Certified Tester Specialist Level Testing with Generative AI (CT-GenAI) Syllabus

v1.0

International Software Testing Qualifications Board

Copyright Notice

Revision History

Version	Date	Remarks
V1.0	2024/12/11	CT-GenAI 1.0 Alpha Release
V1.0	2025/06/01	CT-GenAI 1.0 Beta Release
v1.0	2025/06/10	CT-GenAI v1.0 Release

Table of Contents

Copyright Notice 2
Revision History 3
Table of Contents 4
Acknowledgements 7
0 Introduction 8
0.1 Purpose of this Syllabus 8
0.2 Software Testing with Generative AI 8
0.3 Career Path for Testers 8
0.4 Business Outcomes 8
0.5 Examinable Learning Objectives, Hands-on Objectives and Cognitive Level of Knowledge 9
0.6 The Certified Tester Testing with Generative AI Certificate Exam 10
0.7 Accreditation 10
0.8 Handling of Standards 10
0.9 Level of Detail 10
0.10 How this Syllabus is Organized 11
1 Introduction to Generative AI for Software Testing – 100 minutes 13
1.1 Generative AI Foundations and Key Concepts 14
1.1.1 AI Spectrum: Symbolic AI, Classical Machine Learning, Deep Learning, and Generative AI 14
1.1.2 Basics of Generative AI and LLMs 14
1.1.3 Foundation, Instruction-Tuned and Reasoning LLMs 16
1.1.4 Multimodal LLMs and Vision-Language Models 16
1.2 Leveraging Generative AI in Software Testing: Core Principles 17
1.2.1 Key LLM Capabilities for Test Tasks 17
1.2.2 AI Chatbots and LLM-Powered Testing Applications for Software Testing 18
2 Prompt Engineering for Effective Software Testing – 365 minutes 19
2.1 Effective Prompt Development 21
2.1.1 Structure of Prompts for Generative AI in Software Testing 21
2.1.2 Core Prompting Techniques for Software Testing 22

2.1.3 System Prompt and User Prompt ... 23
2.2 Applying Prompt Engineering Techniques to Software Test Tasks ... 24
2.2.1 Test Analysis with Generative AI ... 24
2.2.2 Test Design and Test Implementation with Generative AI ... 25
2.2.3 Automated Regression Testing with Generative AI ... 27
2.2.4 Test Monitoring and Test Control with Generative AI ... 28
2.2.5 Choosing Prompting Techniques for Software Testing ... 29
2.3 Evaluate Generative AI Results and Refine Prompts for Software Test tasks ... 30
2.3.1 Metrics for Evaluating the Results of Generative AI on Test tasks ... 30
2.3.2 Techniques for Evaluating and Iteratively Refining Prompts ... 31
3 Managing Risks of Generative AI in Software Testing – 160 minutes ... 33
3.1 Hallucinations, Reasoning Errors and Biases ... 34
3.1.1 Hallucinations, Reasoning Errors and Biases in Generative AI ... 34
3.1.2 Identify Hallucinations, Reasoning Errors and Biases in LLM Output ... 34
3.1.3 Mitigation techniques of GenAI hallucinations, reasoning errors and biases in software test tasks 36
3.1.4 Mitigation of Non-Deterministic Behavior of LLMs ... 36
3.2 Data Privacy and Security Risks of Generative AI in Software Testing ... 37
3.2.1 Data Privacy and Security Risks Associated with Using Generative AI ... 37
3.2.2 Data Privacy and Vulnerabilities in Generative AI for Test processes and Tools ... 37
3.2.3 Mitigation Strategies to Protect Data Privacy and Enhance Security in Testing with Generative AI ... 38
3.3 Energy Consumption and Environmental Impact of Generative AI in Software Testing ... 39
3.3.1 The Impact of Using GenAI on Energy Consumption and CO2 Emissions ... 39
3.4 AI Regulations, Standards, and Best Practice Frameworks ... 40
3.4.1 AI Regulations, Standards and Frameworks Relevant to GenAI in Software Testing ... 40
4 LLM-Powered Test Infrastructure for Software Testing – 110 minutes ... 42
4.1 Architectural Approaches for LLM-Powered Test Infrastructure ... 43
4.1.1 Key Architectural Components and Concepts of LLM-Powered Test Infrastructure ... 43
4.1.2 Retrieval-Augmented Generation ... 44
4.1.3 The Role of LLM-Powered Agents in Automating Test processes ... 45
4.2 Fine-Tuning and LLMOps: Operationalizing Generative AI for Software Testing ... 45

4.2.1 Fine-Tuning LLMs for Test tasks 46
4.2.2 LLMOps when Deploying and Managing LLMs for Software Testing 47
5 Deploying and Integrating Generative AI in Test organizations – 80 minutes 48
5.1 Roadmap for the Adoption of Generative AI in Software Testing 49
5.1.1 Risks of Shadow AI 49
5.1.2 Key Aspects of a Generative AI Strategy in Software Testing 49
5.1.3 Selecting LLMs/SLMs for Software Test Tasks 50
5.1.4 Phases when Adopting Generative AI in Software Testing 50
5.2 Manage Change when Adopting Generative AI for Software Testing 51
5.2.1 Essential Skills and Knowledge for Testing with Generative AI 51
5.2.2 Building Generative AI Capabilities in Test Teams 51
5.2.3 Evolving Test Processes in AI-Enabled Test organizations 52
6 References 53
Standards 53
ISTQB® Documents 53
Glossary References 53
Books 53
Articles 53
Web Pages 54
7 Appendix A – Learning Objectives/Cognitive Level of Knowledge 55
Level 1: Remember (K1) 55
Level 2: Understand (K2) 55
Level 3: Apply (K3) 56
8 Appendix B – Business Outcomes traceability matrix with Learning Objectives 57
9 Appendix C – Release Notes 64
10 Appendix D – Generative AI Specific Terms 65
11 Appendix E – Trademarks 68
12 Index 69

Acknowledgements

This document was formally released by the General Assembly of the ISTQB® on 25/07/2025.

It was produced by a team from the International Software Testing Qualifications Board: Abbas Ahmad (product owner), Gualtiero Bazzana, Alessandro Collino, Olivier Denoo, and Bruno Legeard (technical manager).

The team thanks Anne Kramer, Jedrzej Kwapinski, Samuel Ouko and Ina Schieferdecker for their technical review and the review team and the Member Boards for their suggestions and input.

The following persons participated in the reviewing, commenting and balloting of this syllabus:

Albert Laura, Aneta Derkova, Anne Kramer, Arda Ender Torçuk, Baris Sarialioglu, Claire Van Der Meulen, Daniel van der Zwan, Derek Young, Dietmar Gehring, Francisca Cano Ortiz, Gary Mogyorodi, Gergely Ágnecz, Horst Pohlmann, Ina Schieferdecker, Ingvar Nordström, Jan Sabak, Jaroslaw Hryszko, Jedrzej Kwapinski, Joanna Kazun, Karol Frühauf, Katalin Balla, Koray Yitmen, Laura Albert, Linda Vreeswijk, Lucjan Stapp, Lukáš Piška, Mario Winter, Marton Siska, Mattijs Kemmink, Matthias Hamburg, Meile Posthuma, Michael Stahl, Márton Siska, Nele Van Asch, Nils Röttger, Nishan Portoyan, Piet de Roo, Piotr Wicherski, Péter Földházi, Péter Sótér, Radoslaw Smilgin, Ralf Pichler, Renzo Cerquozzi, Rik Marselis, Samuel Ouko, Stephanie Ulrich, Stuart Reid, Tal Pe'er, Tamás Gergely, Thomas Letzkus, Wim Decoutere, Zsolt Hargitai, Mark Rutz, Patrick Quilter, Earl Burba, Taz Daughtrey, Judy McKay, Randall Rice, Thomas Adams, Tom Van Ongeval, Sander Mol, Miroslav Renda, Geng Chen, Chai Afeng, Xinghan Li, Klaudia Dussa-Zieger, Arnd Pehl, Florian Fieber, Ray Gillespie, József Kreisz, Dénes Medzihradszky, Ferenc Hamori, Giorgio Pisani, Giancarlo Tomasig, Young jae Choi, Arnika Hryszko, Andrei Brovko, Ilia kulakov, Praveen, Kostas Pashalidis, Ferdinand Gramsamer, A. Berfin Öztaş, Abdullah Gök, Abdurrahman AKIN, Aleyna Zuhal IŞIK, Anıl Şahin, Atakan Erdemgil, Aysel Bilici, Azmi YÜKSEL, Bilal Gelik, Bilge Yazıcı, Burak Gel, Burcu ÖZEL, Büşra İlayda Çevik Köken, Can Polat, Canan Ayten Dörtkol (Polat), Cansu Mercan Daldaban, Denizcan Orhun Karaca, Didem Çiçek Bay, Duygu Yalçınkaya, Efe Can Yemez, ELİF CERAV, Emine Tekiner, Emre Aman, Emre Can Akgül, Esra Kücük, Gençay GENÇ, Gül Çalışır Açan, Gül Nihal SİNGİL, Güler GÖK, Gulhanim Anulur, Hakan GÜVEZ, Haktan Bilgehan Dilber, Halil Ibrahim Tasdemir, Hasan Küçükayar, Hatice Erdoğan, Hatice Kübra Daşdoğan, Hüseyin Sevki ARI, Hyulya Gyuler, İLKNUR NEŞE TUNCAL, Kaan Eminğlu, Kamil Isik, Koray Danışman, Melisa Canbaz, Merve Guleroglu, Müjde Ceylan, Mustafa Furkan CEYLAN, Nergiz Gençaslan, Nuh Soner Bozkurt, Omer Fatih Poyraz, Onur Ersoy, Özlem Körpe, Özgür Özdemir, Sedat YOLTAY, Selahattin Aliyazıcıoğlu, Sevan Lalikoğlu, Sebastian Malyska, Sevim Öykü Demirel, Tatsiana Beliai, Tayg.

0 Introduction

0.1 Purpose of this Syllabus

This syllabus forms the basis for the International Software Testing Qualification Board for Testing with generative AI (CT-GenAI) qualification. The ISTQB® provides this syllabus as follows:

1. To member boards, to translate into their local language and to accredit training providers. Member boards may adapt the syllabus to their particular language needs and modify the references to adapt to their local publications.
2. To certification bodies, to derive examination questions in their local language adapted to the learning objectives for this syllabus.
3. To training providers, to produce courseware and determine appropriate teaching methods.
4. To certification candidates, to prepare for the certification exam (either as part of a training course or independently).
5. To the international software and systems engineering community, to advance the profession of software and systems testing, and as a basis for books and articles.

0.2 Software Testing with Generative AI

The Testing with generative AI qualification is aimed at anyone involved in using generative AI (GenAI) for software testing. This includes people in roles such as testers, test analysts, test automation engineers, test managers, user acceptance testers and software developers. This Testing with GenAI qualification is also appropriate for anyone who wants a basic understanding of using GenAI for software testing, such as project managers, quality managers, software development managers, business analysts, IT directors and management consultants.

0.3 Career Path for Testers

The ISTQB® scheme provides support for testing professionals at all stages of their careers offering both breadth and depth of knowledge. Individuals who achieve the ISTQB® Certified Tester Testing with generative AI certification may also be interested in Core Advanced Levels (Test Analyst, Technical Test Analyst, Test Manager, and Test Engineering) and thereafter Expert Level (Test Management or Improving the Test Process). Please visit www.istqb.org for the latest information of ISTQB´s Certified Tester Scheme.

0.4 Business Outcomes

This section lists the Business Outcomes expected of a candidate who has achieved the Testing with generative AI certification.

A candidate who has achieved the Testing with Generative AI certification can:

GenAI-BO1	Understand the fundamental concepts, capabilities, and limitations of generative AI
GenAI-BO2	Develop practical skills in prompting large language models for software testing
GenAI-BO3	Gain insight into the risks and mitigations of using generative AI for software testing
GenAI-BO4	Gain insight into the applications of generative AI solutions for software testing
GenAI-BO5	Contribute effectively to the definition and implementation of a generative AI strategy and roadmap for software testing within an organization

0.5 Examinable Learning Objectives, Hands-on Objectives and Cognitive Level of Knowledge

Learning and hands-on objectives support the business outcomes and are used to create certification exams for Testing with Generative AI.

In general, all contents of this syllabus are examinable at a K1, K2 and K3 levels, except for the Introduction, Hands-on Objectives and Appendices. The exam questions will confirm knowledge of keywords at K1 level (see below) or learning objectives at all K-levels.

The specific learning objectives levels are shown at the beginning of each chapter, and classified as follows:

- K1: Remember
- K2: Understand
- K3: Apply

Further details and examples of learning objectives are given in Appendix A.

All terms listed as keywords just below chapter headings shall be remembered, even if not explicitly mentioned in the learning objectives.

The specific hands-on objectives (HO) are shown at the beginning of each chapter. Each HO is linked to a LO at level K2 or K3, with the aim of refining learning through hands-on practice. The level of a HO is classified as follows:

- H0: This can include a live demo of an exercise or recorded video. Since this is not performed by the trainee, it is not strictly an exercise.
- H1: Guided exercise. The trainees follow a sequence of steps performed by the trainer.
- H2: Exercise with hints. The trainee is given an exercise with relevant hints to enable the exercise to be solved within the given timeframe.

0.6 The Certified Tester Testing with Generative AI Certificate Exam

The Certified Tester Testing with Generative AI Certificate exam will be based on this syllabus. Answers to exam questions may require the use of material based on more than one section of this syllabus. All sections of the syllabus are examinable, except for the Introduction, Hands-on objectives and Appendices. Standards, books and articles are included as references, but their content is not examinable, beyond what is summarized in the syllabus itself.

Refer to Exam Structures and Rules V1.0 document for Certified Tester Testing with Generative AI for further details.

Entry Requirement Note: The ISTQB® Foundation Level certificate shall be obtained before taking the ISTQB® Certified Tester Testing with Generative AI certification exam.

0.7 Accreditation

An ISTQB® Member Board may accredit training providers whose course material follows this syllabus. Training providers should obtain accreditation guidelines from the Member Board or body that performs the accreditation. An accredited course is recognized as conforming to this syllabus, and is allowed to have an ISTQB® exam as part of the course.

The accreditation guidelines for this syllabus are defined in the ISTQB CT-GenAI Accreditation Guidelines document.

