UXQCC

Release Version 1.0 EN

July 2026

UXQCC - GenAI Certified Professional in Software Usability with GenAI Foundation Level

Designing, implementing, and reviewing usability, user experience, and accessibility with GenAI



Imprint, Copyright, and Terms of Use

© 2026 Robert Pucher

All contents of this work, in particular texts and graphics, are protected by copyright. The author and sole rights holder is Robert Pucher. The publisher is UXQCC; UXQCC is entitled to use, maintain, edit, translate, and license this work to accredited training providers and examination providers (e.g. GASQ) as part of the certification programme. Accredited training and examination providers are entitled to use and publish this work as part of their activities, for example on their website. Reproduction or duplication of this work, in whole or in part, without the rights holder's consent is otherwise prohibited. In case of infringement, the rights holder reserves the right to take civil and criminal action.

Document information

Title

Syllabus: UXQCC Certified Professional in Software Usability with GenAI – Foundation Level

Subtitle

Designing, implementing, and reviewing usability, user experience, and accessibility with GenAI

Short form

Syllabus: UXQCC GenAI - FL

Publisher

UXQCC – User Experience Quality Certification Center

International Board for Usability and User Experience Qualification

Author

FH-Prof. Dipl.-Ing. Dr. techn. Robert Pucher

Public syllabus edition and provider edition

This syllabus is the public edition of the curriculum for the UXQCC Certified Professional in Software Usability with GenAI – Foundation Level. It describes the professional framework, the business outcomes, learning objectives, key terms, exam scope, and fundamental contents of the certification.

Accredited training providers additionally receive a more detailed provider edition. It contains in-depth explanations, didactic guidance, more extensive professional elaborations, examples, delineations, and advice on training design. It serves the preparation, delivery, and quality assurance of official trainings.

The public syllabus edition alone is therefore not to be understood as complete training material. It defines which contents and learning objectives are authoritative for the certification, but it does not replace the official training materials of accredited training providers.

Acknowledgments

My special thanks go to the many people, in particular Dr. Verena Seibert-Giller, for their valuable professional contributions, critical feedback, and constructive suggestions during the development of this syllabus.

Their diverse perspectives from practice, teaching, software development, quality assurance, usability, and user experience have contributed substantially to sharpening the contents professionally, aligning them with practice, and making them accessible to the target audiences.

I also thank all other people who contributed to the further development of this document through discussions, suggestions, reviews, or professional input. Their contributions have helped to shape this syllabus not only as a professional framework, but also as a practical foundation for training, certification, and application in software development teams.

Version history

Version	Date	Remark
1.0	July 14, 2026	Internally reviewed document
1.0	July 15, 2026	Translated from the German edition 1.0 (Build 1.012)

Current version 1.0 (Build 1.013)

Disclaimer and terms of use

This document provides general information on improving software usability, user experience, and interaction quality with the support of generative artificial intelligence (GenAI), taking into account available ergonomic norms and quality standards, in particular ISO 9241 and ISO/IEC 25010.

The authors and UXQCC accept no liability for the completeness, correctness, or currency of the contents. This document does not replace individual UX consulting or any binding conformity assessment, for example with regard to legal requirements or the European Accessibility Act.

Since GenAI systems produce hallucinations and cannot independently capture the specific domain context of a product, this syllabus also serves as a guide for the critical assessment of AI output. Responsibility for software quality, software usability, accessibility, and legal conformity always remains with the responsible persons and organizations.

For binding implementations, the applicable legal provisions, official standards, in particular WCAG 2.2, and, where necessary, expert advice must be consulted.

Note on the descriptive use of Scrum

Scrum is used in this syllabus purely descriptively as a widely used agile process framework. The syllabus is independent and has no affiliation with Scrum.org, Scrum Alliance, or Scrum Inc. It is neither published, reviewed, endorsed, authorized, nor sponsored by these organizations.

Note on the use of Artificial Intelligence (AI)

This syllabus was partly developed and editorially reviewed with the support of AI tools.

Contents

Imprint, Copyright, and Terms of Use 1
Contents 4
0 Preamble 7
0.1 Purpose of this syllabus 7
0.2 Target audience for this syllabus 8
0.3Exam-relevant learning objectives and cognitive knowledge levels 11
0.4 The exam 12
0.5 Accreditation 12
0.6 Level of detail 13
0.7 Structure of this syllabus 14
0.8 Accompanying prompt library and practice examples 15
0.9 Business Outcomes (BOs) 16
0.10 Abbreviations and acronyms used 21
1 Usability with AI: Why This Topic Matters 27
1.1 Core idea of this chapter 27
1.2 Contribution to the business outcomes 27
1.3 Learning objectives 27
1.4 Key terms 28
1.5 Software usability, UX, and interaction capability as parts of software quality 28
1.6 Usability in development processes 29
1.7 GenAI as support along agile activities 30
1.8 Familiar patterns, unfamiliar context, and AI output 30
1.9 Responsible AI use in the usability context 31
2 Building Foundational Usability Knowledge with AI 32
2.1 Core idea of this chapter 32
2.2 Contribution to the business outcomes 32
2.3 Learning objectives 32
2.4 Key terms 33
2.5 Foundational usability knowledge as a prerequisite for AI-assisted work 34
2.6 Human factors for UX 35
2.7 GenAI as a learning assistant for usability foundations 35
2.8 Prompting as a learning and reflection technique 36
2.9 Assessing AI output critically 36
2.10 Familiar patterns and unfamiliar context 37
2.11 AI-assisted usability reflection and human review 37
3 Discovery: Understanding Users, Context, Problems, and Hypotheses 39
3.1 Core idea of this chapter 39
3.2 Contribution to the business outcomes 39
3.3 Learning objectives 40
3.4 Key terms 40
3.5 Discovery in the software development process 41
3.6 User, user goal, task, and context of use 41
3.7 Typical information sources for usability discovery 42
3.8 GenAI for structuring existing information 43
3.9 From information to usability problem areas 44
3.10 Hypotheses instead of certainties 45
3.11 Using personas and usage scenarios with caution 45
3.12 When real users, domain experts, or additional research activities are required 46
3.13 Turning discovery findings into usable artifacts 47

3.14 AI-assisted discovery and human review 48

4 Backlog Refinement and Usability 49

4.1 Core idea of this chapter 49
4.2 Contribution to the business outcomes 49
4.3 Learning objectives 49
4.4 Key terms 50
4.5 Backlog refinement as a translation point 50
4.6 Usability requirements as part of professional requirements work 51
4.7 Making implicit usability requirements visible 51
4.8 Reviewing user stories from a usage perspective 52
4.9 Reviewing backlog items with GenAI 53
4.10 Complementing acceptance criteria with usability aspects 54
4.11 Definition of Ready and Definition of Done with a usability focus 54
4.12 Professionally classifying AI-generated requirements 55
4.13 AI-assisted refinement and human review 55

5 Planning and Team Organization 57

5.1 Core idea of this chapter 57
5.2 Contribution to the business outcomes 57
5.3 Learning objectives 57
5.4 Key terms 58
5.5 Planning as the transition from backlog to teamwork 58
5.6 Making usability tasks visible in planning 59
5.7 Roles and responsibility in the software development team 59
5.8 Organizationally planning simple usability activities 60
5.9 Where teams can act themselves and where expertise is necessary 61
5.10 Design critiques as a structured format 62
5.11 GenAI for preparing and structuring team activities 62
5.12 Documenting the use of AI 63
5.13 AI-assisted team organization and human review 63

6 UI, Information Architecture, States, and AI-Generated Drafts 65

6.1 Core idea of this chapter 65
6.2 Contribution to the business outcomes 65
6.3 Learning objectives 66
6.4 Key terms 66
6.5 From backlog item to visible solution 67
6.6 UI design foundations for assessing drafts 68
6.7 Information architecture, navigation, and findability 69
6.8 Prototype types and their purpose 70
6.9 AI-assisted rapid prototyping in implementation 70
6.10 Assessing AI-generated UI drafts 71
6.11 Familiar UI patterns as a strength and risk of AI 72
6.12 States, edge cases, and alternative paths 73
6.13 UI patterns, onboarding, and help 74
6.14 Prototypes as a basis for communication, learning, evaluation, and validation 74
6.15 AI-assisted design, prototyping, and human review 75

7 Code Creation, Components, Accessibility, and Handoff 77

7.1 Core idea of this chapter 77
7.2 Contribution to the business outcomes 77
7.3 Learning objectives 77
7.4 Key terms 78
7.5 From usability requirements to the implemented UI 79
7.6 AI-assisted code creation for usability and frontend 79

7.7 Usability in implementation 80
7.8 Components, design systems, and design tokens 81
7.9 AI for deriving UI patterns and design system building blocks 82
7.10 Handoff between design, AI, and development 83
7.11 Accessibility basics in implementation 84
7.12 AI for documentation, tests, and reviews 85
7.13 Risks of AI-generated implementation 86
7.14 Review measures for AI-generated code 87
7.15 AI-assisted implementation and human review 88

8 Evaluation, Testing, and Validation 90

8.1 Core idea of this chapter 90
8.2 Contribution to the business outcomes 91
8.3 Learning objectives 91
8.4 Key terms 92
8.5 Evaluation, testing, and validation in the software development process 92
8.6 Making usability visible in review formats 93
8.7 Heuristic evaluation with AI support 93
8.8 Cognitive walkthrough 94
8.9 Planning usability tests 95
8.10 Formulating test tasks 96
8.11 Facilitation, think-aloud, and structured observation 97
8.12 Structuring observations, findings, and report drafts 98
8.13 Prioritizing usability problems 98
8.14 Deriving testable assumptions from AI suggestions 99
8.15 AI simulations and synthetic users 100
8.16 AI-assisted analysis and human review 101

9 Release, Monitoring, and Improvement 102

9.1 Core idea of this chapter 102
9.2 Contribution to the business outcomes 102
9.3 Learning objectives 103
9.4 Key terms 103
9.5 Observing usability after release 104
9.6 Sources of usability insights after release 104
9.7 Why usability must be further improved after release 105
9.8 UX metrics and the HEART framework 106
9.9 GenAI for structuring feedback, tickets, and metrics 107
9.10 Distinguishing pattern, correlation, cause, and priority 108
9.11 Recognizing UX debt 109
9.12 Turning insights into backlog items and improvement measures 110
9.13 Embedding continuous improvement in the team 111
9.14 External support and the limits of team-internal AI use 112
9.15 AI-assisted monitoring and human review 113

10 Literature 114

11 Annex 1 – Glossary of key terms 116

0 Preamble

0.1 Purpose of this syllabus

This syllabus describes the intended business outcomes, learning objectives, and fundamental concepts for the syllabus *UXQCC Certified Professional in Software Usability with GenAI – Foundation Level.*

The syllabus defines which professional inputs are to be taught in a training of approximately 16 hours. It is aimed at software development teams that want to integrate software usability and user experience into their development process earlier, more pragmatically, and more systematically with the support of generative artificial intelligence.

This syllabus uses the terms software usability, user experience, and interaction capability in their respective meaning for software engineering. Usability refers to the fitness for use of software in a specific context of use. The central question is whether specific users can achieve their goals with a system effectively, efficiently, and satisfactorily. User experience, UX for short, is the broader frame and additionally covers expectations, perceptions, and responses before, during, and after the use of a product or system. In ISO/IEC 25010, interaction capability refers to the product-related quality of human-system interaction in the software quality model. In this syllabus, UX is therefore not understood as downstream visual styling, but as part of software usability, software quality, and verifiable interaction quality.

For better readability, the term usability is predominantly used in the remainder of this text. It stands for software usability as defined above and includes the UX aspects covered in this syllabus. UX is explicitly mentioned where the broader experience frame is meant or where established terms such as UX quality, UX debt, UX metrics, or role designations such as UX designer are used.

The focus is on building solid foundational usability competence in software development teams. GenAI is regarded as a tool for learning, structuring, drafting, reviewing, and support. At the same time, the syllabus emphasizes the limits of AI: AI can use familiar patterns, generate hypotheses, and formulate suggestions, but it does not replace human judgment, domain knowledge, real user observation, or professional responsibility.

The syllabus deliberately follows a typical agile software development process and uses Scrum descriptively as a widely used frame of reference. The contents therefore do not follow a classical HCI taxonomy, but the working steps in which usability in agile software development teams is created, influenced, reviewed, or improved: orientation, competence building, discovery, backlog refinement,

sprint planning, design and prototyping, implementation, review formats, evaluation and testing, as well as release, monitoring, and continuous improvement.

0.2 Target audience for this syllabus

This syllabus is aimed first and foremost at software developers as well as testers and QA teams in software development teams. Supported by AI, they today implement requirements, UI components, code, and tests in very little time and thereby de facto decide on the usability of the resulting software, often without a dedicated UX role being available in the team.

It equally addresses all other roles involved in this process: product owners, requirements engineers and business analysts, UI designers, UX professionals, Scrum Masters and agile coaches, as well as leaders who create the conditions for the responsible use of AI.

UX professionals find in this syllabus the shared language and the process anchors through which their expertise becomes accessible to development teams, and they can use AI in a targeted way to make their work more efficient.

The syllabus is particularly relevant for organizations in which AI tools are already used for requirements, design, code, tests, documentation, or analysis, but usability work is not yet sufficiently and systematically embedded in the development process. This applies especially to teams without a permanently available dedicated UX role, or where usability aspects have so far been considered too late, unsystematically, or only selectively.

The goal is not to train UX professionals, but to build solid foundational usability competence in the team with the support of AI. AI is used to accelerate development work. However, AI output must be critically reviewed from a usage and quality perspective in order to identify risks to usability and to embed usability as part of professional software quality in daily teamwork.

This syllabus is particularly relevant for teams and individuals who

- want to develop software faster with AI and place particular value on integrating usability, understandability, and usefulness,
- do not have a permanently available dedicated UX role in the team,
- want to support user stories, acceptance criteria, UI drafts, prototypes, code, tests, or feedback data with GenAI,
- do not simply want to adopt AI-generated results, but want to assess them professionally, methodically, and from a usage perspective,

- want to make usability, accessibility basics, and UX quality visible earlier in the agile development process,
- want to align their development, testing, and product decisions more closely with user goals, context of use, and verifiable quality criteria,
- want to bring their UX expertise into software development efficiently.

In detail, this syllabus addresses the following target groups:

Software developers

- implement UI-related requirements, components, states, validation, error messages, and accessibility basics
- use AI-assisted coding tools for frontend, components, tests, documentation, and refactoring
- learn to review AI-generated code not only technically, but also from a usability perspective
- recognize that fast code creation only creates value if the resulting software can be used effectively, efficiently, and satisfactorily by the intended users in their respective context of use, and is implemented in an understandable, operable, and robust way

Testers and QA teams

- check not only functional correctness, but also understandability, states, error cases, keyboard operability, and basic accessibility
- use GenAI to prepare test cases, review checklists, observation sheets, and findings
- distinguish between AI-generated hints, heuristic assessments, and real test observations
- help make usability verifiable as a quality characteristic in reviews, tests, and acceptance

Product owners, product leads, and product managers

- formulate user goals, product assumptions, backlog items, and acceptance criteria with a usability focus
- prioritize usability problems, UX debt, and improvement measures
- use GenAI to structure feedback, hypotheses, requirements, and decision options
- ensure that AI-assisted product decisions do not merely sound plausible, but fit the actual context of use

Requirements engineers, business analysts, and domain analysts

- translate user problems, contexts of use, and product assumptions into backlog items and verifiable requirements

- use AI to structure feedback, support tickets, interviews, and domain information
- review AI-generated requirements for domain correctness, contextual fit, implicit assumptions, and validation needs
- reveal where functional requirements need to be complemented by usability and accessibility criteria

UX professionals

- translate their knowledge of users, contexts of use, research, and evaluation methods into the working formats of software development, such as backlog items, acceptance criteria, Definition of Ready and Definition of Done, review formats, and tests
- anchor usability criteria where development teams actually make decisions: in refinement, in planning, in code review, and in evaluation
- use GenAI to scale their expertise within the team, for example through reusable review questions, checklists, heuristic reviews, and the preparation of usability tests
- review AI-generated requirements, drafts, and analyses professionally and methodically, and enable the team to increasingly perform this review itself

UX and UI designers

- integrate GenAI into design, prototyping, variant creation, and design critiques
- assess AI-generated drafts based on user goal, visual hierarchy, mental models, cognitive load, consistency, and accessibility
- support agile teams in building shared usability criteria
- benefit from a shared understanding that eases collaboration with development, QA, product owners, and stakeholders

Scrum Masters, agile coaches, and people responsible for team processes

- support the embedding of usability activities in agile development processes
- make usability tasks, responsibilities, review formats, and learning loops visible within the team
- foster the reflective use of AI in the team without delegating responsibility to AI
- help ensure that UX is understood not as an add-on task at the end, but as part of ongoing teamwork

Leaders, team leads, and decision-makers

- understand usability with GenAI as part of software quality, product quality, and AI governance

- create the organizational conditions for responsible AI use, data protection, traceability, and human oversight
- recognize when teams can carry out simple usability activities themselves and when additional usability, research, accessibility, or domain expertise is required
- can use the training as a pragmatic measure to better connect AI use, software quality, and customer and user satisfaction

External service providers, agencies, and trainers

- can use the syllabus as a basis for trainings, workshops, or qualification programs
- independently add suitable exercises, case studies, and tool demonstrations
- ensure that AI-assisted usability work is taught as an accountable team competence and not as mere tool operation
- support organizations in using GenAI not only as an efficiency tool, but as an opportunity for better usability and quality processes

0.3Exam-relevant learning objectives and cognitive knowledge levels

The learning objectives support the business outcomes and serve as the basis for structuring the training contents and, where applicable, for creating an exam.

Learners can be expected to reproduce central contents of this syllabus, explain relationships, and apply knowledge in typical practice-oriented situations. The focus is on a foundation level: participants should not merely operate GenAI in the context of usability, but be able to classify AI output, review it critically, and transfer it to typical software development artifacts.

The knowledge levels of the respective learning objectives are stated at the beginning of each chapter and are classified as follows:

K1 – Remember:
Recall and reproduce facts, terms, and information.

K2 – Understand:
Grasp meanings, explain relationships, and classify concepts.

K3 – Apply:
Apply knowledge in concrete situations or to typical artifacts, for example user stories, acceptance criteria, UI drafts, prototypes, code, tests, feedback data, or UX debt.

The higher levels K4 to K6 of the revised Bloom taxonomy are not used as separate exam levels in this syllabus. However, they can play a role in advanced trainings, practical exercises, project work, or further certifications.

0.4 The exam

The exam for the *UXQCC Certified Professional in Software Usability with GenAI – Foundation Level* is based on the contents of this syllabus. Answering exam questions may require using contents from several sections of this syllabus. All chapters with professional content are exam-relevant, with the exception of the preamble and any annexes, unless these are explicitly marked as exam-relevant.

Standards, books, articles, online resources, legal regulations, and tool documentation may be used as references. However, their contents are exam-relevant only insofar as they are summarized or explicitly covered in this syllabus.

The exam consists of **40 multiple-choice questions**. Each correct answer is awarded one point. To pass the exam, a minimum score of **65%** is required, that is, at least **26 correctly answered questions**. The time available for the exam is **60 minutes**. If the native language of the person being examined is not the language of the exam, additional exam time of **25%**, that is, **15 minutes**, may be granted.

0.5 Accreditation

Training providers that want to offer trainings for the *UXQCC Certified Professional in Software Usability with GenAI – Foundation Level* must be accredited and use the official training materials. They must ensure that their examples, tool demonstrations, and exercises are consistent with the business outcomes and learning objectives of this syllabus.

Training providers should take particular care that the limits of AI, data protection, human review, governance, usability, accessibility, code quality, and validation are not neglected in favor of mere tool demonstrations.

For accredited training providers, additional materials are available to complement the public syllabus edition. These include, in particular, a detailed, didactically prepared provider syllabus, a base slide set for official trainings, and numerous exercise examples with accompanying sample prompts for practical consolidation of the contents. These materials serve the preparation, delivery, and quality assurance of accredited trainings and are to be used exclusively within the applicable license terms.

0.6 Level of detail

This public edition is deliberately more compact than the detailed provider edition. It describes the exam-relevant business outcomes, learning objectives, key terms, and professional contents in a framework-setting format.

Accredited training providers receive a more comprehensive edition as the basis for their trainings. It contains additional explanations, didactic guidance, examples, delineations, and in-depth professional elaborations for the design of official trainings.

The brevity of this public edition therefore does not mean that trainings are limited to this amount of text. Official trainings can and should elaborate the contents didactically, complement them with exercises, examples, tool demonstrations, and practical cases, and adapt them to the target groups, as long as they remain consistent with the business outcomes and learning objectives of this syllabus.

This syllabus consists of:

- business outcomes that define what participants can accomplish in professional practice after the learning unit in order to use usability with GenAI in software development teams in a meaningful, responsible, and effective way
- general learning goals that describe the purpose of the syllabus
- chapter-specific learning objectives with assigned cognitive knowledge levels
- key terms per chapter that should be understood and used in the respective context
- descriptions of central concepts, approaches, and limits
- relevant literature, standards, and online resources

The content of this syllabus is not a complete account of the entire fields of software usability, user experience, human-computer interaction, accessibility, software engineering, Scrum, or generative AI. It describes the level of detail that can be covered in a compact foundation-level training of approximately 16 hours.

This syllabus defines **which** contents, business outcomes, and learning objectives are covered. However, it does not contain binding specifications on **how** these contents are implemented didactically, which concrete exercises are used, or which tools are employed. The design of practical exercise components is therefore up to the training providers.

For practical implementation, complementary **recommended practice materials** are therefore provided. These include, in particular, structured prompt examples and an accompanying prompt library. These

materials show by example how the contents of this syllabus can be developed, deepened, and critically reviewed in practice with GenAI. The prompt library is to be used within the applicable license terms.

As a professional deepening, the book **Pucher/Simandl, *UX Design mit AI*** can additionally be consulted. It is aligned with the contents of this syllabus and structured to specifically support learning, revising, and practically applying the contents. However, this syllabus remains authoritative for business outcomes, learning objectives, and exam scope.

0.7 Structure of this syllabus

This syllabus is not organized along a classical HCI taxonomy. Its structure deliberately follows a typical software development process with agile working steps so that the contents connect directly to the practice of software development teams. Scrum is used descriptively as a widely used frame of reference.

There are nine chapters with exam-relevant professional content. The heading of each chapter states the intended teaching time; within the chapters, no further binding time allocation is given at sub-level. For the training course, the syllabus suggests a total of approximately **960 minutes**, that is, **16 hours**, of teaching.

The teaching time actually required may depend on the participants' prior knowledge, the examples used, the tools chosen, the share of discussions, practical exercises, the industry context, and the didactic design by the training provider.

Teaching structure and suggested timing

Chapter	Title	Duration in minutes (approx.)	Hours (approx.)
1	Usability with AI: Why This Topic Matters	90	1.5
2	Building Foundational Usability Knowledge with AI	120	2
3	Discovery: Understanding Users, Context, Problems, and Hypotheses	120	2
4	Backlog Refinement and Usability	120	2
5	Planning and Team Organization	90	1.5
6	UI, Information Architecture, States, and AI-Generated Drafts	135	2.25

Chapter	Title	Duration in minutes (approx.)	Hours (approx.)
7	Code Creation, Components, Accessibility, and Handoff	135	2.25
8	Evaluation, Testing, and Validation	150	2.5
9	Release, Monitoring, and Improvement	90	1.5
	Total	**960**	**16**

0.8 Accompanying prompt library and practice examples

This syllabus describes which contents, business outcomes, and learning objectives are to be covered in a training. It thereby sets the professional framework, but does not describe in detail how individual contents are developed didactically, implemented in practice with GenAI, or deepened in concrete exercises. To make this application level tangible as well, a structured prompt library with accompanying practice examples is provided to complement the syllabus.

These materials are aimed at participants, training providers, and self-learners. They show not only which contents are to be covered in the sense of this syllabus, but also how GenAI can be concretely guided, queried, reviewed, and used iteratively in typical situations. The prompt library is therefore not a mere collection of individual inputs; it explains the purpose, the context of use, the expected type of AI output, typical limits, and the necessary human review steps.

Since GenAI tools, available functions, use cases, and typical tool workflows evolve very quickly, these practice materials are preferably provided online and continuously maintained. This allows prompt examples, tool guidance, and application scenarios to be updated without changing the stable professional framework of this syllabus.

This creates a bridge between the professional framework of the syllabus and the practical application of usability with GenAI in trainings, workshops, self-study, and project-based learning settings.

0.9 Business Outcomes (BOs)

General business outcomes

BO-G 1

Embed usability work in software development teams earlier, more pragmatically, and more systematically with the help of GenAI.

BO-G 2

Critically assess AI-assisted usability results and distinguish between recognized patterns, plausible hypotheses, AI-generated suggestions, and actually validated findings.

BO-G 3

Consider fundamental usability criteria, including perception, understandability, cognitive load, accessibility, and expectable behavior, in discovery, requirements, backlog items, UI drafts, prototypes, code, review formats, evaluation, testing, and monitoring.

BO-G 4

Assess the benefits and limits of AI in the usability context, in particular with regard to familiar patterns, unfamiliar context of use, domain knowledge, user contact, and validation needs.

BO-G 5

Locate responsibility for usability, data protection, accessibility, code quality, AI use, and professional decisions within the software development team, and derive appropriate review and governance measures.

BO-G 6

Turn insights from AI-assisted analysis, evaluation, real usage, user and customer feedback, and support data into concrete improvements, backlog items, and continuous usability improvement.

Chapter 1

BO 1.1

Consider and justify usability as part of professional software quality in development processes.

BO 1.2

Use GenAI as a low-threshold entry into concrete usability work, especially in teams without their own UX professionals, sufficient budget, time, or methodological background.

BO 1.3

From the principle “AI is strong with familiar patterns but weak with unfamiliar context”, derive where AI provides meaningful support in the usability process and where human judgment, domain knowledge, or real user observation remain necessary.

