Alt + (drag mouse)	Column (box) selection
Ctrl+Shift+Alt + (arrow key)	Column (box) selection
Ctrl+Shift+Alt +PgUp/PgDn	Column (box) selection page up/down

Rich languages editing

Ctrl+Space	Trigger suggestion
Ctrl+Shift+Space	Trigger parameter hints
Shift+Alt+F	Format document
Ctrl+K Ctrl+F	Format selection

File management

Ctrl+N	New File
Ctrl+O	Open File...
Ctrl+S	Save
Ctrl+Shift+S	Save As...
Ctrl+K S	Save All
Ctrl+F4	Close
Ctrl+K Ctrl+W	Close All
Ctrl+Shift+T	Reopen closed editor
Ctrl+K Enter	Keep preview mode editor open
Ctrl+Tab	Open next
Ctrl+Shift+Tab	Open previous
Ctrl+K P	Copy path of active file
Ctrl+K R	Reveal active file in Explorer
Ctrl+K O	Show active file in new window/instance

Display

F11	Toggle full screen
Shift+Alt+0	Toggle editor layout (horizontal/vertical)
Ctrl+ = / -	Zoom in/out
Ctrl+B	Toggle Sidebar visibility
Ctrl+Shift+E	Show Explorer / Toggle focus
Ctrl+Shift+F	Show Search
Ctrl+Shift+G	Show Source Control
Ctrl+Shift+D	Show Debug
Ctrl+Shift+X	Show Extensions
Ctrl+Shift+H	Replace in files
Ctrl+Shift+J	Toggle Search details
Ctrl+Shift+U	Show Output panel
Ctrl+Shift+V	Open Markdown preview
Ctrl+K V	Open Markdown preview to the side
Ctrl+K Z	Zen Mode (Esc Esc to exit)

Debug

10 heuristics for the design of user interfaces

Jakob Nielsen

Principle 7: Flexibility and efficiency of use - **EFFICIENCY**

Accelerators that the inexperienced user does not see can often speed up the interaction for the experienced user, making the system suitable for both inexperienced and experienced users. Allow users to customize **frequent actions**.

Visual Studio Code

Keyboard shortcuts for Windows

General

Ctrl+Shift+P, F1	Show Command Palette
Ctrl+P	Quick Open, Go to File...
Ctrl+Shift+N	New window/instance
Ctrl+Shift+W	Close window/instance
Ctrl+,	User Settings
Ctrl+K Ctrl+S	Keyboard Shortcuts

Basic editing

Ctrl+X	Cut line (empty selection)
Ctrl+C	Copy line (empty selection)
Alt+ ↑ / ↓	Move line up/down
Shift+Alt + ↓ / ↑	Copy line up/down
Ctrl+Shift+K	Delete line
Ctrl+Enter	Insert line below
Ctrl+Shift+Enter	Insert line above
Ctrl+Shift+\	Jump to matching bracket
Ctrl+] / [	Indent/outdent line
Home / End	Go to beginning/end of line
Ctrl+Home	Go to beginning of file
Ctrl+End	Go to end of file
Ctrl+↑ / ↓	Scroll line up/down
Alt+PgUp / PgDn	Scroll page up/down
Ctrl+Shift+[	Fold (collapse) region
Ctrl+Shift+]	Unfold (uncollapse) region
Ctrl+K Ctrl+[	Fold (collapse) all subregions
Ctrl+K Ctrl+]	Unfold (uncollapse) all subregions
Ctrl+K Ctrl+0	Fold (collapse) all regions

Ctrl+M	Toggle Tab moves focus

Search and replace

Ctrl+F	Find
Ctrl+H	Replace
F3 / Shift+F3	Find next/previous
Alt+Enter	Select all occurences of Find match
Ctrl+D	Add selection to next Find match
Ctrl+K Ctrl+D	Move last selection to next Find match
Alt+C / R / W	Toggle case-sensitive / regex / whole word