0.8 Handling of Standards

There are standards associated with quality characteristics and software testing, namely the ones referenced in the Foundational Level syllabus like by IEEE and ISO. The purpose of these references is to provide a framework or to provide a source of additional information if desired by the reader. Please note that syllabi are using the standard documents as reference. Standards documents are not intended for examination. Refer to Chapter 6 for more information on Standards.

0.9 Level of Detail

The level of detail in this syllabus allows internationally consistent courses and exams. In order to achieve this goal, the syllabus consists of:

- General instructional objectives describing the intention of the ISTQB® Certified Tester Testing with Generative AI certification
- A list of terms that students must be able to recall
- Learning objectives for each knowledge area, describing the cognitive learning outcome to be achieved
- A description of the key concepts, including references to sources such as accepted literature or standards

- A description for each hands-on objective of the recommended practice to support learning

The syllabus content is not a description of the entire knowledge area of testing with GenAI; it reflects the level of detail to be covered in ISTQB® Certified Tester Testing with Generative AI training courses. It focuses on test concepts and techniques that can apply to all software projects when using generative AI for testing.

The syllabus uses the terminology (i.e. the name and meaning) of the terms used in software testing and quality assurance according to the ISTQB® Glossary.

0.10 How this Syllabus is Organized

There are 5 chapters with examinable content. The top-level heading for each chapter specifies the time for the chapter; timing is not provided below chapter level. For accredited training courses, the syllabus requires a minimum of 13,6 hours of instruction, distributed across the 5 chapters as follows:

- Chapter 1: 100 minutes Introduction to Generative AI for Software Testing
  - The tester learns basics of large language models (LLMs), including tokenization and multi-modal capabilities.
  - The tester explores applications of Generative AI (GenAI) in software testing, distinguishing AI chatbot from LLM-powered test tools, and experimenting with tokenization, context windows, and multi-modal prompts.
- Chapter 2: 365 minutes Prompt Engineering for Effective Software Testing
  - The tester learns to craft effective, structured prompts for GenAI in software testing.
  - The tester gains hands-on experience with prompt engineering techniques for software test tasks and applies them.
- Chapter 3: 160 minutes Managing Risks of Generative AI in Software Testing
  - The tester learns to identify and mitigate hallucinations, reasoning errors, and biases when testing with GenAI.
  - The tester learns to address data privacy and security issues of GenAI in software testing.
  - The tester learns energy consumption and environmental impact of GenAI in software testing.
  - The tester learns AI regulations, standards and best practices for ethical, transparent, and secure GenAI use in software testing.
- Chapter 4: 110 minutes LLM-Powered Test Infrastructure for Software Testing
  - The tester explores GenAI architecture like Retrieval-Augmented Generation and GenAI agents.
  - The tester learns the process to fine-tune LLMs for software test tasks.

    - The tester learns Large Language Model Operations (LLMOps) concepts for deploying and managing LLMs in software testing.
- Chapter 5: 80 minutes Deploying and Integrating Generative AI in Test Organizations
    - The tester learns a structured roadmap for integrating GenAI into test processes.
    - The tester learns organizational transformation for GenAI integration into test processes.

1 Introduction to Generative AI for Software Testing – 100 minutes

Keywords

None

Generative AI Specific Keywords

AI chatbot, context window, deep learning, embedding, feature, foundation LLM, generative AI, generative pre-trained transformer, instruction-tuned LLM, large language model, machine learning, multimodal model, reasoning LLM, symbolic AI, tokenization, transformer

Learning Objectives and Hands-on Objectives for Chapter 1:

1.1 Generative AI Foundations and Key Concepts

GenAI-1.1.1	(K1)	Recall different types of AI: symbolic AI, classical machine learning, deep learning, and generative AI
GenAI-1.1.2	(K2)	Explain the basics of generative AI and large language models
HO-1.1.2	(H1)	Practice tokenization and token count evaluation when using an LLM for a software test task
GenAI-1.1.3	(K2)	Distinguish between foundation, instruction-tuned and reasoning LLMs
GenAI-1.1.4	(K2)	Summarize the basic principles of multimodal LLMs and vision-language models
HO-1.1.4	(H1)	Write and execute a prompt for a multimodal LLM using both textual and image inputs for a software test task

1.2 Leveraging Generative AI in Software Testing: Core Principles

GenAI-1.2.1	(K2)	Give examples of key LLM capabilities for test tasks
GenAI-1.2.2	(K2)	Compare interaction models when using GenAI for software testing

1.1 Generative AI Foundations and Key Concepts

Generative Artificial Intelligence (GenAI) is a branch of artificial intelligence that uses large, pre-trained models to generate human-like output, such as text, images, or code. Large language models (LLMs) are GenAI models that are pre-trained on large textual datasets, enabling them to determine context and produce relevant responses according to user prompts.

Key concepts include tokenization (i.e. breaking text into units for efficient processing), context windows (limiting the amount of information considered at once to maintain relevance), and multimodal models (capable of processing multiple data types such as text, images, and audio for rich interactions).

In software testing, these LLMs can support tasks such as reviewing and improving acceptance criteria, generating test cases or test scripts, identifying potential defects, analyzing defect patterns, generating synthetic test data, or supporting documentation generation, across the entire test process.

1.1.1 AI Spectrum: Symbolic AI, Classical Machine Learning, Deep Learning, and Generative AI

Artificial Intelligence (AI) is a broad field that encompasses different types of technologies, each with its own unique way of solving problems, such as symbolic AI, classical machine learning, deep learning, and GenAI (among other technologies that are outside the scope of this syllabus):

- Symbolic AI uses a rule-based system to mimic human decision-making. Essentially, symbolic AI represents knowledge using symbols and logical rules.
- Classical machine learning is a data-driven approach that requires data preparation, feature selection and model training, and can be used for tasks such as defect categorization and predicting software problems.
- Deep learning uses machine learning structures called neural networks to automatically learn features from data. Deep learning models can find patterns in very large and complex datasets, such as images, video, audio, or text, without the need for users to manually define features, though in practice, it may still require human involvement in tasks such as data annotation, model tuning, or result validation.
- Generative AI uses deep learning techniques to create new content (text, images, code) by learning and mimicking patterns from its training data. Models such as LLMs can generate text, write code, and simulate reasoning or problem-solving within the scope of their training.

In summary, the field of AI has evolved in several directions, each with different strengths and limitations. The key advantage of using GenAI for software testing is that it uses pre-trained models that can be applied directly to test tasks without the need for an additional training phase, although this does come with some risks (see Section 3.1).

1.1.2 Basics of Generative AI and LLMs

Based on the generative pre-trained transformer deep learning model, LLMs are trained on very large datasets, including books, articles, and websites. Small language models (SLMs) are compact models with fewer parameters compared to large language models, designed to provide lightweight and focused GenAI solutions.

LLMs can handle language nuances as well as generate coherent content. Two key concepts that help LLMs process and generate content are tokenization and embeddings. Tokenization and embeddings convert language into a numerical form that the model can process effectively.

- Tokenization in language models is the process of breaking down text into smaller units called tokens. Tokens can be as small as a character or as large as a sub-word or word. When an LLM processes a sentence, it first tokenizes the input so that each token can be understood individually, while maintaining the overall context.
- Embeddings are numerical representations of tokens that encode their semantic, syntactic, and contextual relationships in a format suitable for processing by generative AI models. Each token is transformed into a vector in a high-dimensional space, capturing nuanced information about its meaning and usage. Tokens with similar meanings or contextual roles have embeddings that are positioned closely together in this space. This proximity enables LLMs to understand word relationships, retain context, and generate coherent and contextually appropriate responses.

LLMs utilize a neural network architecture known as the transformer model. Transformer models excel in language tasks by processing the context of extensive text sequences and learning how tokens relate to each other. During inference, LLMs predict the next token in a sequence, leveraging these learned relationships to generate coherent and contextually appropriate text. The transformer model can be used to generate new text that is statistically plausible, based on training data and the prompt. But plausible is not necessarily correct.

LLMs exhibit non-deterministic behavior primarily due to the probabilistic nature of their inference mechanisms and hyper-parameter settings. This inherent randomness can lead to variations in outputs even when the same input is provided multiple times.

In the realm of LLMs, the context window refers to the amount of preceding text, measured in tokens, that the model can consider when generating responses. A larger context window allows the model to maintain coherence over longer passages, for example when analyzing large test logs. However, increasing the number of tokens in the context window also increases the computational complexity and processing time required for the model to perform effectively.

Hands-On Objective 1.1.2 (H1): Practice Tokenization and Token Count Evaluation

This hands-on activity is designed to help trainees develop a practical understanding of tokenization and its implications when working with LLMs. The exercise is divided into two key parts:

- Tokenization: Use a tokenizer to break down a sample text into individual tokens. Examine the output to see how words, punctuation, and phrases are represented, and identify patterns or nuances in tokenization.
- Token Count Evaluation: Measure the number of tokens generated from various input texts. Analyze how token count influences model performance, particularly in relation to the model's context window limits and efficiency considerations.

By the end of this exercise, trainees will be able to better anticipate how different text structures and input lengths can affect interactions with LLMs.

1.1.3 Foundation, Instruction-Tuned and Reasoning LLMs

Large Language Models are developed through progressively specialized training stages to enhance their effectiveness across a wide range of tasks. These stages give rise to three main categories: foundation LLMs, instruction-tuned LLMs, and reasoning LLMs.

- Foundation LLMs: These are general-purpose models trained on vast and diverse datasets comprising text, code, images, and other modalities. Their extensive pretraining enables them to support various tasks across domains such as natural language processing, computer vision, and speech recognition. While powerful and flexible, foundation models typically require further adaptation to meet specific task requirements.
- Instruction-tuned LLMs: Derived from foundation models, instruction-tuned LLMs are fine-tuned using datasets that pair prompts with expected responses. This stage enhances their alignment with human instructions, improving usability in real-world applications. The tuning process involves optimizing for task adherence, instruction following, and response coherence, thereby improving the model's ability to interpret and act on user intent effectively.
- Reasoning LLMs: Reasoning models extend instruction-tuned models by emphasizing structured cognitive abilities such as logical inference, multi-step problem-solving, and chain-of-thought reasoning. These models are further trained or fine-tuned on carefully selected tasks that demand contextual understanding, intermediate reasoning steps, and synthesis of complex information. As a result, they are better suited for high-cognitive-load tasks, including those in technical domains.

In the context of GenAI applications for software testing, both instruction-tuned (sometimes referred to as non-reasoning) and reasoning LLMs are utilized. The selection depends on the complexity and reasoning demands of the specific testing task at hand.

1.1.4 Multimodal LLMs and Vision-Language Models

Multimodal LLMs extend the traditional transformer model to process multiple data modalities, including text, images, sound, and video. These models are trained on large and diverse datasets that enable them to learn relationships between different types of data. To handle various modalities, tokenization is adapted for each data type—for example, images are converted into embeddings using vision-language models before being processed in the transformer model.

Vision-language models, a subset of multimodal LLMs, specifically integrate visual and textual information to perform tasks such as image captioning, visual question answering, and analyzing the consistency between textual and visual input.

In software testing, multimodal LLMs, especially LLMs augmented with vision-language models offer significant opportunities. They can analyze visual elements of applications, such as screenshots and GUI wireframes, along with associated textual descriptions, such as defect reports or user stories. This capability allows testers to identify discrepancies between expected results and actual visual elements on a

screenshot. In addition, LLMs augmented with vision-language models can generate rich, realistic test cases that incorporate both textual data and visual cues, thereby increasing overall coverage.

Hands-On Objective HO-1.1.4 (H1): Review and execute a given prompt addressing a test task using a multimodal LLM model

This exercise involves reviewing and executing a given prompt for a multimodal LLM using both text and image input to solve a test task in two steps:

- Review the inputs: Review the prompt and the input data (text and image).
- Execute the prompt and verify the result: Use a multimodal LLM to input both image and text and check the LLM's response.

This exercise demonstrates how to use multimodal LLMs for a task involving both text and image input in software testing use cases, including recognizing the benefits and potential challenges involved.

1.2 Leveraging Generative AI in Software Testing: Core Principles

GenAI provides transformative capabilities in various test activities. LLMs excel at processing natural language and code, generating coherent text and code, answering questions, summarizing information, translating languages, and analyzing images in a multimodal context.

Test professionals in all roles can leverage GenAI in two complementary ways: through GenAI chatbots that provide instant responses to queries, and through LLM-powered applications integrated into test tools.

1.2.1 Key LLM Capabilities for Test Tasks

LLMs can interpret requirements, specifications, screenshots, code, test cases, and defect reports, making them tools for understanding and clarifying the information needed throughout the test process and generating elements of the testware. Below are some of the key LLM capabilities relevant to software testing:

- Requirements analysis and improvement: LLMs can help analyze requirements, and other elements of the test basis, by identifying ambiguities, inconsistencies, or missing information. They can generate meaningful questions to help clarify requirements during discussions with stakeholders.
- Test case creation support: LLMs can help generate test cases and suggest test objectives based on system requirements, user stories or any other elements of the test basis.
- Test oracle generation: LLMs can help generate expected results.
- Test data generation: LLMs can generate datasets, set boundary values, and create different combinations of test data.

- Test automation support: LLMs can help generate test scripts from test case description and improve existing test scripts by suggesting changes and identifying appropriate test design techniques.
- Test result analysis: LLMs can help analyze test results by creating summaries and classifying anomalies based on severity and priority.
- Testware creation: LLMs can help create various documents, including test plans, test reports and defect reports, and keep them updated as the project evolves.

These capabilities demonstrate how LLMs can impact various aspects of software testing through the whole test process.

1.2.2 AI Chatbots and LLM-Powered Testing Applications for Software Testing

AI chatbots and LLM-powered testing applications can both assist testers, though they differ in functionality, flexibility, and integration approaches.

AI Chatbots provide a user-friendly, conversational interface that enables testers to communicate directly with LLMs. This natural language interaction allows testers to input questions, commands, or prompts and receive immediate, contextually aware responses. Through techniques such as prompt chaining, testers can iteratively refine outputs, making chatbots particularly effective for routine tasks, exploratory testing, and even onboarding new testers by providing quick access to testing knowledge and practices.

These AI chatbots are especially beneficial in scenarios requiring fast feedback, clarification of test concepts, or dynamic exploration of requirements and potential test cases. Their intuitive interface makes them accessible even to non-technical stakeholders, broadening the potential user base and encouraging wider adoption.