BO 1.4

Apply fundamental rules for responsible AI use in the usability context, in particular human oversight, data protection, transparency, traceability, and governance.

Chapter 2

BO 2.1

Use GenAI in a targeted way to learn, revise, and apply central usability foundations, including human perception, mental models, cognitive load, understandability, and accessibility, to one's own development artifacts.

BO 2.2

Build a shared basic understanding of usability in the software development team by using AI to explain and compare central terms, quality criteria, and typical usability problems and relate them to the team's own development artifacts.

BO 2.3

Critically review AI output and distinguish between plausible-sounding suggestions, useful hypotheses, and reliable findings.

BO 2.4

Use AI support for familiar usability patterns and specifically verify it for unfamiliar context, implicit requirements, and domain-specific questions.

Chapter 3

BO 3.1

Use existing information such as support tickets, customer feedback, analytics, interviews, or domain knowledge to identify usability-relevant problem areas.

BO 3.2

Use GenAI to structure and summarize existing user and customer feedback, support hints, observations, and data, and to derive verifiable usability hypotheses from them.

BO 3.3

From information structured with AI support, derive concrete user problems, usage hypotheses, and verifiable product assumptions that serve as a basis for backlog items, prototypes, or tests.

BO 3.4

Recognize when AI-assisted hypotheses are not sufficient and real users, domain experts, or additional research activities must be involved.

Chapter 4

BO 4.1

Integrate usability requirements as part of professional requirements work into user stories, acceptance criteria, and Definition-of-Done criteria.

BO 4.2

Use GenAI to check backlog items for missing usability aspects, unclear wording, happy-path fixation, missing states, and possible edge cases.

BO 4.3

Integrate usability-relevant criteria into Definition of Ready and Definition of Done so that user goal, context of use, states, error cases, accessibility basics, and validation needs become verifiable.

BO 4.4

Professionally classify AI-generated requirements and acceptance criteria and decide which suggestions are relevant in the specific product and usage context.

Chapter 5

BO 5.1

Make usability tasks visible in planning and clarify basic responsibilities for UX within the software development team.

BO 5.2

Organizationally plan simple usability activities in the team, such as design critiques, heuristic reviews based on defined criteria, form tests, accessibility basic checks, and system state checks.

BO 5.3

Use GenAI for the preparation, structuring, and documentation of usability-related team activities without delegating responsibility and decision-making authority to AI.

BO 5.4

Document the use of AI in usability-relevant team activities in a traceable way, in particular purpose, inputs used, adopted results, and human review.

Chapter 6

BO 6.1

Assess AI-assisted UI drafts, wireframes, prototypes, and frontend suggestions based on fundamental usability and human-factors criteria, in particular perceivability, understandability, cognitive load, consistency, accessibility, and expectable behavior.

BO 6.2

Recognize typical weaknesses of AI-generated drafts, in particular missing states, edge cases, error paths, unclear navigation, insufficient accessibility, and excessive happy-path orientation.

BO 6.3

Use AI-generated prototypes as a basis for communication, learning, and validation without prematurely treating them as professionally correct or production-ready solutions.

BO 6.4

Compare AI-generated UI variants based on user goal, visual hierarchy, understandability, consistency, accessibility basics, and expectable behavior, and select among them with justification.

Chapter 7

BO 7.1

Professionally classify AI-assisted coding tools for UI-related development tasks, components, validation, documentation, and tests, and use them in a targeted way.

BO 7.2

Consider fundamental usability quality aspects in implementation, in particular consistent components,

states, semantic HTML, keyboard operability, focus order, labels, error messages, and accessibility basics.

BO 7.3

Check AI-generated UI implementations for their conformity with usability requirements, acceptance criteria, states, component rules, and tests.

BO 7.4

Recognize risks of AI-generated implementation and derive appropriate review measures such as code review, tests, security review, accessibility checks, and professional acceptance.

Chapter 8

BO 8.1

Classify fundamental review and evaluation methods for usability, such as heuristic evaluation, cognitive walkthrough, and usability testing, within development processes and use them appropriately.

BO 8.2

Use GenAI for the preparation, structuring, and documentation of usability reviews and usability tests, in particular for test tasks, observation notes, findings, and report drafts.

BO 8.3

Recognize why AI simulations and heuristic AI analyses do not replace validation with real users and when real tests are necessary.

BO 8.4

Prioritize usability problems by severity, frequency, user impact, risk, implementation effort, and business relevance, and critically assess AI suggestions in this regard.

BO 8.5

From AI-generated usability suggestions, derive testable assumptions and questions that can be verified through heuristic reviews, usability tests, or usage data.

Chapter 9

BO 9.1

Monitor usability after release based on feedback, support tickets, usage data, and simple UX metrics, and identify improvement potential.

BO 9.2

Use GenAI to structure user and customer feedback, tickets, metrics, and UX debt, to reveal patterns, and to derive hypotheses for improvements.

BO 9.3

Recognize the limits of data- and AI-based analysis, in particular the difference between pattern, correlation, cause, priority, and a reliable product decision.

BO 9.4

Turn insights from feedback, metrics, support data, and AI-assisted analysis into prioritized backlog items, UX debt measures, or new discovery questions.

0.10 Abbreviations and acronyms used

List of abbreviations

Abbreviation	Full name	Explanation
AI	Artificial Intelligence	English term for artificial intelligence. In this syllabus, AI is used as an umbrella term for systems that perform tasks such as text generation, analysis, pattern recognition, code assistance, or decision support.
API	Application Programming Interface	Programming interface through which software components or systems communicate. Relevant in the usability context where frontend, data, validation, and system responses interact.
ARIA	Accessible Rich Internet Applications	W3C specification for adding semantic information for assistive technologies. ARIA should only be used where native HTML elements are not sufficient.
BO	Business Outcome	Outcome-oriented description of what participants should be able to accomplish in professional practice after the training.
CI/CD	Continuous Integration / Continuous Delivery	Development and delivery practices in which code is regularly integrated, tested, and deployed. Relevant for quality assurance, testing, and continuous improvement.

Abbreviation	Full name	Explanation
CSS	Cascading Style Sheets	Stylesheet language for styling web interfaces, such as layout, colors, typography, spacing, and responsive presentation.
DoD	Definition of Done	Shared team agreement on when a backlog item or increment counts as done. This syllabus emphasizes that usability, accessibility, test, and quality criteria can be made explicit in it.
DoR	Definition of Ready	Shared team agreement on when a backlog item is sufficiently prepared to be taken into implementation or into a sprint. Usability-relevant aspects include user goal, context, states, error cases, and open assumptions.
DSGVO	Datenschutz-Grundverordnung	European data protection regulation for the protection of personal data (German abbreviation). Relevant when AI is used with user feedback, research data, support tickets, or personal information. See GDPR.
EAA	European Accessibility Act	European directive on accessibility requirements for certain products and services. Relevant in this syllabus as the legal framework for digital accessibility.
EN	European Norm	European standard. Particularly relevant in this syllabus in connection with EN 301 549 on accessibility requirements for ICT products and services.
EU	European Union	Political and legal framework, particularly relevant for data protection, AI regulation, and digital accessibility.
FL	Foundation Level	Foundation level of the certification. The foundation level describes a fundamental, practice-oriented qualification level at which central terms, concepts, and typical applications are understood and can be applied to simple professional situations.
GenAI	Generative Artificial Intelligence	Generative artificial intelligence. Systems that can create new content such as texts, images, UI drafts, code, summaries, or test cases.
GDPR	General Data Protection Regulation	English term for the European data protection regulation. See DSGVO.

Abbreviation	Full name	Explanation
HCI	Human-Computer Interaction	Field of research and practice on the interaction between humans and computer systems. This syllabus, however, is not structured along classical HCI lines, but along a typical agile software development process.
HEART	Happiness, Engagement, Adoption, Retention, Task Success	UX metrics framework by Google for the structured consideration of user happiness, engagement, adoption, retention, and task success.
HTML	HyperText Markup Language	Markup language for web content. Semantic HTML is a foundation for accessibility, robust structure, and assistive technologies.
ICT	Information and Communication Technology	Information and communication technology. The term is used, for example, in standards on the accessibility of digital products and services.
IDE	Integrated Development Environment	Integrated development environment, for example for code editing, debugging, testing, and AI-assisted development support.
IEC	International Electrotechnical Commission	International standardization organization for electrotechnology and related technologies. Relevant in combination with ISO, for example in ISO/IEC standards.
ISO	International Organization for Standardization	International standardization organization. Relevant in this syllabus for usability, human-centered design, and software quality, for example ISO 9241 and ISO/IEC 25010.
ISO/IEC	International Organization for Standardization / International Electrotechnical Commission	Joint standards of ISO and IEC, particularly relevant for software and system quality.
K1	Knowledge Level 1 – Remember	Learning objective level according to the revised Bloom taxonomy. Participants can reproduce terms, facts, or basic principles.
K2	Knowledge Level 2 – Understand	Learning objective level according to the revised Bloom taxonomy. Participants can explain relationships, describe differences, and classify concepts.

Abbreviation	Full name	Explanation
K3	Knowledge Level 3 – Apply	Learning objective level according to the revised Bloom taxonomy. Participants can apply knowledge to typical tasks, artifacts, or situations.
K4	Knowledge Level 4 – Analyze	Higher learning objective level according to Bloom. Not used as a separate exam level in this foundation-level syllabus.
K5	Knowledge Level 5 – Evaluate	Higher learning objective level according to Bloom. Not used as a separate exam level in this foundation-level syllabus.
K6	Knowledge Level 6 – Create	Higher learning objective level according to Bloom. Not used as a separate exam level in this foundation-level syllabus.
KPI	Key Performance Indicator	Metric for assessing performance or goal achievement. Relevant in the usability context when UX metrics are linked to product or business goals.
LO	Learning Objective	Learning objective. Describes concretely what participants should know, understand, or be able to apply after a chapter.
LLM	Large Language Model	Large language model trained on extensive text data that can analyze, generate, summarize, or transform text. The basis of many GenAI systems.
MVP	Minimum Viable Product	Minimal functional product version used to verify central assumptions or to provide initial value.
NIST	National Institute of Standards and Technology	US institution for standards and technology. Relevant in this syllabus through the AI Risk Management Framework.
OECD	Organisation for Economic Co-operation and Development	International organization for economic co-operation and development. Relevant through its AI Principles and governance orientation.
PO	Product Owner	Role in Scrum and in many agile product teams, responsible for the product backlog, prioritization, and product value. Important in the usability context for user

Abbreviation	Full name	Explanation
		goals, acceptance criteria, and the prioritization of usability problems.
RMF	Risk Management Framework	Framework for risk management. Particularly relevant in this syllabus as part of the NIST AI Risk Management Framework.
Scrum	Not an acronym	Framework for agile product development. The term is not an abbreviation, but is often used like one.
SQuaRE	Systems and software Quality Requirements and Evaluation	ISO/IEC series of standards on the quality of systems and software. Relevant in connection with ISO/IEC 25010, software product quality, interaction capability, and quality in use.
SUS	System Usability Scale	Standardized questionnaire for measuring perceived usability. Relevant for usability evaluation and UX metrics.
UI	User Interface	User interface of a system, that is, the visible and operable elements such as screens, forms, navigation, controls, and interactions.
UNESCO	United Nations Educational, Scientific and Cultural Organization	United Nations organization for education, science, and culture. Relevant through its recommendations on the ethics of artificial intelligence.
UX	User Experience	The overall experience of using a product or system. UX covers expectations, perceptions, and responses before, during, and after use. In this syllabus, UX is understood as the broader frame that includes software usability but goes beyond it.
UXQCC	User Experience Quality Certification Center	International certification organization for user experience and usability. UXQCC develops and maintains certifications for UX and usability competence, based on internationally recognized standards, in particular the ISO 9241 series.
W3C	World Wide Web Consortium	International organization for the development of web standards. Relevant for HTML, ARIA, WCAG, and web accessibility.

Abbreviation	Full name	Explanation
WAI	Web Accessibility Initiative	W3C initiative promoting digital accessibility. Develops resources and standards such as the WCAG materials.
WCAG	Web Content Accessibility Guidelines	Guidelines by the W3C for accessible web content. Relevant in this syllabus for accessibility basics and digital accessibility.

1 Usability with AI: Why This Topic Matters

Teaching time: approx. 90 minutes

1.1 Core idea of this chapter

This chapter introduces the basic logic of this syllabus. Software usability and user experience are understood as parts of professional software quality. For software development teams it is relevant that this quality is already created in problem understanding, requirements, user stories, acceptance criteria, states, error messages, components, tests, reviews, and monitoring.

GenAI can enable a low-threshold entry into usability work by structuring information, explaining terms, generating variants, reviewing artifacts, or preparing analyses. AI output, however, must be reviewed by humans, connected to the product and usage context, and placed appropriately within the development process. A central basic principle is:

AI is strong with familiar patterns but weak with unfamiliar context.

1.2 Contribution to the business outcomes

This chapter particularly supports business outcomes BO 1.1 to BO 1.4. The complete and authoritative wording of the business outcomes can be found in section 0.9.

1.3 Learning objectives

LO	Learning objective	K
LO 1.1	Be able to reproduce the terms software usability, user experience, and UX quality in the context of software development.	K1
LO 1.2	Be able to explain why usability is part of professional software quality.	K2
LO 1.3	Be able to explain why agile processes do not automatically ensure good UX.	K2
LO 1.4	Be able to describe how GenAI enables a low-threshold entry into usability work.	K2

LO	Learning objective	K
LO 1.5	Be able to explain the basic rule “AI is strong with familiar patterns but weak with unfamiliar context”.	K2
LO 1.6	Be able to name typical fields of application of GenAI along typical agile development activities.	K1
LO 1.7	Be able to classify possible uses of AI in agile development processes based on their benefits and limits.	K3
LO 1.8	Be able to explain fundamental rules for responsible AI use in the usability context.	K2

1.4 Key terms

Software usability, user experience, usability, UX quality, software quality, interaction capability, quality in use, ISO/IEC 25010, agile development, Scrum, backlog, discovery, delivery, GenAI, AI output, pattern recognition, hallucination, context of use, human oversight, data protection, transparency, AI governance.

1.5 Software usability, UX, and interaction capability as parts of software quality

Software usability and user experience are parts of the quality of interactive software. Software usability describes the extent to which specific users can use a system in a specific context of use to achieve specific goals effectively, efficiently, and with satisfaction.

User experience, UX for short, is the broader term. In addition to actual use, UX also covers expectations, perceptions, and responses before, during, and after the use of a product or system. UX includes software usability but goes beyond it.

In software quality, the product-related quality of human-system interaction is described, among other things, by the term interaction capability. This includes characteristics such as learnability, operability, user error protection, user assistance, self-descriptiveness, and accessibility. For software development teams this is relevant because such characteristics can be considered in requirements, acceptance criteria, reviews, tests, Definition-of-Done criteria, and continuous improvement.

Software is therefore not good simply because it works technically. It must also be usable in such a way that people understand their tasks, find relevant information, make decisions, and avoid or recover from errors. Every function has a usage perspective in addition to a technical one.

UX is not to be equated with visual design. A visually appealing interface can be hard to understand or inefficient. Conversely, a simple interface can have high UX quality if it clearly supports user goals and makes the interaction comprehensible.

For agile software development teams it is important that UX quality does not only emerge at the end of a project. It already shows in backlog items, acceptance criteria, prototypes, components, tests, reviews, and in how feedback is handled after release.

1.6 Usability in development processes

Agile processes help software development teams plan, implement, and review work iteratively. Scrum is used descriptively in this syllabus as a widely used agile frame of reference. However, neither Scrum nor other agile approaches automatically ensure that real user needs are understood, usability requirements are formulated, or contexts of use are sufficiently considered.

Usability-relevant decisions are often made before an interface becomes visible: when a problem is described, a user group is assumed, a user story is formulated, an acceptance criterion is defined, or a technical solution path is chosen. Usability effects also arise during implementation, for example through naming, states, error messages, validations, components, navigation logic, focus order, or loading behavior.

A product backlog can be fully groomed and still omit essential usability aspects. A user story can be functionally correct and yet contain no indication of understandability, error cases, states, accessibility, or context of use. A sprint review can show that a function was implemented without checking whether users actually understand or can successfully use that function.

In agile development processes, UX can be made visible in discovery activities, backlog refinements, acceptance criteria, Definition of Ready, Definition of Done, design critiques, heuristic reviews, usability tests, and post-release monitoring, among others.

1.7 GenAI as support along agile activities

GenAI can make usability work in agile software development teams accessible earlier and easier to connect to, especially when no permanently available dedicated UX role exists or there is little methodological UX experience.

In discovery, AI can structure existing feedback, summarize support tickets, organize interview notes, or formulate initial hypotheses. In backlog refinement, AI can review user stories, add acceptance criteria, flag unclear wording, uncover missing states, or reduce an excessive happy-path orientation. In agile planning, AI can help structure usability tasks, prepare review questions, or create checklists.

In design and prototyping, AI can generate UI variants, navigation models, wireframes, or clickable prototypes. In implementation, AI can support UI components, code, tests, documentation, and accessibility guidance. In review, evaluation, and testing, AI can prepare heuristic assessments, formulate test tasks, structure observation notes, or summarize findings. After release, AI can analyze feedback, tickets, or metrics and provide indications of UX debt.

These possible uses do not replace professional usability work, but they can enable an initial, pragmatic engagement with UX. They should be classified by benefit and limits, in particular by whether AI mainly provides structuring, variant creation, wording, technical support, or initial analysis, and which human review remains necessary.

1.8 Familiar patterns, unfamiliar context, and AI output

Generative AI does not think in the human sense. It has no perception of its own, no usage experience of its own, and no real understanding of specific users. It produces output based on learned patterns and probabilities.

AI can be helpful for tasks that occur frequently, are well documented, and rest on familiar patterns. Examples include typical login flows, standard forms, simple navigation patterns, error messages, button texts, onboarding sequences, simple accessibility guidance, or initial heuristic assessments.

AI becomes less reliable as soon as the specific context of use is decisive. This includes real user goals, specialized domains, organizational conditions, implicit requirements, rare exceptional cases, legal questions, safety-critical applications, or prioritization decisions. Good UX depends on who uses a system, in what situation, with what prior knowledge, under what constraints, and with what goal.

AI output can appear convincing in language or visuals even though it is professionally inaccurate, incomplete, or unsuited to the specific context. Plausibility is therefore no proof of quality. In the usability context, AI output is initially to be treated as a hypothesis, a variant, or a structuring suggestion.

1.9 Responsible AI use in the usability context

Using AI in the usability context requires human oversight. AI output must be reviewed, classified, and, where necessary, discarded or adapted. This applies to texts, user stories, acceptance criteria, UI drafts, prototypes, code, test cases, and analyses.

AI is to be understood as a drafting partner, not a decision-maker. It can generate variants, formulate hypotheses, suggest structures, and point out alternatives. Decisions about product, users, quality, risk, and implementation remain with the team or with the responsible persons and organizations.

Data protection and confidentiality must be considered before using an AI tool. Personal data, confidential project data, or sensitive research data must not be entered into external AI services without reflection. Where possible, data should be anonymized, pseudonymized, or replaced with synthetic examples.

Transparency and traceability are required when AI output flows into requirements, designs, code, tests, or product decisions. It should remain traceable where AI was used, for what purpose, which results were adopted, and what human review took place.

AI governance creates the organizational framework for responsible AI use. This includes rules on permitted tools, data protection, documentation, human review, quality assurance, and responsibilities.

AI can suspect usability problems, but it cannot replace real usage. Where central assumptions are uncertain, user feedback, domain expertise, usability tests, or real usage data must be consulted.

2 Building Foundational Usability Knowledge with AI

Teaching time: approx. 120 minutes

2.1 Core idea of this chapter

This chapter treats GenAI not primarily as a tool for creating usability artifacts, but as a learning and reflection tool for software development teams. Teams need a shared basic understanding of usability in order to professionally assess AI-generated suggestions, UI drafts, requirements, error messages, prototypes, or code.

Foundational usability knowledge covers central terms such as user goal, context of use, mental model, conformity with user expectations, cognitive load, visual hierarchy, and accessibility. It also includes fundamental human factors such as perception, attention, memory, motor abilities, situational constraints, and cognitive load.

AI can explain these foundations, connect them with examples, and help apply them to the team's own development artifacts. AI output, however, is not to be treated as reliable knowledge. It must be critically classified based on fundamental usability criteria, the context of use, and human review.

2.2 Contribution to the business outcomes

This chapter particularly supports business outcomes BO 2.1 to BO 2.4. The complete and authoritative wording of the business outcomes can be found in section 0.9.

2.3 Learning objectives

LO	Learning objective	K
LO 2.1	Be able to reproduce central basic usability terms such as user goal, context of use, mental model, conformity with user expectations, cognitive load, visual hierarchy, and accessibility.	K1
LO 2.2	Be able to name fundamental human factors for UX, in particular perception, attention, memory, cognitive load, motor abilities, and situational constraints.	K1
LO 2.3	Be able to explain why human perception is selective, limited, and context-dependent.	K2

LO	Learning objective	K
LO 2.4	Be able to explain why mental models, conformity with user expectations, and cognitive load are central to assessing user interfaces.	K2
LO 2.5	Be able to explain why software development teams need a shared basic understanding of usability in order to assess AI output meaningfully.	K2
LO 2.6	Be able to describe how GenAI can support learning and revising usability foundations.	K2
LO 2.7	Be able to use the team's own development artifacts such as screens, user stories, error messages, prototypes, or code samples for usability reflection with AI support.	K3
LO 2.8	Be able to describe prompting techniques for usability learning, in particular explaining, comparing, generating counterexamples, checking assumptions, and having improvement suggestions justified.	K2
LO 2.9	Be able to classify AI output as hypotheses and not as reliable findings.	K2
LO 2.10	Be able to recognize and critically question plausible but potentially incorrect or incomplete AI answers.	K3
LO 2.11	Be able to check AI-assisted usability explanations and improvement suggestions against fundamental criteria of human perception, understandability, conformity with user expectations, and accessibility.	K3

2.4 Key terms

User, user goal, task, context of use, software usability, usability, user experience, UX quality, foundational usability knowledge, human-centered design, perception, attention, memory, cognitive load, mental model, conformity with user expectations, understandability, visual hierarchy, motor abilities, situational constraints, accessibility, AI output, prompting, hypothesis, plausibility, hallucination, apparent plausibility, pattern recognition, unfamiliar context, critical review, human oversight

Note on the key terms

The terms listed in this section form the central terminological basis for chapter 2. They name the concepts that are particularly important for understanding foundational usability knowledge, human factors, and the critical assessment of AI output.

The glossary in Annex 1 briefly summarizes the most important terms of the entire syllabus and serves as orientation for a shared basic understanding of UX, agile development, Scrum, and GenAI in the context of this curriculum.

In addition, a separate extended glossary document is available. It contains additional terms, translations, synonyms, examples, delineations, and normative and professional references. It serves as a learning and reference aid, but does not constitute separate additional exam material. Terms are exam-relevant only insofar as they are professionally covered in the chapters, business outcomes, and learning objectives of this syllabus.

2.5 Foundational usability knowledge as a prerequisite for AI-assisted work

Software development teams need a shared basic understanding of usability in order to assess AI output meaningfully. Without shared terms, statements about usage often remain vague, for example when an interface is merely described as “not intuitive”, “cluttered”, or “not appealing”. For professional product development, such impressions must be translated into verifiable quality questions.

Central terms such as user goal, task, context of use, software usability, user experience, mental model, conformity with user expectations, cognitive load, visual hierarchy, and accessibility help describe usage problems professionally. They make it possible to check AI-generated suggestions not only by taste or plausibility, but based on comprehensible criteria.

GenAI can generate texts, screens, requirements, acceptance criteria, error messages, prototypes, or code suggestions very quickly. This shifts the bottleneck from creation to assessment. Foundational usability knowledge thus becomes a review capability: teams must be able to judge whether AI output is understandable, in line with user expectations, consistent, accessible, and plausible for the context of use.

This capability is a team competence. Product owners, developers, testers, requirements engineers, usability and UI roles, and people with process responsibility jointly influence requirements, implementation, review, and prioritization. A shared basic understanding of usability supports collaboration between these roles.

2.6 Human factors for UX

Human perception is selective, limited, and context-dependent. People do not perceive all information completely and objectively; they direct their attention to what seems relevant to their current task, expectation, or situation. For UI design this means that visibility alone is not enough. An element must be perceivable at the right moment, in the right place, with sufficient clarity, and in the appropriate context.

Attention and memory are limited. User interfaces that contain too many options, unclear priorities, contradictory signals, or unnecessary information increase cognitive load. This is particularly relevant for forms, complex dashboards, multi-step processes, error messages, security prompts, and professionally demanding systems.

Mental models describe how users believe a system works or should work. They arise from previous experience, familiar patterns, domain knowledge, and context of use. Conformity with user expectations means that a system behaves as users expect based on their experience and task. When a button, a navigation, or a process reacts differently than expected, errors, uncertainty, or loss of trust can result.

Users differ in visual, motor, cognitive, linguistic, and technical abilities. In addition, situational constraints can occur, such as poor lighting, mobile use, time pressure, stress, distraction, one-handed operation, or a noisy environment.

Accessibility is also a fundamental part of good UX. AI can provide indications of accessibility problems, but it cannot guarantee a complete accessibility review or conformity.

2.7 GenAI as a learning assistant for usability foundations

GenAI can support teams in learning, revising, and structuring usability foundations. It can explain terms, work out differences between concepts, generate examples, describe typical mistakes, or relate usability criteria to artifacts from everyday development work.

Using the team's own development artifacts is particularly relevant. Screens, user stories, error messages, prototypes, components, code samples, test tasks, or user feedback can be used for usability reflection with AI support. A team can, for example, have AI explain why an error message seems hard to understand, why a form might be cognitively demanding, or which assumptions about user goals a user story contains.