Multi-cursor and selection

Alt+Click	Insert cursor
Ctrl+Alt+ ↑ / ↓	Insert cursor above / below
Ctrl+U	Undo last cursor operation
Shift+Alt+I	Insert cursor at end of each line selected
Ctrl+L	Select current line
Ctrl+Shift+L	Select all occurrences of current selection
Ctrl+F2	Select all occurrences of current word
Shift+Alt+→	Expand selection
Shift+Alt+←	Shrink selection
Shift+Alt + (drag mouse)	Column (box) selection
Ctrl+Shift+Alt + (arrow key)	Column (box) selection
Ctrl+Shift+Alt +PgUp/PgDn	Column (box) selection page up/down

Rich languages editing

Ctrl+Space	Trigger suggestion
Ctrl+Shift+Space	Trigger parameter hints
Shift+Alt+F	Format document
Ctrl+K Ctrl+F	Format selection

File management

Ctrl+N	New File
Ctrl+O	Open File...
Ctrl+S	Save
Ctrl+Shift+S	Save As...
Ctrl+K S	Save All
Ctrl+F4	Close
Ctrl+K Ctrl+W	Close All
Ctrl+Shift+T	Reopen closed editor
Ctrl+K Enter	Keep preview mode editor open
Ctrl+Tab	Open next
Ctrl+Shift+Tab	Open previous
Ctrl+K P	Copy path of active file
Ctrl+K R	Reveal active file in Explorer
Ctrl+K O	Show active file in new window/instance

Display

F11	Toggle full screen
Shift+Alt+0	Toggle editor layout (horizontal/vertical)
Ctrl+ = / -	Zoom in/out
Ctrl+B	Toggle Sidebar visibility
Ctrl+Shift+E	Show Explorer / Toggle focus
Ctrl+Shift+F	Show Search
Ctrl+Shift+G	Show Source Control
Ctrl+Shift+D	Show Debug
Ctrl+Shift+X	Show Extensions
Ctrl+Shift+H	Replace in files
Ctrl+Shift+J	Toggle Search details
Ctrl+Shift+U	Show Output panel
Ctrl+Shift+V	Open Markdown preview
Ctrl+K V	Open Markdown preview to the side
Ctrl+K Z	Zen Mode (Esc Esc to exit)

Debug

Offer shortcuts for frequently used functions in addition to buttons.

E.g., RTS/MOBAs

Several options for navigation, arrow keys, etc.

10 heuristics for the design of user interfaces

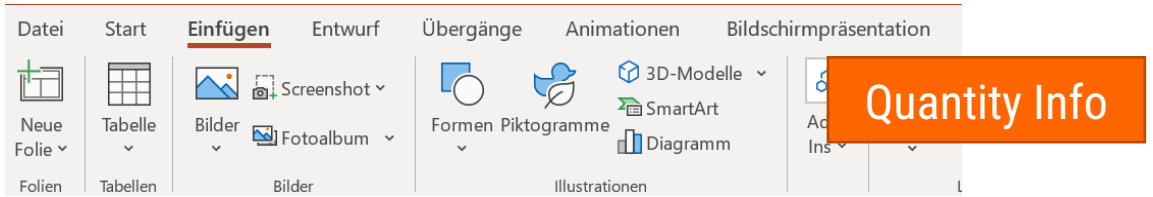
Jakob Nielsen

Principle 8: Aesthetic and minimalist design - **DESIGN**

Dialogs should not contain information that is irrelevant or rarely needed.

Each additional information unit in a dialog competes with the relevant information units and reduces their relative visibility.

The principles of **contrast**, **repetition**, **alignment** and **proximity** should be observed in the visual design.



Alignment

10 heuristics for the design of user interfaces

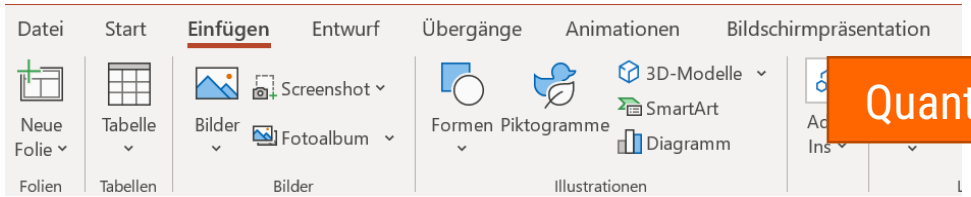
Jakob Nielsen

Principle 8: Aesthetic and minimalist design - **DESIGN**

Dialogs should not contain information that is irrelevant or rarely needed.