LLM-Powered Testing Applications, in contrast, involve the integration of LLM capabilities via APIs to perform well-defined and often automated testing tasks. These applications offer greater customization and scalability, allowing organizations and tool vendors to embed generative AI into existing test frameworks. This enables the automation of repetitive or complex tasks, such as test case generation, defect analysis, or test data synthesis. In more advanced implementations, organizations can create AI agents specifically designed to perform certain testing roles (see Chapter 4).

Regardless of how the tester interacts with LLMs,—whether through chatbots or integrated LLM-powered applications—successful implementation of generative AI in testing requires strong prompt engineering (see Chapter 2). Carefully designed prompts and clear, specific instructions are essential to ensure that LLM-generated outputs are accurate, relevant, and aligned with testing objectives. This practice helps maximize the value derived from generative AI and ensures consistent, reliable support for a wide range of testing activities.

2 Prompt Engineering for Effective Software Testing – 365 minutes

Keywords

acceptance criteria, test script, test case, test condition, test data, test design, test report

Generative AI Specific Keywords

few-shot prompting, meta prompting, natural language processing, one-shot prompting, prompt, prompt chaining, prompt engineering, system prompt, user prompt, zero-shot prompting

Learning Objectives and Hands-on Objectives for Chapter 2:

2.1 Effective Prompt Development

GenAI-2.1.1	(K2)	Give examples of the structure of prompts used in generative AI for software testing
HO-2.1.1	(H0)	Observe several given prompts for software test tasks, identifying the components of role, context, instruction, input data, constraints and output format in each
GenAI-2.1.2	(K2)	Differentiate core prompting techniques for software testing
HO-2.1.2a	(H0)	Observe demonstrations of prompt chaining, few-shot prompting, and meta prompting applied to software test tasks
HO-2.1.2b	(H1)	Identify which prompt engineering techniques are being used in given examples
GenAI-2.1.3	(K2)	Distinguish between system prompts and user prompts

2.2 Applying Prompt Engineering Techniques to Software Test tasks

GenAI-2.2.1	(K3)	Apply generative AI to test analysis tasks
HO-2.2.1a	(H2)	Practice multimodal prompting to generate acceptance criteria for a user story based on a GUI wireframe
HO-2.2.1b	(H2)	Practice prompt chaining and human verification to progressively analyze a given user story and refine acceptance criteria
GenAI-2.2.2	(K3)	Apply generative AI to test design and test implementation tasks
HO-2.2.2a	(H2)	Practice functional test case generation from user stories with AI using prompt chaining, structured prompts and meta-prompting
HO-2.2.2b	(H2)	Use few-shot prompting technique to generate Gherkin style test conditions and test cases from user stories
HO-2.2.2c	(H2)	Use prompt chaining to prioritize test cases within a given test suite, taking into account their specific priorities and dependencies
GenAI-2.2.3	(K3)	Apply generative AI to automated regression testing

HO-2.2.3a	(H2)	Practice few-shot prompting to create and manage keyword-driven test scripts
HO-2.2.3b	(H2)	Practice structured prompt engineering for test report analysis
GenAI-2.2.4	(K3)	Apply generative AI to test control and monitoring tasks
HO-2.2.4	(H0)	Observe test monitoring metrics prepared by AI from test data
GenAI-2.2.5	(K3)	Select and apply appropriate prompting techniques for a given context and test task
HO-2.2.5	(H1)	Select and apply context-appropriate prompting techniques for a given test task

2.3 Evaluate Generative AI Results and Refine Prompts for Software Test Tasks

GenAI-2.3.1	(K2)	Understand the metrics for evaluating the results of Generative AI on test tasks
HO-2.3.1	(H0)	Observe how metrics can be used for evaluating the result of generative AI on a test task
GenAI-2.3.2	(K2)	Give examples of techniques for evaluating and iteratively refining prompts
HO-2.3.2	(H1)	Evaluate and optimize a prompt for a given test task

2.1 Effective Prompt Development

Effective prompt design ensures that GenAI tools perform software test tasks accurately and efficiently and that testers obtain useful results from the chatbot. A structured prompt includes different components (see section 2.1.1). Each of these components contributes to the clarity and precision of a prompt that effectively communicates requirements and expectations to LLMs.

Various prompt engineering techniques enhance the effectiveness of prompts in software testing. Techniques such as prompt chaining, few-shot prompting, and meta prompting help address complex testing challenges (see section 2.1.2).

The combination of structured prompts (see section 2.1.1) with core prompting techniques is aimed at achieving good results when querying an LLM for software testing tasks (see section 2.1.3).

2.1.1 Structure of Prompts for Generative AI in Software Testing

A structured prompt for software testing typically includes six components:

- Role: The role defines the perspective or persona that the GenAI model should take when generating a response. Specifying the role helps the LLM determine its responsibilities and adopt an appropriate tone or approach, such as acting as a tester, test manager, or test automation engineer.
- Context: Context provides the background information that the GenAI model needs to determine the test conditions. This includes details about the test object, the specific functionality to be tested, and any relevant contextual information.
- Instruction: Instructions are directives given to the GenAI that outline the specific task to be performed. Clear, imperative and concise instructions include a task description and any relevant requirements for the task.
- Input data: Input data includes any information needed to perform the task, such as user stories, acceptance criteria, screenshots, code, existing test cases or output examples. Providing detailed and structured input data helps the LLM to generate more accurate and context-aware results.
- Constraints: Constraints outline any restrictions or special considerations that the LLM should adhere to. Constraints help to specify how instructions should be applied to input data.
- Output format: Output specifications denote the expected format, structure or characteristics of the response. These indicators help shape the output of the LLM.

These components form the basic structure of the prompt. This structure should be combined with the implementation of prompting techniques (see Section 2.1.2), depending on the task to be performed and the LLM to be used.

Hands-On Objective HO-2.1.1 (H0): Observe and analyze prompt components

In a demonstration, several structured prompts are experimented with on an AI chatbot, each tailored to specific software testing tasks. These prompts follow a structured format consisting of six key components: role, context, instruction, input data, constraints, and output format. The demonstration

aims to facilitate observation and analysis of these structured prompts, highlighting how each component contributes to providing accurate, relevant, and actionable insights to an LLM used for a software testing task.

2.1.2 Core Prompting Techniques for Software Testing

In recent years, many LLM prompting techniques have been proposed for different GenAI use cases (Schulhoff 2024). Among these, three core prompting techniques are commonly used for test tasks with GenAI in conjunction with the 6-component prompt structure described above (see section 2.1.1): prompt chaining, few-shot prompting, and meta prompting.

- Prompt chaining involves breaking a task into a series of intermediate steps (multiple prompts). The result of each step is manually or automatically checked and refined before proceeding to the next step. This approach leads to greater accuracy as each response informs the next prompt. Prompt chaining is particularly useful in test processes where tasks are complicated and require decomposition into subtasks and systematic checking of intermediate LLM outputs. It also allows for dynamic interactions in test processes.
- Few-shot prompting involves providing the LLM with examples in the prompt. While zero-shot prompting (no example) relies on the model's pre-existing knowledge to generate a response, one-shot prompting provides one example to demonstrate the desired outcome for a given input. Few-shot prompts contain more than one example (a few) to further consolidate the desired response behavior of the model.

  This technique helps guide the model by providing a clear reference and ensuring that results are consistent and in line with expectations. Few-shot prompting is particularly effective for tasks where examples can illustrate the required behavior, allowing the model to generalize effectively and produce reliable results.
- Meta prompting leverages the AI's ability to generate or refine its own prompts. In an iterative cycle, the LLM can generate prompts that can be evaluated and refined by the tester. This approach optimizes prompt quality by by taking advantage of the LLMs knowledge about optimized prompts. Meta prompting is especially beneficial when efficiency and prompt optimization are critical, as it reduces the manual effort required to design effective prompts. Another advantage of meta prompting is that if the tester is unsure how to craft an effective prompt, they can collaborate with the LLM to co-create it. This reflects a form of pairing with the GenAI tool where the tester and the AI work together interactively to achieve a shared goal. This concept of pairing highlights a new way of collaborating with AI tools, enhancing both productivity and learning not only in prompt engineering, but also in pair programming and pair testing.

These prompting techniques can be used effectively in combination to improve LLM outcomes (see section 2.2.5).

Hands-On Objective HO-2.1.2a (H0): Observing and discussing prompt chaining, few-shot prompting, and meta prompting in software test tasks

Participants will experience with prompt chaining, few-shot prompting, and meta prompting on an AI chatbot, each applied to specific software test tasks. The demonstration aims to explore and discuss these prompting techniques in the context of software testing, emphasizing how each technique contributes to the accuracy and completeness of LLM outputs.

Hands-On Objective HO-2.1.2b (H1): Identifying prompt engineering techniques in given examples

Participants will read a set of prompt examples related to software testing to identify the core prompting techniques applied. The focus is on recognizing techniques such as prompt chaining, few-shot prompting, and meta prompting, while highlighting their distinct features and practical applications.

This activity aims to deepen participants' understanding of how different prompting techniques enhance the effective use of GenAI in software testing.

2.1.3 System Prompt and User Prompt

System prompts and user prompts serve different purposes in interactions with LLMs, each playing a distinct role in shaping the conversation. The system prompt is typically defined by the developer or tester, to guide the overall behavior of the LLM, and is not visible or editable by the chatbot's user in most interfaces.

A system prompt acts as a predefined command set that defines the LLM's behavior, personality, and operational parameters. Operational parameters determine how the LLM responds — for example, using a formal tone, keeping answers concise, respecting domain-specific rules or avoiding certain behavior. The system prompt sets the rules for the entire conversation. It may contain parts of a structured prompt such as the role, context and constraints.

The system prompt stays constant throughout the interaction session and establishes the fundamental framework for how the LLM should respond. For example, a system prompt might say: "You are a professional software testing assistant. Always respond clearly, use formal language, and focus on ISTQB-aligned practices. Avoid speculation and cite testing principles when relevant."

The user prompt, on the other hand, represents the actual input or question from the chatbot's user. It changes with each interaction and can include specific instructions, questions, or tasks that the chatbot's user wants the LLM to address. Unlike the system prompt, user prompts are directly visible and form the immediate context for each response.
For example, a user prompt might be: "List the key differences between black-box and white-box testing with examples."

Typical usage involves setting the system prompt once at the start of the conversation, then sending successive user prompts for each interaction. The LLM generates responses by considering both the unchanging system prompt and the current user prompt together. For effective implementation, system prompts should be clear and specific about the LLM's role and possible constraints. It may also contain context and general instructions, e.g. regarding the expected output.

User prompts must be focused and well-structured, including explicit instructions as well as additional relevant context and output format instructions.

2.2 Applying Prompt Engineering Techniques to Software Test Tasks

Applying prompt engineering techniques to software testing enables GenAI to support test tasks such as test analysis, test design, test automation, test case prioritization, defect detection, coverage analysis, and test monitoring and test control. By using and combining techniques such as prompt chaining, few-shot prompting, and meta prompting, teams can tailor AI prompts to the specific test objectives, making outputs more precise, relevant, and effective. High-quality input is crucial for meaningful AI results.

2.2.1 Test Analysis with Generative AI

GenAI can support test analysis tasks by generating and prioritizing test conditions, identifying defects in the test basis and providing coverage analysis. The input data includes requirements, user stories, technical specifications, GUI wireframes and other relevant information. The output consists of typical test analysis work products, such as prioritized test conditions (e.g., acceptance criteria).

Here are some typical test analysis tasks that can be supported by GenAI:

- **Identify potential defects in the test basis**: GenAI can help analyze the test basis for inconsistencies, ambiguities, or incomplete information that could lead to defects. By comparing similar requirement patterns or applying knowledge from previous defect reports, the LLM can flag potential anomalies and suggest improvements.
- **Generating test conditions based on the test basis,** for example on requirements/user stories: LLMs can analyze requirements and user stories to generate test conditions. Using natural language processing, they can interpret the meaning of requirements and break them down into measurable, testable statements. This can help translate requirements into specific test conditions.
- **Prioritize test conditions based on risk level:** With information on the risk likelihood and risk impact of failure for each test condition, an LLM can help prioritize test effort. By considering aspects such as regulatory compliance, user-facing features (e.g., login functionality or payment processing), and historical defect data, the LLM can recommend priority levels.
- **Support coverage analysis:** By mapping requirements and user stories to test conditions, an LLM can perform coverage analysis to determine whether all aspects of the test basis are covered. This is particularly useful for projects with complex requirements, where gaps in coverage can lead to escaped defects.
- **Suggest test techniques:** GenAI can suggest relevant test techniques (e.g., boundary value analysis, equivalence partitioning) based on the type of requirement or user story being tested. This can help testers apply the most effective test techniques for specific test conditions.

The quality and relevance of inputs provided to the LLM in relation to the task to be completed directly impact the accuracy and precision of the output generated by the LLM.

Hands-On Objective 2.2.1a (H2): Practice creating structured multimodal prompts to generate acceptance criteria for a user story based on a GUI wireframe

This is an exercise to practice writing structured prompts using multimodal input (text and image). The goal is to generate high quality (i.e. well-formed, clear and complete) acceptance criteria from a user story and a GUI wireframe. Other text elements can be added to provide context, such as constraints on input fields or business rules to be applied to data processing.

The results obtained from the LLM are compared to assess the impact of different formulations of the structured prompt (role, context, instruction, textual and image input data, constraints, and output format) for a test analysis task.

This exercise provides practical experience in the importance of prompt structuring, the contribution of precise instructions, and the importance of both textual and image contextual data in obtaining accurate and relevant results from the LLM.

Hands-On Objective 2.2.1b (H2): Practice prompt chaining and human verification to progressively analyze a given user story and refine acceptance criteria

This is an exercise to practice prompt chaining to analyze a given user story and refine acceptance criteria, first by identifying ambiguities, then by evaluating testability, and finally by evaluating completeness. This exercise encourages a step-by-step approach, refining the analysis at each step to ensure that the acceptance criteria are well-formed and actionable to achieve the test objectives. At each step, the results provided by the LLM are manually verified and corrected, if necessary, either by adjusting the output or through a prompt chaining process with the LLM. In this way, the next stage uses a clean result from the previous stage to address another aspect of improving the acceptance criteria.

This exercise provides practical experience of the benefits of breaking down a complex task into subtasks, with human verification of the results of each stage.

2.2.2 Test Design and Test Implementation with Generative AI

As described in [ISTQB_CTFL_SYL], test design involves the elaboration and refinement of test conditions, which are then translated into test cases and other testware. Test implementation entails the creation or acquisition of the necessary testware to perform the tests.