Participants should be able to analyze their own development artifacts with AI support, check the generated hints against fundamental usability criteria, and derive questions for further professional clarification from them.

AI-assisted learning, however, is not to be equated with reliable expert knowledge. AI explanations can be incomplete, too general, or wrong. Participants should therefore use AI as a learning aid and reflection tool, not as an authority. The value lies in the fact that AI can generate questions, variants, justifications, and counterexamples that are professionally reviewed and discussed within the team.

2.8 Prompting as a learning and reflection technique

In the usability context, prompting can be used as a learning and reflection technique. This is not about memorizing individual prompt wordings, but about the ability to use AI in a targeted way for explanation, comparison, checking assumptions, and justifying improvement suggestions.

Prompts for explanation can be used to clarify usability terms in relation to software development. One should ask not only for definitions, but for meaning, typical examples, risks, and application to concrete artifacts. A term like cognitive load can, for example, be considered in the context of forms, dashboards, or multi-step processes.

Prompts for comparing and justifying can help discuss variants of button texts, error messages, form structures, onboarding steps, or navigation solutions. Criteria such as understandability, user goal, conformity with user expectations, cognitive load, accessibility, error prevention, and consistency should be considered.

Prompts for counterexamples and critical review support reflection on one's own assumptions. Teams can ask AI to name possible problems of a solution, to reveal implicit assumptions, or to describe in which contexts of use a suggestion might be unsuitable. Evaluating the results remains the team's task.

2.9 Assessing AI output critically

AI output can appear complete, professional, and convincing. In the usability context, this can lead to suggestions being accepted too quickly. However, a clear wording or a visually appealing draft does not mean that the suggestion is professionally correct, complete, or suitable for the context of use.

Participants should be able to distinguish between plausibility, hypothesis, suggestion, and reliable finding. A plausible AI answer can be a useful starting point, but it is not a validated finding. Reliability

only arises through appropriate review, for example through domain knowledge, comparison with requirements, user feedback, evaluation, usability tests, or real usage data.

A common weakness of AI is filling context gaps with plausible assumptions. If no information about user group, task, domain, risks, or conditions is available, AI still produces an answer. This answer may seem sensible, but it may rest on generic patterns.

Participants should therefore recognize context gaps as early as possible. Important review questions are which information is missing, which user group is implicitly assumed, which domain rules are relevant, which usage situations need to be described, and which risks would otherwise be carried along invisibly in early requirements, UI drafts, or prototypes.

2.10 Familiar patterns and unfamiliar context

AI is particularly helpful with familiar, frequently occurring, and well-documented usability patterns. These include standard forms, login flows, search functions, error messages, call-to-action texts, simple dashboards, onboarding steps, or basic UI patterns. In such areas, AI can suggest typical variants, name known weaknesses, or formulate improvements.

The more a usability question depends on the specific context of use, the more cautiously AI output must be treated. This applies, for example, to medical applications, industrial control systems, financial systems, safety-critical processes, internal domain applications, or systems with specialized user groups.

In such cases, generic usability patterns are not sufficient. What matters is how specific users work under real conditions, which domain language they use, which errors are severe, and which expectations arise from their practice. Specialist literature, standards, guidelines, human-factors analyses, domain knowledge, and real user observation may be required to classify AI output appropriately.

For agile software development teams, this results in a **simple review logic**: AI can help quickly with familiar patterns. However, when specific users, domain knowledge, workflows, risks, or legal requirements are decisive, AI suggestions must be reviewed through human expertise, user feedback, or tests.

2.11 AI-assisted usability reflection and human review

In this chapter, AI can be used particularly well as a learning and reflection tool.

AI is good at	Explaining terms; explaining human-factors criteria; analyzing the team's own artifacts; comparing variants; generating counterexamples; pointing out typical usability problems.
Limitations and typical risks	No guaranteed professional truth: terms can be weighted incorrectly, examples can be too general, and explanations can be convincing but incomplete; domain-specific particularities are overlooked; AI does not know the perception, mental models, and real behavior of specific user groups from its own experience.
Requires human review	Comparison against the criteria of human perception, understandability, conformity with user expectations, cognitive load, and accessibility; consideration of the specific context of use; clarification of which statements need further verification through user feedback, usability tests, usage data, or domain knowledge.

What matters for competence building is that terms are not merely repeated, but understood and applied to the team's own artifacts and decision situations.

3 Discovery: Understanding Users, Context, Problems, and Hypotheses

Teaching time: approx. 120 minutes

3.1 Core idea of this chapter

This chapter covers discovery for usability before or alongside concrete implementation. The central question is how software development teams, with AI support, can build a better understanding of users, tasks, contexts of use, problems, and product assumptions before solutions are concretely designed or implemented.

In this syllabus, discovery is not understood as an extensive research program that necessarily requires dedicated specialist roles, long lead times, or large budgets. It is about a pragmatic start: using existing information, recognizing indications of user problems, revealing assumptions, formulating hypotheses, and deciding where further clarification is needed.

GenAI can support this process by structuring support tickets, user and customer feedback, interview notes, qualitative responses, analytics data, internal observations, or domain descriptions, and by working out recurring patterns. From this, initial problem hypotheses, research questions, usage scenarios, or product assumptions can be derived.

AI-assisted discovery, however, remains particularly context-dependent. AI can organize existing information, but it does not know real users, their tasks, or their context of use from its own experience. AI-generated statements are therefore to be treated as verifiable assumptions and not as reliable findings.

3.2 Contribution to the business outcomes

This chapter particularly supports business outcomes BO 3.1 to BO 3.4. The complete and authoritative wording of the business outcomes can be found in section 0.9.

3.3 Learning objectives

LO	Learning objective	K
LO 3.1	Be able to reproduce the terms user, user goal, task, context of use, problem hypothesis, and product assumption.	K1
LO 3.2	Be able to explain why usability work must begin before the solution idea with understanding user, task, and context.	K2
LO 3.3	Be able to name typical information sources for usability discovery, such as support tickets, customer feedback, analytics, interviews, domain knowledge, and existing documentation.	K1
LO 3.4	Be able to describe how GenAI can structure and summarize existing user and customer feedback, support hints, observations, and data.	K2
LO 3.5	Be able to derive initial usability-relevant problem areas from existing information with AI support.	K3
LO 3.6	Be able to formulate AI-generated user assumptions as hypotheses and distinguish them from reliable findings.	K3
LO 3.7	Be able to explain why AI-generated personas and scenarios can be problematic without reliable data.	K2
LO 3.8	Be able to judge when real users, domain experts, or additional research activities are required.	K3
LO 3.9	Be able to derive verifiable product assumptions for backlog items, prototypes, or tests from discovery findings.	K3

3.4 Key terms

User, user group, user goal, task, context of use, domain knowledge, problem hypothesis, product assumption, research question, user feedback, customer feedback, support ticket, support data, analytics, usage data, interview note, observation, persona, data-based persona, hypothetical persona, usage scenario, AI-assisted structuring, pattern, hypothesis, evidence, validation need, discovery, discovery result, backlog connection

3.5 Discovery in the software development process

Discovery describes activities that help to better understand a problem, a target group, a task, or a context of use before a solution is implemented. In agile development processes, discovery can take place before an implementation iteration or run in parallel to implementation when open questions for upcoming work steps are clarified.

Discovery is particularly important for agile teams because backlog items are often already formulated as solutions. A story then describes a new feature without it being clear which user problem it is supposed to solve. Discovery helps take a step back: What problem is at hand? For whom is it relevant? In what situation does it occur? What assumptions are we making? What information is missing?

Discovery is not only valuable when extensive research activities are possible. Small steps can also be relevant: reviewing existing feedback, clustering support tickets, collecting open questions, formulating assumptions, sharpening user goals, or obtaining domain knowledge. What matters is distinguishing between solution, problem, assumption, and finding.

In agile teams, different roles can contribute to discovery. Product owners can bring in customer feedback, developers can analyze support cases or technical constraints, testers can document recurring errors and irritations, stakeholders can contribute domain knowledge, and, where necessary, real users can be interviewed or observed.

Discovery is only effective if its results are carried into subsequent work. A problem area can become a backlog item. An uncertain assumption can become a test question. An unclear user group can become a research question. A suspected problem can become a prototype used to verify a solution idea.

3.6 User, user goal, task, and context of use

At the beginning of every usability-oriented discovery stands the question of who a system or a feature is intended for. “The user” is often too vague. Different user groups can have different goals, roles, permissions, prior experience, working conditions, and expectations.

User groups should therefore not be described only demographically. What matters is their relationship to the task and the system. Relevant aspects include role, goal, experience, domain knowledge, technical competence, frequency of use, responsibility, decision latitude, and possible constraints.

A distinction must also be made between system function and user goal. A function such as “export as CSV” describes what the system should do. The underlying user goal, however, can vary: handing data

over to accounting, preparing a report, running an external analysis, or fulfilling documentation obligations. Usability work therefore begins before the solution idea with user, task, and context.

A task describes what users have to do to achieve a goal. For UX, it is relevant how this task is embedded in everyday work: Which steps come before? What information is needed? Which decisions must be made? Which errors are likely? What are the consequences of an error? Which interruptions or time constraints exist?

The context of use covers the conditions under which a system is used. This includes physical, technical, organizational, social, and psychological conditions. A system is experienced differently depending on whether it is used in the office, on the go on a smartphone, under time pressure, in a call center, on a factory floor, in a hospital, or in an exam situation.

GenAI can help derive possible user goals, tasks, or context dimensions from functional descriptions. These derivations, however, are hypotheses. The team must check whether they fit the product reality, the user group, the domain, and the actual usage situations.

3.7 Typical information sources for usability discovery

Typical information sources for usability discovery are:

- support tickets, customer feedback, reviews, and free-text comments,
- analytics and usage data,
- interviews, observation notes, and workshop minutes,
- functional specifications, process descriptions, and existing documentation,
- domain knowledge within the team.

Support tickets and service desk data contain indications of where users fail, which terms they do not understand, which processes repeatedly cause problems, or which functions frequently need to be explained. GenAI can group such tickets by topic, identify recurring problems, summarize similar cases, or formulate initial problem hypotheses.

Customer feedback, app store reviews, survey comments, or free-text fields can provide indications of frustration, wishes, expectations, or recurring irritations. AI can summarize and cluster such qualitative responses, describe their tone, and work out recurring topics. The results are to be understood as an initial structuring and must be professionally reviewed.

Analytics and usage data can show what users do: where they abandon a process, which pages they visit frequently, which functions are rarely used, or how long certain processes take. However, these

data do not automatically explain why something happens. A high abandonment rate can have different causes, such as unclear texts, technical errors, lack of trust, wrong expectations, or high cognitive load.

Interviews, observations, and existing documentation often contain valuable information about users, tasks, and context. AI can help summarize such materials, extract central topics, flag open questions, or uncover contradictions. A distinction must be made between actual observation, human interpretation, and AI-generated addition.

Domain knowledge within the team or among stakeholders is particularly important because AI does not reliably know specific domain logic. Domain experts, support staff, testers, product owners, compliance officers, or experienced users can help classify AI-generated hypotheses professionally and identify blind spots.

All of the sources mentioned have limits. Support tickets only show reported problems. Analytics data show behavior, but not automatically causes. Interviews show the perspectives of individual people. Domain knowledge can be valuable, but it also contains assumptions. AI can structure such sources, but it cannot automatically check whether they are complete, representative, or correctly interpreted.

3.8 GenAI for structuring existing information

For GenAI to structure existing information meaningfully, this information must be prepared. It must be checked which data may be used. Personal, confidential, or sensitive data must not be entered into AI systems without reflection. Depending on the context, data must be anonymized, pseudonymized, shortened, or replaced with synthetic examples.

AI can summarize and cluster existing user feedback, customer feedback, support hints, interview notes, observations, or usage data, and turn them into initial hypotheses. The quality of this structuring depends directly on the quality, completeness, and admissibility of the inputs. Poor, one-sided, outdated, unclear, or inadmissible data lead to correspondingly uncertain results.

Clustering can reveal topics such as login problems, error messages, navigation, performance, comprehension problems, or missing functions. Such clusters, however, are only a structuring aid. The categories suggested by AI must be reviewed because they can be too coarse, too specific, or professionally unsuitable. Minority topics can also become invisible through clustering even though they are professionally, legally, or commercially relevant.

Initial problem areas can be derived from clusters. Note that not every cluster already describes a precise usability problem. From a cluster such as “many tickets about passwords”, it must first be worked

out more precisely whether it is about password rules, the reset process, an error message, email delivery, account lockout, or trust.

AI can also point out recurring wordings, emotional signals, contradictions, or open questions. An important benefit is not only generating answers, but recognizing missing information, implicit assumptions, and further clarification needs. From this, research questions, clarification questions for stakeholders, or test questions for later usability tests can emerge.

3.9 From information to usability problem areas

A problem area describes an area in which users repeatedly experience difficulties, uncertainties, errors, or abandonments. It should not be prematurely formulated as a solution. A statement such as “We need a new wizard” already describes a possible solution, but not yet the underlying problem.

A problem-oriented wording is better, for example: “In the current process, users do not understand which entries are required in step two.” This wording shows where the difficulty lies without already determining how it should be solved.

A good problem area describes, where possible, the user group, task, situation, observed or suspected problem, and possible impact. This makes it usable for requirements, prototyping, and testing. General topics such as “navigation”, “password”, or “onboarding” must therefore be further specified before they can be worked on as usage-related problem areas.

Such problem areas can be derived with AI support from existing information such as support tickets, feedback, observations, analytics data, or domain knowledge. AI can merge individual hints and suggest wordings. The team must check whether the suggested problem areas are supported by the available evidence, sufficiently concrete, and still formulated solution-neutrally.

In discovery, a distinction must also be made between symptoms and causes. A symptom can be that users abandon a process. Possible causes can be unclear mandatory fields, lack of trust, professional uncertainty, technical errors, or unsuitable system feedback. AI can suggest possible causes, but these remain hypotheses.

Problem areas can already be roughly classified. Criteria include frequency, user impact, risk, business relevance, strategic importance, regulatory relevance, or proximity to implementation. AI can support an initial sorting. The actual assessment, however, must be done by the team, because AI does not automatically know business goals, risks, regulatory requirements, technical dependencies, and organizational constraints.

3.10 Hypotheses instead of certainties

A hypothesis is a verifiable assumption. It describes something that could be plausible but is not yet sufficiently substantiated. Discovery often produces hypotheses about user goals, problems, causes, expectations, or possible solutions.

AI-generated user assumptions are therefore not to be treated as reliable findings. An AI statement such as “users prefer step-by-step guidance” is not a finding without data. It can, however, be formulated as a hypothesis: “We assume that users need step-by-step guidance in the current process because they cannot see the overall process.”

A good hypothesis is concrete, verifiable, and tied to a user group, task, or situation. “The interface is bad” is not a usable hypothesis. “First-time users do not find the export because the function is only available in a context menu” is easier to verify.

Participants should be able to use AI to sharpen unclear assumptions: Who is affected? Which task is relevant? In what situation does the problem occur? What cause is suspected? Which observation or test could verify the assumption?

Hypotheses are to be distinguished from findings. A finding arises through reliable observation, data analysis, evaluation, usability testing, or professional review. A hypothesis, by contrast, is a possible explanation that still needs to be verified. This distinction is central because AI output is often worded as if assumptions were already reliable findings.

Hypotheses should be documented so that it remains traceable in the further development process which assumptions underlie a story, a prototype, or a decision. A hypothesis can later be confirmed, discarded, or adjusted.

3.11 Using personas and usage scenarios with caution

Personas and usage scenarios can help make user perspectives more tangible within the team. A persona describes typified characteristics, goals, needs, and conditions of a user group. A scenario describes a concrete usage situation or task.

Their value lies in making discussions more concrete and considering requirements from the user perspective. Instead of talking generally about “the users”, a team can ask more precisely what role a user group has, what goal it pursues, what prior experience it brings, and in what situation a function is used.

AI can generate personas and scenarios quickly. Without reliable data, however, such artifacts are problematic. They can appear professional but rest on assumptions, stereotypes, or generic patterns. AI-generated personas and scenarios should therefore only be used as hypotheses or as a basis for discussion as long as they are not supported by real data, domain knowledge, interviews, observations, or other reliable sources.

A data-based persona rests on interviews, observations, usage data, or reliable domain knowledge. A hypothetical persona, by contrast, serves to reveal assumptions. Both forms can be useful, but they must be clearly distinguished and labeled.

For every persona, it must be checked which statements are substantiated and which are only assumed. Statements about goals, tasks, problems, domain knowledge, frequency of use, constraints, or expectations should be traced back to comprehensible sources wherever possible. This prevents AI-generated personas from being treated as facts within the team.

AI-generated usage scenarios are also to be understood as hypotheses. They are useful when they generate concrete questions for the backlog, a prototype, or a test. Participants should check scenarios for plausibility: Does the task fit the user group? Is the context realistic? Are important constraints considered? Which assumptions would need to be verified through users, domain experts, or data?

3.12 When real users, domain experts, or additional research activities are required

AI can structure and summarize existing information and derive initial hypotheses. However, it has no experience of its own with how real users work, think, hesitate, make mistakes, or make decisions in a concrete situation. When the central question is how users actually behave, AI-based assumptions are not sufficient.

Additional research activities are particularly important in cases of unclear user goals, contradictory feedback, high uncertainty, safety-critical functions, new target groups, professionally complex processes, or high-impact decisions. In such cases, real users, domain experts, observations, interviews, usability tests, or additional data sources may be required.

Domain experts can provide important hints about domain rules, risks, workflows, exceptional situations, or organizational conditions. These include, for example, internal subject-matter experts, support staff, trainers, compliance officers, or particularly experienced users. Their expertise is especially relevant when AI produces generic assumptions, domain language remains unclear, or regulatory requirements are affected.

Real users are required when assumptions about behavior, understanding, expectations, mental models, or actual usage situations are to be verified. AI can simulate possible user reactions and thereby generate hypotheses. Such simulations, however, are not validation, because they observe no real behavior, no genuine hesitation, no actual erroneous actions, and no situational influences.

Additional research activities do not always have to be large-scale. Short interviews, observations, internal domain reviews, short usability tests, follow-up questions to support, or analysis of individual user paths are all possible. GenAI can prepare such activities, for example through interview questions, observation guides, hypothesis lists, or test tasks. Execution, assessment, and responsibility remain human tasks.

3.13 Turning discovery findings into usable artifacts

Discovery results should be worded so that they can be used in the further agile development process. They can be turned into problem hypotheses, product assumptions, backlog items, prototype questions, acceptance criteria, or test tasks.

A problem area can become a backlog item when it is clear which user problem is to be addressed, which assumption lies behind it, and which next clarification or implementation step makes sense. It is important not to formulate backlog items prematurely as finished solutions. If the cause, user goal, or suitable solution is still uncertain, a backlog item can initially also be a clarification, prototyping, or testing task.

If it is unclear which solution is suitable, discovery findings can be translated into prototype questions. It must be clarified which assumption is to be revealed or verified and which user reaction is relevant. AI can help derive such questions from hypotheses. The team must check whether the suggested prototype actually addresses the relevant assumption or merely visualizes a superficial solution.

Uncertain assumptions can also be turned into verifiable test questions. From the assumption “users do not understand the next step”, the usability test question can arise, for example: “After completing step 1, do users recognize what they need to do in step 2?” This translates a vague suspicion into a concrete question that can be verified through observation, prototyping, or usability tests.

Discovery results should be documented in a traceable way. This includes which sources were used, which AI support was employed, which hypotheses emerged, which assumptions remain uncertain, and which next steps are proposed. AI can support the documentation. The team, however, must check that the documentation is professionally correct and does not suggest more certainty than actually exists.

3.14 AI-assisted discovery and human review

In discovery, AI is particularly good at structuring existing information and making it more accessible.

AI is good at	Clustering support tickets and feedback; summarizing interview notes; pointing out recurring topics; formulating possible user problems; deriving hypotheses and research questions; generating usage scenarios as a basis for discussion; structuring discovery results for backlog, prototyping, or tests.
Limitations and typical risks	Does not replace real users and does not know their goals, expectations, and barriers from its own experience; cannot reliably judge the representativeness of data and carries forward the biases of one-sided feedback; it becomes problematic when AI assumptions are treated as reliable findings, synthetic personas appear as research results, symptoms are prematurely interpreted as causes, or solutions are formulated before the problem is understood.
Requires human review	Which user group is actually meant; which user goal or task is at the center; in which context of use the problem occurs; which data support the assumption; which statements are substantiated and which are hypothetical; whether domain experts, real users, or additional research activities are required.

AI-assisted discovery is helpful when it reveals assumptions, sharpens questions, and makes next steps actionable.

4 Backlog Refinement and Usability

Teaching time: approx. 120 minutes

4.1 Core idea of this chapter

This chapter covers backlog refinement as the central point in the software development process where usability can be influenced early. Many usage problems do not first arise in design or implementation, but already where requirements are described too generally, too functionally, or too strongly from a system perspective.

Backlog items, user stories, and acceptance criteria determine what the team pays attention to during implementation. If only the function is described there, user goal, context of use, understandability, system states, error cases, accessibility, expectable behavior, and usability often remain invisible.

GenAI can help uncover such gaps, for example unclear wording, missing usability aspects, happy-path fixation, missing states, edge cases, or accessibility aspects. AI-generated suggestions, however, are not binding requirements. They must be professionally reviewed, prioritized, and adapted to product, user group, and context of use.

The goal of this chapter is to treat usability in the backlog not as an additional design task, but as part of professional requirements work.

4.2 Contribution to the business outcomes

This chapter particularly supports business outcomes BO 4.1 to BO 4.4. The complete and authoritative wording of the business outcomes can be found in section 0.9.

4.3 Learning objectives

LO	Learning objective	K
LO 4.1	Be able to reproduce the terms usability requirement, user story, acceptance criterion, Definition of Ready, and Definition of Done.	K1
LO 4.2	Be able to explain why usability requirements are part of professional requirements work.	K2

LO	Learning objective	K
LO 4.3	Be able to describe typical implicit usability requirements, such as understandability, error tolerance, system feedback, system states, and accessibility basics.	K2
LO 4.4	Be able to analyze user stories from a usage perspective, in particular with regard to user goal, context of use, and expected behavior.	K3
LO 4.5	Be able to use GenAI to check backlog items for unclear wording, missing usability aspects, and happy-path fixation.	K3
LO 4.6	Be able to complement acceptance criteria with usability aspects such as error messages, input validation, system states, keyboard operability, or understandable system feedback.	K3
LO 4.7	Be able to explain usability-relevant criteria for Definition of Ready and Definition of Done.	K2
LO 4.8	Be able to professionally assess AI-generated suggestions for requirements and acceptance criteria and decide whether they must be adopted, adapted, or excluded with justification.	K3
LO 4.9	Be able to explain why AI cannot determine binding prioritization or professional necessity in the specific product context.	K2

4.4 Key terms

Usability requirement, user requirement, user story, backlog item, acceptance criterion, Definition of Ready, Definition of Done, user goal, context of use, implicit requirement, quality criterion, error tolerance, system feedback, input validation, validation need, state / system state, edge case, happy path, accessibility basics, expectable behavior, AI-assisted refinement, hypothesis, evidence, professional review, prioritization.

4.5 Backlog refinement as a translation point

Backlog refinement forms the transition between problem understanding and implementation. In this phase, ideas, requirements, hypotheses, and product decisions are sharpened so that they become workable in implementation. This is often where it is decided whether usability and the usage experience are deliberately considered or whether understandability, states, error cases, and context of use remain unattended.

If discovery findings are not turned into clear backlog items, they remain non-binding. If user stories contain only technical or functional aspects, the team lacks indications of how the solution should work for users. If acceptance criteria describe only the successful flow, error cases, alternative states, and understandability remain open.

A good refinement therefore clarifies not only what is to be built, but also for whom, in what context of use, with what goal, and under what quality conditions. Usability requirements are not an addition outside the requirements; they describe quality aspects of use.

GenAI can review backlog items from an additional perspective. It can point out missing user goals, unclear wording, missing acceptance criteria, happy-path fixation, unconsidered error cases, or missing accessibility aspects. The professional decision remains with the team.

4.6 Usability requirements as part of professional requirements work

Functional requirements describe what a system should do. Usability requirements describe under which conditions this performance can be used effectively, efficiently, and with satisfaction by specific users in a specific context of use. This includes understandability, conformity with user expectations, error tolerance, clear system feedback, appropriate system states, and accessibility basics.

Functional requirements and usability requirements belong together. For a search, the pure function “display search results” is not enough. Usage quality also covers understandable inputs, sensible sorting, empty results, loading states, filter logic, keyboard operability, and clear system feedback.

Usability requirements arise from user goal, task, and context of use. A user calmly editing a personal profile needs different support than a user making a critical decision under time pressure. An occasional user needs different orientation than a daily professional user.

Backlog items should therefore describe not only which function is needed, but also which goal is to be achieved with it. From a story such as “As a caseworker, I want to filter open cases so that I can process urgent applications first”, usability requirements arise for sorting, priority display, filter state, empty results, and feedback when filters are changed.

Usability requirements should be worded so that they can be verified. General statements such as “the interface should be intuitive” are too imprecise. Better are criteria that can be verified or validated, such as understandable error messages, visible system states, keyboard operability, recognizable mandatory fields, or clear feedback after saving.

4.7 Making implicit usability requirements visible

Many usability requirements are not explicitly formulated because they seem self-evident. Users expect understandable error messages, clear system feedback, comprehensible states, readable texts, and

expectable system behavior. Precisely because these expectations seem self-evident, they are often not described in requirements.

Typical implicit usability requirements concern understandability, system feedback, error tolerance, orientation, consistency, system states, and accessibility basics. These include recognizable mandatory fields, understandable validation, loading states, empty states, error states, unambiguous confirmations, keyboard operability, and visible focus states.

If such requirements are not made visible, they are interpreted differently or overlooked during implementation. This leads to inconsistent interfaces, unclear flows, missing error states, or poorly usable components.