Each additional information unit in a dialog competes with the relevant information units and reduces their relative visibility.

The principles of **contrast**, **repetition**, **alignment** and **proximity** should be observed in the visual design.



Quantity Info

Visual indicators that are consistent

Buttons that belong together are grouped together

Common color "codes", red rather for demolition

Maintain "style"

10 heuristics for the design of user interfaces

Jakob Nielsen

Principle 9: Support users in detecting, diagnosing and correcting faults - **RECOVERY**

Error messages should be formulated in simple language (no codes), state the problem precisely and constructively suggest a solution.

10 heuristics for the design of user interfaces

Jakob Nielsen

Principle 9: Support users in detecting, diagnosing and correcting faults - **RECOVERY**

Error messages should be formulated in simple language (no codes), state the problem precisely and constructively suggest a solution.

Error code yes/no?

→ Both are best

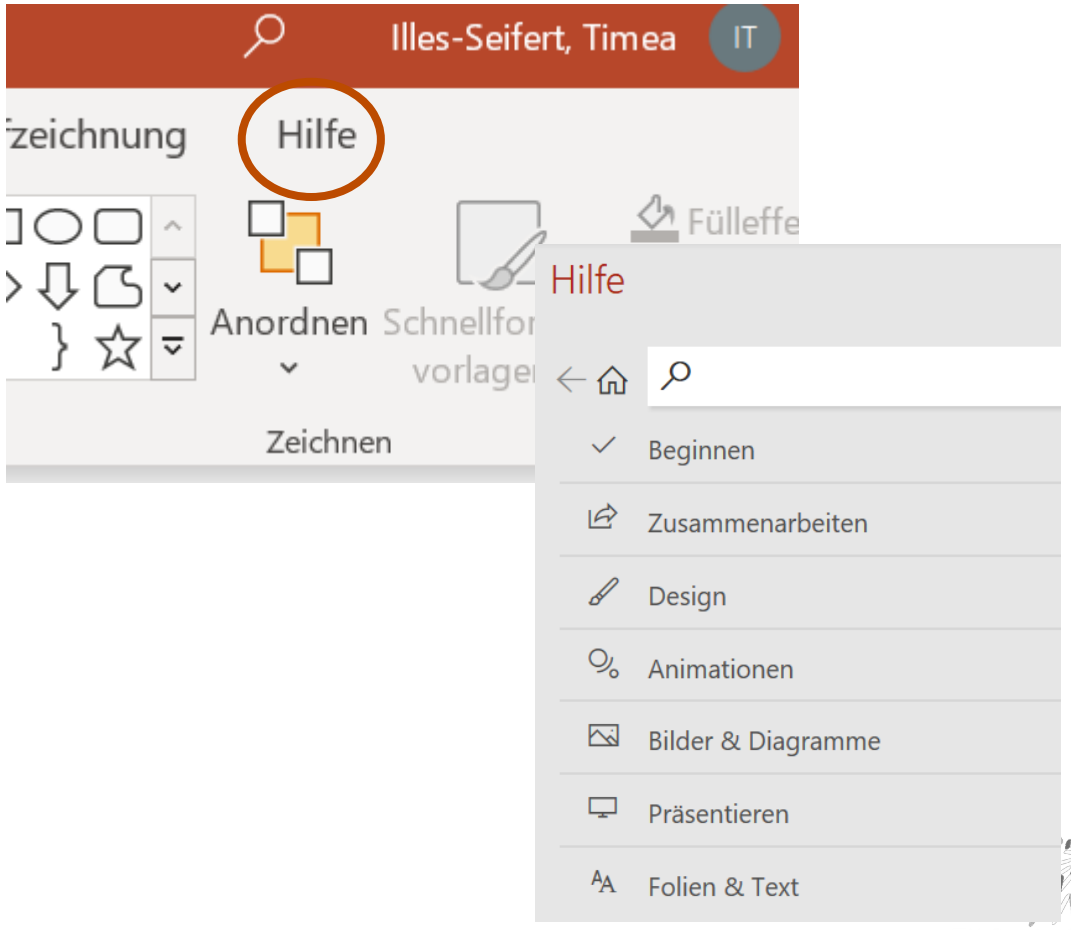
If Character gets stuck at a corner, the user gets help, e.g. press x to get out.