Both manual tests and automated test scripts can be created, prioritized, and arranged within a test execution schedule with the support of GenAI. GenAI can significantly support this large group of test activities by assisting in the creation and evaluation of various testware, including test cases, test data, test scripts, and test environments.

Here are some typical test design and test implementation tasks that can be supported by GenAI:

- **Test case generation**: Natural language processing enables GenAI to create draft test cases based on functional and non-functional requirements. When prompted with suitable information, an LLM can suggest test preconditions and inputs, expected results, and coverage criteria, producing test cases that meet different test objectives, from basic functional verification to complex end-to-end testing.

- **Test data synthesis**: GenAI can create representative, data privacy-preserving synthetic test data that resembles production data, covering extreme situations and varied test conditions. This synthetic test data can be used for functional and non-functional testing. AI-generated test data can be tailored to application requirements, simulating realistic scenarios without exposing sensitive information.
- **Automated test script generation**: GenAI can generate manual test procedures and automated test scripts from structured test cases, interpreting test steps and translating them into code compatible with various test automation frameworks. These test scripts can be updated or extended based on new requirements.
- **Test execution scheduling and prioritization**: GenAI can analyze test cases and their interdependencies, optimizing test execution schedules based on priority, associated risks, resource availability and test objectives.

Hands-On Objective 2.2.2a (H2): Practice functional test case generation from user stories with AI using prompt chaining, structured prompts and meta prompting

This exercise focuses on developing functional test cases from user stories with GenAI, using prompt chaining, structured prompts, and meta prompting techniques to ensure thorough coverage. The first step is to create a prompt that instructs the AI to generate functional test cases based on given acceptance criteria following a specific output format. A second step is to verify the completeness of the generated test cases. Here, the prompt verifies that each acceptance criterion is covered by having the AI generate a table summarizing the coverage. Finally, a third step is to create a meta-prompt to aid in the creation of end-to-end test procedures. This meta-prompt helps refine the prompt to generate comprehensive end-to-end tests, encouraging iterative improvements to maximize effectiveness.

This exercise enhances the understanding of using LLMs for test case generation, coverage validation, and end-to-end testing.

Hands-On Objective 2.2.2b (H2): Use the few-shot prompting technique to generate Gherkin style test cases from given user stories

This exercise is about using few-shot prompting to generate Gherkin style test cases from given user stories. Starting with a review of predefined examples and Gherkin syntax, step 1 is to select n examples to include in the prompt, each with a user story, test conditions, and expected given-when-then style test cases to model the desired output. This prompt is then applied to a new user story, generating Gherkin scenarios that reflect the original test conditions. If the results are inaccurate, the prompt or examples should be refined.

This exercise helps to gain experience in applying few-shot prompting techniques to realistic test design and test implementation tasks.

Hands-On Objective 2.2.2c (H2): Use prompt chaining to prioritize test cases within a given test suite, taking into account their specific priorities and dependencies

This exercise focuses on using GenAI to improve test case prioritization within a given test suite with associated risk analysis and dependencies between test cases. The session begins with a brief overview of different test approaches, such as risk-based, coverage-based, and requirements-based, and a review of the given test suite. Participants will then engage in creating prompts to generate actionable prioritization plans for various test prioritization strategies. The results of the LLM based on the prompt and the given input data should be manually verified to detect any errors in the LLM's reasoning.

The goal of this exercise is to experiment with GenAI on test tasks that require multi-criteria reasoning capabilities (here, the different risks and dependencies to be considered for test case prioritization).

2.2.3 Automated Regression Testing with Generative AI

As each new iteration or release is completed, the number of regression test cases to be run often increases, making them ideal candidates for automation, particularly in Continuous Integration / Continuous Delivery (CI/CD) pipelines due to the high frequency of test execution. GenAI can streamline this process by assisting in the creation, maintenance, and optimization of automated regression test suites. By dynamically adapting to codebase changes and performing impact analysis, GenAI can identify which areas of the software are most likely to be affected by recent modifications, focusing regression test efforts where they are most needed.

Here are some typical automated regression testing and test reporting activities that can be supported by GenAI prompting:

- **Automated test script implementation with keyword-driven automation**: LLMs can be used to implement test scripts based on keyword-driven test automation frameworks, where pre-defined keywords represent common test steps. GenAI can map these keywords to specific test cases, generate test scripts and assist testers and test automation engineers in their work.
- **Impact analysis and test optimization**: GenAI can be used to analyze code changes in order to identify high-risk areas, thereby enabling targeted regression testing where it is most needed.
- **Self-healing and adaptive tests**: GenAI can be used to automatically adjust test scripts to handle minor UI or API changes, preventing unnecessary failures from small modifications and ensuring that test suites remain stable over time.
- **Automated test reporting and insights**: GenAI enables the generation of detailed, timely available test reports with success metrics, failures, and key insights, providing stakeholders with dashboards that highlight testing trends and offer predictive insights on potential failure points.
- **Enhanced defect reporting and root cause analysis**: GenAI can support the automatic compilation of comprehensive defect reports with test logs, screenshots, and test environment data.

These activities can be applied to a variety of regression tests, including functional and non-functional regression tests. However, the testers must be aware that GenAI can make mistakes. The generated output must therefore be carefully checked, depending on the associated risk (see chapter 3).

Furthermore, GenAI can assist end-to-end GUI and API-based automated regression tests, each with its distinctive challenges and solutions. GUI tests frequently become unstable due to recurrent changes to the user interface. GenAI can automatically adapt test scripts to handle changes like dynamic locators

and modified interactions, reducing the need for manual intervention. API regression tests face challenges such as changing request/response formats, endpoints, and authentication. GenAI can adapt test scripts automatically to evolving API specifications and generate diverse test data, maintaining comprehensive coverage and reducing the need for manual updates.

Hands-On Objective 2.2.3a (H2): Practice few-shot prompting to create and manage keyword-driven test scripts

This exercise focuses on developing and automating test scripts for a given web application using a GUI test automation framework. The exercise is structured into two main sections: test automation and test script debugging. The first part of the exercise provides guidance on creating documentation for a keyword library, generating initial test scripts, having AI validate these test scripts, and expanding the coverage with additional test scripts. The second part places an emphasis on debugging support, using system prompts to create an AI assistant that can check and correct test scripts.

This exercise combines traditional test automation with AI-assisted validation, demonstrating how few-shot prompting can be effectively used to create, maintain, and debug keyword-driven test scripts.

Hands-On Objective 2.2.3b (H2): Practice writing structured prompts for test report analysis in the context of regression testing

This exercise illustrates a methodical approach to analyzing regression test reports, utilizing structured prompts. The process begins with an analysis of the provided test results and a comparison with the test specification. It then progresses to the clustering of similar defects, the maintenance of a known anomalies list, and a cross-checking of findings. Each step is linked to the next one in a single LLM conversation.

The step-by-step approach demonstrates how structured prompts can be used to transform regression test results and test logs into actionable insights, thereby supporting effective test report analysis in the context of regression testing.

2.2.4 Test Monitoring and Test Control with Generative AI

Test monitoring tasks require the retrieval of large quantities of (sometimes unstructured) data, which are often already available in test management tools that GenAI can help analyze and synthesize.

GenAI facilitates a number of test monitoring and test control tasks, including:

- **Test monitoring and metrics analysis:** GenAI can facilitate the automation of test monitoring, as well as the analysis of trends to predict potential risks and alert teams of any deviations from the plan. This enables teams to remain informed and take action to maintain quality standards.
- **Test control:** GenAI can assist with test control by providing insights for reprioritizing tests, adjusting test schedules, and reallocating resources as needed. This ensures that testing remains flexible and focused on high-priority areas.

- **Test completion insights and continuous learning:** GenAI can assist by generating test completion reports, highlighting successes and lessons learned. This allows teams to refine test strategies and improve future test processes.
- **Enhanced test metrics visualization and reporting:** GenAI can assist in the creation of dynamic dashboards and natural language summaries, ensuring that all stakeholders have access to the relevant metrics. This assistance provides the information needed to make quick decisions and gives a clear view of test progress.

Hands-On Objective 2.2.4 (H0): Observe Test Monitoring Metrics Prepared by AI from Test Data

This demonstration illustrates how GenAI can assist test teams by transforming test data into actionable test monitoring metrics, thereby facilitating informed decision-making. Starting from test data extracted from test tools, an LLM processes it to generate key metrics like test progress, defect trends, or coverage, highlighting potential risks. These AI-generated metrics may then be displayed on a dashboard and summarized in natural language for easy understanding by all stakeholders.

This demonstration illustrates how GenAI turns test data into practical insights, helping test teams monitor test progress, manage quality, and adapt quickly to changes.

2.2.5 Choosing Prompting Techniques for Software Testing

The following table shows the suitability of the three prompting techniques mentioned in section 2.1.2 according to the characteristics of the test task.

Prompting Technique	Recommended Use Case	Key Features & Applications
Prompt chaining	Complex tasks requiring precision with human verification at each step	Breaks tasks into smaller steps, useful for test analysis, test design and test automation, where each test step is checked for accuracy.
Few-shot prompting	Repetitive or specific/constrained output format tasks	Provides examples to GenAI for repetitive generation with a specific pattern, for example in Gherkin style test case (e.g scenario-based), keyword-driven testing or test reporting with a specific output format.
Meta prompting	Flexible, dynamic tasks, useful for crafting prompts for new tasks	General description of the objective and the task to be performed, which guides the LLM in the creation of the prompt. Useful for all kinds of complex tasks such as test report analysis and anomaly detection.

It is even possible to use multiple techniques for a single use case. For example, meta prompting can be used to create an initial prompt. This generated prompt may contain examples that must be adapted and can be enhanced (few-shot prompting). Finally, it can be useful to divide the task into smaller subtasks to enable validation of the intermediate steps (prompt chaining).

Hands-On Objective 2.2.5 (H1): Selecting Context-Appropriate Prompting Techniques for Given Test Tasks

This exercise focuses on selecting appropriate prompting techniques for different test tasks. Participants are given several test tasks with different challenges. For each test task, participants should evaluate the nature of the task - whether it requires precision or repetitive structure - and suggest the prompting technique(s) that best fits the context and meets the specific needs of the task. The choices are discussed in the group.

This exercise is designed to deepen understanding of how different prompting techniques can be used effectively in practical test efforts.

2.3 Evaluate Generative AI Results and Refine Prompts for Software Test tasks

Evaluating the performance of GenAI in software testing requires a clear set of metrics to assess the quality, relevance, and effectiveness of the generated outputs (Li 2024). These metrics, whether general or task-specific, help optimize LLM prompting.

2.3.1 Metrics for Evaluating the Results of Generative AI on Test tasks

Several metrics can be used to evaluate the quality and efficiency of GenAI results on a test task:

Metric	Description	Example
Accuracy	Measures the overall correctness of the generated output against expert-written test cases, requirements, or other standards.	The degree to which the generated test cases cover all specified requirements.
Precision	Evaluates the correctness of the generated output with respect to a specific objective.	The degree to which the generated test cases correctly identify anomalies.
Recall	Measures the ability of a model to identify all relevant instances within a dataset.	The degree to which generated test cases cover valid and invalid equivalence partition of a data class.
Relevance and Contextual Fit	Determines whether the generated output is applicable and appropriate for a given context.	The degree to which the generated test cases are consistent with the test basis and integrate the domain-specific requirements.
Diversity	Ensures a wide range of inputs and scenarios are covered, avoiding repetition.	The degree to which the generated test cases cover various user behaviors and to which they explore edge cases.

Execution Success Rate	Measures the proportion of generated test cases or test scripts that can be executed successfully.	Determining how many of the generated test scripts can be executed without syntax errors or output format issues in an otherwise working test environment.
Time Efficiency	Evaluates the time saved compared to manual test efforts.	Time required by the AI to generate test cases versus the time a human would take to manually create equivalent tests.

In addition to these general metrics, task-specific metrics can be tailored to evaluate how well the GenAI supports specific test activities.

To evaluate these metrics effectively, testers may perform manual reviews or automate them e.g. by comparing the LLM output against a predefined reference. Given the non-deterministic nature of GenAI, the metrics must be based on statistically relevant data.

Hands-On Objective 2.3.1 (H0): Observe how metrics can be used for evaluating the result of generative AI on a test task

During a demonstration on a given test task, task-adapted metrics for evaluating GenAI results are shown, as well as their concrete application to the results obtained with an LLM on that test task.

This demonstration illustrates the importance of evaluation metrics in providing confidence in the results of generative AI for software testing.

2.3.2 Techniques for Evaluating and Iteratively Refining Prompts

Building on the metrics presented above, specific techniques for prompt evaluation and refinement are used to improve AI results:

- **Iterative prompt modification:** Start with a base prompt and iteratively modify it based on observed results, gradually adding more context or adjusting wording (e.g. regarding terminology) to improve specificity and relevance.
- **A/B testing of prompts:** Create multiple versions of prompts and evaluate which version produces better results based on predefined metrics. This approach helps determine which prompt phrasing or prompt structure produces the most accurate and relevant results.
- **Output analysis:** Examine AI-generated output for inaccuracies or inconsistencies, e.g. with respect to test basis. Understanding the types of errors and inconsistencies can help refine prompts to avoid similar defects in future iterations.
- **Integrate user feedback:** Gather input from testers about the usefulness and clarity of generated output, e.g. regarding the level of detail of generated tests. Analyze their insights and use them to refine prompts to better meet real-world testing needs.

- **Adjust prompt length and specificity:** Experiment with different prompt lengths and levels of detail. Sometimes adding more context can improve the quality of the response. In other cases shorter prompts may yield better generalization.

By using these techniques, test teams can organize prompt evaluation and optimization sessions to ensure continuous improvement of GenAI prompts. Sharing practices across the test team or test organization not only helps standardize prompt techniques and maintain consistent quality but also promotes a culture of learning and iterative improvement. This collaborative approach contributes to the evolution of GenAI test methodologies by enabling test teams to build on collective insights, avoid repeated errors, and refine their use of GenAI tools more effectively over time, e.g. by sharing prompt libraries.

Hands-On Objective 2.3.2 (H1): Evaluate and optimize a prompt for a given test task

This exercise focuses on applying prompt optimization techniques to a given test task. Participants will start with an initial prompt and iteratively refine it to improve the AI-generated results. They will use techniques such as A/B testing and human verification to evaluate and improve the quality of the prompts. The goal is for participants to experience how iterative refinement leads to more effective and contextually relevant test case generation.