GenAI can help make implicit requirements visible. A backlog item can be specifically checked for which usability aspects are missing, which states should be considered, which error cases are likely, and which questions remain open before implementation.

AI can name typical gaps, for example: What happens with invalid input? What does the user see while loading? What happens if no data is available? How is success confirmed? How is an error corrected? What information does the user need before, during, and after the action? Such questions support the refinement, but they do not replace the professional decision on which requirements are actually relevant.

4.8 Reviewing user stories from a usage perspective

A user story usually describes role, goal, and benefit. The familiar form “As a … I want … so that …” can be helpful when it is not used purely formally. What matters is that the story expresses a genuine user goal.

A weak story often describes only a system function, for example: “As a user, I want to have a button so that I can save.” From a usage perspective it is more relevant which goal is to be achieved, for example: “As a user, I want to save my changes and receive an unambiguous confirmation so that I can be sure my entries have been accepted.”

During refinement, it should be checked whether the user goal is clear. If the goal is missing, often only a system function is implemented. AI can help derive possible user goals from a story. The team must check whether these goals apply, especially for internal domain applications, regulated processes, or specialized workflows.

The context of use must also be sufficiently clarified. Relevant questions are: How often is the function used? Under what time pressure? On which device? With what prior knowledge? Which errors would be critical? Which data or preconditions exist? This information does not always have to be fully contained in the story, but it should be known in the refinement or documented as an open question.

User stories should also not describe only the successful flow. Expected behavior also covers system feedback, validations, error messages, states, and other relevant system reactions. When users submit a form, saving is not enough. The system should indicate whether the operation was successful, which inputs are invalid, whether data is still being processed, or whether a later retry is necessary.

4.9 Reviewing backlog items with GenAI

GenAI can be used to check backlog items for unclear wording, missing usability aspects, and happy-path fixation. This support is particularly useful for familiar patterns such as forms, login, search, upload, tables, filters, notifications, or checkout processes.

Backlog items often contain unclear terms such as “simple”, “intuitive”, “fast”, “clear”, or “user-friendly”. Such terms can be meaningful, but without concretization they are not verifiable. AI can flag unclear wording and suggest more precise alternatives. The team decides which criteria are relevant and realistic.

AI can also check whether user goal, context, system feedback, error cases, states, accessibility, consistency, and expectable behavior are missing. One should not ask generally whether a story is “good”, but specifically which usability requirements are missing, which acceptance criteria cover only the happy path, which states must be considered, and which accessibility basics are relevant.

Many backlog items describe only the successful flow. Usability problems, however, often arise in deviations: invalid inputs, missing data, permission problems, network errors, slow response times, duplicate actions, or abandonment of a process. AI can suggest typical non-happy-path situations. Not all of them need to be implemented, but they should be deliberately checked.

A good refinement also uncovers open questions. The point is not to answer all questions immediately, but to document uncertainties deliberately. Open questions can lead to clarification tasks, spike stories, prototypes, or test tasks.

4.10 Complementing acceptance criteria with usability aspects

Acceptance criteria describe when a backlog item counts as fulfilled. They are particularly important for usability because they determine which aspects are checked during implementation. If acceptance criteria contain only technical success conditions, usability is not systematically checked.

Acceptance criteria should therefore also cover user behavior, understandability, system feedback, error cases, system states, and accessibility basics. Examples are an unambiguous confirmation after saving, an understandable correction instruction for invalid input, retention of already entered data after an error, a visible loading state for longer processing, or an explanatory empty state for zero results.

System states are not merely technical details. Loading state, empty state, error state, success state, disabled state, missing permission, incomplete data, or aborted actions directly influence whether users understand a flow and can correct errors.

Accessibility requirements should be included in acceptance criteria when they are relevant for the respective backlog item. These include keyboard operability, visible focus, understandable labels, sufficient contrast, semantic structure, and understandable error messages. Whether a specific WCAG conformance level is required must be determined depending on product, target group, legal framework, and organizational rules.

AI can suggest such criteria. Complete accessibility conformance or professional correctness, however, cannot be derived from them. The criteria must be reviewed by the team and adapted to product, context of use, and quality requirements.

4.11 Definition of Ready and Definition of Done with a usability focus

The Definition of Ready describes when a backlog item is sufficiently prepared to be taken into implementation or into a sprint. Usability criteria can help recognize unclear or prematurely formulated stories.

Possible usability criteria for Ready are a described user goal, a known user group, a sufficiently clarified context of use, documented open questions, central states and error cases, relevant accessibility basics, and a clarified validation need.

The Definition of Done describes when an increment is considered complete. Usability-relevant criteria can ensure that not only functionality but also usage quality has been considered. These include fulfilled acceptance criteria including usability aspects, relevant system states, understandable error messages,

verified keyboard operability, consistent UI texts, basic accessibility checks, humanly reviewed AI results, and documented open usability risks.

Definition of Ready and Definition of Done are team agreements. For UX, it is decisive that they fit the product type, team maturity, and risk situation. A small internal tool needs different criteria than a public e-commerce system or a safety-critical domain application.

AI can generate suggestions for DoR and DoD criteria. The selection, however, must fit the specific product, context of use, quality ambition, and risk.

4.12 Professionally classifying AI-generated requirements

AI-generated requirements and acceptance criteria are initially raw material. They can be useful, but also excessive, generic, unsuitable, or professionally wrong. A common mistake is adopting AI suggestions in full because they sound formally good.

Participants should check AI suggestions against clear criteria: Does the suggestion fit the user goal? Does it fit the context of use? Is it professionally correct? Is it verifiable? Is it relevant for implementation? Is it necessary or merely desirable? Does it create disproportionate effort?

Relevance and priority are to be distinguished. Not every relevant usability aspect has the same urgency. A missing visible focus can be critical for keyboard users. An easily improvable help text can be important but less urgent. A misleading error message in a critical process can be severe.

AI can suggest priorities, but it does not automatically know business goals, risks, user numbers, regulatory requirements, or technical dependencies. It also cannot determine binding professional necessity in the specific product context. Prioritization and responsibility remain the team's task.

Professional correctness must be checked particularly critically for domain-specific processes, legal requirements, medical applications, financial systems, or internal domain logic. A suggestion that would make sense in a standard application can be unsuitable in a regulated or safety-critical process.

AI also tends to generate extensive criteria lists. These can be helpful, but they can also overload a backlog item. Good refinement decides what is required in the next implementation, what can be worked on later, and what is deliberately not part of the item.

4.13 AI-assisted refinement and human review

In backlog refinement, AI can be used as a review and structuring tool.

AI is good at	Checking user stories for user goal and context of use; flagging unclear wording; naming missing usability aspects; suggesting acceptance criteria; adding states and error cases; uncovering happy-path fixation; suggesting accessibility basics; formulating open questions; preparing DoR and DoD criteria.
Limitations and typical risks	Does not decide in a binding way which requirements are professionally necessary, commercially prioritized, or sufficient from a regulatory perspective; the product context is only as good as the information provided, and missing or incorrect input still leads to plausible suggestions; typical risks: acceptance criteria adopted without review, overloaded backlog items, professionally wrong assumptions, superficially treated accessibility, edge cases that remain unconsidered, overly generic DoR or DoD criteria, usability requirements not worded in a testable way.
Requires human review	Is the user goal clearly described; is the context of use sufficiently known; are important states and error cases considered; is the system feedback understandable and actionable; are accessibility basics relevant and appropriately considered; are acceptance criteria verifiable; are AI suggestions professionally correct, prioritized, and implementable.

What remains decisive is that the team adopts, adapts, or excludes AI suggestions deliberately and with justification.

5 Planning and Team Organization

Teaching time: approx. 90 minutes

5.1 Core idea of this chapter

This chapter covers the organizational embedding of usability in the software development team. After the relevant aspects have been named in backlog refinement, they must be planned, worked on, reviewed, and accounted for in implementation.

Usability does not arise simply because a backlog item contains the relevant criteria. These criteria must be turned into concrete tasks, responsibilities, review questions, and review formats. This requires a shared understanding of which usability tasks the team can take on itself and when additional expertise is required.

GenAI can support team organization by suggesting usability tasks, creating checklists, preparing design critiques, formulating review questions, structuring meeting results, or summarizing decision options. However, AI must not become a hidden decision-making authority. Responsibility for UX quality, prioritization, professional assessment, and approval remains with the team or the responsible roles.

The goal of this chapter is to treat usability in implementation not as a diffuse expectation of “good design”, but as plannable, verifiable, and accountable teamwork.

5.2 Contribution to the business outcomes

This chapter particularly supports business outcomes BO 5.1 to BO 5.4. The complete and authoritative wording of the business outcomes can be found in section 0.9.

5.3 Learning objectives

LO	Learning objective	K
LO 5.1	Be able to name typical usability tasks in implementation, such as usability check, design critique, system state check, form test, accessibility basic check, and preparation of review formats or review questions.	K1
LO 5.2	Be able to explain why usability tasks should be made visible in planning and not merely expected implicitly.	K2

LO	Learning objective	K
LO 5.3	Be able to describe the contributions of typical roles in agile software development teams to UX quality, such as product owners, development, testers, quality assurance, Scrum Masters, agile coaches, and, where available, UX professionals.	K2
LO 5.4	Be able to organizationally plan simple usability activities in implementation planning.	K3
LO 5.5	Be able to explain where agile software development teams can take on simple usability tasks themselves and where additional expertise is required.	K2
LO 5.6	Be able to describe design critiques as a structured format for assessing drafts.	K2
LO 5.7	Be able to use GenAI to prepare usability meetings, review questions, checklists, and decision options.	K3
LO 5.8	Be able to document the use of AI in usability-relevant team activities in a traceable way, in particular purpose, inputs, adopted results, and human review.	K3

5.4 Key terms

Sprint planning, usability task, team responsibility, product owner, developers, testers, quality assurance, Scrum Master, agile coach, UX professional, design critique, usability check, heuristic check, heuristic evaluation, form test, system state check, accessibility basic check, review question, review format, decision option, facilitation, responsibility, AI-assisted preparation, documentation, human review.

5.5 Planning as the transition from backlog to teamwork

Agile planning translates selected backlog items into concrete work for implementation. Sprint planning is a common format for this. If usability aspects became visible in the backlog, they must be turned into tasks, responsibilities, and checks. Otherwise, usability criteria remain formally present but are not actively worked on during implementation.

Agile planning is therefore not just effort estimation and technical decomposition. It is also the point at which usability work is made visible. An acceptance criterion on understandable error messages should, for example, lead to tasks: formulating the error message, implementing it, testing it, and, where necessary, checking it from a usability perspective.

In agile teams, UX quality arises through several roles and activities. Product owners influence UX through goal clarification, prioritization, and acceptance criteria. Developers influence UX through

interaction logic, states, components, error messages, performance, and accessibility. Testers and quality assurance reveal usability problems by checking behavior, edge cases, understandability, and error states. Scrum Masters and agile coaches can help embed usability activities in team processes. UX professionals, where available, contribute methodological, design, and evaluation expertise.

UX is thus not the task of a single role alone. Even though specialized UX competence is valuable, all team members must understand how their decisions influence use.

5.6 Making usability tasks visible in planning

Usability tasks are implicitly expected in many teams. It is assumed that someone “keeps an eye on” whether an interface is understandable, whether error messages fit, or whether a form is easy to operate. This implicit expectation rarely leads to clear responsibility.

If usability tasks are not planned visibly, typical gaps arise: error messages are worded late, empty states are missing, loading states are not designed, keyboard operability is not checked, acceptance criteria are tested only functionally, or AI-generated drafts and texts are not critically assessed.

Typical usability tasks in agile implementation are usability checks, design critiques, heuristic checks, form tests, system state checks, accessibility basic checks, review preparations, and the review of UI texts. These tasks do not have to be large. Small, clearly described activities are often enough to improve UX quality considerably.

Usability tasks should be derived from user stories, acceptance criteria, Definition of Ready, and Definition of Done. If an acceptance criterion demands an understandable error state, this can give rise to a text-wording task, an implementation task, a testing task, and a review question.

GenAI can help derive usability-relevant tasks from a backlog item. It can suggest which usability checks make sense, which states should be considered, or which review questions can be asked. The suggestions, however, must be adapted to sprint goal, team capacity, technical dependencies, and product priorities.

5.7 Roles and responsibility in the software development team

Usability is not exclusively the task of a single dedicated role. It arises through decisions in product ownership, requirements, design, development, testing, quality assurance, and operations. A dedicated UX role can contribute important expertise, but it does not replace the team's responsibility for concrete product quality.

The product owner influences UX above all through goal clarification, prioritization, and acceptance. Backlog items should not only describe functions, but also name the user goal, context, and relevant quality criteria. The product owner co-decides which usability problems are prioritized and which risks are deliberately accepted.

Developers influence UX through the technical implementation. Many usability properties arise directly in code: input validation, loading behavior, focus order, error messages, keyboard operability, component logic, responsiveness, performance, and system states. AI-assisted coding tools can support this, but they must be checked against the usability requirement, the architecture, and the context.

Testers and quality assurance can make UX quality demonstrable by checking not only functional correctness, but also understandability, system states, error cases, input validation, keyboard operability, and user guidance. A form is not only to be checked for whether invalid inputs are technically rejected, but also for whether the error message, field reference, correction help, and retention of already entered data are usable.

Scrum Masters and agile coaches influence UX not through professional design decisions, but through the way the team works. They can point out when usability tasks regularly remain unplanned, when review formats become too opinion-based, or when learning loops from user feedback are missing.

UX professionals, where available, contribute methodological, design, and evaluation expertise. They can professionally deepen discovery, prototyping, research, evaluation, usability tests, accessibility, or design systems. In many teams, however, such a role is not permanently available. Agile teams must therefore be able to take on simple usability activities themselves while recognizing when additional expertise is required.

In connection with GenAI, clarifying responsibility becomes particularly important. AI can generate suggestions, structure tasks, formulate usability criteria, or prepare decision bases. However, it takes no responsibility. Teams should therefore clarify who operationally works on an AI-assisted task, who decides professionally, who must be involved, and who is informed. The RACI principle can offer simple orientation here; AI itself takes no role in the responsibility matrix.

5.8 Organizationally planning simple usability activities

Development teams can integrate simple usability activities into planning and implementation without turning them into a heavyweight additional process. These include design critiques, heuristic checks based on defined criteria, form tests, system state checks, accessibility basic checks, and review preparations.

A design critique is a structured review format for the professional discussion of a draft. Its purpose is not to collect personal preferences, but to check a draft against comprehensible usability criteria, such as user goal, understandability, conformity with user expectations, consistency, relevant states, accessibility, and risks.

A heuristic check examines an interface based on defined usability principles or design criteria. Within a sprint, a simplified heuristic check can help uncover coarse usability problems early. It replaces neither a complete heuristic evaluation nor a usability test with real users.

A form test can check whether labels are understandable, mandatory fields are recognizable, validation works sensibly, error messages help, and inputs are retained after errors. A system state check examines relevant states such as loading, empty, error, success, disabled, missing permission, incomplete data, network errors, or aborted actions.

An accessibility basic check verifies selected requirements that are relevant for many interfaces, such as keyboard operability, visible focus, programmatically associated labels, sufficient contrast, semantic structure, and understandable error messages. It replaces no complete audit against WCAG or EN 301 549, but it helps identify common barriers early.

UX should also not be mentioned only in passing in review formats. A review preparation can clarify which user goal is supported, which states have been implemented, which error cases have been checked, which assumptions remain open, and which user reactions must be validated later.

5.9 Where teams can act themselves and where expertise is necessary

Agile software development teams can take on many simple usability tasks themselves if they know fundamental criteria. These include initial usability checks, simplified heuristic checks, form tests, understandable error messages, system state checks, accessibility basic checks, design critiques, and the critical assessment of AI-generated suggestions.

These activities do not require full UX specialization. They do, however, require a shared basic understanding, clear criteria, and the willingness not to treat UX as a matter of taste.

Additional expertise becomes necessary when uncertainty, risk, or complexity are high. This concerns, for example, extensive user research, safety-critical systems, complex domains, legal questions, in-depth accessibility audits, strategic usability decisions, design system governance, or critical conversion optimization.

Whether a team can act itself or should call in expertise depends on several factors: possible consequences of erroneous operation, specialization of the domain, uncertainty about user needs, legal or regulatory requirements, indications of severe usage problems, the team's experience with the method, and the need for validation with real users.

AI can prepare this decision, for example through checklists or risk questions. The decision remains with the team or the organization, both organizationally and professionally.

5.10 Design critiques as a structured format

Design critiques help teams assess UI drafts, prototypes, or implemented functions in a structured way from the perspective of usability, accessibility, and expectable behavior before they are implemented or developed further. They are particularly important when AI quickly generates several variants. Without criteria, there is a risk that the visually most convincing variant is chosen even though it is not the most suitable one from a usage perspective.

A design critique does not merely ask whether a draft is liked. It checks whether the draft supports the user goal in an understandable, expectation-conformant, consistent, error-tolerant, and accessible way.

A good design critique uses clear assessment criteria. These include user goal, context of use, visual hierarchy, understandability, consistency, cognitive load, relevant states, error prevention, accessibility basics, and expectable behavior.

Different roles can contribute in a design critique. Product owners pay attention to user value and priority. Developers check technical feasibility and component logic. Testers pay attention to verifiable states and error cases. UX professionals, where available, check design, interaction, and method. Scrum Masters, agile coaches, or people responsible for team processes can structure the session.

A design critique should lead to clear results: identified problems, necessary changes, open assumptions, later checkpoints, and decisions made. AI can help prepare criteria, review questions, or summaries. Assessment and decision remain with the team.

5.11 GenAI for preparing and structuring team activities

GenAI can prepare and structure team activities on usability when goal, context, and intended outcome are clearly defined. These include design critiques, refinement sessions, review preparations, retrospectives, or decision workshops.

AI can suggest agendas, guiding questions, checklists, and role hints. From a backlog item or prototype, AI can generate review questions, for example on missing states, invalid inputs, the next step, the success message, keyboard operability, or open assumptions. Such checklists should remain short and context-related, because overly long AI-generated lists can overload meetings.

AI can also structure decision options. When several UI variants or solution paths exist, AI can summarize advantages and disadvantages, risks, open questions, and possible assessment criteria. It thereby provides structure for a decision, but it does not decide itself.

AI can also summarize meeting results, findings, or decisions. These summaries must be reviewed. Especially for decisions, it is important that AI does not add resolutions that were not made, does not conceal uncertainties, and does not word hypotheses as facts.

5.12 Documenting the use of AI

When AI is used in usability-relevant team activities, it should remain traceable where and for what purpose it was used. This is important because AI results can later flow into requirements, designs, code, tests, or product decisions.

Without documentation, it can become unclear whether a statement comes from user data, domain knowledge, a team decision, or an AI suggestion. This increases the risk that AI-generated content is later treated as a reliable basis.

The documentation should remain purposeful and lightweight. Important elements are the purpose of the AI use, inputs or data sources used, essential AI results, adopted results, discarded or adapted suggestions, human review, and open assumptions or risks.

Documentation supports AI governance. It shows how AI was used within the team, which responsibility remained with humans, and how quality was checked. For AI-assisted usability activities, it should additionally be recorded who created or prepared the AI output, who professionally reviewed it, and who made the decision on adoption, adaptation, or exclusion.

For agile teams, this is particularly relevant when AI output flows into backlog items, acceptance criteria, UI drafts, tests, or code. The closer an AI result gets to productive implementation or a professional decision, the more important traceability becomes.

5.13 AI-assisted team organization and human review

In team organization, AI is particularly good at supporting preparation, structuring, and documentation.

AI is good at	Deriving usability tasks from backlog items; creating checklists; formulating review questions; structuring design critique criteria; summarizing decision options; documenting meeting results; logging AI use in a traceable way.
Limitations and typical risks	Takes no team responsibility and does not decide on roles, priorities, or conflicts; does not sufficiently know team dynamics, organizational culture, capacities, and political conditions; typical risks: checklists adopted without review, overloaded planning or review formats, unclear responsibilities, inaccurate summaries, hypotheses documented as resolutions, undocumented AI use, external expertise called in too late.
Requires human review	Which usability tasks arise from the backlog items; whether these tasks are scheduled in agile planning; who takes on which responsibility; which usability checks are necessary for the implementation goal; which AI-generated checklists and review questions are relevant; which decisions were actually made; whether additional usability, accessibility, research, or domain expertise is required.

AI can provide structure, but it cannot take on responsibility, priority, or professional decisions.

6 UI, Information Architecture, States, and AI-Generated Drafts

Teaching time: approx. 135 minutes

6.1 Core idea of this chapter

This chapter covers the design and early concretization of solutions in implementation. The focus is on UI drafts, information architecture, navigation, interaction flows, system states, prototypes, and AI-generated suggestions for user interfaces. In this syllabus, these artifacts are not understood as pure design showcases, but as means of making usability and interaction quality visible, discussable, verifiable, and, where necessary, testable.

A terminological distinction is needed here: software usability refers to the fitness for use of software in a specific context of use. UX refers to the overall experience of using a product or system. UI refers to the concrete user interface with screens, forms, navigation, controls, texts, and interaction elements. UI design shapes this interface. Frontend refers to the technical implementation of the visible and operable parts of a digital system. Good UI drafts are therefore not to be judged only visually, but by whether they support user goals, context of use, understandability, expectable behavior, accessibility, and relevant system states.

GenAI can generate wireframes, mockups, screen variants, interface texts, navigation models, information structures, clickable prototypes, or frontend-oriented structures. This shrinks the distance between idea, draft, and testable solution. At the same time, AI-generated drafts can appear professional and complete even though central usability questions remain unresolved.

The goal of this chapter is to critically assess AI-generated UI drafts and prototypes based on fundamental usability and human-factors criteria. These include user goal, context of use, human perception, visual hierarchy, mental models, understandability, cognitive load, consistency, accessibility, and expectable behavior.

6.2 Contribution to the business outcomes

This chapter particularly supports business outcomes BO 6.1 to BO 6.4. The complete and authoritative wording of the business outcomes can be found in section 0.9.

6.3 Learning objectives

LO	Learning objective	K
LO 6.1	Be able to reproduce fundamental UI design terms such as visual hierarchy, layout, spacing, typography, consistency, interaction, state, and control.	K1
LO 6.2	Be able to explain why UI design foundations and human-factors criteria are necessary to assess AI-generated drafts.	K2
LO 6.3	Be able to explain the importance of information architecture, navigation, naming, mental models, and findability for usability.	K2
LO 6.4	Be able to name typical prototype types and their purpose, in particular wireframe, mockup, low-fidelity prototype, high-fidelity prototype, clickable prototype, functional prototype, horizontal prototype, vertical prototype, and scenario prototype.	K1
LO 6.5	Be able to describe how AI-assisted rapid prototyping can be used in agile development processes.	K2
LO 6.6	Be able to assess AI-generated UI drafts based on user goal, context of use, human perception, mental models, visual hierarchy, understandability, cognitive load, consistency, accessibility, and expectable behavior.	K3
LO 6.7	Be able to explain typical strengths of AI with familiar UI patterns, such as login, form, dashboard, navigation, search, filter, or onboarding.	K2
LO 6.8	Be able to recognize typical weaknesses of AI-generated drafts, in particular missing states, edge cases, error paths, permissions, alternative usage situations, and incomplete accessibility consideration.	K3
LO 6.9	Be able to compare AI-generated UI variants based on defined usability criteria and select among them with justification	K3
LO 6.10	Be able to use prototypes as a basis for communication, learning, evaluation, and, where applicable, validation within the team.	K3

6.4 Key terms

Software usability, user experience, UX quality, interaction capability, quality in use, UI, user interface, UI design, interaction design, visual hierarchy, layout, spacing, typography, readability, consistency,

interaction, information architecture, navigation, naming, findability, tree testing, mental model, conformity with user expectations, cognitive load, Gestalt principles, wireframe, mockup, prototype, low-fidelity prototype, high-fidelity prototype, clickable prototype, functional prototype, horizontal prototype, vertical prototype, scenario prototype, rapid prototyping, state, loading state, empty state, error state, success state, disabled state, happy path, edge case, alternative usage path, UI pattern, onboarding, AI-generated draft, frontend-oriented suggestion, human-factors criteria, accessibility, human review.

6.5 From backlog item to visible solution

The step from backlog item to visible solution is also a step from abstract requirements to concrete usability: user goals, system reactions, naming, states, error paths, and operability become visible as concrete interaction for the first time.

After discovery, refinement, and agile planning, there is ideally a clear user goal, a described context of use, relevant acceptance criteria, and visible usability tasks. In implementation, these foundations are translated into concrete solution ideas. These include UI drafts, information structures, navigation models, interaction flows, system states, prototypes, and initial technical implementation considerations.

Design and prototyping are not to be understood as an isolated design phase. They serve to reveal assumptions. A backlog item can seem plausible in text, but only a draft shows whether the solution appears understandable, complete, findable, and in line with expectations.

AI can accelerate this step by generating initial UI drafts and prototypes from requirements, user stories, acceptance criteria, or descriptions. This lets teams discuss, compare, and review faster. An AI-generated draft, however, is initially a suggestion that must be checked against user goal, context of use, usability criteria, human-factors criteria, and professional requirements.

A UI draft shows which information is emphasized, which actions are offered, which terms are used, what order a flow has, and how users are guided through a task. This makes it an object of review for UX quality.

Prototypes connect team discussion, evaluation, and, where applicable, validation. They can be used for internal discussions, design critiques, stakeholder alignment, expert reviews, or usability tests. Validation, however, only arises when a concrete assumption is verified in a methodologically appropriate way, for example through real users, domain experts, usability tests, usage data, or other reliable sources.

6.6 UI design foundations for assessing drafts

In this syllabus, UI design foundations are not understood as specialized aesthetic knowledge, but as a basis for assessing usability and interaction quality. Visual hierarchy, layout, spacing, typography, consistency, and readability influence whether users perceive information, understand tasks, execute actions correctly, and can avoid errors.