10 heuristics for the design of user interfaces

Jakob Nielsen

Principle 10: Help and documentation - **HELP**

Although it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Such information should **be easy to find**, focus on the user's task, list specific steps and not be too extensive.

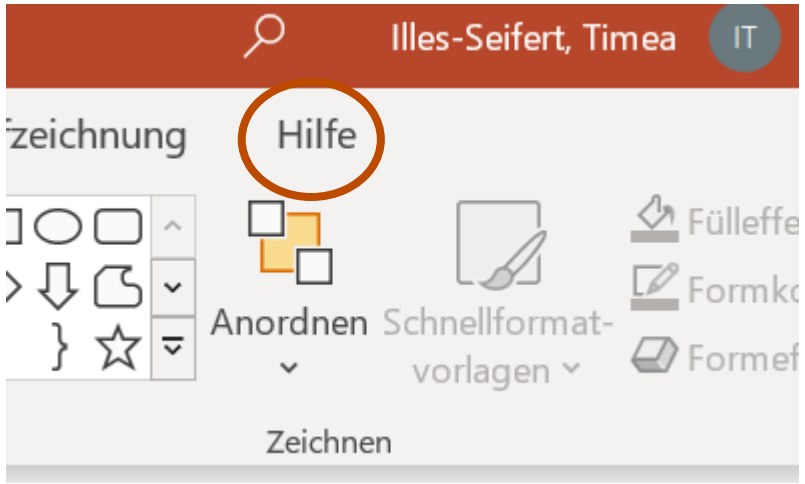


10 heuristics for the design of user interfaces

Jakob Nielsen

Principle 10: Help and documentation - **HELP**

Although it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Such information should **be easy to find**, focus on the user's task, list specific steps and not be too extensive.



Self-explanatory

Community

Tutorials

- What is software testing? → Execution of the program with the objective of finding errors.
- How do you ensure quality? → Through constructive, analytical and organizational measures
- When to test? → As early as possible, always
- Why do you test? → To find errors
- Who tests? → Everyone, all team members/project members are responsible for the quality of the software.

- How much testing do you do? → Software can be delivered with acceptable risks.
- How do you test? Systematically. Black box, white box, equivalence scale analysis (we will deal with this in the next semester ;-)
- What is a failure? → Visible occurrence of a fault.
- What is a defect? → Cause of a failure
- Can you test your own results well? → There are psychological aspects that need to be taken into account.

Verification vs. Validation

- Requirements:

Made of plastic, blue

Container that can be filled with water

Capacity 10 L

With handle

- Fulfilled?

Verification

Verification vs. Validation

- Requirements:

Made of plastic, blue

Container that can be filled with water

Capacity 10 L

With handle

- Useful?

Validation

Motivation and Challenges

How to ensure Quality?

Software Test

Validation

Reviews

Question

What is a review?

Review

→ Definitions

Review *A type A type of static testing in which the quality of a work product or process is evaluated by individuals.*

ISTQB Certified Tester Glossary

Question

When is a review useful?

always.

As soon as you say someone should look over my (interim) result.

Before milestones.

Before customer appointments.

Before handover to the next phase.

Before you accept an (interim) result.

. . .

Question

Who carries out reviews?

everyone.

Everyone is responsible for the quality of a product!

But: Quality cannot be tested!

Question

How good are you at reviewing your own results, e.g. requirements?

Question

When do you conduct reviews?

always.

As soon as you say someone should look over my (interim) result.

Before milestones.

Before customer appointments.

Before handover to the next phase.

Before you accept an (interim) result.

. . .

The earlier a defect is found, the cheaper it is to rectify.