By the end of the exercise, participants will have performed several iterations of prompt refinement and evaluated each iteration using the metrics discussed to improve AI output quality.

3 Managing Risks of Generative AI in Software Testing – 160 minutes

Keywords

security, vulnerability, data privacy

Generative AI Specific Keywords

hallucination, temperature, reasoning error, bias

Learning Objectives and Hands-on Objectives for Chapter 3:

3.1 Hallucinations, Reasoning Errors and Biases

GenAI-3.1.1	(K1)	Recall the definitions of hallucinations, reasoning errors and biases in Generative AI systems
GenAI-3.1.2	(K3)	Identify hallucinations, reasoning errors and biases in LLM output
HO-3.1.2a	(H1)	Experiment with hallucinations in testing with GenAI
HO-3.1.2b	(H1)	Experiment with reasoning errors in testing with GenAI
GenAI-3.1.3	(K2)	Summarize mitigation techniques for GenAI hallucinations, reasoning errors and biases in software test tasks
GenAI-3.1.4	(K1)	Recall mitigation techniques for non-deterministic behavior of LLMs

3.2 Data Privacy and Security Risks of Generative AI in Software Testing

GenAI-3.2.1	(K2)	Explain key data privacy and security risks associated with using generative AI in software testing
GenAI-3.2.2	(K2)	Give examples of data privacy and vulnerabilities in using Generative AI in software testing
GenAI-3.2.3	(K2)	Summarize mitigation strategies to protect data privacy and enhance security in Generative AI for software testing
HO-3.2.3	(H0)	Recognize data privacy and security risks in a given Generative AI for testing case study

3.3 Energy Consumption and Environmental Impact of Generative AI for Software Testing

GenAI-3.3.1	(K2)	Explain the impact of task characteristics and model usage on the energy consumption of Generative AI in software testing
HO-3.3.1	(H1)	Use a simulator to calculate the energy and CO_2 emissions for given test tasks with Generative AI

3.4 AI Regulations, Standards and Best Practice Frameworks

GenAI-3.4.1	(K1)	Recall examples of AI regulations, standards and best practice frameworks relevant to Generative AI in software testing

3.1 Hallucinations, Reasoning Errors and Biases

GenAI systems, especially LLMs, are prone to certain defects, including hallucinations, reasoning errors, and biases. These defects reduce the quality of GenAI output on test tasks, resulting in generated testware that fails to meet testers' expectations. These hallucinations, reasoning errors, and biases need to be identified by testers in the LLM output, and measures should be taken to mitigate these risks.

The non-deterministic behavior of LLMs (see section 1.1.2) makes it difficult to fix these types of defects; they may appear to be fixed for one LLM output but reappear in another conversation with the same LLM.

3.1.1 Hallucinations, Reasoning Errors and Biases in Generative AI

Hallucinations occur when an LLM generates output that appears factually incorrect or irrelevant to a given task. In software testing, hallucinations can manifest as LLMs creating fictitious or irrelevant test cases, generating incorrect or non-functioning test scripts, or suggesting test cases that verify non-existent acceptance criteria. This can mislead testers and compromise the validity of test outputs.

Reasoning errors occur when LLMs misinterpret logical structures, such as cause-and-effect relationships, conditional logic, or step-by-step problem-solving processes, leading to incorrect conclusions. Unlike humans, LLMs lack true logical reasoning and rely on pattern matching, which can lead to faulty logic when performing tasks such as mathematical reasoning (Mirzadeh 2024). Test planning and test case prioritization are examples of test tasks that require logical reasoning and where LLMs can make reasoning errors.

LLM biases (Gallegos 2024) come from the data on which the model was trained. These biases can lead to outputs that favor certain types of information, approaches, or assumptions. For example, LLMs trained primarily on English-language data may underrepresent non-English perspectives. In software testing, biases can influence LLM responses when, for instance, generating test data or refining acceptance criteria for test cases.

The hallucinations, reasoning errors and biases in GenAI output result from the nature of their training data and the inherent limitations of the transformer model (see Chapter 1). Recognizing and addressing these challenges increases the quality of generative AI results in test processes.

3.1.2 Identify Hallucinations, Reasoning Errors and Biases in LLM Output

Effective integration of GenAI systems into software testing requires the ability to detect hallucinations, reasoning errors and biases in LLM output. Depending on the type of problem, different approaches to detection can be applied. The following are common approaches that are applied through review or a combination of review and automated verification:

Hallucination detection:

- Cross-verification: Compare AI-generated output with existing documentation, requirements, and known system behavior. Automated tools can help cross-reference the output with established data sources to flag discrepancies.
- Domain expertise consultation: Engage subject matter experts to validate the accuracy of generated content. Their expertise is essential for capturing nuanced insights that automated systems might overlook.

- Consistency checks: Verify that generated outputs are consistent with each other and with known information. Automated systems can help identify patterns and flag inconsistencies.

Reasoning error detection:

- Logical validation: Evaluate the logical flow (e.g., the consistency, coherence, and structured reasoning within the generated text) of AI-generated content for coherence and correctness through review cycles. Automated tools can help, but complex cases may require human judgment.
- Output testing: For example, running the generated test cases or test scripts against the test objects to verify the test results. This can be partially or fully automated, depending on the type of testware being generated.

Bias detection:

- Reviewing how generated testware, such as synthetic test data, is fairly and accurately represented relative to the test strategy
- Assessing biases related to test types, such as underrepresented non-functional tests in the generated output of the LLM.

The actual implementation of these detection methods will depend on the estimated risk level of hallucinations, reasoning errors or biases in the test task being performed with GenAI.

Hands-On Objective 3.1.2a (H1): Experiment with Generative AI hallucinations related to a software test task

This exercise focuses on experimenting with examples of GenAI hallucinations in relation to the software testing body of knowledge. Participants will attempt to confront at least two LLMs with a situation in which the LLMs invent irrelevant elements, e.g., add unrelated criteria that do not exist in the given contextual data. Variations in prompting are tested to examine the influence of prompting on hallucinations.

This exercise increases understanding of identifying GenAI hallucinations in software testing.

Hands-On Objective 3.1.2b (H1): Experiment with Generative AI reasoning errors in a test planning task

This exercise focuses on presenting an example of a GenAI reasoning error. An example of a problem to be solved in the area of test planning, such as estimation of test effort and prioritization of test cases (see [ISTQB_CTFL] - Chapter 5). The exercise is designed with a certain complexity of input data, which requires problem-solving skills and highlights the limitations of LLMs for this purpose. The result of the LLM will be compared with the exact result that should be achieved. Three different LLM types will be tried (LLM, SLM, and reasoning model), and variations of the prompt will be used to try to improve the results.

This exercise increases understanding of how to identify GenAI reasoning errors in software test tasks that require logical problem-solving skills.

3.1.3 Mitigation techniques of GenAI hallucinations, reasoning errors and biases in software test tasks

To minimize undesirable outcomes of GenAI in software testing, several strategies can be employed to reduce hallucinations, reasoning errors and biases. These problems are more likely to occur when prompts are not properly designed (see Chapter 2) or when relevant contextual input data is lacking for a given test task. Key techniques for mitigating risks associated with AI hallucinations, reasoning errors and biases include:

- Provide complete context: Ensure that the prompt contains all relevant information (see section 2.1.1), offering a comprehensive context to guide the AI in producing accurate results.
- Divide prompts into manageable segments: Split complex prompts into smaller steps by using prompt chaining techniques (see section 2.1.2), systematically verifying each output before moving to the next. This step-by-step approach can help detect reasoning errors early in the generation process.
- Use clear, interpretable data formats: Avoid formats that may be ambiguous or challenging for the GenAI to interpret. Structured, straightforward formats help the model focus on the essential aspects of the task.
- Select the appropriate GenAI model for the task: Use an LLM specifically trained for the task at hand (see section 5.1.3).
- Compare results across models: When appropriate, evaluating the prompt with several LLMs and comparing outputs helps detect output errors and select the most reliable results.

Chapter 4 introduces two complementary techniques for improving LLM results: Retrieval-Augmented Generation and Fine-Tuning.

3.1.4 Mitigation of Non-Deterministic Behavior of LLMs

The inherent non-deterministic behavior of LLMs (Shuyin 2023) can lead to variations in output, even when the same input is provided. This arises from the probabilistic sampling processes used during inference. Consequently, achieving consistent and reproducible results when using LLMs can be challenging, particularly for long outputs, which increases the risk of variability.

While complete reproducibility cannot be guaranteed, certain strategies can help reduce variability:

- Adjusting LLM's temperature parameter settings: Lowering the temperature during response generation (inference) narrows the probability distribution, reducing randomness and resulting in more consistent outputs. However, this will also limit creativity and diversity in responses, making outputs more repetitive or overly deterministic.
- Setting random seeds: Some LLM implementations allow setting a seed value for the random number generator, ensuring the same pseudo-random (i.e., deterministic random values) sequence is used, which improves reproducibility.

Reducing the risk of hallucinations and reasoning errors in LLM output involves addressing this non-deterministic behavior, e.g., by automating some aspects of output verification to ensure a structured and consistent evaluation process.

3.2 Data Privacy and Security Risks of Generative AI in Software Testing

GenAI in testing introduces risks related to data privacy and security due to the handling of sensitive information and potential vulnerabilities in LLM-powered test infrastructure. Robust data protection is essential to prevent breaches, unauthorized access, and exposure of confidential data.

3.2.1 Data Privacy and Security Risks Associated with Using Generative AI

GenAI can process large amounts of data that may contain sensitive or personally identifiable information. This raises the following data privacy concerns:

- Unintentional data exposure: GenAI models may generate outputs that accidentally reveal sensitive information.
- Lack of control over data usage: GenAI tools may store and process sensitive data without explicit user consent or control. This can lead to potential misuse or unauthorized access.
- Compliance risks: Using GenAI tools without complying with data protection regulations, such as the General Data Protection Regulation (GDPR, Regulation (EU) 2016/679), could lead to legal disputes.

Additionally, specific security risks arise when testing with GenAI, such as:

- LLM-powered test infrastructure can be vulnerable to security attacks, such as data breaches or unauthorized access.
- Malicious actors can exploit vulnerabilities in LLMs, like manipulative attacks (see section 3.2.2), to alter their behavior or extract sensitive information.
- Attackers can intentionally introduce malicious input data to mislead LLMs and compromise their accuracy or security.

3.2.2 Data Privacy and Vulnerabilities in Generative AI for Test processes and Tools

The following table gives some examples of attack vectors in GenAI test processes and tools.

Attack vector	**Description**	**Example**
Data exfiltration	Sending requests designed to extract confidential training data.	Exceeding the LLM contextual window with long prompts to overload the AI's memory could lead it to reveal random snippets of its training data and potentially expose sensitive information.
Request manipulation	Introducing data that disrupts the AI's output.	Images that lure the AI into a different context, thus provoking hallucinations on e.g., acceptance criteria.

Data poisoning	Manipulating training data.	Providing fake evaluations when rating the results of an AI-generated test report.
Malicious code generation	Manipulating an LLM to generate backdoors (e.g., external command calls) during use.	Generation of code to open a communication channel with a specific, malicious IP.

3.2.3 Mitigation Strategies to Protect Data Privacy and Enhance Security in Testing with Generative AI

As GenAI becomes mainstream, and with the inherent risks involved, regulations and standards emerge to mitigate them (see section 3.4.1).

Data protection regulations like GDPR do not restrict the applications of GenAI explicitly but do provide safeguards that may limit what can be done, particularly regarding lawfulness and limitations on purposes of collection, processing, and storage of data.

To mitigate these risks, organizations should implement robust data privacy measures, including:

- Data minimization: Avoiding the processing of sensitive data unless legally permitted and using only the necessary amount of non-sensitive data in AI testing to reduce data privacy risks.
- Data anonymization and pseudonymization: Masking or replacing sensitive information with non-identifiable data.
- Secure data storage and transmission: Implementing strong encryption and access controls.
- Resources training: Organizations should establish clear training programs and policies to ensure the responsible use of GenAI tools, promote ethical practices, and mitigate potential risks.

Additional mitigation strategies can be considered when implementing GenAI for testing:

- Systematic review of the generated output: Human evaluation is essential for ensuring quality and accuracy of GenAI-powered test tasks.
- Evaluation by comparison with another LLM: This involves using several LLMs on a given task to evaluate outputs by comparing their responses.
- Choice of a secure, operational environment: Depending on the level of confidentiality required, organizations can opt for different secure solutions: Using a commercial, secure offering from an LLM provider, operating the LLM in a secure cloud or installing the LLM in the organization's infrastructure.
- Regular security audits and vulnerability assessments: Identifying and addressing weaknesses in GenAI systems.
- Staying updated with security best practices: Keeping up to date with the latest security guidelines and technologies.

The strategies are often complementary to each other and a combination of these is required to ensure data security while using GenAI. It is highly recommended to involve senior Security Engineers, Legal counsel, the Chief Technology Officer (CTO), or the Chief Information Security Officer (CISO), if present in the organization.

Hands-On Objective 3.2.3 (H0): Recognize data privacy and security risks in a given Generative AI for testing case study

This demonstration illustrates how data privacy and security risks can arise when using GenAI in software testing. Participants will explore case studies to identify potential threats, such as model vulnerabilities, unauthorized data access, or malicious use of generated outputs. They will explore mitigation strategies, including secure data handling, robust access controls, and AI monitoring practices, while reflecting on the ethical and practical implications.

By the end, participants will understand data privacy principles and learn to recognize and address security risks in GenAI test conditions.

3.3 Energy Consumption and Environmental Impact of Generative AI in Software Testing

Studies such as (Luccioni 2024a) show that training and processing LLMs require intensive use of a large number of specialized computing resources. LLMs are made available as web-based services, and their use increases the load on devices, networks, and data centers, leading to higher energy consumption.

3.3.1 The Impact of Using GenAI on Energy Consumption and CO2 Emissions

The environmental impact of GenAI should not be underestimated, as energy consumption rises sharply as usage increases. The complexity of the task and the computational resources required influence energy consumption. For example, generating a single image using a powerful AI model can consume as much energy as fully charging a smartphone, while generating text consumes only a small percentage of a smartphone's charge (Heikkilä 2023).

Even if it is hard to get accurate data on the environmental impact of GenAI (Luccioni 2024b), it is clear that these energy-intensive operations collectively contribute to significant CO_2 emissions (Berthelot 2024). While a single search or text generation task may seem negligible, their cumulative effect across millions of users worldwide results in substantial environmental strain.