Visual hierarchy describes which elements of an interface are perceived first and what importance is attributed to them. Size, position, contrast, spacing, color, typography, grouping, and motion influence what users pay attention to. AI-generated drafts can look visually modern and still have a poor visual hierarchy, for example when several actions are emphasized equally strongly.

Layout and spacing structure content. Spacing shows which elements belong together and which are separated. A consistent spacing system supports orientation and reduces visual noise. Fundamental Gestalt principles are relevant here, such as proximity, similarity, common region, closure, and good form. Elements that are close together or similarly designed are often perceived as belonging together.

Typography influences whether information can be read quickly and reliably. Relevant aspects are font size, line length, line spacing, font weight, contrast, information density, and clear distinction between different information levels. Readability is a quality criterion especially for error messages, legal notes, tables, dashboards, or complex domain information.

Consistency means that similar things look similar and work similarly. The same actions should be named the same, the same states should be displayed the same, and recurring components should behave in an expectable way. AI can produce good variants in individual screens but introduce inconsistencies across multiple screens.

Interaction describes how users act with the system and how the system responds. This includes clicks, inputs, selection, navigation, confirmation, cancelation, system feedback, and error handling. A static UI draft often shows only what something looks like. Good UX additionally requires that interaction flows, system states, and feedback are considered explicitly.

Controls carry familiar mental models. Radio buttons typically stand for a choice among mutually exclusive options, checkboxes for several independent choices, toggles for immediate switching on or off, buttons for actions, links for navigation, and tabs for related content areas. When controls are used differently than users expect, irritation, erroneous operation, or additional cognitive load arise.

6.7 Information architecture, navigation, and findability

Information architecture, navigation, and findability are central components of usability. They determine whether users find relevant content, functions, and next steps, and whether the structure of a system fits their mental models and tasks.

Information architecture describes how content, functions, and objects are structured. It influences whether users find what they need and whether they understand where they are in the system. Usability problems often arise not from individual bad screens, but from unclear structure, unexpected categories, or inconsistent terms.

A mental model describes how users believe a system, a process, or a domain object is structured and works. This concept is central for information architecture: users look for functions and information where they expect them based on their mental model. If the structure of an application primarily follows internal system logic, it can be plausible for the development team but hard to understand for users.

Navigation helps users move through a system. Good navigation answers questions such as: Where am I? What can I do here? How do I get back? What is the next sensible step? Which areas belong together? Which function belongs to which task?

Naming influences findability and understanding. A term that is self-evident within the development team can be unclear for users. Conversely, domain language can be necessary in specialized domains. AI can suggest alternative names and simplify terms. This is useful, but risky when domain language is needed with precision.

Findability describes whether users can find relevant functions, information, or objects. It depends on information architecture, navigation, search, filtering, naming, visual hierarchy, and context of use. AI can point out possible problems and prepare review questions. The assessment only becomes reliable when it is checked against user knowledge, domain knowledge, or empirical evidence.

A simple method for checking information architecture and navigation is tree testing. Here, no finished interface is tested, but a reduced, text-based navigation structure. Test participants receive typical tasks and indicate where they would expect the information or function they are looking for. AI can prepare tasks, alternative navigation labels, or initial result structures, but it does not replace testing with suitable people.

6.8 Prototype types and their purpose

Prototypes are not meant to show a seemingly finished solution as early as possible. They help make assumptions about structure, flow, understandability, interaction, system states, and context of use visible and verifiable.

A prototype is a preliminary representation or implementation of a solution. It serves to make ideas visible, discussable, verifiable, or testable. Its value does not depend on how finished it looks, but on whether it fits the question at hand.

Wireframes show structure and basic arrangement without putting visual details in the foreground. They are suitable for discussing information architecture, content, order, and basic interaction.

Mockups show a more visually elaborated picture of an interface. They usually contain colors, typography, spacing, and concrete UI elements. They are suitable for discussing visual hierarchy, consistency, readability, and understandability.

Low-fidelity prototypes have a low level of detail. They are suitable for early discussions, structural questions, initial variants, and basic flows. High-fidelity prototypes more closely resemble the later product and are suitable when concrete flows, visual hierarchy, understandability, or more realistic usage situations are to be checked.

Horizontal prototypes show many areas or screens of a system in breadth, but elaborate individual functions only superficially. Vertical prototypes work out a selected functional area or flow in more depth. Scenario prototypes represent a concrete usage path or a typical usage situation.

Clickable prototypes make flows tangible. Users or team members can navigate through screens and follow initial interaction paths. Functional prototypes behave more like real applications and can contain data, simple logic, forms, or interactions.

AI makes it easy to quickly generate detailed or functional prototypes. This does not mean, however, that a complex prototype is always sensible. The level of detail should depend on the purpose: first clarify which question is to be answered, then choose the appropriate prototype.

6.9 AI-assisted rapid prototyping in implementation

AI-assisted rapid prototyping can quickly generate initial UI drafts and prototypes from requirements, user stories, acceptance criteria, sketches, or text descriptions. This lets teams see earlier how a solution could look and work.

The faster AI leads from an idea to a seemingly finished prototype, the more important the team's deliberate review becomes. Speed does not replace clarifying user goal, context of use, states, error paths, accessibility, and professional correctness.

The advantage lies not only in speed. Drafts reveal assumptions. They show which information is needed, which steps are planned, which terms are used, which system states have been considered, and which states may be missing.

AI can generate several variants of a screen or flow. Variants can set different emphases: more guidance, fewer steps, stronger focus on data, different navigation, different language, different structure, or different interaction patterns. A variant should be selected based on defined criteria, such as user goal, context of use, understandability, visual hierarchy, cognitive load, consistency, accessibility, expectable behavior, and professional correctness.

Prompt-to-UI, prompt-to-app, and prompt-to-frontend tools can generate UI drafts, frontend-oriented structures, or functional app prototypes. This moves prototyping closer to implementation and code. The closer an AI tool gets to runnable code, the more important requirements review, usability review, accessibility review, code review, security review, and professional acceptance become.

Rapid prototyping is particularly valuable when it is clear which hypothesis or question is to be verified. Without a clear question, AI produces variants, but not automatically better product decisions.

6.10 Assessing AI-generated UI drafts

AI-generated UI drafts are not to be judged by whether they look professional, but by whether they support the intended use. What matters is whether user goal, context of use, information structure, understandability, expectable behavior, accessibility, system states, and domain rules are sufficiently considered.

The most important checkpoint is the user goal. A UI draft should not merely display functions, but support a goal. If it is unclear which goal users are supposed to achieve, the draft cannot be meaningfully assessed either. It must also be checked whether the assumed user goal is substantiated or merely suspected.

A UI draft can only be assessed in connection with context of use and task. An interface well suited for occasional users can be too slow or too explanatory for experienced professional users. A guided step sequence can support beginners but be inefficient in a frequently repeated domain task.

AI-generated drafts must be checked for whether important information is perceivable. Decisive factors are contrast, size, position, grouping, spacing, order, and visual weighting. The review should ask what stands out first, whether this is the most important thing, whether related elements are recognizably grouped, and whether the visual order corresponds to the task.

A draft should not force users to think unnecessarily. Texts, naming, order, grouping, interaction logic, and information density should be understandable. Cognitive load arises from too many options, unclear terms, missing orientation, unfamiliar patterns, memory demands, unclear system states, or contradictory signals.

Users expect familiar elements to work in an expectable way. A link behaves differently than a button. A search should deliver comprehensible search results. A filter should be visibly activatable and resettable. A save button should give feedback after a successful action.

AI-generated drafts must also be checked for basic accessibility aspects. These include contrast, focus, keyboard operability, understandable labels, semantic structure, sufficient font size, clear error messages, and information that is not conveyed by color alone. AI can provide hints, but it cannot guarantee complete accessibility.

6.11 Familiar UI patterns as a strength and risk of AI

Familiar UI patterns can support usability when they fit the context of use. However, they can also do harm when they merely feel familiar, oversimplify domain processes, or conceal necessary exceptions.

AI is particularly useful with familiar UI patterns because these are documented in many examples. Login, registration, password reset, forms, search, filters, tables, dashboards, onboarding, upload components, confirmation dialogs, or simple checkout processes often follow recurring structures.

AI can quickly reproduce such patterns and generate variants. This is helpful for teams because they do not have to start from scratch. Especially in early draft phases, AI can help point out typical elements, possible states, standard flows, and alternative forms of presentation.

This strength, however, rests on pattern recognition. AI recognizes common solutions and combines them into plausible suggestions. It does not automatically follow that a suggested pattern is suitable for the specific context of use.

Standard patterns can be unsuitable. An onboarding flow can make sense for a simple app but be an unnecessary disturbance in a domain application with experienced users. A wizard can support beginners but slow down experts. A dashboard can look attractive but emphasize irrelevant metrics.

Teams must therefore check whether a UI pattern fits the task, the user group, the frequency of use, the risk, accessibility, and the users' mental model.

6.12 States, edge cases, and alternative paths

States, edge cases, and alternative paths are central components of usability. They determine whether a system remains understandable, robust, error-tolerant, and trustworthy even outside the ideal happy path.

AI-generated drafts often show the ideal flow. Users enter correct data, the system responds quickly, permissions fit, data is available, and actions succeed. This happy path is important, but not sufficient.

Many usability problems arise in non-ideal states. When such states are missing, users are left without orientation. Error cases, empty data situations, loading times, aborted processes, or missing permissions often determine whether a system is experienced as understandable and trustworthy.

Important system states include:

- loading, empty, error, success, and disabled state,
- missing permission and incomplete data,
- interrupted connection and failed validation,
- aborted action, active selection, and active filters,
- unsaved editing.

Not all of these states are always relevant, but they should be deliberately checked.

Empty states should explain why nothing is displayed and which next action is possible. Error states should explain understandably what happened and what users can do. Permission cases should make clear why an action is not possible and which alternative may exist.

Edge cases are special or rarer situations outside the ideal flow. These include very long names, unusual data formats, missing values, duplicate records, expired sessions, parallel editing, or unexpected inputs. AI can suggest edge cases, but it does not reliably recognize which of them are relevant or critical in the specific product.

6.13 UI patterns, onboarding, and help

UI patterns, onboarding, and help are not to be understood as decorative additions. They support usability when they help users understand functions, perform tasks safely, and receive orientation or support when needed.

UI patterns are proven solutions for recurring interaction problems. Examples are tab navigation, breadcrumbs, search filters, modal dialogs, steppers, cards, toast messages, or inline validation. They help create consistent and expectable interfaces.

AI can suggest and explain UI patterns. Teams should, however, check whether a pattern fits the user goal, the task, and the context of use. A familiar pattern is not automatically suitable. A modal dialog can be helpful for a focused decision but disruptive when users need information from the background. A stepper can structure complex processes but slow down experienced users.

Onboarding supports users when entering a system or a function. AI can suggest onboarding texts, step sequences, tooltips, or help concepts. It must be checked whether onboarding is actually necessary or whether the interface itself should be made more understandable.

Good onboarding does not merely explain functions; it supports users in achieving a first meaningful goal. If a system only becomes understandable through extensive explanations, this can be an indication of structural usability problems.

AI frequently suggests help texts or explanations. These can be useful, but too many hints increase cognitive load. Progressive disclosure means showing information when it is needed. Teams should check whether an explanation is necessary or whether structure, naming, visual hierarchy, or system feedback should be improved.

6.14 Prototypes as a basis for communication, learning, evaluation, and validation

Prototypes reveal assumptions. They help software development teams talk early about user goal, flow, information structure, states, error cases, and feasibility before decisions become expensive or AI-generated suggestions are prematurely turned into code.

Prototypes make abstract requirements tangible. They help identify misunderstandings, compare variants, and concretize assumptions about structure, flow, language, or interaction. For team communication, it is important that a prototype is understood as a working tool, not as a final solution.

Prototypes also support learning. When a prototype is reviewed, questions about user goal, understandability, system states, cognitive load, mental models, and accessibility become concrete. Teams do not learn because AI generates a solution, but because they assess AI results against usability criteria and check them against context of use, requirements, and domain rules.

Prototypes can be used to verify assumptions. For this, it must be clear which assumption or research question is to be verified. A prototype without a question produces feedback, but not necessarily usable findings.

The term validation should be used carefully. A prototype is not itself the validation. It can enable or support validation when an assumption is verified in a methodologically appropriate way, for example with real users, domain experts, usability tests, or real usage data.

A prototype is also not automatically a complete specification. It can omit details, simplify, or represent only certain paths. When prototypes serve as a basis for implementation, missing system states, rules, texts, interactions, accessibility criteria, and domain requirements must be added.

6.15 AI-assisted design, prototyping, and human review

In design and prototyping, AI is particularly good at contributing to fast variant creation.

AI is good at	Generating initial UI drafts from requirements, user stories, or acceptance criteria; suggesting familiar UI patterns; drafting information structures; varying interface texts; preparing prototypes; generating review questions for design critiques and team discussions.
Limitations and typical risks	Does not guarantee that a draft fits the actual context of use; AI knows real users, domain logic, risks, and implicit requirements only if they are correctly provided and subsequently reviewed; completeness of system states, edge cases, and accessibility is not ensured; typical risks: drafts appear more finished than they are, visual attractiveness is confused with UX quality, only the happy path is considered, generic UI patterns meet unsuitable domain contexts, prototypes are confused with specifications, frontend-oriented results move toward implementation too quickly.
Requires human review	Which user goal the draft supports; whether the context of use is sufficiently considered; which assumptions are contained in the draft; whether visual hierarchy and information structure are sensible; whether terms are understandable and professionally correct; whether relevant

	system states and error cases are considered; whether accessibility basics are fulfilled; whether the prototype is suitable for the intended question.

AI results remain draft variants and hypotheses; they are reviewed by humans professionally, methodologically, and from a usage perspective.

7 Code Creation, Components, Accessibility, and Handoff

Teaching time: approx. 135 minutes

7.1 Core idea of this chapter

This chapter covers the implementation of usability requirements, UI drafts, and prototypes in code. The central question is how drafts are turned into implemented, consistent, accessible, and maintainable UI components.

A UI draft or prototype makes a solution visible and discussable. The UI implementation translates this solution into concrete components, code, states, interaction logic, semantic structure, validation, error messages, accessibility properties, and tests. UX quality therefore arises not only in the draft, but also in many implementation decisions.

AI-assisted coding tools can generate UI components, analyze existing codebases, suggest tests, prepare validation logic, write documentation, create stories for a component workshop, or provide accessibility guidance. This shortens the path from usability requirement to implemented UI.

AI-generated code, however, is not automatically production-ready, secure, maintainable, accessible, or professionally correct. What matters is that implemented UI solutions conform to usability requirements, acceptance criteria, system states, component rules, design system rules, accessibility basics, and tests. Responsibility for this review remains with the team.

7.2 Contribution to the business outcomes

This chapter particularly supports business outcomes BO 7.1 to BO 7.4. The complete and authoritative wording of the business outcomes can be found in section 0.9.

7.3 Learning objectives

LO	Learning objective	K
LO 7.1	Be able to name typical AI-assisted coding tools and fields of application for UI-related development.	K1

LO	Learning objective	K
LO 7.2	Be able to explain how AI-assisted code creation changes the interface between usability requirements, UI design, and implementation.	K2
LO 7.3	Be able to describe fundamental usability quality aspects in implementation, in particular system states, component logic, error messages, labels, focus order, keyboard operability, and expectable behavior.	K2
LO 7.4	Be able to explain the importance of consistent, reusable components for usability, maintainability, and accessibility.	K2
LO 7.5	Be able to describe the function of design systems and design tokens as the connection between design, UI implementation, and code.	K2
LO 7.6	Be able to check AI-generated UI implementations against usability requirements, acceptance criteria, system states, component rules, and design system rules.	K3
LO 7.7	Be able to use AI to prepare test cases, documentation, stories for a component workshop, review questions, or accessibility guidance.	K3
LO 7.8	Be able to name fundamental accessibility basics in implementation, in particular semantic HTML, contrast, labels, focus order, keyboard operability, and ARIA only where necessary.	K1
LO 7.9	Be able to explain risks of AI-generated implementation, in particular security vulnerabilities, accessibility problems, inconsistencies, maintainability problems, performance problems, and professionally wrong logic.	K2
LO 7.10	Be able to derive appropriate review measures for AI-generated code, in particular code review, functional tests, UI tests, security review, accessibility checks, and professional acceptance.	K3
LO 7.11	Be able to explain how AI can be used to derive recurring UI patterns, inconsistencies, and initial design system building blocks from an existing application.	K2

7.4 Key terms

AI-assisted code creation, coding agent, UI implementation, frontend, UI component, component logic, component library, design system, design token, minimal viable design system, retroactive design system, UI inventory, component inventory, inconsistency analysis, handoff, traceability, component workshop, story, semantic HTML, accessibility, accessibility basics, keyboard operability, focus order, label, ARIA, validation, input validation, error message, system state, loading state, empty state, error state, success state, disabled state, expectable behavior, code review, security review, accessibility check, professional acceptance, test case, maintainability, performance, production readiness.

7.5 From usability requirements to the implemented UI

In development processes, usability arises not only through requirements, drafts, and prototypes, but also through their concrete implementation. A well-worded user story, a clean prototype, or a convincing UI draft does not yet produce well-usable software if the implementation omits central details.

In implementation, many usability-relevant decisions become concrete: Which component is used? How does a button behave in the disabled state? How is input validation displayed? Are inputs retained after errors? Is the focus visible? Can the flow be operated without a mouse? Is a loading state displayed? Are labels correctly associated? Are error messages understandable and actionable?

AI-assisted coding tools can accelerate this work. However, they can also produce faulty or incomplete implementations. Implementation must therefore be understood as usability-relevant quality work. The central question is not only whether code works, but whether the implemented UI supports the user goal in the context of use in an understandable, accessible, consistent, and expectable way.

In this context, handoff is not merely a transfer from design to development. It is an ongoing alignment between requirements, draft, component rules, system states, interaction logic, accessibility requirements, tests, and code. Especially with AI-generated drafts or implementations, it must remain clear which parts are professionally confirmed and which AI additions are merely assumptions.

What is implemented in this chapter is checked in later reviews, tests, and evaluations. Test cases, review questions, accessibility checks, and professional acceptance must therefore already be considered during implementation. The more strongly AI supports productive implementation, the more important traceable review measures become, such as code review, functional tests, UI tests, accessibility checks, security review, and professional acceptance.

7.6 AI-assisted code creation for usability and frontend

AI-assisted coding tools support software development directly in code. These include IDE assistance systems, coding agents, code completion tools, pull request assistants, prompt-to-code tools, and systems that can suggest or execute larger changes in codebases.

Typical fields of application in the usability-related frontend are UI components, forms, input validation, system states, error messages, tests, accessibility guidance, documentation, stories for a component workshop, and refactorings. These tools can accelerate development and help uncover usability-relevant details earlier. Their output, however, remains a suggestion that must be checked against requirements, code quality, accessibility, security, maintainability, and professional correctness.

AI-assisted code creation changes the interface between UX draft, UI design, and implementation. Initial implementations can arise directly from descriptions, screens, component libraries, or existing codebases. This shortens the path from idea to implemented UI. At the same time, the risk increases that unclear requirements are translated directly into code.

If user goal, context of use, system states, error cases, permissions, or accessibility are not described, AI may fill these gaps with generic assumptions. Usability requirements, acceptance criteria, component rules, design system rules, and relevant system states should therefore be as clear as possible before code creation.

AI can produce better implementation suggestions when the context is provided precisely:

- the framework used and the existing component library,
- design system rules and design tokens,
- acceptance criteria, system states, and accessibility requirements,
- test requirements and domain constraints,
- code conventions and security requirements.

The less context is provided, the more strongly AI orients itself toward generic patterns.

AI-generated code should be treated as a suggestion, not as a finished solution. An AI-generated form can technically process inputs but contain unclear labels, poor focus order, missing error messages, or insufficient keyboard operability. AI code must therefore be checked technically, professionally, from a usability perspective, and with regard to accessibility, security, and maintainability.

7.7 Usability in implementation

In implementation, usability arises through concrete implementation details: consistent components, complete system states, semantic HTML, keyboard operability, sensible focus order, understandable labels, helpful error messages, expectable behavior, and basic accessibility.

System states are a central part of implementation. Good UX requires that users understand what is currently happening and how they can act next. This includes loading state, empty state, error state, success state, disabled state, missing permission, and failed input validation. These states must not only exist visually; they must be triggered correctly and communicated understandably.

Error messages are usability-critical. They should be understandable, concrete, and action-oriented. A good error message describes which problem occurred, where it occurred, and how users can fix it. In implementation, it is also important that error messages are associated with the correct field, remain

visible, can be captured by assistive technologies, and that inputs are not lost unnecessarily.

Labels give input fields meaning. Placeholder texts do not replace labels because they disappear while typing and are often not sufficiently accessible. Help texts can provide additional orientation but should be used in a targeted way. AI can suggest labels and help texts. The team must check whether the terms fit the users' domain language, their mental model, the task, and the context of use.

Focus order is important for keyboard use, screen reader use, and efficient operation. Users must be able to recognize which element is active. The tab order should be logical. Dialogs, menus, and modal windows must set focus correctly and return it sensibly after closing.

Keyboard operability means that interactive elements can be reached and operated without a mouse. Human review is necessary especially for custom components, modals, dropdowns, tabs, or complex forms, because AI-generated components do not always handle focus management and keyboard operability correctly.

Expectable behavior connects UI design, interaction design, and implementation. A button should trigger an action, a link should navigate, a toggle should change a state, a tab should switch between related contents, a search should deliver comprehensible results, and a filter should be visibly activatable and resettable. When control, behavior, and system feedback do not fit together, irritation, errors, or additional explanation needs arise.

Performance and perceived responsiveness also influence UX. Long loading times, missing feedback, or blocked interactions can irritate users. In implementation, this means displaying loading states, explaining blocked actions, preventing duplicate execution, and indicating progress when operations take longer.

7.8 Components, design systems, and design tokens

Components, design systems, and design tokens are not merely means of visual consistency. They support usability, maintainability, and accessibility because they systematically unify recurring interactions, states, naming, and behaviors.

Reusable components help ensure consistency, maintainability, and accessibility. When buttons, forms, error messages, dialogs, or tables are implemented anew again and again, inconsistencies arise. These increase cognitive load and make use more difficult.

A component consists of more than its visual presentation. It contains behavior, states, variants, accessibility properties, and technical interfaces. A form field component, for example, comprises the input field, label, help text, error message, mandatory-field marking, validation state, focus behavior, and

programmatic association for assistive technologies.

Components are particularly important for AI-assisted development. When AI uses existing components, the application remains more consistent. When AI instead creates new one-off solutions for every screen, variants quickly arise that look visually similar but work differently.

Design systems describe reusable UI components, design rules, interaction patterns, texts, system states, and quality criteria. They connect design, code, accessibility, documentation, and governance. Especially with AI-assisted implementation, they help align AI output with existing rules.

Not every team needs an extensive design system right away. A minimal viable design system can start with a few central elements:

- colors, typography, and spacing,
- buttons and form fields,
- error messages, dialogs, and tables,
- system states and basic accessibility rules.

What matters is not completeness, but bindingness for frequent and usability-critical elements.

Design tokens are named values for design properties such as colors, spacing, font sizes, border radius, or shadows. They connect design and code. Tokens help improve consistency and maintainability because changes can be controlled centrally. For AI-assisted implementation, it is important that AI knows the existing tokens and design system rules. Otherwise it may produce deviating values or new variants.

AI can support design system work, for example through component documentation, detection of inconsistencies, suggestions for tokens, usage guidelines, stories for a component workshop, comparison of the implementation with design system rules, and support in refactoring. However, it cannot decide on its own which design system rules should apply. Governance, approval, and maintenance remain human tasks.

7.9 AI for deriving UI patterns and design system building blocks

Many teams do not start with a finished design system, but with a grown application. Different buttons, form fields, spacings, colors, error messages, table variants, or dialogs have emerged over several sprints or releases. This creates inconsistencies that can be confusing for users and hard to maintain for teams.

AI can help derive initial design system building blocks from an existing application. To do so, AI can analyze screenshots, code excerpts, components, CSS rules, or UI descriptions and list recurring patterns:

Which button variants exist? Which form fields are used? Which colors and spacings occur? Which system states are implemented? Where are inconsistencies? Which components could be unified?

The value lies in the fact that a team does not have to start with an ideal design system. It can build a minimal viable design system out of the existing product: initially central colors, typography, spacing, buttons, form fields, error messages, tables, dialogs, and system states. From this, component rules, design tokens, documentation, and review criteria emerge step by step.

AI can accelerate this process, but it cannot automatically decide which variant should be the future standard. This decision must be made based on usability, accessibility, technical maintainability, brand requirements, existing use, and team conventions. A frequently occurring variant is not automatically the best variant. Frequency must therefore not be confused with quality.

7.10 Handoff between design, AI, and development

A good handoff describes not only what something looks like. It describes how a UI works and which requirements for usability, accessibility, professional correctness, and tests must be fulfilled. This includes:

- user goal and context of use,
- acceptance criteria,
- components, variants, and system states,
- error cases, texts, and input validation,
- accessibility requirements and data dependencies,
- test hints and open assumptions.

With AI-generated drafts, it must be checked in particular which assumptions are contained, which system states are missing, which components are to be used, which texts are placeholders, which interactions are not specified, which input validation is intended, and which accessibility criteria apply. An AI-generated draft should not be turned into code without review.

AI can help create a handoff description from a draft. It can suggest component lists, system states, open questions, test hints, accessibility guidance, or acceptance criteria. Such documents must be reviewed because AI can add details that were not contained in the draft. These additions are to be labeled as assumptions if they are not professionally confirmed.

Traceability is particularly important in AI-assisted implementation. It should be recognizable which usability requirement is covered by which component, which code, which system state, or which test. When it is traceable which requirement is addressed by which implementation, gaps, misunderstandings,

and unintended deviations are easier to detect.