Error occurrence and troubleshooting in the product life cycle

Model from product development transferable to SE

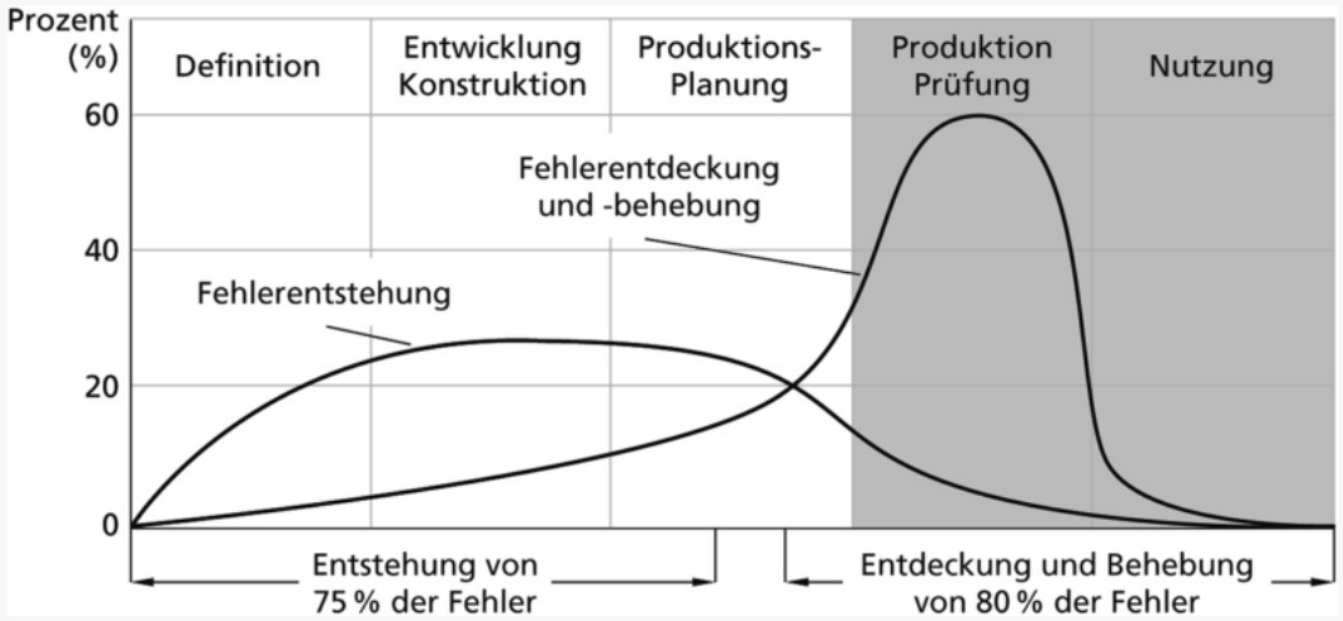


Abb. 3.16
Fehlerentstehung und Fehlerbehebung im Produktlebenslauf.
(In Anlehnung an Pfeifer)

"Studies have shown that the majority of errors are caused during development. In contrast, errors are often only noticed very late during testing or even during use. However, according to the so-called "Rule of Ten", they cost many times more to rectify."

Source: Innovation and product development, Alexander Schloske, published in Fabrikbetriebslehre 1, Springer Verlag, 2020