Adopting best practices, such as limiting unnecessary model interactions, is critical to mitigating the environmental risks posed by GenAI.

Hands-On Objective 3.3.1 (H1): Use a simulator to calculate the energy and CO_2 emissions for given test tasks with Generative AI

This exercise focuses on evaluating the energy consumption and associated CO_2 emissions of various generative AI tasks within software testing. Participants will use simulations to calculate these metrics and examine how different task characteristics and model usage affect the environmental impact.

By observing how different factors affect energy consumption and emissions, participants understand the drivers of energy consumption with LLMs.

3.4 AI Regulations, Standards, and Best Practice Frameworks

GenAI is transforming software testing by assisting testers in a variety of test tasks (see Chapter 2). However, these opportunities also bring significant risks, such as reasoning errors, data privacy, vulnerabilities, and environmental impacts (see sections 3.1, 3.2, and 3.3). Addressing these risks should consider general regulations, standards, and best practice frameworks for AI.

3.4.1 AI Regulations, Standards and Frameworks Relevant to GenAI in Software Testing

Below is an overview of key guidelines relevant to the use of GenAI in software testing:

Name / Type	Description	Application in software testing
ISO/IEC 42001:2023 Information technology – Artificial intelligence-Management system Type: Standard	Specifies requirements for managing AI systems within an organization.	Ensures that GenAI in testing adheres to recommended practices, promoting consistency and reliability.
ISO/IEC 23053:2022 Framework for Artificial Intelligence (AI) Systems Using Machine Learning Type: Standard	Provides a framework for AI lifecycle processes, emphasizing fault tolerance and transparency.	Provides a framework for data quality, transparency, and fault tolerance when using GenAI for testing.
EU AI Act Type: Regulation	Establishes a legal framework addressing AI risks, classifying applications by risk level. Source: (AI Act 2024)	Mandates compliance in transparency, accountability, and bias mitigation for GenAI used in testing.
NIST AI Risk Management Framework (US) Type: Framework	Offers guidelines for managing AI risks, focusing on fairness, transparency, and security. Source: (NIST AI RMF 1.0)	Ensures fairness and mitigates risks in GenAI, preventing biased test results.

As AI technologies and their regulatory landscapes continue to evolve, it is imperative for test organizations to stay updated on the development of regulations, standards, national laws, and best practice frameworks, such as those in this table.

4 LLM-Powered Test Infrastructure for Software Testing – 110 minutes

Keywords

test infrastructure

Generative AI Specific Keywords

fine-tuning, LLM-powered agent, Large Language Model Operations , retrieval-augmented generation, vector database

Learning Objectives and Hands-on Objectives for Chapter 4:

4.1 Architectural Approaches for LLM-Powered Test Infrastructure

GenAI-4.1.1	(K2)	Explain key architectural components and concepts of LLM-powered test infrastructure
GenAI-4.1.2	(K2)	Summarize Retrieval-Augmented Generation
HO-4.1.2	(H1)	Experiment with Retrieval-Augmented Generation for a given test task
GenAI-4.1.3	(K2)	Explain the role and application of LLM-powered agents in automating test processes
HO-4.1.3	(H0)	Observe how an LLM-powered agent assists in automating a repetitive test task

4.2 Fine-Tuning and LLMOps: Operationalizing Generative AI for Software Testing

GenAI-4.2.1	(K2)	Explain the fine-tuning of language models for specific test tasks
HO-4.2.1	(H0)	Observe an example of a fine-tuning process for a given test task and language model
GenAI-4.2.2	(K2)	Explain LLMOps and its role in deploying and managing LLMs for test tasks

4.1 Architectural Approaches for LLM-Powered Test Infrastructure

AI chatbots and LLM-powered test tools are two types of test infrastructures using LLMs. (see section 1.2.2).

Beyond the basic architecture of an LLM-powered test infrastructure (see section 4.1.1), Retrieval-Augmented Generation (see section 4.1.2) and LLM-powered agent architectures (see section 4.1.3) extend the functionality and usefulness of using LLMs in software testing.

4.1.1 Key Architectural Components and Concepts of LLM-Powered Test Infrastructure

An LLM-powered test infrastructure refers to a system that integrates an LLM into the software testing process to enhance automation, reasoning, and decision-making. Unlike a traditional AI chatbot, which primarily focuses on conversational interactions, an LLM-powered test tool is designed to support software testing by processing test-related queries, analyzing requirements, generating test cases, and evaluating outputs.

The typical architecture of an LLM-powered test infrastructure follows a multi-component design that facilitates secure and efficient interaction with the LLM. The architecture consists of front-end and back-end components, along with external data sources and an integrated LLM:

- The front-end serves as the user interface where testers interact with the system by inputting queries or commands.
- The back-end processes user input and manages critical functions such as authentication, data retrieval, prompt preparation, and interaction with the LLM.
- The LLM, which may be hosted as a third-party service (accessed via API) or a custom in-house model, generates responses based on structured prompts.

This architecture goes beyond a traditional client-server model by incorporating intelligent processing components, such as LLMs and multi-source back ends:

1. The LLM is not just a server but a smart processing component that interprets and reasons based on test products.
2. Unlike rule-based chatbots that follow scripted responses, an LLM-powered test infrastructure generates test insights dynamically from context—such as requirements, code, or test results.
3. The back end integrates multiple data sources, such as:
    - Relational databases (for structured data used in testing, such as test cases).
    - Vector databases (for semantic retrieval of related content using embeddings; see section 4.1.2).
4. The back end enhances the LLM's raw output through post-processing, ensuring its responses align with the test conditions of the test process before presenting them to the front-end.

4.1.2 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) enhances LLMs by incorporating additional data sources into their response generation process (Zhao 2024), thereby increasing the relevance and accuracy of their outputs.

RAG combines retrieval systems with language models to generate context-aware responses. During preprocessing, large documents are broken into smaller chunks (e.g., 256-512 tokens) to ensure focused retrieval and compatibility with the model context window. Each chunk is cleaned, processed, and encoded into a high-dimensional vector (embedding) using pre-trained models. These embeddings, that can be stored in vector databases, enable efficient similarity-based retrieval at runtime (inference). A user query is encoded, relevant chunks are retrieved based on semantic similarity, and these chunks are used as context for the language model to generate a grounded response.

A relevant response is essentially an output generated by the language model that is deeply rooted in relevant, accurate, and contextually appropriate information gathered during the retrieval process. It ensures that the response is not only based on the model's pre-existing training but also enriched with precise data pertinent to the prompt. This synergy between retrieval and generation enhances the accuracy and relevance of the responses, making them more reliable and informative for the user.

In the user prompt processing phase, a RAG system works through a two-step process:

1. Retrieval: Given a user query, the system retrieves relevant information from the previously created vector databases. This retrieval is typically based on semantic similarity between the embeddings of the prompt and those of the chunks.
2. Generation: The retrieved information is then fed to the LLM, which generates a response that combines its existing knowledge with the newly acquired data, resulting in more accurate and contextually appropriate output.

RAG in software testing enables LLM-powered test infrastructure to access the company's enterprise data sources such as databases, documentation, and repositories to retrieve contextual information in real time, ensuring that test tasks such as test analysis or test design are aligned with the latest specifications, requirements, and existing test data.

Hands-On Objective 4.1.2 (H1): Experiment with Retrieval-Augmented Generation for a given test task

This hands-on exercise focuses on the application of RAG techniques for a given test task. Participants will experiment with a RAG system by incorporating documents and observe how it generates more or less accurate answers based on complex information. Participants will compare the output of the LLM with and without RAG on the given test task. This exercise aims to identify the strengths and limitations of the RAG system in handling different types of test tasks.

By examining the retrieved data and generated results, participants will gain insight into the role of RAG in enhancing LLM-powered test processes.

4.1.3 The Role of LLM-Powered Agents in Automating Test processes

LLM-powered agents (Wang 2024) are specialized GenAI applications powered by LLMs and designed for semi-autonomous or autonomous processing of defined tasks. At their core, these agents rely on LLMs for natural language understanding and generation, complemented by the possibility to process instructions, retrieve context, and take intelligent actions.

Unlike traditional AI chatbots that focus solely on question-response interactions, LLM-powered agents can perform tasks or "act" by invoking a predefined set of functions, commonly referred to as "tools." This capability allows them to interact with and manipulate external systems, making them highly versatile in task execution. LLM-powered agents' degree of autonomy can vary:

- Autonomous agents operate independently, performing tasks with minimal human intervention using predefined rules, reinforcement learning, and adaptive feedback loops.
- Semi-autonomous agents perform tasks with periodic human oversight to ensure that the output meets user-defined goals.

Multi-agent architectures involve a collaborative system where several agents, each with specialized roles, communicate and coordinate to solve complex problems more efficiently than a single agent. This coordinated effort among multiple AI agents is known as "orchestration."

In test processes, LLM-powered agents can automate test tasks by emulating human reasoning and decision making. However, these agents suffer from the same problems of possible hallucinations, reasoning errors, and biases observed when using LLMs (see Section 3.1). These agents can produce incorrect or misleading results, which can weaken the reliability of automated test processes. These risks can be mitigated by implementing automated verification procedures for the agents' results or using semi-autonomous agents for critical tasks.

Hands-On Objective 4.1.3 (H0): Observe how an LLM-powered agent assists in automating a repetitive test task

The demonstration focuses on a test task performed by an LLM-powered agent. The input data passed to the agent, its behavior, and the results of its actions will be demonstrated to illustrate the various aspects of integrating agent-based solutions into a test process.

This demonstration shows a concrete example of an LLM-powered agent in the context of a test task.

4.2 Fine-Tuning and LLMOps: Operationalizing Generative AI for Software Testing

Two key practices for operationalizing LLM-powered test infrastructure for testing include fine-tuning LLMs and managing the operational pipeline through LLMOps (Mailach 2024).

4.2.1 Fine-Tuning LLMs for Test tasks

Fine-tuning adapts a pre-trained Language Model (LM), such as an LLM or a Small Language Model (SLM, see section 1.1.2), to perform specific tasks or tailor it to particular domains (Parthasarathy 2024). This involves further training the model on a targeted dataset, allowing it to learn domain-specific knowledge and nuances. By fine-tuning, the model's performance is enhanced for specialized applications, making it more accurate and relevant to the intended use case.

In practice, fine-tuning is suitable for equipping generic LLMs with specialized reasoning abilities relevant to a specific domain or to adopt a vocabulary unique to that field. Fine-tuning can also be applied to smaller models, known as SLMs which are less resource intensive. By fine-tuning an SLM, one can achieve higher performance levels for specific tasks without the same computational overhead required for LLMs. This comparison highlights the flexibility and efficiency in using both LLMs and SLMs based on the specific requirements of the task.

For example, in software testing, fine-tuning can enable an LLM or SLM to generate test cases from user stories in an output format specific to the organization's context. By training the model on the organization's user stories and corresponding test cases, the model aligns with the organization's specific test process and terminology.

Fine-tuning a GenAI model for software testing presents several challenges:

- Avoid biased or inaccurate results by ensuring the use of high-quality, task-specific training datasets.
- Mitigating overfitting (model becomes too specialized to the training data, negatively impacting its performance on new, unseen data) to maintain generalization across different scenarios.
- Addressing opacity (lack of transparency in how an LLM makes its decisions or produces its outputs) in the model's reasoning, which complicates debugging and validation
- Managing the significant computational resources required for the fine-tuning process (for LLMs).

Hands-On Objective 4.2.1 (H0): Observe an example fine-tuning process for a given test task and LLM/SLM

This demonstration shows the various steps involved in fine-tuning an LLM for a given test task. It starts with selecting an appropriate LLM or SLM. Next, a data set is presented that is tailored to the given test task. Then an exemplary solution for the fine-tune process is shown (e.g. a machine learning framework). Finally, a prompt is sent to the fine-tuned model, and the quality of the generated output is discussed.

This demonstration of the LLM/SLM fine-tuning process for a test task shows several key aspects of this process and addresses in particular the quality of training data.

4.2.2 LLMOps when Deploying and Managing LLMs for Software Testing

LLMOps, or Large Language Model Operations, refers to the set of practices, tools, and processes designed to streamline the development, deployment, and maintenance of LLMs in production environments (Sinha 2024).

The use of generative AI in an organization's test processes can be accomplished in a few different ways, which will influence the LLMOps decisions to be made. Here are three possible approaches:

- Using an AI chatbot: The primary considerations for this approach include managing data privacy and security risks while optimizing cost. Organizations might use LLM-as-a-Service platforms if the necessary assurances are given or deploy in-house infrastructure using open-source licensed LLMs for greater control. A rigorous assessment of vendor assurances or internal capabilities is critical to mitigate data privacy and security risks (see section 3.2) and ensure operational efficiency.
- Using a test tool with generative AI capabilities: This approach has similar considerations to AI chatbots, such as data privacy, security, and operational costs. In addition, organizations must evaluate the data security and performance assurances offered by the test tool provider. These test tools typically complement existing test processes, which require a thorough cost-benefit analysis and risk assessment.
- In-house development of a test tool based on generative AI: This approach emphasizes comprehensive control of data privacy and security risks, as well as careful planning for AI operating costs such as computational resources, data storage, and staff training. Organizations must also establish structured processes for validating and maintaining developments specific to GenAI. Developing in-house solutions requires expertise in implementing and deploying an LLM-powered test infrastructure.

These approaches are not mutually exclusive since an organization might utilize an AI chatbot for some tasks while developing custom tools for others. Thus, they may be implemented simultaneously depending on the specific test activities involved. Furthermore, they can incorporate additional technologies, such as RAG and fine-tuning of LLMs/SLMs, to enhance the effectiveness and adaptability of the test processes with GenAI.

5 Deploying and Integrating Generative AI in Test organizations – 80 minutes

Keywords

None

Generative AI Specific Keywords

shadow AI

Learning Objectives and Hands-on Objectives for Chapter 5:

5.1 Roadmap for Adoption of Generative AI in Software Testing

GenAI-5.1.1	(K1)	Recall the risks of shadow AI
GenAI-5.1.2	(K2)	Explain the key aspects to consider when defining a Generative AI strategy for software testing
GenAI-5.1.3	(K2)	Summarize key criteria for selecting LLMs/SLMs for software test tasks in a given context
HO-5.1.3	(H1)	Estimate the recurring costs of using Generative AI for a given test task
GenAI-5.1.4	(K1)	Recall key phases in the adoption of Generative AI in a test organization

5.2 Manage Change when Adopting Generative AI for Software Testing

GenAI-5.2.1	(K2)	Explain the essential skills and knowledge areas required for testers to work effectively with generative AI in test processes
GenAI-5.2.2	(K1)	Recall strategies for cultivating AI skills within test teams to support the adoption of Generative AI in test activities
GenAI-5.2.3	(K1)	Recognize how test processes and responsibilities shift within a test organization when adopting Generative AI

5.1 Roadmap for the Adoption of Generative AI in Software Testing

A test strategy with GenAI must carefully consider key aspects such as the test objectives to be achieved, appropriate LLM selection, issues with the input data used for prompting, and compliance with AI standards and regulations. Based on this strategy, the organization can establish a roadmap and monitor progress in integrating GenAI into test processes.