In agile teams, handoff is not a one-time transfer point. Requirements, drafts, implementation, and tests evolve iteratively. Handoff should therefore be understood as an ongoing alignment: What was decided? What is implemented? What is open? What must be checked?

7.11 Accessibility basics in implementation

Accessibility basics are part of usability and professional UI implementation. They ensure that central tasks can be performed with different abilities, assistive tools, devices, and usage situations.

Accessibility is not a downstream add-on topic and not a purely formal compliance topic. It concerns the fundamental usability of interactive systems for people with different abilities, impairments, and usage situations. This includes permanent impairments as well as situational conditions, such as mobile use, poor lighting, time pressure, stress, or operation without a mouse.

Standards and guidelines such as WCAG, EN 301 549, and ISO/IEC 40500 form important professional reference points for digital accessibility. In the foundation-level context, accessibility basics means a targeted selection of fundamental requirements that agile software development teams must know and consider in implementation. These include, in particular, semantic HTML, sufficient contrast, understandable labels, keyboard operability, visible and sensible focus order, accessible error messages, and the careful use of ARIA.

This selection replaces neither dedicated accessibility training nor an expert accessibility audit. Digital accessibility is a field of its own with its own standards, methods, testing procedures, and specializations. This syllabus covers accessibility insofar as it is directly relevant for AI-assisted usability work and UI-related implementation at foundation level.

Semantic HTML means using appropriate HTML elements according to their meaning and function. A button should trigger an action, a link should navigate to another destination, headings should be marked up as headings, and form fields should be connected to labels. AI-generated code does not always use the semantically appropriate structure, for example when a clickable div is used instead of a real button.

Keyboard operability means that all interactive elements can be reached, operated, and focused in a sensible order using the keyboard. Forms, menus, dialogs, dropdowns, tabs, and complex components are particularly important. Custom components that visually look like standard controls do not automatically behave like native controls.

Focus order shows where the current interaction is. A visible focus is necessary so that keyboard users

know which element is active. For dynamic areas, dialogs, menus, or modal windows, the focus must be set and returned sensibly. Errors in focus order can make an interface difficult or impossible to operate for certain user groups.

Labels, help texts, and error messages must be understandable, professionally correct, and accessible. Placeholder texts do not replace labels. Error messages should be clearly associated with the affected field or area and be programmatically recognizable for assistive technologies. Good error messages explain what happened, how users can fix the problem, and preserve the work done so far wherever possible.

Sufficient contrast is a prerequisite for texts, icons, controls, and states to be perceivable. Color must not be the only means of conveying meaning. An error state should therefore not only be marked in red, but also be made understandable through text, an icon, or structural association.

ARIA can improve accessibility when native HTML elements are not sufficient. Incorrectly used ARIA, however, can make accessibility worse. In principle, native HTML elements should be preferred when they already offer the required semantics and function. ARIA should not be used to repair incorrect HTML when a suitable native element is available.

AI and automated tools can uncover accessibility problems, but they find only part of the problems. Accessibility checking should therefore comprise automated checks, manual review, a keyboard test, a screen reader test where applicable, and specialized accessibility expertise where risks are relevant. AI can support this process, but it cannot replace it.

7.12 AI for documentation, tests, and reviews

AI can suggest test cases from acceptance criteria, code, component rules, or handoff documents. This is particularly useful for system states, error cases, input validation, and UI behavior. Such tests should consider not only the happy path, but also invalid inputs, retained input data after errors, loading states, empty states, disabled states, keyboard operability, focus behavior, and dialog behavior.

A component workshop is a development environment in which individual components are displayed and checked in isolation from the rest of the application, in different variants and system states. Each displayed manifestation of a component is called a story there. AI can generate stories for components, such as Button Primary, Button Disabled, Button Loading, Form Field Error, or Modal Open. It is important that not only the standard case is represented, but also error states, empty states, long texts, disabled variants, keyboard behavior, and accessibility-relevant properties.

AI can generate review questions and checklists for UI code. These can concern component usage,

semantic HTML, keyboard operability, focus order, system states, error messages, consistency, tests, security risks, and professional correctness. Checklists should remain context-related. Overly general lists quickly become impractical.

AI can also formulate component documentation, usage guidelines, code comments, or handoff texts. This documentation is helpful when it is correct, concise, and traceable. It is particularly important that AI does not invent a rationale that was never decided within the team. Documentation must reflect actual decisions, reviewed requirements, known constraints, and open assumptions.

AI can also help align code, acceptance criteria, and tests with one another. For example, it can check whether tests exist for relevant system states or whether a component covers the described variants. This alignment supports traceability, but it does not replace a professional decision. If AI finds no gap, that does not mean the implementation is complete or correct.

7.13 Risks of AI-generated implementation

Risks of AI-generated implementation concern not only technical correctness. They can also impair usability, accessibility, security, maintainability, performance, domain logic, and trust in the system.

AI-generated code can contain security vulnerabilities. This concerns insecure input processing, missing or incorrect input validation, insecure API use, insufficient authentication, faulty permission checks, or insecure data handling. Frontend code can also be security-relevant, for example when sensitive information is displayed, permissions are wrongly assumed, or inputs are handled insufficiently.

AI can generate visually fitting components that are problematic in accessibility terms. Common problems are missing labels, wrong semantics, insufficient focus order, inoperable custom components, too little contrast, meaning marked by color only, or incorrect ARIA use. These problems are not always detectable through visual inspection.

AI can create new variants instead of using existing components. This creates inconsistencies in behavior, styling, texts, system states, or interaction patterns. Inconsistencies are not merely aesthetic problems; they can lead users to interpret the same functions differently or to understand the same states differently.

AI-generated code can be hard to maintain when it is too complex, redundant, or not compliant with project standards. Maintainability is also usability-relevant, because hard-to-maintain UI structures make later improvements, accessibility corrections, or consistency work more difficult.

AI can misinterpret domain rules. This is particularly critical for input validation, permissions, workflow steps, status logic, or regulatory requirements. An interface can appear efficient while violating central

process rules.

AI can generate documentation that sounds convincing but does not correspond to the actual decisions. This creates an appearance of traceability. Teams should therefore check whether documentation reflects actual decisions, reviewed requirements, and known constraints.

When building a design system retroactively, there is the additional risk that AI merely describes existing patterns without evaluating them. A frequently occurring button variant is not automatically the best variant. An existing inconsistency does not become the standard just because it occurs several times in the application.

7.14 Review measures for AI-generated code

Review measures for AI-generated code must consider technical quality and usage quality together. These include code review, functional tests, UI tests, accessibility checks, security review, professional acceptance, and checking usability in the specific context of use.

AI-generated code should therefore be reviewed like other code, often even more carefully. Code review should check not only style and function, but also usability-relevant aspects: system states, error messages, focus order, component usage, accessibility, consistency, and expectable behavior.

Functional tests and UI tests should not only check the successful flow. They should also consider error cases, alternative paths, and system states. Important test areas are:

- input validation and error states,
- loading states and empty data situations,
- permissions,
- keyboard operation and focus behavior,
- component variants,
- abort and resumption of flows.

Accessibility should be checked with automated tools, manual review, and, where needed, specialized expertise. Manual checks should cover, in particular, keyboard flow, visible focus, focus logic for dialogs and dynamic content, understandability of error messages, association of labels and error messages, the screen reader experience, semantic structure, and suitability for the task.

Security review is required when AI creates or changes code. This concerns data processing, authentication, authorization, external libraries, API access, and input validation. Security is also relevant from a usability perspective, because login, session timeouts, permissions, error messages, and

confirmations must be implemented securely and, at the same time, understandably.

Professional acceptance checks whether the implementation fits the domain, the process, and the user goal. A feature can work technically and still be professionally wrong. Professional acceptance ensures that roles, permissions, domain terms, legal or organizational conditions, and process logic have been considered correctly.

Production readiness means that an implementation not only works locally or looks visually fitting, but is suitable for productive use. This includes technical stability, security, accessibility, performance, maintainability, professional correctness, tests, monitoring, and traceable approval. AI can provide hints, test suggestions, and documentation for this, but it cannot take on approval responsibility.

7.15 AI-assisted implementation and human review

In implementation, AI is particularly good at supporting recurring UI and development tasks; the clearer the usability requirements, acceptance criteria, component rules, and design system rules, the better the results.

AI is good at	Generating components; explaining and refactoring code; adding system states; preparing forms and input validation; suggesting error messages; generating test cases; creating stories for a component workshop and component documentation; providing accessibility guidance; uncovering inconsistencies in code or components.
Limitations and typical risks	Guarantees no production readiness: security, maintainability, accessibility, performance, and professional correctness are not ensured; AI knows project standards, architecture decisions, domain rules, and regulatory requirements only if they are sufficiently provided and subsequently reviewed.
Requires human review	Which usability requirement the implementation is supposed to fulfill; whether the code fulfills the acceptance criteria; whether relevant system states are implemented; whether error messages are understandable and correctly associated; whether existing components, design tokens, and design system rules are used; whether keyboard operation and focus order work; whether semantic HTML is correct and ARIA is necessary and correctly used; whether fundamental WCAG-relevant requirements are considered; whether security risks exist; whether the solution is maintainable and performant; whether the domain logic is correct. The

	review is documented in a traceable way, especially when AI has generated substantial parts of code, components, documentation, or tests.

What matters is not only that something works, but that it is implemented correctly, understandably, accessibly, securely, and maintainably in the context of use.

8 Evaluation, Testing, and Validation

Teaching time: approx. 150 minutes

8.1 Core idea of this chapter

This chapter covers the structured verification of usability in the software development process. The central question is how teams recognize whether a solution is actually understandable, usable, in line with user expectations, and effective.

Several terms must be distinguished for this. Evaluation is the umbrella term for the systematic assessment of a draft, prototype, product, or usage path with regard to usability, user goal, and context of use. Evaluation can be conducted with real users, for example through usability tests, or expert-based, for example through heuristic evaluation or a cognitive walkthrough.

Review formats, such as a sprint review, are team formats in which work results are shown and discussed. They can make usability questions visible, but they are not automatically evaluation methods in the narrower sense and do not replace validation with real users.

Testing can mean different things in the software context. Functional tests check whether specified functions work correctly. Usability tests check usability not only technically, but through observation of real or representative users performing concrete tasks. This makes comprehension problems, errors, search movements, uncertainties, and obstacles visible.

Validation means verifying assumptions about users, tasks, understandability, behavior, or context of use with suitable sources. Real user observation, usability tests, domain knowledge, usage data, or other reliable evidence can contribute to this.

GenAI can prepare and structure evaluation, testing, and validation. It can formulate review questions, prepare heuristic evaluations, structure cognitive walkthrough steps, draft test tasks, organize observation notes, cluster findings, and generate report drafts. AI, however, does not replace real users. AI-assisted analyses are to be understood as support, a source of hypotheses, or a structuring aid, not as a substitute for evaluation or validation.

The goal of this chapter is to enable software development teams to verify usability in a structured way with simple methods, to use AI sensibly for preparation and analysis, and to clearly recognize the limits of synthetic or heuristic AI analyses.

8.2 Contribution to the business outcomes

This chapter particularly supports business outcomes BO 8.1 to BO 8.5. The complete and authoritative wording of the business outcomes can be found in section 0.9.

8.3 Learning objectives

LO	Learning objective	K
LO 8.1	Be able to name fundamental evaluation methods for usability as well as expert-based procedures, in particular heuristic evaluation, cognitive walkthrough, and usability test.	K1
LO 8.2	Be able to explain the purpose of evaluation and validation in software development processes.	K2
LO 8.3	Be able to explain why review formats, such as the sprint review, should consider not only functionality, but also user goal, understandability, system states, and error tolerance.	K2
LO 8.4	Be able to prepare a heuristic evaluation with AI support and critically classify its results.	K3
LO 8.5	Be able to explain the basic idea of the cognitive walkthrough as an expert-based procedure.	K2
LO 8.6	Be able to describe the central components of a simple usability test, in particular research question, test participants, test object, test tasks, facilitation, observation, and analysis.	K2
LO 8.7	Be able to formulate test tasks in a goal-oriented way without prescribing the solution path.	K3
LO 8.8	Be able to explain the purpose and limits of think-aloud and structured observation.	K2
LO 8.9	Be able to use GenAI to structure observation notes, findings, and report drafts.	K3
LO 8.10	Be able to prioritize usability problems by severity, frequency, user impact, risk, implementation effort, and business relevance.	K3
LO 8.11	Be able to derive testable assumptions and research questions from AI-generated usability suggestions.	K3
LO 8.12	Be able to explain why AI simulations, AI-assisted heuristic pre-checks, and synthetic users do not replace validation with real users.	K2

8.4 Key terms

Review format, sprint review, evaluation, formative evaluation, summative evaluation, validation, verification, expert review, expert-based procedure, heuristic evaluation, usability heuristic, cognitive walkthrough, usability test, moderated usability test, unmoderated usability test, remote usability test, formative usability test, summative usability test, research question, test question, hypothesis, test object, test task, test participant, facilitation, observation, think-aloud, thinking aloud, finding, severity, frequency, user impact, risk, business relevance, implementation effort, report draft, A/B test, AI-assisted analysis, AI simulation, synthetic user, human review.

8.5 Evaluation, testing, and validation in the software development process

In software development processes, review formats, such as the sprint review, are central moments for showing work results. Often, the main thing shown is that a function has been implemented. For UX, that is not enough. A function can be technically implemented and still be hard to understand from the user perspective, error-prone, incomplete, or not in line with user expectations.

Review formats can be used to make usability-relevant questions visible: Is the user goal supported? Is the flow understandable? Are relevant system states implemented? Are error messages, system feedback, and interactions expectable? Are accessibility basics considered? Are there open assumptions that still need to be verified?

A review format, however, is not automatically a usability evaluation method in the narrower sense. It is first of all a structured discussion or check by the team, stakeholders, or subject-matter experts. It can provide indications of possible usability problems, but it replaces neither a methodologically appropriate evaluation nor validation with real users.

Evaluation refers to the systematic assessment of a draft, prototype, product, or usage path based on criteria, methods, or observations. Evaluation can be expert-based or involve real users. Examples are heuristic evaluation, cognitive walkthrough, and usability test.

In the software development process, testing can mean functional tests, UI tests, accessibility tests, technical checks, or usability tests. In the usability context, it must therefore always be stated clearly which kind of test is meant.

Validation means verifying central assumptions about users, tasks, understandability, behavior, or context of use with suitable evidence. An AI analysis can, for example, generate the hypothesis: “Users might overlook the filter function.” Validation means verifying this assumption through suitable sources, for

example through a usability test, observation, domain expertise, usage data, or other reliable evidence.

Results from evaluation, testing, and validation must be turned into decisions. They can lead to new backlog items, extended acceptance criteria, design adjustments, technical tasks, further tests, or documented UX debt. A distinction must be made between observed problems, professional assessments, AI-generated hints, and open hypotheses.

8.6 Making usability visible in review formats

Review formats should not only consider functionality. Functionality describes whether a system delivers a certain performance. Usability describes whether specific users can use this performance effectively, efficiently, and with satisfaction in a specific context of use.

An export function is technically implemented when it produces a file. From a usability perspective, it is additionally relevant whether users find the function, understand which format is produced, recognize when the export is complete, and receive understandable feedback in case of errors.

A review format can be structured using simple usability review questions: Which user goal does this solution support? Is the next step recognizable? Are important system states implemented? Are error messages understandable and actionable? Is system feedback visible and in line with expectations? Are relevant edge cases and accessibility basics considered? Are there open assumptions about user behavior?

GenAI can prepare review formats by deriving review questions from user stories, acceptance criteria, prototypes, or implemented functions. These questions must be adapted to context of use, target group, and professional requirements.

Results from review formats should be documented. A distinction should be made between confirmed problem, open question, improvement idea, AI-generated hint, hypothesis for further evaluation, and the backlog item derived from it. A finding should rest on observation, verification, or professional assessment. An AI suspicion remains a hypothesis until it has been verified.

8.7 Heuristic evaluation with AI support

A heuristic evaluation is an expert-based procedure. A user interface is systematically checked against defined usability heuristics, interaction principles, or design criteria in order to uncover possible usability problems. Without such a frame of reference, it is not a heuristic evaluation but a general assessment.

Possible reference points are visibility of system status, match with user expectations, consistency, error

prevention, recognition rather than recall, support for error recovery, controllability, suitability for the task, or understandable system feedback.

For agile teams, heuristic evaluation is useful because it can be conducted relatively quickly and requires no elaborate test infrastructure. It can uncover coarse problems early and is suitable as a pre-check of drafts, prototypes, or implemented screens. It does not, however, replace usability tests. A heuristic evaluation shows possible problems from an expert perspective, but not reliably how real users actually behave.

GenAI can prepare or support a heuristic evaluation when clear heuristics or criteria are provided. AI can analyze an interface, a screenshot, a description, or a prototype against these criteria, name possible problems, formulate improvement suggestions, and generate review questions.

The results of AI-assisted heuristic assessments must be critically classified. AI can detect problems that are not relevant in the context, or overlook problems that do not become visible without domain knowledge. Results can therefore be classified as obviously applicable, plausible but needing verification, unclear or context-dependent, not relevant, or as a question for further evaluation.

The separation between heuristic hint and evidence-based finding is important. A heuristic hint can point to a possible problem. A finding, by contrast, should rest on concrete verification, observation, or a professionally comprehensible assessment.

8.8 Cognitive walkthrough

A cognitive walkthrough is an expert-based procedure. A usage path is walked through step by step from the perspective of a user. The central question is whether users can recognize, understand, and perform the necessary steps to achieve a specific goal.

The method is particularly useful for new users, rarely used functions, or flows where orientation, learnability, and initial understandability are important. It is also suitable for checking early whether a draft or prototype fits the users' assumed mental models.

The procedure works with four guiding questions: Will the user recognize what the next sensible step is? Will the user find the right element? Will the user understand that this element fits the task? Will the user recognize after the action whether it was successful? In practice, two further questions are often added: Will the user be able to carry out the right action? Will the user know what to do in case of an error? These two questions do not belong to the original procedure but are widespread in application. In addition, a streamlined version exists, the Streamlined Cognitive Walkthrough, which works with two guiding questions.

These guiding questions should always relate to a concrete usage path. A cognitive walkthrough does not assess “the whole interface” in general; it checks step by step whether a specific task can presumably be accomplished with a specific interface.

GenAI can help decompose a flow into steps, formulate possible comprehension problems per step, and create a walkthrough plan from a user story, a prototype, or a process description. The actual review, however, requires knowledge of users, task, domain, and interface.

A cognitive walkthrough remains an expert or team check. It shows possible problems, but not whether real users actually fail. When central assumptions are uncertain or risks are high, real users should be involved.

8.9 Planning usability tests

Usability tests are a central method for verifying usability with real or representative users. They show whether users understand concrete tasks, find suitable functions, cope with error situations, and can achieve their goals in the context of use.

In this syllabus, usability tests are not mere test cases with pass/fail. They are a usage-related evaluation method with which software development teams check whether real or representative users can understand, perform, and complete concrete tasks.

Test participants work on concrete tasks with a product, prototype, draft, or application. What is observed is how they proceed, where they search, hesitate, make errors, misunderstand something, or abandon a task.

The goal of a usability test is not to collect general opinions about a design. The focus is on observable behavior in connection with concrete tasks. This makes comprehension problems, obstacles, erroneous operation, and unexpected usage patterns visible.

Usability tests can be conducted moderated or unmoderated, on-site or remote, formative or summative. Moderated tests are particularly suitable when the team wants to understand why users hesitate, fail, or choose different paths than expected. Unmoderated tests can be conducted faster and with more people, but they provide less context. Formative tests serve improvement during development; summative tests serve rather the assessment of a more advanced or finished solution.

Before a usability test, the research question must be clarified. A question like “Is the new screen good?” is too general. Better is, for example: “Do new users recognize how they can create a project after their first login?” The research question determines test participants, test object, test tasks, and relevant

observations.

Test participants should fit the relevant user group. For initial pragmatic tests, large samples are not always necessary. Even a few suitable people can uncover important problems. What matters is that the people have a meaningful proximity to actual use.

The test object must fit the research question. A test can be conducted with a wireframe, a clickable prototype, a functional prototype, a staging version, an existing application, or an individual flow. The system state must also be defined, for example an empty application, existing data, specific permissions, a triggered error state, or a partially completed process.

A simple usability test needs a goal and research question, suitable test participants, a suitable test object and defined system state, concrete test tasks, a short introduction, an observation guide, rules for facilitation, documentation of the observations, and analysis of the findings. AI can support test scripts, task drafts, observation sheets, and analysis grids.

8.10 Formulating test tasks

Good test tasks translate usability questions into observable tasks. They describe a goal from the user perspective without prescribing the solution path or the relevant UI element.

Test tasks should be derived from the research question. The research question describes what the team wants to find out. The test task creates a situation in which it can be observed whether users can actually accomplish the relevant task.

A good test task describes a goal from the user perspective without prescribing the solution path. If the task already says which button users should click, it is no longer tested whether they find and understand the function themselves.

Weak would be: “Click the ‘Export’ button.” Better is: “You want to pass on the data for the monthly statement. Find out how you can get this data out of the system.” The second task checks whether users can find, understand, and use the export function.

Test tasks should describe realistic situations. A task like “Use the filter” is less helpful than: “You want to display only open applications that have been unprocessed for more than seven days. Find out how you can create this list.”

Test tasks should not use terms that point directly to UI elements when it is precisely their findability that is to be tested. If the goal is to test whether users find the “Archive” tab, the task should not be: “Open the

Archive tab.” Better is: “You want to look at a completed application from last month again.”

A test task should be worded so that it is observable whether it was completed successfully. For this, it must be clear which result is expected. If no observable success criterion exists, the analysis becomes blurred.

AI can formulate test tasks from user stories, acceptance criteria, hypotheses, or research questions. These tasks must be checked: Are they derived from the research question? Do they describe a realistic user goal? Do they give away the solution? Do they fit the target group? Is success observable? Do they fit the test object and system state?

8.11 Facilitation, think-aloud, and structured observation

Facilitation in a usability test means explaining the procedure, giving the task, enabling observation, and asking neutral follow-up questions where needed. Facilitation does not mean helping, explaining, or guiding users toward the right behavior.

When a test participant hesitates, help should not be given immediately. It is precisely the hesitation that is an important observation. Good facilitation creates a safe setting. Test participants should understand that it is not they who are being tested, but the system.

Neutral follow-up questions help verbalize thoughts without steering. Examples are: “What do you expect to happen here?”, “What are you looking for right now?”, “What makes you unsure?”, or “What would you do next?” Not neutral would be hints such as: “Do you see the button at the top right?” or “Would you click Export now?”

Think-aloud means that test participants say out loud during the task what they think, expect, look for, or understand. This can make mental models, uncertainties, expectations, and misunderstandings visible. Think-aloud, however, does not replace the observation of actual behavior. What users say and what they do can diverge.

Think-aloud has limits. Thinking aloud can influence task performance, increase cognitive load, slow down behavior, or lead to rationalizations. It is therefore to be understood as a supporting method, not as a complete explanation of user behavior.

Structured observation should document as closely to the observation as possible where test participants hesitate, what is overlooked, which terms are misunderstood, which error messages help or do not help, which wrong steps are chosen, whether help is needed, or whether a task is abandoned.

For software development teams, it is important not to prematurely interpret observations as an individual user problem. Hesitation, detours, questions, or errors are much more often indications of problems with usability, understandability, or system feedback.

AI can prepare facilitation guides, observation sheets, and protocol structures. After a test, AI can organize notes and work out initial patterns. During analysis, it must be checked whether AI correctly reproduces actual observations or is already interpreting, summarizing, or weighting them.

8.12 Structuring observations, findings, and report drafts

A common problem in analysis is the mixing of observation and interpretation. An observation describes what was actually seen, heard, or recorded. An interpretation explains what this could mean.

Observation: “Three out of five test participants looked for the export function in the ‘Reports’ menu and not in the table view.” Interpretation: “The export function is probably not placed where users expect it.” Both statements are useful, but they should remain separate.

A good finding describes what was observed, with whom or in which task it occurred, what impact it had, why it is relevant, and whether there are indications of cause or solution. A finding should be concrete, comprehensible, and evidence-based. It should not merely reproduce an opinion or an AI suspicion.

Findings can arise from different sources: usability tests, heuristic evaluation, cognitive walkthrough, professional review, support data, usage data, or AI-assisted hints. These sources should be clearly labeled. An observed problem from a usability test has a different significance than an AI suspicion.

AI can cluster observation notes, merge similar problems, and formulate initial findings. This is helpful when many notes or several tests are analyzed. The AI analysis, however, must be checked: Were observations reproduced correctly? Were several problems merged inadmissibly? Were minority problems overlooked? Were interpretations worded as facts? Is the source of the finding clearly recognizable?

AI can also generate report drafts. Such drafts save time, but they must not be adopted without review. It is particularly important that AI does not add evidence that was not observed. If only two test participants were involved, this must not become a general statement about “most users”.

8.13 Prioritizing usability problems

Not every usability problem has the same significance. Some problems prevent task completion entirely. Others create small irritations. Some occur frequently, others rarely, but are severe in critical situations.

Prioritization helps turn findings into implementable decisions. Without prioritization, a long list of problems often emerges, but no clear next step. Prioritization is not a purely mechanical assessment. It connects evidence, user impact, risk, frequency, business relevance, effort, and product strategy.

Usability problems can be assessed by severity, frequency, user impact, risk, business relevance, implementation effort, regulatory significance, and strategic significance. A problem is not automatically high priority just because it occurs frequently. A rare problem can be critical when it leads to wrong decisions, data loss, security risks, legal problems, or a substantial loss of trust.

AI can make severity or prioritization suggestions. These can be helpful, but they are uncertain without context. AI does not automatically know business goals, legal risks, technical dependencies, user numbers, costs, or strategic priorities.

The team must therefore check which evidence exists, which source a finding comes from, how severe the impact is for users, how often the problem occurs, which risks arise, which business relevance exists, how costly the fix is, and what happens if the problem is not fixed.