Rule of ten of troubleshooting costs

Model from product development transferable to SE

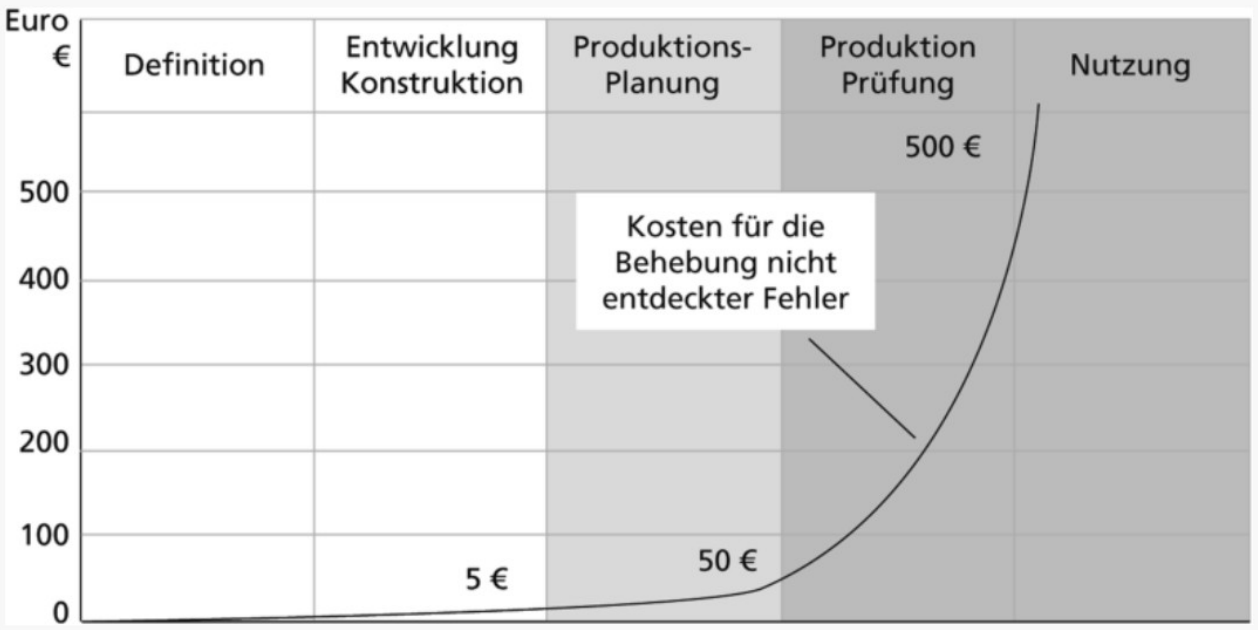


Abb. 3.17
Rule of Ten der Fehlerbehebungskosten.
(In Anlehnung an Pfeifer)

"According to the Rule of Ten, the costs of rectifying faults increase by a factor of ten from one development stage to the next. In some cases, the recall costs can reach 1000 times the manufacturing costs. Current recall campaigns in the automotive industry show that the costs for a recall can add up to €1.4 billion."

Source: Innovation and product development, Alexander Schloske, published in Fabrikbetriebslehre 1, Springer Verlag, 2020

Question

How do you carry out reviews?

systematically.

Checklist-based: A checklist created in advance is used as a basis.

Role-based: Review from a specific perspective, e.g. customer, user, PO, developer

Perspective-based: Certain aspects defined in advance are reviewed, e.g. consistency or compliance, etc..

ad-hoc.

Formal vs. informal.

A combination of both is effective.

Question

What does review documentation look like?

Review documentation

- **Checklists** for the checklist/perspective-based review.
- If applicable, process description for formal review types.
- **Review protocol** contains all findings of a review as well as a summarized evaluation.

Rules for formulating review findings

- Formulate the findings **neutrally and objectively.**
- Classify the errors according to their severity.
- Please note that a review refers to an **artifact** and **not to a person!**
- Provide an opportunity for queries!

Literature

- **[Ludewig 2013]** Ludewig, Lichter: Software Engineering. Grundlagen, Menschen, Prozesse, Techniken, dpunkt.verlag.
- **[Paech 2021]** Barbara Paech: Lecture Software Engineering, Uni-Heidelberg.
- **[Balzert, 2009]**HHelmut Balzert: Lehrbuch der Softwaretechnik, Basiskonzepte und Requirements Engineering, 3. Auflage, 2009, Springer.
- **[Sommerville 2018]** Ian Sommerville: Software Engineering, Pearson.
- **[Rupp, 2021]** Rupp & die SOPHISTen: Requirements Engineering und – Management, Hanser Verlag, 7. Auflage, Hanser Verlag, 2021.

Thank you for your attention!

Software Engineering

Prof. Dr. J. v. Kistowski

TH Aschaffenburg
university of applied sciences