5.1.1 Risks of Shadow AI

Shadow AI can lead to risks regarding security, compliance, and data privacy:

- Information security and data privacy weaknesses: Personal AI tools may lack robust security, leading to potential data breaches.
- Compliance and regulatory issues: Using unapproved AI tools can lead to non-compliance with industry standards and regulations (See Section 3.4.1), potentially resulting in legal consequences.
- Vague intellectual property: The use of AI tools with unclear licensing agreements can expose LLM users to intellectual property disputes, especially if copyrighted data is processed without proper authorization.

A strategy and steps for integrating and deploying GenAI can help test organizations avoid the risk of shadow AI.

5.1.2 Key Aspects of a Generative AI Strategy in Software Testing

To successfully implement a GenAI strategy in testing, organizations must carefully consider several key factors to ensure smooth integration and optimal results. This begins with defining measurable test objectives for GenAI, such as increasing test productivity, shortening test cycles, and improving test quality. Selecting the right LLMs is critical (see section 5.1.3) and should be aligned with these test objectives, while ensuring compatibility with existing test infrastructure and meeting system scalability requirements.

Data quality plays a critical role, as the effectiveness of LLM-powered testing depends on accurate, relevant input data, protected by robust security procedures. Maintaining high quality input data is therefore key to achieving results that can be trusted to be correct.

Comprehensive training programs should be provided to ensure that test teams have the technical and ethical skills necessary to use GenAI tools effectively. In addition to training, specific metrics should be collected to measure the effectiveness of GenAI results (See section 2.3.1).

To ensure compliance with regulatory standards and adherence to ethical guidelines, organizations should establish process guidelines for the use of GenAI, including rules for the use of sensitive data, transparency obligations (e.g., what was generated using GenAI), and quality gates with review of generated testware.

5.1.3 Selecting LLMs/SLMs for Software Test Tasks

There is a wide range of LLMs/SLMs, each with different functional capabilities (e.g., multimodal input, reasoning capabilities), technical features (e.g., context window size), and licensing types (e.g., commercial vs. open source). While many benchmarks are available to evaluate LLMs/SLMs for tasks such as natural language processing, code generation, or image analysis, only a few are specifically focused on software test tasks (Wenhan 2024). Therefore, selecting LLMs/SLMs for test tasks requires careful consideration of several key criteria:

- Model performance: Evaluate the model's performance for the targeted test tasks against the organization's benchmarks using metrics such as those presented in section 2.3.1.
- Fine-tuning potential: Evaluate whether it is possible and useful to fine-tune the language model (LLM or SLM) with domain-specific data to improve performance for a given use case, increasing accuracy and relevance in specialized contexts.
- Recurring cost: Consider the recurring costs of using the LLM/SLM, including licensing fees and operational expenses, to ensure that it fits within the organization's budget for the targeted test tasks.
- Community and support: Choose models with active community support and detailed documentation to aid in implementation and troubleshooting.

By carefully evaluating these criteria, test organizations can select one or more LLMs/SLMs that meets their specific needs and organizational constraints.

Hands-On Objective 5.1.3 (H1): Estimating the recurring costs of using Generative AI for a given test task

This exercise focuses on estimating the recurring costs of using GenAI for a specific test task based on various assumptions. These include factors such as the number of tokens in the input and output data, the prompts used, and the frequency of the task. Pricing models from several LLM/SLM vendors will be explored and compared, including at least one commercial solution and one open source licensed model.

This exercise provides an opportunity to calculate and experiment with the recurring costs of GenAI using practical test conditions, helping to understand the financial implications of different approaches and vendors.

5.1.4 Phases when Adopting Generative AI in Software Testing

Adopting GenAI within a test organization involves three key phases:

1. Discovery: The first phase focuses on awareness and capability building. Activities include training test teams on GenAI concepts, providing access to LLMs/SLMs, and experimenting with initial use cases to familiarize testers with GenAI and build confidence.
2. Initiation and usage definition: Once the basic understanding is established, the second phase focuses on identifying and prioritizing practical use cases for GenAI in software testing. This

phase includes evaluating LLM-powered test infrastructure, developing expertise, and ensuring alignment with the organization's needs (see **[ISTQB_CTFL_SYL]** section 6).

3. Utilization and iteration: At this advanced phase, organizations fully integrate GenAI into their test processes. Continuous monitoring of the progress of GenAI for software testing and related tools is in place, as well as measurement and management of the transformation to ensure sustainable benefits and scalability.

These phases can run in parallel for different use cases. For example, test report analysis may be further along the roadmap while test automation is in the early phases. It's also important to recognize and address early concerns such as fear of job displacement, which can impact adoption and team morale.

5.2 Manage Change when Adopting Generative AI for Software Testing

Successful implementation of GenAI in a test organization requires a structured approach to change management processes. Key aspects include building essential GenAI skills and evolving traditional testing roles to embrace AI-enabled test processes. The transformation involves both technical skills and organizational aspects.

5.2.1 Essential Skills and Knowledge for Testing with Generative AI

Successful integration of GenAI into testing requires mastering prompt engineering techniques, understanding model context windows, and developing test review methods. Testers must combine domain and test expertise with AI skills to evaluate LLM-driven testing in tasks such as test case generation, defect report analysis, and test data generation.

Key competencies include assessing LLM capabilities, understanding prompt refinement techniques, and evaluation of AI-generated testware. Essential knowledge includes understanding the inherent risks of GenAI, along with awareness of common mitigation strategies. Testers should understand the data security implications of sharing testware with LLMs, implement proper data sanitization (removing or masking sensitive, personal, or confidential information), and follow data privacy-preserving prompt engineering practices. Environmental considerations include optimizing model selection and usage patterns to reduce computational overhead, selecting right-sized models for test tasks, and balancing the benefits of GenAI automation with the impact on cost and energy consumption.

5.2.2 Building Generative AI Capabilities in Test Teams

A hands-on approach is essential to strategically train test teams in GenAI for testing. This includes practicing with various LLMs/SLMs, following structured learning paths, and gradually developing know-how through sharing within the organization. The focus of training is on developing practical skills through guided exercises, peer learning, and the gradual integration of AI into daily test tasks.

Test team members progress from mastering basic prompt creation to using more focused techniques, such as test-specific prompts. A prompt pattern is a reusable template for crafting effective prompts to guide GenAI toward consistent and reliable outputs. Internal communities of practice support ongoing knowledge sharing, with regular meetings to highlight successful GenAI applications, discuss challenges, and refine best practices. These communities promote continuous improvement by sharing prompt

pattern libraries and documenting lessons learned from GenAI for test implementations across projects and domains.

5.2.3 Evolving Test Processes in AI-Enabled Test organizations

The integration of GenAI transforms the traditional test processes of testers and test managers within test organizations.

Testers evolve from test design and test execution specialists to AI-assisted test specialists, combining their expertise in test techniques with skills to guide and verify AI-generated testware. Their test tasks expand to include review of the overall AI-based output, prompt refinement, and maintenance of test-specific prompt libraries.

The responsibilities of test managers are updated to include the development of an AI-based test strategy, AI-based risk management, and monitoring and control of AI-based test processes. Test managers focus on balancing human and AI capabilities, establishing AI governance frameworks for use cases, and ensuring that their test teams maintain both traditional testing competencies and AI literacy. Test managers will not only lead human testers but also coordinate with GenAI-powered test agents, requiring new management skills for overseeing hybrid teams of people and GenAI tools.

6 References

Standards

- **ISO/IEC 42001:2023** (2023), Information technology — Artificial intelligence — Management system
- **ISO/IEC 23053:2022** (2022), Framework for Artificial Intelligence (AI) Systems Using Machine Learning (ML)

ISTQB® Documents

- **[ISTQB_CTFL_SYL]** ISTQB® Foundation Level Syllabus v4.0, 2023

Glossary References

- **ISTQB® Glossary** https://glossary.istqb.org/

Books

- Winteringham M. (2024) Software Testing with Generative AI, Manning Publications (5 Mar. 2025), ISBN-13 : 978-1633437364, 10 Dec. 2024 - 304 pages

Articles

- (Berthelot 2024) Berthelot, Adrien, et al. "Estimating the environmental impact of Generative-AI services using an LCA-based methodology." *Procedia CIRP* 122 (2024): 707-712.
- (Gallegos 2024) Gallegos, Isabel O., et al. "Bias and fairness in large language models: A survey." *Computational Linguistics* (2024): 1-79.
- (Li 2024) Yihao Li, Pan Liu, Haiyang Wang, Jie Chu, W. Eric Wong, Evaluating Large Language Models for Software Testing, Computer Standards & Interfaces (2024), doi: https://doi.org/10.1016/j.csi.2024.103942
- (Luccioni 2024a) Luccioni, Sasha, Yacine Jernite, and Emma Strubell. "Power hungry processing: Watts driving the cost of AI deployment?." *The 2024 ACM Conference on Fairness, Accountability, and Transparency*. 2024.
- (Mailach 2024) Mailach, Alina, et al. "Practitioners' Discussions on Building LLM-based Applications for Production." *arXiv preprint arXiv:2411.08574* (2024).
- (Mirzadeh 2024) Mirzadeh, Iman et al. “GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models.” ArXiv abs/2410.05229 (2024)
- (NIST AI RMF 1.0) National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1, U.S. Department of Commerce, 2023, https://doi.org/10.6028/NIST.AI.100-1.

- (Parthasarathy 2024) Parthasarathy, Venkatesh Balavadhani, et al. "The ultimate guide to fine-tuning llms from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities." arXiv preprint arXiv:2408.13296 (2024).
- (Schulhoff 2024) Schulhoff, S., "The Prompt Report: A Systematic Survey of Prompting Techniques", Art. no. arXiv:2406.06608, 2024. doi:10.48550/arXiv.2406.06608.
- (Shuyin 2023) Ouyang, Shuyin, et al. "LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation." arXiv preprint arXiv:2308.02828 (2023).
- (Sinha 2024) Sinha, Megha, Sreekanth Menon, and Ram Sagar. "LLMOps: Definitions, Framework and Best Practices." 2024 International Conference on Electrical, Computer and Energy Technologies (ICECET. IEEE, 2024.
- (Wang 2024) Wang, Yanlin, et al. "Agents in Software Engineering: Survey, Landscape, and Vision." arXiv preprint arXiv:2409.09030 (2024).
- (Wenhan 2024) Wang, Wenhan, et al. "TESTEVAL: Benchmarking Large Language Models for Test Case Generation." *arXiv preprint arXiv:2406.04531* (2024).
- (Zhao 2024) Zhao, Penghao, et al. "Retrieval-augmented generation for ai-generated content: A survey." arXiv preprint arXiv:2402.19473 (2024).

Web Pages

(AI Act 2024) European Commission. "European Approach to Artificial Intelligence." *Shaping Europe's Digital Future*, European Commission, https://digital-strategy.ec.europa.eu/en/policies/european-approach-artificial-intelligence. Accessed 24 Nov. 2024.

(Heikkilä 2023) Heikkilä, M. (2023, December 1). Making an image with generative AI uses as much energy as charging your phone. MIT Technology Review. Retrieved from https://www.technologyreview.com/2023/12/01/1084189/making-an-image-with-generative-ai-uses-as-much-energy-as-charging-your-phone/

(Luccioni 2024b) Luccioni, S. (2024, February 22). Generative AI's environmental costs are soaring. Nature. Retrieved from https://www.nature.com/articles/d41586-024-00478-x

(Google Dev Glossary 2024) Google Developers. (n.d.). Machine learning glossary: Generative AI. Retrieved November 24, 2024, from https://developers.google.com/machine-learning/glossary/generative

(MIT 2024) "Glossary of Terms: Generative AI Basics." *MIT Sloan Teaching & Learning Technologies*, MIT Sloan School of Management, https://mitsloanedtech.mit.edu/ai/basics/glossary. Accessed 24 Nov. 2024.

The previous references point to information available on the Internet and elsewhere. Even though those references were checked at the time of publication of this syllabus, the ISTQB® cannot be held responsible if the references are not available anymore.

7 Appendix A – Learning Objectives/Cognitive Level of Knowledge

The specific learning objectives applying to this syllabus are shown at the beginning of each chapter. Each topic in the syllabus will be examined according to the learning objective for it.

The learning objectives begin with an action verb corresponding to its cognitive level of knowledge as listed below.

Level 1: Remember (K1)

The candidate will remember, recognize and recall a term or concept.

Action verbs: Recall, recognize.

Examples
Recall the concepts of the test pyramid.
Recognize the typical objectives of testing.

Level 2: Understand (K2)

The candidate can select the reasons or explanations for statements related to the topic, and can summarize, compare, classify and give examples for the testing concept.

Action verbs: Classify, compare, differentiate, distinguish, explain, give examples, interpret, summarize

Examples	**Notes**
Classify test tools according to their purpose and the test activities they support.	
Compare the different test levels.	Can be used to look for similarities, differences or both.
Differentiate testing from debugging.	Looks for differences between concepts.
Distinguish between project and product risks.	Allows two (or more) concepts to be separately classified.
Explain the impact of context on the test process.	
Give examples of why testing is necessary.	

Examples	Notes
Infer the root cause of defects from a given profile of failures.	
Summarize the activities of the work product review process.	

Level 3: Apply (K3)

The candidate can carry out a procedure when confronted with a familiar task, or select the correct procedure and apply it to a given context.

Action verbs: Apply, implement, prepare, use

Examples	Notes
Apply boundary value analysis to derive test cases from given requirements.	Should refer to a procedure / technique / process etc.
Implement metrics collection methods to support technical and management requirements.	
Prepare installability tests for mobile apps.	
Use traceability to monitor test progress for completeness and consistency with the test objectives, test strategy, and test plan.	Could be used in a LO that wants the candidate to be able to use a technique or procedure. Similar to 'apply'.