Prioritization only becomes effective when concrete measures result from it. Findings can lead to an immediate fix, a new backlog item, an addition to acceptance criteria, an adjustment of a prototype, a technical task, an accessibility check, further evaluation, or documentation as UX debt.

8.14 Deriving testable assumptions from AI suggestions

AI can name possible usability problems from a screen, prototype, backlog item, or an implemented function. These can point to blind spots, but they are neither confirmed findings nor validated insights.

An AI suggestion such as “The filter function could be hard to find” can be turned into a testable research question: “Do users find the filter function when they want to narrow a results list down to open applications?” AI's strength here lies not in proving the problem, but in revealing possible assumptions.

A testable research question describes what is to be verified and under which conditions an observation would be meaningful. Relevant questions are: Which user group is affected? Which task is to be performed? Which screen, prototype, or flow is affected? Which behavior is to be observed? How does one recognize success, uncertainty, or a problem? Which decision is the verification supposed to support?

Not every question needs a usability test. Some questions can be addressed through heuristic evaluation, cognitive walkthrough, expert review, tree testing, analytics, support data, or a domain check. Other questions need real users. The team decides which method fits the research question, the risk, the state of development, and the available evidence.

Open assumptions should be documented. If an AI analysis suspects a problem but no verification takes place, this must be kept documented. Suitable documentation can record: the AI-generated hint, the hypothesis derived from it, a possible research question, a suggested method, the current evidence status, the team's decision, and the next step.

Not every hypothesis leads immediately to a backlog item. Some hypotheses must be verified first. Others can be documented as an open question, an improvement idea, or UX debt. AI can structure these categories and suggest initial wordings. The decision remains with the team.

8.15 AI simulations and synthetic users

AI simulations can be helpful for preparing usability tests, for example to collect possible questions, risks, or test tasks. On their own, however, they are no evidence of usability and no validation of real use.

AI can simulate how a certain user type might react to an interface. It can formulate typical questions, misunderstandings, expectations, or possible barriers. This can be helpful for generating hypotheses, recognizing blind spots, or preparing an evaluation.

AI simulations, however, rest on patterns, not on the real behavior of specific people. They do not show what real users actually do. They capture no real search movements, no actual hesitation, no nonverbal signals, no real frustration, no situational conditions, and no actual erroneous operation.

Synthetic users are AI-generated user profiles, roles, or simulated reactions. They can help make assumptions explicit or mentally play through different perspectives. However, they must not be confused with real users.

A synthetic user is not based on own experience, real context, or actual behavior. It is a generated representation that can sound plausible but provides no evidence of real use. Synthetic users can therefore serve at most as an aid for hypothesis generation.

A particular risk is that AI simulations create a sense of certainty even though no real verification has taken place. When AI writes that “users will probably understand the flow”, this can be reassuring. Without real observation, it remains a supposition.

Real tests are particularly important when central user assumptions are uncertain, a function is business-critical, errors have severe consequences, target groups are little known, domain language or domain logic is complex, several plausible solutions exist, accessibility or special usage situations are relevant, or a result is to become the basis of important product decisions.

8.16 AI-assisted analysis and human review

In evaluation, testing, and validation, AI is particularly good at supporting preparation, structuring, and documentation.

AI is good at	Deriving review questions from backlog items; structuring heuristic criteria; preparing cognitive walkthrough steps; formulating test tasks; drafting facilitation guides and observation sheets; clustering observation notes; pre-formulating findings; creating report drafts and prioritization suggestions; translating AI suggestions into testable hypotheses.
Limitations and typical risks	Validates no real use: AI cannot observe behavior, hesitation, failure, nonverbal signals, actual frustration, and real working environments; typical risks: AI analyses are misunderstood as user tests, synthetic users are confused with real ones, heuristic hints are documented as validated findings, observation and interpretation are mixed, minority problems are overlooked, test tasks give away the solution, report drafts suggest more evidence than exists.
Requires human review	Which research question is being verified and which method fits it; whether it is a review, an evaluation, a test, or a validation; which source a finding rests on; whether test tasks are worded in a goal-oriented way and without solution hints; whether test participants fit the user group; whether observation and interpretation were separated; whether findings are concrete and evidence-based; whether the prioritization is comprehensible; which AI statements were adopted, adapted, or discarded; what is evidence and what is hypothesis; what is fixed immediately, what is documented as UX debt, and what is turned into backlog items, acceptance criteria, or tests.

AI can make usability evaluation more pragmatic and more accessible, but it must not be confused with evidence-based evaluation or validation; assessment and responsibility remain with the team or the responsible roles.

9 Release, Monitoring, and Improvement

Teaching time: approx. 90 minutes

9.1 Core idea of this chapter

This chapter covers usability after release. Work on usability does not end with the implementation of an increment, nor with a review format such as the sprint review. Only in actual use does it become apparent whether assumptions from discovery, backlog refinement, design, implementation, evaluation, and testing hold up in practice.

In this chapter, the term feedback is used in the sense of user, customer, support, or operations feedback. It refers to responses from actual use or from the productive environment, such as hints, complaints, wishes, observations, or experience reports. This is to be distinguished from system feedback, that is, the system's response to users during interaction, such as error messages, loading states, success messages, or confirmations.

After release, new indications of usability problems arise. Users give feedback, support tickets accumulate around certain topics, usage data shows abandonments or rarely used functions, qualitative responses reveal irritations, and metrics show whether certain goals are being achieved.

GenAI can help structure such responses and data. AI can cluster feedback, summarize support tickets, recognize patterns in free texts, prepare UX debt lists, explain metrics, or formulate hypotheses for usability improvements. At the same time, patterns in data must not be confused with causes. An abandonment rate shows that something is happening. It does not automatically explain why it is happening.

The goal of this chapter is to enable software development teams to observe usability systematically after release, to critically classify AI-assisted analyses, and to turn insights into backlog items, UX debt measures, or new discovery questions.

9.2 Contribution to the business outcomes

This chapter particularly supports business outcomes BO 9.1 to BO 9.4. The complete and authoritative wording of the business outcomes can be found in section 0.9.

9.3 Learning objectives

LO	Learning objective	K
LO 9.1	Be able to name typical sources of usability insights after release, in particular user feedback, customer feedback, support tickets, usage data, technical signals, and UX metrics.	K1
LO 9.2	Be able to explain why usability must be further observed and improved after release.	K2
LO 9.3	Be able to describe fundamental usability metrics such as task success, error rate, abandonment rate, time on task, return rate, use of central functions, and support volume.	K2
LO 9.4	Be able to explain the basic idea of the HEART framework and classify the dimensions happiness, engagement, adoption, retention, and task success as a structuring aid for UX goals and UX metrics.	K2
LO 9.5	Be able to use GenAI to structure feedback, tickets, metrics, and indications of usability problems or UX debt and to reveal patterns.	K3
LO 9.6	Be able to explain the difference between pattern, correlation, cause, priority, and a reliable product decision.	K2
LO 9.7	Be able to recognize UX debt based on typical usability-related examples, such as incomplete system states, inconsistent components, unclear texts, missing accessibility, or unresolved usability problems.	K3
LO 9.8	Be able to turn insights from feedback, metrics, support data, and AI-assisted analysis into backlog items, UX debt measures, usability improvements, or new discovery questions.	K3

9.4 Key terms

Release, monitoring, feedback, user feedback, customer feedback, support feedback, operations feedback, system feedback, support ticket, support data, usage data, analytics, technical signals, UX metric, task success, error rate, abandonment rate, time on task, return rate, retention, adoption, use of central functions, support volume, HEART framework, happiness, engagement, adoption, retention, task success, Goals–Signals–Metrics, KPI, pattern, correlation, cause, hypothesis, evidence, priority, product decision, UX debt, technical debt, backlog item, improvement measure, discovery question, continuous improvement, AI-assisted analysis, human review, data protection, anonymization.

9.5 Observing usability after release

After release, usability shows itself in the real context of use. Teams then recognize whether users actually complete tasks, where they abandon them, which functions are rarely used, which error messages lead to support cases, and which assumptions from discovery, backlog refinement, design, implementation, or testing do not hold.

Review formats, evaluation, and testing before release provide important indications of UX quality. However, they do not fully show how a solution works in real use. Some usability problems only become visible when users work with real data, real tasks, real devices, time pressure, distraction, permissions, organizational conditions, or domain context.

After release, it becomes visible whether the assumptions from previous process steps hold. Discovery hypotheses can be confirmed, refined, or refuted. Acceptance criteria can prove too narrow or incomplete. Prototypes may not have shown problems that occur in productive use. Tests may have overlooked edge cases.

Monitoring comprises the systematic observation of user feedback, customer feedback, support cases, usage data, technical signals, and UX metrics. It is not a purely technical activity. Technical signals such as loading times, error rates, timeouts, or abandonments can also be usability-relevant when they influence goal achievement.

For agile software development teams, monitoring is particularly useful when insights are fed back into backlog refinement, discovery, design, implementation, or technical improvement. Otherwise, reports are produced, but no improvement.

UX debt arises when usability problems are left unaddressed, deliberately or not. After release, UX debt often becomes visible: users report the same problems, certain flows produce abandonments, help texts are missing, system states are incomplete, components are inconsistent, or accessibility problems persist.

9.6 Sources of usability insights after release

Typical sources of usability insights after release are user feedback, customer feedback, support tickets, support data, usage data, analytics, technical signals, and UX metrics. These sources show different excerpts of actual use and should not be considered in isolation.

Direct user feedback and customer feedback can come from feedback forms, interviews, comments, ratings, support conversations, internal responses, customer communication, or product surveys.

Feedback often shows what users experience as disturbing, unclear, helpful, or missing. It is valuable, but not automatically representative.

Support tickets show where users need help or fail. If tickets accumulate around a function, this can point to comprehension problems, missing system feedback, technical errors, unclear terms, missing permission information, or gaps in the documentation. Support data often shows symptoms. The cause must be narrowed down through further analysis.

Usage data and analytics show what users actually do. They can provide indications of abandonments, rarely used functions, long task times, repeated errors, frequent backtracking, uncompleted transactions, or unusual paths. Analytics primarily answers the question of what happens. It does not automatically answer why it happens.

Qualitative responses such as app store reviews, product ratings, survey comments, or internal stakeholder responses often contain indications of frustration, expectations, or recurring misunderstandings. AI can cluster such responses and describe their tone. Tone, however, is not the same as priority.

Technical signals can also be usability-relevant. These include loading times, error rates, timeouts, crash rates, validation errors, API latencies, uncompleted transactions, or repeated attempts to trigger the same action. When users trigger actions multiple times, visible system feedback may be missing.

AI can structure these sources, reveal patterns, and formulate cause hypotheses. The professional interpretation remains the team's task.

9.7 Why usability must be further improved after release

Usability is not a property completed once and for all. After release, usage, expectations, technical conditions, support volume, and product goals change. Teams must therefore observe which assumptions hold, which problems newly arise, and which improvements should be prioritized.

Tests, evaluations, and review formats before release are important, but they do not fully represent real use. In real use, people work with their own data, real consequences, time pressure, distraction, different devices, individual expectations, and organizational conditions.

Some problems only arise through real data volumes, rare roles, unusual permissions, slow connections, parallel work, mobile use, or complex combinations of functions. Monitoring after release is therefore necessary to develop UX quality further.

Many decisions in the development process rest on assumptions. The team assumes that users find a function, understand a term, accept a process, or interpret an error message correctly. After release, these assumptions can be verified using feedback, support data, usage data, metrics, or targeted evaluation.

UX is not a one-time activity. Products change, user groups change, requirements grow, and new functions influence existing flows. An interface that originally worked well can become complex, inconsistent, or hard to understand through later extensions.

Continuous improvement means observing UX regularly, prioritizing problems, and improving in a targeted way. AI can support this learning loop by working out patterns in feedback, support tickets, usage data, and metrics, and by preparing improvement hypotheses. Responsibility for interpretation, prioritization, and decision remains with the team.

9.8 UX metrics and the HEART framework

UX metrics can help make usability and the usage experience measurable after release. They do not replace qualitative assessment, but they show whether users complete tasks successfully, where they abandon them, how long they need for central flows, how often errors occur, and whether important functions are adopted. Metrics should be connected to user goals and product goals. A metric is only helpful when it is clear what it says and which decision it is supposed to support.

Task successdescribes whether users can successfully complete a relevant task. This metric is particularly important for UX because it is directly connected to goal achievement. A task, however, can be formally completed even though users take unnecessary detours, make wrong assumptions, or only reach the goal with considerable effort.

Error rate describes how often users make errors while working on a task. This can include wrong inputs, aborted actions, repeated validation errors, misclicks, misunderstandings, or incorrectly performed steps. Not every error has the same significance. The error rate should therefore be connected with severity, context, and user impact.

Abandonment rate describes at which points users leave a process, a task, or an interaction before the intended goal is reached. A high abandonment rate can indicate usability problems, such as unclear next steps, excessive cognitive load, lack of trust, technical problems, or incomprehensible error messages. An abandonment, however, is not automatically a usability problem.

Time on task describes how long users need to complete a specific task. A long task time can indicate unnecessary complexity, unclear terms, poor information architecture, too many steps, or missing orientation. Shorter, however, is not always better, for example in safety-critical or legally relevant

decisions.

Support volume describes how often users need help, ask questions, or report problems. This metric is particularly helpful because usability problems are often not named directly as usability problems. Recurring support requests can point to unclear flows, misleading terms, missing system feedback, poor findability, or insufficient error tolerance.

The HEART framework structures UX metrics into five dimensions:

HEART dimension Meaning Example metrics
Happiness Subjective assessment of the usage experience, such as satisfaction, trust, or perceived simplicity. Satisfaction rating, survey score, SUS, qualitative responses, complaint frequency
Engagement Intensity or frequency of use of a product or function. Active sessions, interactions per use, frequency of use, active days
Adoption Uptake of a new product, a new function, or a new workflow. First use of a function, activation rate, use after introduction
Retention Return or continued use over a certain period. Return rate, renewed use after a period, churn rate
Task Success Successful completion of relevant tasks in the context of use. Completion rate, error rate, time on task, abandonment rate, support tickets for a task

What matters is the translation into Goals, Signals, and Metrics. This procedure is called Goals–Signals–Metrics and belongs to the HEART framework. Goals describe which user or product goal is to be improved. Signals are observable indications of whether this goal is being achieved. Metrics are concrete measures with which these indications are captured. Only in this way does a general UX dimension become a measure that can support a decision.

HEART is not a rigid mandatory scheme. Not every dimension is equally important for every product. For an internal domain application, task success can be more important than engagement. For a learning platform, retention and task success can be relevant together. AI can suggest possible HEART dimensions, goals, signals, and metrics. The team must decide which goals are actually relevant and which metrics are interpretable and decision-relevant.

9.9 GenAI for structuring feedback, tickets, and metrics

AI can be particularly helpful for turning large amounts of unstructured responses into possible usability problem areas.

In particular, AI can cluster large amounts of free-text feedback, support tickets, product ratings, survey

answers, or internal responses. Possible clusters are navigation, performance, error messages, search, onboarding, understandability, accessibility, missing functions, or support needs. Such clusters are a first ordering step. They show topics, but they prove neither cause nor priority.

AI can filter out recurring terms, complaints, wishes, or irritations. From responses such as “I cannot find my old invoices”, “Where are completed invoices?”, and “Archive is unclear”, AI can derive the pattern “findability of archived invoices”. The team must then check whether this pattern actually describes a usability problem and which cause is likely.

AI can explain metrics and suggest possible interpretations. If a function has low adoption, AI can name possible causes, such as lack of visibility, unclear benefit, poor communication, wrong target group, technical problems, or lack of relevance. These causes are hypotheses. They must be checked against data, user feedback, support information, usability tests, domain knowledge, or technical analysis.

AI can prepare a UX debt list from feedback, tickets, evaluation results, known weaknesses, and existing backlog items. Such a list can describe problems, name affected areas, formulate possible impacts, and make initial prioritization suggestions. It is, however, a working artifact that must be reviewed, cleaned up, and prioritized by the team.

In AI-assisted analysis, data protection and confidentiality must be observed. Feedback, support tickets, chat logs, usage data, and free texts can contain personal or confidential information. Depending on the context, anonymization, pseudonymization, aggregation, or the use of approved company systems may be necessary.

9.10 Distinguishing pattern, correlation, cause, and priority

For decisions about usability, it is not enough to find patterns in data. Teams must distinguish whether a pattern actually shows a problem, whether it merely correlates with another event, which cause is plausible, and whether a prioritized product decision should follow from it.

A pattern is a recurring observation. Several users report the same problem, many tickets concern the same function, or analytics shows repeated abandonments at a certain point. Patterns are important because they direct attention. But they are not yet an explanation.

A correlation describes that two things occur together. For example, users abandon more often on mobile devices than on desktop. This can be an important hint, but not yet a certain cause. Different causes are possible, such as a poor layout on small screens, a slower connection, a different usage situation, or a technical error.

A cause explains why a problem arises. Recognizing causes is harder than recognizing patterns. It often requires additional data, usability tests, user observation, support analysis, technical analysis, or domain knowledge. AI can formulate cause hypotheses, but it cannot prove causes from a pattern alone.

Priority describes what should be worked on next. Priority does not result from frequency alone. It also depends on severity, user impact, risk, business relevance, technical feasibility, regulatory relevance, accessibility relevance, and strategic importance. AI can provide prioritization suggestions, but it does not know the complete context.

A reliable product decision combines several sources of information: user feedback, support data, usage data, evaluation results, domain knowledge, technical feasibility, business goals, and risks. AI can structure this information, point out possible relationships, and formulate decision options.

The central learning point is: data shows hints. AI recognizes patterns. Humans make accountable decisions.

9.11 Recognizing UX debt

UX debt refers to deferred or incompletely solved problems of the usage experience. In software development teams, UX debt often shows very concretely as a usability problem: incomplete system states, inconsistent components, unclear texts, missing accessibility basics, hard-to-find functions, unnecessary work steps, or unresolved usability problems.

UX debt arises when usability problems are not solved, are only treated provisionally, or are created through quick decisions. Similar to technical debt, UX debt can cause higher costs later, because user problems, inconsistencies, workarounds, support effort, and rework increase.

UX debt is not every improvement idea. What is meant are weaknesses that impair usefulness, understandability, consistency, accessibility, efficiency, or satisfaction, or that make future changes more difficult.

Typical forms of UX debt are:

- incomplete system states and missing system feedback after actions,
- inconsistent components, non-uniform forms, and non-uniform terms,
- poor error messages and unclear navigation,
- missing or incomplete accessibility,
- unreviewed AI-generated drafts and outdated help texts,
- excessive cognitive load and unresolved usability problems,

- undocumented workarounds and recurring support problems without a sustainable solution.

In grown applications, UX debt often arises over a longer period. Different teams, different frameworks, fast features, changing priorities, or AI-generated one-off solutions can lead to inconsistencies. AI can help identify such inconsistencies and document them as UX debt. The team must decide which variant should apply in the future, which differences are professionally necessary, and which inconsistencies should be reduced.

UX debt should be documented. Good documentation names the affected area, the observed problem, the impact on users, the evidence or source, a possible measure, the risk of non-remediation, a rough priority, and the next step.

UX debt and technical debt can be related, but they are not identical. Technical debt primarily concerns technical structure, maintainability, architecture, or code quality. UX debt concerns the usage experience, such as understandability, consistency, findability, accessibility, or error tolerance.

9.12 Turning insights into backlog items and improvement measures

Insights after release only create value when they are turned into workable artifacts. For software development teams, this means: observations, feedback, metrics, and AI-assisted hypotheses must be translated into backlog items, acceptance criteria, UX debt measures, technical tasks, or new discovery questions.

A pattern from feedback or data only becomes actionable when a backlog item, a discovery question, or an improvement measure arises from it.

Example pattern:
“Many users report that they cannot find archived documents.”

Possible backlog item:
“As a professional user, I want to be able to find archived documents via search and filters so that I can quickly retrieve completed cases when questions arise.”

Possible discovery question:
“Which terms do users use for archived documents, and where do they expect them in the system?”

AI can suggest such translations. The team must check which form makes sense. If cause and user goal are clear enough, a backlog item can be created. If cause or user goal is unclear, a discovery question should be formulated first.

UX debt measures should be concrete. “Improve UX” is too imprecise. Better are measures such as unifying error messages in the registration process, adding empty states for project lists, consolidating table filters, checking focus order in the dialog system, unifying button naming, or fixing contrast problems in secondary actions.

Not every problem should be immediately translated into a solution. If cause or user goal is unclear, a new discovery question makes sense. If analytics shows that a function is rarely used, possible causes can be a lack of need, poor findability, unclear benefit, an unsuitable point in the workflow, missing permission, or technical problems.

Improvement measures should be prioritized based on user impact, risk, frequency, effort, strategic importance, accessibility relevance, regulatory significance, and technical dependency. AI can suggest an initial sorting, but prioritization remains a team and product decision.

When an improvement measure has been implemented, its effect should be observed. Renewed usability tests, usage data, the development of support tickets, feedback, or defined UX metrics can be used for this. This closes the learning loop: hints are recognized, hypotheses are formulated, measures are implemented, and their effect is observed again.

9.13 Embedding continuous improvement in the team

Continuous improvement means regularly including usability in teamwork, prioritization, and product decisions. Feedback, support data, metrics, and UX debt should not only be collected, but recurringly assessed, prioritized, and fed back into product development.

Teams can reflect on usability insights regularly. This can be part of a retrospective, a review format, a refinement, or a dedicated short usability checkpoint. What matters is not the specific meeting format, but that usability insights from real use, feedback, support data, metrics, and evaluation results are considered regularly.

Suitable questions are: Which usability problems have occurred repeatedly? Which assumptions were confirmed, refuted, or remain open? Which feedback topics are accumulating? Which support tickets point to comprehension or findability problems? Which UX debt items are growing? Which improvement should be scheduled next?

UX debt should remain visible, for example as a dedicated category in the backlog, as a label, as a list, as a quality board, or as part of a regular product quality review. Visibility prevents small problems from being left unaddressed permanently and condensing into larger usability problems over several releases.

The use of AI itself should also be reflected upon. Teams can observe where AI delivered good suggestions, where context was missing, where hallucinations occurred, and which prompts were particularly helpful. This improves not only usability work, but also AI competence within the team.

The insights after release flow back into discovery, backlog refinement, design, implementation, evaluation, and testing. This creates a continuous cycle:

Feedback and data → recognize patterns → formulate hypotheses → verify causes → prioritize → derive backlog items or UX debt measures → implement → test → observe again.

AI can support this cycle by structuring, summarizing, generating hypotheses, and preparing documentation. Monitoring without a feedback loop produces reports; monitoring with a feedback loop produces learning.

9.14 External support and the limits of team-internal AI use

External usability, research, accessibility, or domain expertise is particularly important when problems carry high risks. This concerns safety-critical processes, legal requirements, medical or financial applications, large user groups, accessible use, business-critical flows, or decisions with substantial impact on users.

External support can also make sense when a team is not sure which method is suitable. This concerns, for example, complex usability tests, quantitative UX measurement, accessibility audits, research design, A/B tests, statistical interpretation, or strategic usability decisions.

If certain usability problems persist over several releases, cause frequent support cases, or cannot be solved through internal measures, additional expertise should be considered. Recurring problems can indicate that the cause lies deeper, for example in information architecture, the mental model, product strategy, the domain process, accessibility, technical architecture, or organizational conditions.

AI-assisted self-help is valuable, but it has limits. When there is high uncertainty, high impact, or high responsibility, professionally reliable review is needed. AI cannot resolve such situations; at most, it can provide hints that further review is necessary.

External usability, research, accessibility, data analysis, or domain expertise is particularly necessary when decisions carry high risks, user groups are hard to reach, data is contradictory, accessibility conformance must be verified in a binding way, or AI analyses are not sufficiently reliable.

9.15 AI-assisted monitoring and human review

After release, AI is particularly good at structuring large amounts of unstructured information, uncovering UX debt, and formulating initial improvement hypotheses.

AI is good at	Clustering and summarizing feedback and support tickets; pointing out recurring topics; analyzing free texts; connecting technical signals with usability hypotheses; explaining metrics; formulating cause hypotheses; preparing UX debt lists; suggesting improvement measures and backlog items; deriving new discovery questions; preparing reports for reviews, retrospectives, or product decisions.
Limitations and typical risks	Proves no causes and sets no binding product priorities; business context, risk, strategy, regulatory requirements, and technical dependencies are often missing; typical risks: patterns are interpreted as causes and correlations as proof, feedback is assumed to be representative, loud feedback crowds out quiet but important problems, AI prioritization is adopted without review, UX debt is documented but not worked on, technical metrics are interpreted without usability context, data protection and confidentiality are neglected in analyses.
Requires human review	Which sources the hints come from and whether the data is representative or biased; whether personal or confidential data is affected and whether data protection, anonymization, and permissions have been considered; whether it is a pattern, a correlation, or a possible cause; which alternative explanations are possible; which user group is affected and how severe the impact is; which business, accessibility, or regulatory relevance exists; whether measures are concrete enough for the backlog.

AI-assisted monitoring should not only produce reports, but lead to reliable improvement decisions; interpretation, prioritization, and decision remain with the team or the responsible roles.

10 Literature

The following list contains central standards, professional foundations, and references on which this syllabus builds. It serves professional orientation and deepening.

External sources are exam-relevant only insofar as their contents are explicitly covered in this syllabus. Complete mastery of the standards, books, or online resources mentioned is not required for the foundation-level exam.

Further literature, practical examples, prompt examples, tool guidance, and in-depth sources are provided in separate accompanying materials.

Basic literature and central references

Norms and standards

[1] ISO 9241-11, Ergonomics of human-system interaction – Part 11: Usability: Definitions and concepts.

[2] ISO 9241-210, Ergonomics of human-system interaction – Part 210: Human-centred design for interactive systems.

[3] ISO/IEC 25010, Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.