Reference

(For the cognitive levels of learning objectives)

Anderson, L. W. and Krathwohl, D. R. (eds) (2001) A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon

8 Appendix B – Business Outcomes traceability matrix with Learning Objectives

This section lists the traceability between the Business Outcomes and the Learning Objectives of Certified Tester Testing with Generative AI. Hands-on objectives are not mentioned in this table as each HO is associated with a single LO. Traceability between an HO and a BO is via the LO to which the HO is associated.

Business Outcomes: Certified Tester Testing with Generative AI			BO1	BO2	BO3	BO4	BO5
GenAI-BO1	Understand the fundamental concepts, capabilities, and limitations of Generative AI		8				
GenAI-BO2	Develop practical skills in prompting large language models for software testing			10			
GenAI-BO3	Gain insight into the risks and mitigations of using Generative AI for software testing				11		
GenAI-BO4	Gain insight into the applications of Generative AI solutions for software testing					19	
GenAI-BO5	Contribute effectively to the definition and implementation of a Generative AI strategy and roadmap for software testing within an organization						13

Business Outcomes: Certified Tester Testing with Generative AI			BO1	BO2	BO3	BO4	BO5
Unique LO	Learning Objective	K-Level					
1	Introduction to Generative AI for Software Testing						
1.1	Generative AI Foundations and Key Concepts						
GenAI-1.1.1	Recall different types of AI: symbolic AI, classical machine learning, deep learning, and Generative AI	K1	X				
GenAI-1.1.2	Explain the basics of Generative AI and Large Language Models	K2	X				
GenAI-1.1.3	Distinguish between foundation, instruction-tuned and reasoning LLMs	K2	X				
GenAI-1.1.4	Summarize the basic principles of multimodal LLMs and vision-language models	K2	X				
1.2	Leveraging Generative AI in Software Testing: Core Principles						
GenAI-1.2.1	Give examples of LLM capabilities for test tasks	K2	X			X	
GenAI-1.2.2	Compare interaction models when using GenAI for software testing	K2	X			X	

Business Outcomes: Certified Tester Testing with Generative AI			BO1	BO2	BO3	BO4	BO5
2	Prompt Engineering for Effective Software Testing						
2.1	Effective Prompt Development						
GenAI-2.1.1	Give examples of the structure of prompts used in generative AI for software testing	K2		X			
GenAI-2.1.2	Differentiate core prompting techniques for software testing	K2		X			
GenAI-2.1.3	Distinguish between system prompts and user prompts	K2		X			
2.2	Applying Prompt Engineering Techniques to Software Test tasks						
GenAI-2.2.1	Apply generative AI to test analysis tasks	K3		X			
GenAI-2.2.2	Apply generative AI to test design and test implementation tasks	K3		X			
GenAI-2.2.3	Apply generative AI to automated regression testing	K3		X			
GenAI-2.2.4	Apply Generative AI for test monitoring tasks	K3		X			

Business Outcomes: Certified Tester Testing with Generative AI			BO1	BO2	BO3	BO4	BO5
GenAI-2.2.5	Select and apply appropriate prompting techniques for a given context and testing task	K3		X		X	
2.3	Evaluate Generative AI Results and Refine Prompts for Software Test tasks						
GenAI-2.3.1	Understand the metrics for evaluating the results of Generative AI on test tasks	K2		X	X	X	
GenAI-2.3.2	Give examples of methods for evaluating and iteratively refining prompts	K2		X	X	X	
3	Managing Risks of Generative AI in Software Testing						
3.1	Hallucinations, Reasoning Errors and Biases						
GenAI-3.1.1	Recall the definitions of hallucinations, reasoning errors and biases in Generative AI systems	K1	X		X	X	
GenAI-3.1.2	Analyze hallucinations, reasoning errors and biases in LLM output	K3			X	X	
GenAI-3.1.3	Summarize mitigation techniques for GenAI hallucinations, reasoning errors and biases in software test tasks	K2			X	X	
GenAI-3.1.4	Recall mitigation techniques for non-deterministic behavior of LLMs	K1	X		X	X	

Business Outcomes: Certified Tester Testing with Generative AI			BO1	BO2	BO3	BO4	BO5
3.2	Data Privacy and Security Risks of Generative AI in Software Testing						
GenAI-3.2.1	Explain key data privacy and Security risks associated with using Generative AI in software testing	K2			X	X	
GenAI-3.2.2	Give examples of data privacy and vulnerabilities in using Generative AI in software testing	K2			X	X	
GenAI-3.2.3	Summarize mitigation strategies to protect data privacy and enhance security in Generative AI for software testing	K2			X	X	
3.3	Energy Consumption and Environmental Impact of Generative AI in Software Testing						
GenAI-3.3.1	Explain the impact of task characteristics and model usage on the energy consumption of Generative AI in software testing	K2			X	X	
3.4	AI Regulations, Standards and Best Practice Frameworks						
GenAI-3.4.1	Recall examples of AI regulations, standards and best practice frameworks relevant to Generative AI in software testing	K1			X	X	X
4	LLM-Powered Test Infrastructure for Software Testing						
4.1	Architectural Approaches for LLM-Powered Test Infrastructure						

Business Outcomes: Certified Tester Testing with Generative AI			BO1	BO2	BO3	BO4	BO5
GenAI-4.1.1	Explain key architectural components and concepts of LLM-powered test infrastructure	K2				X	X
GenAI-4.1.2	Summarize Retrieval-Augmented Generation	K2				X	X
GenAI-4.1.3	Explain the role and application of LLM-powered agents in automating test processes	K2				X	X
4.2	Fine-Tuning and LLMOps: Operationalizing Generative AI for Software Testing						
GenAI-4.2.1	Explain the fine-tuning of language models for specific test tasks	K2				X	X
Ge2AI-4.2.2	Explain LLMOps and its role in deploying and managing LLMs for test tasks	K2				X	X
5	Deploying and Integrating Generative AI in Test organizations						
5.1	Roadmap for the Adoption of Generative AI in Software Testing						
GenAI-5.1.1	Recall the risks of shadow AI	K1					X
GenAI-5.1.2	Explain the key aspects to consider when defining a Generative AI strategy for software testing	K2					X

Business Outcomes: Certified Tester Testing with Generative AI			BO1	BO2	BO3	BO4	BO5
GenAI-5.1.3	Summarize key criteria for selecting LLMs/SLMs for software test tasks in a given context	K2					X
GenAI-5.1.4	Recall key phases in the adoption of Generative AI in a test organization	K1					X
5.2	Manage Change when Adopting Generative AI for Software Testing						
GenAI-5.2.1	Explain the essential skills and knowledge areas required for testers to work effectively with generative AI in test processes	K2					X
GenAI-5.2.2	Recall strategies for cultivating AI skills within test teams to support the adoption of Generative AI in test processes	K1					X
GenAI-5.2.3	Recognize how test processes and responsibilities shift within a test organization when adopting Generative AI for testing	K1					X

9 Appendix C – Release Notes

This version is V1.0. No release notes for this first version.

10 Appendix D – Generative AI Specific Terms

Term Name	Definition
AI chatbot	A conversational agent that uses LLMs to process queries and generate human-like text responses, enabling interactive communication with users.
Context window	The span of text, measured in tokens, that a language model considers when generating responses, influencing the relevance and coherence of its outputs.
Deep learning	ML using neural networks with multiple layers.
Embedding	A technique used to represent tokens as dense vectors in a continuous space, learned during training to capture semantic, syntactic, and contextual relationships.
Feature	An individual measurable attribute of the input data used for training by an ML algorithm and for prediction by an ML model
Few-shot prompting	A technique where a model is given a few examples within the prompt to guide it in generating appropriate responses.
Fine-tuning	A supervised learning process using a dataset of labeled examples to update LLM weights and adapt them for specific tasks or domains.
Foundation LLM	General-purpose models pre-trained on a wide range of text data, capable of predicting the next word based on learned linguistic patterns. Synonym: Base LLM
Generative AI (GenAI)	A type of artificial intelligence system that uses machine learning models to generate (new) intellectual content that resembles human-created content.
Generative pre-trained transformer (GPT)	A type of transformer-based deep learning model pre-trained on vast amounts of text data to understand and generate human-like text.
Hallucination	Wrong information created by an LLM.
Instruction-tuned LLM	A foundation LLM trained to follow instructions, often reinforced by feedback to encourage correct answers.
Large language model (LLM)	A computer program that uses very large collections of language data in order to understand and produce text in a way that is similar to the way humans do.
LLM-powered agent	An application that integrates LLM reasoning, decision-making, and memory, using tools to perform tasks.

LLMOps	Practices and tools focused on deploying, monitoring, and maintaining LLMs in production environments.
machine learning (ML)	The process using computational techniques to enable systems to learn from data or experience (ISO/IEC TR 29119-11).
Meta prompting	The crafting of higher-level instructions that generate specific prompts for exploring or automating capabilities.
Multimodal model	GenAI models that are capable of processing and generating content across multiple data types, such as text, images, and audio.
natural language processing (NLP)	The processing of data encoded in natural language by computers to retrieve information and for knowledge representation.
One-shot prompting	A prompt writing technique where the prompt contains one example to guide the LLM's response.
Prompt	A natural language input provided to elicit specific response in Generative AI and large language models.
Prompt chaining	A prompting technique that involves using the output of one prompt as the input for another, creating a sequence of prompts.
Prompt engineering	The process of designing and refining input prompts to guide LLMs toward producing desired outputs.
Reasoning LLM	An LLM building upon instruction-tuned models by refining their ability to emulate human-like reasoning processes
Retrieval-augmented generation (RAG)	A technique combining LLM capabilities with a retriever to fetch relevant data for generating accurate, contextually relevant responses.
Shadow AI	The use of GenAI tools or systems within an organization without formal approval or oversight.
Small language model (SML)	Language models that are intentionally designed and trained to be small, offering a balance between efficiency and task-specific language understanding.
Symbolic AI	An AI approach that uses symbols, rules, and structured knowledge to model reasoning.
System prompt	A predefined instruction set, typically hidden from the chatbot's users, that consistently establishes the context, tone, and boundaries for an LLM's responses and guides its behavior throughout interactions.
Temperature	A parameter that controls the randomness or creativity of a LLM's outputs.

Tokenization	The process of breaking down text into smaller units for processing by language models.
Transformer	A deep learning model architecture that utilizes self-attention mechanisms to capture long-range dependencies in input sequences.
User prompt	An instruction or query entered by a user into a Large Language Model (LLM) that directs the model's response to fulfil specific tasks or provide desired information.
Vector database	A database optimized for storing and querying high-dimensional vector representations of data
Vision-language model	A GenAI system that jointly processes visual and textual data to perform tasks by linking and generating content across both modalities.
Zero-shot prompting	A prompt writing technique where the prompt contains no examples, relying on the model's pre-existing knowledge to generate a response.

11 Appendix E – Trademarks

ISTQB® is a registered trademark of International Software Testing Qualifications Board

12 Index

All testing terms are defined in the ISTQB® Glossary (http://glossary.istqb.org/).

acceptance criteria, 14, 19, 21, 24, 25, 26, 34

AI chatbot, 11, 13, 21, 23, 43, 47

biases, 11, 33, 34, 35, 36, 45, 60

chatbots, 17, 45, 47

context window, 13, 15, 44, 50

data privacy, 11, 26, 33, 37, 38, 39, 40, 47, 49, 51, 61

deep learning, 13, 14, 58, 65

embedding, 13, 44

feature, 13, 14

few-shot prompting, 19, 20, 21, 22, 23, 24, 26, 28

fine-tuning, 42, 45, 46, 47, 54, 62

foundation LLM, 13, 65

GenAI, 1, 3, 8, 9, 10, 11, 12, 13, 14, 17, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 42, 45, 46, 47, 48, 49, 50, 51, 52, 57, 58, 59, 60, 61, 62, 63, 65, 66

generative AI, 8, 9, 11, 13, 19, 20, 31, 33, 34, 40, 47, 48, 54, 59, 63

generative pre-trained transformer, 13, 14

hallucination, 33

large language model, 13

LLM, 11, 13, 15, 17, 18, 21, 22, 23, 24, 25, 27, 28, 29, 30, 31, 33, 34, 35, 36, 37, 38, 42, 43, 44, 45, 46, 47, 49, 50, 51, 53, 54, 58, 61, 62, 65, 66, 67

LLMOps, 12, 42, 45, 47, 54, 62, 66

LLMs, 11, 12, 13, 14, 15, 16, 17, 18, 21, 23, 24, 26, 27, 29, 33, 34, 35, 36, 37, 38, 39, 40, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 58, 60, 65, 66

machine learning, 13, 14, 58

meta prompting, 19, 21, 22, 23, 24, 26, 29

multimodal model, 13

natural language processing, 19, 24, 50

one-shot prompting, 19

prompt, 11, 13, 15, 17, 19, 20, 21, 22, 23, 24, 25, 26, 27, 31, 32, 35, 36, 43, 44, 51, 52, 65, 66, 67

prompt chaining, 19, 21, 22, 23, 24, 25, 26, 36

prompt engineering, 11, 19, 20, 21, 23, 24, 51

RAG, 44, 47, 66

reasoning, 11, 13, 14, 27, 33, 34, 35, 36, 40, 43, 45, 46, 50, 58, 60, 65, 66

Retrieval-Augmented Generation, 11, 36, 42, 43, 44, 62, 66

security, 11, 33, 37, 38, 39, 40, 47, 49, 51, 61

shadow AI, 48

SLMs, 14, 46, 47, 48, 50, 51

symbolic AI, 13, 14, 58

system prompt, 19, 23

System prompts, 23

temperature, 33, 36

test automation, 8, 19, 21, 24, 27, 28, 29, 51

test case, 18, 19, 24, 26, 27, 29, 32, 34, 51

test condition, 19, 24, 30

test data, 14, 17, 19, 20, 25, 26, 28, 29, 34, 43, 44, 51

test design, 18, 19, 24, 25, 26, 29, 44, 52, 59

test infrastructure, 37, 42, 43, 44, 45, 47, 49, 51, 62

test report, 19, 20, 28, 51
tokenization, 11, 13, 14, 15, 16
Tokenization, 15, 67
tokens, 15, 44, 50, 65
transformer, 13, 15, 16, 34, 65
user prompt, 19, 23
user prompts, 14, 19, 23, 59
vector database, 42
Vision-language models, 16
vulnerability, 33, 38
zero-shot prompting, 19