[4] W3C, Web Content Accessibility Guidelines (WCAG) 2.2.

[5] EN 301 549, Accessibility requirements for ICT products and services.

[6] ISO/IEC 40500, Information technology – W3C Web Content Accessibility Guidelines (WCAG) 2.0.

Usability, user experience, and human-centered design

[7] J. Nielsen, “10 Usability Heuristics for User Interface Design,” Nielsen Norman Group, 1994.

[8] J. Nielsen and R. Molich, “Heuristic evaluation of user interfaces,” in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 1990.

[9] Nielsen Norman Group, Articles and resources on usability, user experience, usability testing, information architecture and interaction design.

Agile development

[10] K. Schwaber and J. Sutherland, The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game.

Artificial intelligence and governance

[11] European Union, Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act).

[12] NIST, Artificial Intelligence Risk Management Framework (AI RMF 1.0).

Accompanying materials

[13] R. Pucher and M. Simandl, UX Design mit AI. Accompanying textbook and practice book for this syllabus (in German).

[14] UXQCC, Accompanying prompt library and practice examples for the UXQCC Certified Professional in Software Usability with GenAI – Foundation Level.

Note for accredited training providers

For accredited training providers, more extensive training and accompanying materials are available in addition to the public syllabus edition. These can include, in particular, a provider edition of the syllabus, didactic guidance, practical examples, prompt examples, tool guidance, exercise suggestions, and further literature.

These materials support concrete training design and deepening. Exam-relevant remain the business outcomes, learning objectives, and explicitly covered contents formulated in this syllabus.

11 Annex 1 – Glossary of key terms

This glossary explains central terms that are essential for understanding this syllabus. It serves the consistent use of important terms from user experience, usability, accessibility, agile software development, and generative AI.

The glossary does not replace a comprehensive professional glossary on UX, accessibility, software engineering, or AI. Terms are exam-relevant only insofar as they are covered in the business outcomes, learning objectives, and chapter contents of this syllabus.

Term	Explanation
Acceptance criteria	Acceptance criteria are verifiable conditions that must be fulfilled for a backlog item to count as implemented. Usability-relevant acceptance criteria can concern, for example, understandability, system states, error cases, system feedback, expectable behavior, or accessibility basics.
Accessibility	Accessibility refers to designing digital systems so that they can also be used by people with different abilities, impairments, and usage situations. Accessibility is part of good UX and professional software quality. Important reference points include WCAG 2.2, EN 301 549, and ISO/IEC 40500.
AI governance	AI governance refers to rules, responsibilities, and review processes for the responsible use of AI in an organization. This includes, among other things, data protection, transparency, documentation, quality assurance, human review, and accountability.
AI output	AI output is the result produced by an AI system. In the usability context, AI output is to be treated as a suggestion, hypothesis, draft, or structuring aid, not as verified truth.
AI simulation	An AI simulation is an AI-generated hypothetical representation of possible user reactions, usage situations, or perspectives. It can generate hypotheses, but it replaces neither the observation of real users nor validation.
AI-assisted analysis	AI-assisted analysis refers to the use of AI for structuring, summarizing, or making an initial assessment of existing information. It can provide hints and hypotheses, but it replaces neither professional review, nor evaluation, nor validation.
AI-assisted evaluation of results	AI-assisted evaluation of results refers to the use of AI for ordering, clustering, summarizing, or preparing data, feedback, observations, or findings. The results

Term	Explanation
	must be reviewed by humans, in particular with regard to evidence, cause, priority, and professional relevance.
Analytics	Analytics refers to the collection and analysis of usage data, such as click paths, abandonments, frequency of use, or process completions. Analytics primarily shows what happens, but it does not automatically explain why it happens.
Anonymization	Anonymization refers to modifying data so that individuals can no longer be identified. It is particularly relevant when feedback, support tickets, usage data, or free texts are analyzed with AI.
ARIA	ARIA refers to technical attributes with which additional information can be provided for assistive technologies. ARIA should only be used when native HTML elements are not sufficient. Incorrectly used ARIA can make accessibility worse.
Attention	Attention refers to the limited ability to consciously process certain information. User interfaces must support attention in a targeted way and must not overwhelm users with unnecessary information, unclear priorities, or contradictory signals.
Backlog item	A backlog item is an entry in the product backlog or a comparable work list. It describes a requirement, task, improvement, correction, or clarification that can be prioritized and implemented.
Backlog refinement	Backlog refinement refers to the ongoing refinement of backlog items within the team. Items are clarified, split, prioritized, and provided with acceptance criteria. Refinement is central for usability because user goal, context of use, system states, error cases, and accessibility basics can be named there.
Business relevance	Business relevance describes the significance of a usability problem or an improvement for business goals, product strategy, costs, risks, usage, customer loyalty, or support effort. It is a criterion for prioritization.
Cause	A cause explains why an observed problem arises. Causes are harder to determine than patterns or correlations and often require additional data, tests, user observation, technical analysis, or domain knowledge.
Checkbox	A checkbox is a UI control for independent choices. Several checkboxes can be selected at the same time. It should not be used for mutually exclusive options.
Code review	Code review refers to the structured review of code by other people. For AI-generated code, attention should be paid not only to functionality, but also to

Term	Explanation
	usability-relevant aspects such as system states, error messages, focus order, accessibility, security, and maintainability.
Cognitive load	Cognitive load refers to the mental effort users need to understand information, make decisions, and perform tasks. Good UX reduces unnecessary cognitive load through clear structure, understandable language, recognizable priorities, and consistent flows.
Cognitive walkthrough	The cognitive walkthrough is an expert-based procedure in which a usage path is walked through step by step from the perspective of a user. The procedure works with four guiding questions: whether users recognize what the next sensible step is, whether they find the right element, whether they understand that this element fits the task, and whether they recognize after the action whether they were successful.
Component / UI component	A UI component is a reusable building block of a user interface, such as a button, form field, dialog, table, or error message. It comprises not only presentation, but also behavior, states, accessibility properties, and technical interfaces.
Component workshop	A component workshop is a development environment in which individual UI components are displayed, clicked through, and checked in isolation from the rest of the application, in different variants and system states. Each displayed manifestation of a component is called a story. A component workshop supports development, handoff, testing, and design system work.
Conformity with user expectations	Conformity with user expectations means that a system behaves as users expect based on their experience, task, and situation. Conformity with user expectations reduces learning effort, irritation, and erroneous operation.
Context of use	The context of use describes the conditions under which a system is used. This includes users, goals, tasks, devices, environment, organization, domain, risks, and situational conditions.
Control	A control is an operable UI element, such as a button, link, checkbox, radio button, dropdown, toggle, or tab. Controls should be used according to their expected mental model.
Correlation	Correlation means that two observations occur together, such as high abandonments on mobile devices. A correlation can be an important hint, but it does not automatically prove a cause.

Term	Explanation
Data protection	Data protection refers to the protection of personal data. In the context of AI-assisted usability work, it must be checked in particular which data may be entered into AI systems, stored, processed, or passed on.
Definition of Done / DoD	The Definition of Done is a team agreement on when work counts as done. Usability aspects can be included in it explicitly, such as verified system states, understandable error messages, accessibility basics, or fulfilled usability acceptance criteria.
Definition of Ready / DoR	The Definition of Ready is a team agreement on when a backlog item is sufficiently prepared to be taken into implementation or into a sprint. Usability-relevant aspects include a clear user goal, a known context of use, open assumptions, system states, error cases, and validation needs.
Design critique	A design critique is a structured review format for the professional assessment of drafts based on comprehensible criteria such as user goal, understandability, conformity with user expectations, consistency, system states, and accessibility. It differs from a matter-of-taste discussion in that criteria and results are documented.
Design system	A design system describes reusable UI components, design rules, interaction patterns, texts, system states, and quality criteria. It supports consistent, accessible, and maintainable user interfaces.
Design token	Design tokens are named values for design properties such as colors, spacing, font sizes, border radius, or shadows. They connect design and code and support consistency, maintainability, and design systems.
Discovery	Discovery refers to activities with which users, context of use, tasks, problems, goals, risks, and hypotheses are better understood before or during implementation.
Edge case	An edge case is a rare but possible special case. Edge cases are important because in real use they can lead to uncertainty, errors, data loss, or support needs.
Effectiveness	Effectiveness describes whether users can achieve their goals with a system correctly and completely. In the software context, this concerns, for example, successful task completion, correct inputs, or finding relevant functions.
Efficiency	Efficiency describes the effort with which users achieve their goals. This includes, among other things, time, number of steps, search effort, unnecessary repetitions, and cognitive load.

Term	Explanation
Empty state	An empty state is a system state in which no data or contents are available. Good empty states explain why nothing is displayed and which next action is possible.
Error message	An error message informs users about a problem and should be understandable, concrete, action-oriented, and associated with the affected area. Good error messages support error recovery and avoid unnecessary frustration.
Error rate	The error rate describes how often users make errors in a task, such as wrong inputs, repeated validation errors, misclicks, or misunderstood steps. It must be interpreted in connection with severity, context, and user impact.
Error state	An error state is a system state in which an error has occurred. Good error states explain understandably what happened, what impact this has, and how users can proceed.
Evaluation	Evaluation refers to the systematic assessment of a draft, prototype, product, or usage path with regard to UX, usability, user goal, and context of use. Evaluation can be conducted with real users, for example through usability tests, or expert-based, for example through heuristic evaluation or a cognitive walkthrough.
Evidence	Evidence refers to the basis on which a statement, a finding, or a decision rests. Evidence can come from observations, tests, usage data, support data, domain knowledge, or professional review. AI output without review is no reliable evidence.
Expert review / expert-based procedure	An expert review is a structured check by subject-matter experts or UX experts. Examples are heuristic evaluation or the cognitive walkthrough. Expert reviews can uncover possible usability problems, but they do not replace usability tests with real users.
Feedback	In this syllabus, feedback refers to responses from users, customers, support, operations, or other stakeholders on the actual use or the productive environment of a system. This is to be distinguished from system feedback, that is, the system's response to users during interaction.
Finding / UX finding	A finding is a documented insight from evaluation, usability testing, expert review, support analysis, or another check. A good finding describes observation, context, impact, evidence, and, where applicable, cause or recommendation.

Term	Explanation
Focus order	Focus order describes how the input focus is guided through an interface in a visible, logical, and expectable way. It is particularly important for keyboard operability, screen reader use, dialogs, menus, and complex forms.
Formative and summative evaluation	Formative evaluation serves improvement during development, for example through early usability tests on prototypes. Summative evaluation assesses a more advanced or finished solution, for example for acceptance or comparison. Both forms can be expert-based or involve real users.
Frontend	Frontend refers to the technically implemented part of a digital system with which users interact directly. This includes UI code, components, interaction logic, system states, presentation, and accessibility properties.
GenAI	GenAI stands for generative artificial intelligence. GenAI can create new content, such as texts, code, UI suggestions, summaries, test ideas, analysis suggestions, or prototypes.
Gestalt principles	Gestalt principles describe principles of human perception, such as proximity, similarity, common region, closure, or good form. They help understand how users group and structure visual elements.
Goals–Signals–Metrics	Goals–Signals–Metrics is the procedure by which a general UX dimension becomes a measure relevant to decisions. Goals describe which user or product goal is to be improved. Signals are observable indications of whether this goal is being achieved. Metrics are the concrete measures with which these indications are captured. The procedure belongs to the HEART framework.
Hallucination	A hallucination is a plausible-seeming but wrong or unsubstantiated output of an AI system. Hallucinations are particularly risky when they are adopted uncritically into requirements, designs, tests, reports, or code.
Handoff	Handoff refers to the alignment and transfer of information between design, AI support, and development. A good handoff describes not only what something looks like, but also user goal, components, states, behavior, texts, accessibility requirements, acceptance criteria, and open assumptions.
Happy path	The happy path is the ideal flow in which everything works as planned. Good UX additionally considers error cases, abandonments, empty states, alternative paths, permissions, and special cases.
HEART framework	The HEART framework structures UX metrics into the dimensions happiness, engagement, adoption, retention, and task success. It helps connect UX goals, signals, and metrics systematically.

Term	Explanation
Heuristic evaluation	Heuristic evaluation is an expert-based procedure in which an interface is checked against defined usability heuristics or design principles. It provides indications of possible usability problems, but it does not replace a usability test with real users.
High-fidelity prototype	A high-fidelity prototype has a high level of detail and more closely resembles the later product. It can contain visual design and more realistic interactions, but it can also prematurely create the impression of a finished solution.
Horizontal prototype	A horizontal prototype shows many areas or screens of a system in breadth, but elaborates individual functions only superficially. It is particularly suitable for discussing structure, navigation, and scope.
Human oversight	Human oversight means that AI results are deliberately reviewed, assessed, and approved by responsible persons. AI can support, but it cannot take on professional responsibility.
Human review	Human review refers to the professional, methodological, and context-related control of AI output, drafts, analyses, or code by responsible persons. It is necessary because AI results can be wrong, incomplete, or blind to context.
Human-centered design	Human-centered design is a design approach in which users, their goals, requirements, abilities, and contexts of use are considered systematically. The goal is to design interactive systems so that they are usable, useful, and appropriate for people.
Human-factors criteria	Human-factors criteria bring human characteristics, abilities, and limits into design. These include perception, attention, memory, mental models, cognitive load, motor abilities, and situational constraints.
Hypothesis	A hypothesis is a verifiable assumption about users, tasks, problems, causes, or solutions. AI-generated usability suggestions are often hypotheses and must be verified through professional review, evaluation, domain knowledge, usage data, or usability tests.
Information architecture	Information architecture describes how contents, functions, and objects are structured, named, and made findable. It influences whether users understand where they are and where they expect relevant information or functions.
Interaction design	Interaction design refers to the design of the interaction between human and system. This includes flows, system states, controls, system feedback, error handling, controllability, and expectable behavior.

Term	Explanation
Keyboard operability	Keyboard operability means that interactive elements can be reached and operated without a mouse. It is a fundamental prerequisite for accessibility and also relevant for efficient operation.
KPI	KPI stands for key performance indicator and refers to a measure for assessing important product, business, or quality goals. UX metrics can be connected to KPIs, but they are not automatically the same.
Label	A label refers to the caption of an input field or control. Labels should be understandable, professionally correct, and programmatically associated with the respective element. Placeholder texts do not replace labels.
Loading state	A loading state is a system state indicating that data is being loaded or an action is being processed. It helps users understand that the system is working and that no final answer is available yet.
Low-fidelity prototype	A low-fidelity prototype has a low level of detail, such as a sketch, a simple wireframe, or a rough click flow. It is suitable for early discussions, structural questions, and variants.
Mental model	A mental model is the idea users have of how a system, process, or domain object works or should work. When a system works against this mental model, errors, uncertainty, or frustration often arise.
Minimal viable design system	A minimal viable design system is a deliberately small, maintainable initial version of a design system. It contains central elements such as colors, typography, spacing, buttons, form fields, error messages, system states, and fundamental accessibility rules.
Mockup	A mockup is a visually elaborated representation of an interface. It shows layout, colors, typography, and UI elements, but it does not necessarily represent interaction or system behavior completely.
Monitoring	Monitoring refers to the systematic observation of feedback, support data, usage data, technical signals, and UX metrics after release. It serves to recognize indications of problems, patterns, UX debt, and improvement opportunities.
Observation	Observation refers to systematically capturing what users actually do, say, look for, overlook, misunderstand, or abandon. In a usability test, observation is central and must be distinguished from interpretation.

Term	Explanation
Operability	Operability means that a system can actually be controlled and used. This includes, among other things, keyboard operability, visible focus, sufficient target sizes, predictable interaction, and usability with different input methods.
Pattern	A pattern is a recurring observation, such as frequent abandonments at the same point or recurring support tickets for a function. Patterns direct attention, but they do not yet automatically explain causes.
Pattern recognition	Pattern recognition refers to the ability of AI systems to recognize familiar patterns in texts, interfaces, data, or flows and to derive suggestions from them. AI is particularly helpful with well-known, frequently occurring patterns.
Perceivability	Perceivability means that information and controls must be recognizable for users. This includes, for example, sufficient contrast, clear structure, text alternatives, and information that is not conveyed by color alone.
Perception	Perception is the process by which people take in and process information from their environment. Perception is selective, limited, and context-dependent. For UX this means: visibility alone is not enough; information must be understandably perceivable at the right moment.
Persona	A persona is a typified description of a user group with goals, tasks, prior experience, and context of use. Data-based personas rest on interviews, observations, usage data, or reliable domain knowledge. Hypothetical personas, including AI-generated ones, make assumptions explicit, but they are not research results and must be labeled as hypotheses.
Priority	Priority describes what should be worked on next. It does not result from frequency alone, but from severity, user impact, risk, business relevance, effort, accessibility relevance, and strategic importance.
Product decision	A product decision is an accountable decision about the direction, priority, implementation, or change of a product. It should rest on suitable evidence, user impact, business relevance, risks, and technical feasibility.
Product owner	The product owner is a role in Scrum and in many agile product teams. In the usability context, this role is important because user goals, prioritization, acceptance criteria, and UX debt are strongly influenced by product decisions.
Prompt	A prompt is an input or task instruction to an AI system. In the usability context, a prompt can, for example, describe an interface, have a user story reviewed, or request improvement suggestions.

Term	Explanation
Prototype	A prototype is a preliminary representation or implementation of a solution. It serves to make ideas visible, discussable, verifiable, or testable. Prototypes are not finished products and no automatic validation.
Radio button	A radio button is a UI control for a choice among several mutually exclusive options. It should not be used for multiple selection.
Readability	Readability describes how well texts can be visually captured and read. It depends, among other things, on font size, contrast, line length, line spacing, font weight, and information density.
Release	A release refers to making a product, increment, or feature available for actual use. After release, feedback, usage data, and metrics can provide indications of real UX quality.
Remote usability test	A remote usability test is a usability test conducted at a distance. It can be moderated or unmoderated and is useful when users are geographically distributed or should be tested in their own context of use.
Research question	A test question or research question describes what is to be found out through an evaluation or a usability test. It determines test participants, test object, test tasks, and observation criteria.
Retention	Retention describes whether users return after a first use or continue to use a product or function permanently. It is a possible UX or product metric.
Review format	A review format is a structured discussion or check of a work result, for example within the team, with stakeholders, or with subject-matter experts. Review formats can make usability-relevant questions visible, but they are not automatically evaluation methods and do not replace validation with real users.
Robustness	Robustness means that digital contents and interactions work reliably with different devices, browsers, and assistive technologies. Robustness is particularly important for user interfaces that are stable and accessible in the long term.
Satisfaction	Satisfaction describes how users subjectively experience the use of a system. It comprises, among other things, trust, frustration, safety, appropriateness, and the feeling of being able to accomplish a task well.
Scrum	Scrum is a widely used agile process framework for product and software development. In this syllabus, Scrum is used descriptively to classify typical agile roles, artifacts, and events. The syllabus, however, is not limited to Scrum.

Term	Explanation
Security review	A security review is a check of security-relevant aspects of a solution or implementation. For AI-generated code, it is important to check whether security requirements, permissions, data processing, and input validation have been implemented correctly.
Semantic HTML	Semantic HTML means using appropriate HTML elements according to their meaning and function, such as buttons for actions, links for navigation, headings for headings, and labels for form fields. It is a foundation for accessibility and robust UI implementation.
Severity	Severity describes the significance of a usability problem. It can result from user impact, frequency, risk, task relevance, business relevance, and fixability.
Sprint	A sprint is a time-boxed work cycle in Scrum. In this syllabus, the term is used when concrete Scrum-related planning, implementation, or review situations are described. More generally, the syllabus speaks of agile development processes, agile planning, or agile implementation.
Sprint review	The sprint review is a Scrum event in which work results are shown and discussed with stakeholders. In this syllabus, it is used descriptively as an example of an agile review format in which usability-relevant questions can also be considered.
State / system state	A state is a concrete state of an interface, such as the default state, loading state, empty state, error state, warning state, success state, or disabled state. Missing or unclear states are a frequent cause of poor UX.
Support data	Support data arises from support tickets, helpdesk requests, chat logs, internal questions, or recurring complaints. It can provide indications of usability problems, but it often initially shows symptoms and not automatically causes.
Synthetic user	A synthetic user is an AI-generated user profile or a simulated user reaction. Synthetic users can generate perspectives and hypotheses, but they replace neither real users, nor research data, nor usability tests.
System feedback	System feedback refers to the system's responses to users during interaction, such as loading states, error messages, success messages, confirmations, or hints after an action. It is to be distinguished from user feedback or customer feedback.
Task	A task is an activity that users perform with a system in order to achieve a user goal. Tasks are always related to user goal and context of use.

Term	Explanation
Task success	Task success describes whether users can successfully complete a relevant task. Task success is an important UX metric and closely connected to effectiveness.
Test object	The test object is the artifact or system that is checked in an evaluation or a usability test, such as a wireframe, prototype, staging version, existing application, or individual flow.
Test participant	A test participant is a person who works on tasks in a usability test. They should fit the relevant user group or the context of use to be checked as closely as possible.
Test task	In a usability test, a test task describes a goal or a situation from the user perspective without prescribing the solution path. It should be derived from a research question and enable observable success.
Think-aloud	Think-aloud is a supporting method in which test participants say out loud while working on a task what they think, look for, expect, or do not understand. The method can make mental models visible, but it can also influence behavior and increase cognitive load.
Toggle	A toggle is a UI control for switching a state on or off. Users often expect a toggle to take effect immediately or to show clearly when a change has been applied.
Traceability	Traceability refers to the ability to trace requirements, decisions, implementation, tests, and results. In the usability context, it helps recognize which usability requirement is covered by which implementation and which check.
Transparency of AI use	Transparency of AI use means that it remains traceable where AI was used, with what goal, which results were adopted, and what human review took place. Transparency prevents AI-generated contents from later being wrongly treated as a reliable professional basis.
Tree testing	Tree testing is a procedure for checking information architecture. Test participants receive tasks and look for suitable information or functions in a reduced, text-based navigation structure.
UI / user interface	The user interface is the concrete interface of a system. This includes screens, forms, navigation, controls, texts, interaction elements, and visible system states.

Term	Explanation
UI design	UI design refers to the design of the concrete user interface. This includes layout, visual hierarchy, typography, controls, texts, states, and interaction possibilities.
UI pattern	A UI pattern is a proven solution for a recurring interaction problem, such as breadcrumb, modal dialog, tab navigation, search filter, or inline validation. Patterns must fit the user goal and the context of use.
Understandability	Understandability means that users can comprehend contents, terms, states, error messages, and next steps. Understandability concerns language, structure, system feedback, navigation, and system behavior.
Unfamiliar context	Unfamiliar context refers to information about users, domain, tasks, risks, or real usage situations that AI does not know. With unfamiliar context, human review, domain knowledge, and, where necessary, real user observation are required.
Usability	Usability describes the extent to which specific users can use a product in a specific context of use to achieve specific goals effectively, efficiently, and with satisfaction. Usability is therefore not a matter of taste, but an assessable quality of use.
Usability heuristic	A usability heuristic is a general assessment principle or rule with which typical usability problems can be made visible. It serves as a frame of reference for heuristic evaluation.
Usability requirement	A usability requirement is a requirement on the usage quality of a system. Usability requirements concern, for example, understandability, efficiency, error prevention, system feedback, accessibility, expectable behavior, cognitive relief, or system states.
Usability test	A usability test is an evaluation method in which real or representative users work on concrete tasks with a product, prototype, or system. What is observed is behavior, comprehension problems, errors, hesitation, search movements, and task success.
Usage data	Usage data describes how a system is actually used, such as abandonments, frequency of use, click paths, task times, or repeated errors. It shows what happens, but it does not automatically explain why it happens.
User	A user is a person who actually uses or is supposed to use a product, system, or service. The user is not necessarily identical with the buyer, client, product owner, or stakeholder.

Term	Explanation
User experience (UX)	User experience comprises a person's perceptions and responses that result from the use or anticipated use of a product, system, or service. UX covers experiences before, during, and after use and thus goes beyond pure usability.
User feedback	User feedback refers to responses from users on actual use, such as hints, complaints, wishes, ratings, or experience reports. User feedback is valuable, but not automatically representative.
User goal	A user goal is the goal that users want to achieve with the help of a system. Good UX supports users in achieving their goals effectively, efficiently, and understandably.
User requirement	A user requirement is a requirement that results from user goals, tasks, and the context of use. It describes what users need in a specific situation in order to achieve their goal.
User story	A user story is a short description of a requirement from the user or stakeholder perspective. In the usability context, a user story should not only describe functionality, but also consider user goal, context of use, and expected use.
UX debt	UX debt refers to accumulated or deliberately deferred weaknesses in the usage experience of a product. This includes, for example, incomplete system states, inconsistent components, unclear terms, poor error messages, missing accessibility, or unresolved usability problems. UX debt can later cause additional effort, support needs, erroneous operation, or loss of trust.
UX metric	A UX metric is a measure for assessing use and user experience. Examples are task success, error rate, time on task, satisfaction, abandonment rate, return rate, or support volume. UX metrics show indications, but not automatically causes.
UX quality	UX quality refers to the quality of the use and usage experience of a system. This includes, among other things, understandability, conformity with user expectations, efficiency, error prevention, satisfaction, accessibility, and fit with the context of use.
Validation	Validation means checking whether assumptions, requirements, or solutions hold up for the actual users, tasks, and contexts of use. In the usability context, validation can be supported by usability tests, real usage data, domain expertise, targeted user observation, or other suitable evidence. AI simulations and AI analyses provide hypotheses, but no validation.

Term	Explanation
Verification	Verification means checking whether a solution conforms to the specified requirements. It answers the question of whether something was implemented according to specification. Validation, by contrast, answers whether the solution is suitable for users in the context of use.
Vertical prototype	A vertical prototype elaborates a selected functional area or flow in more depth. It is suitable when a concrete flow, a function, or a critical interaction path is to be checked more closely.
Wireframe	A wireframe is a schematic representation of the structure, content, navigation, and basic function of an interface. Visual details are not the focus.