PASCAL PUCHTLER
Software Entwicklung + Beratung

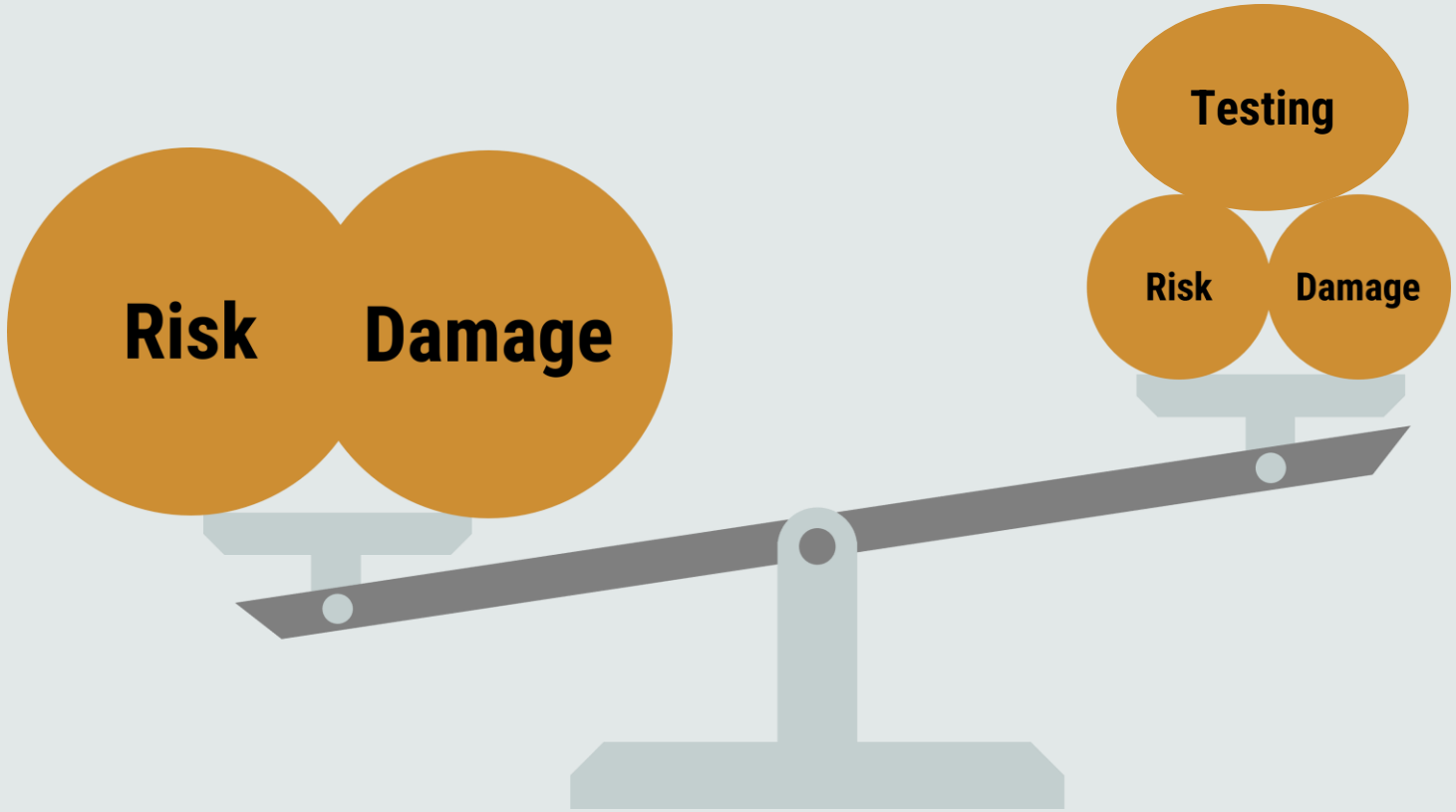
# Which kind of software tests do I really need?

A basic lecture

PyCon DE & PyData Berlin 2024

# Why do we test?

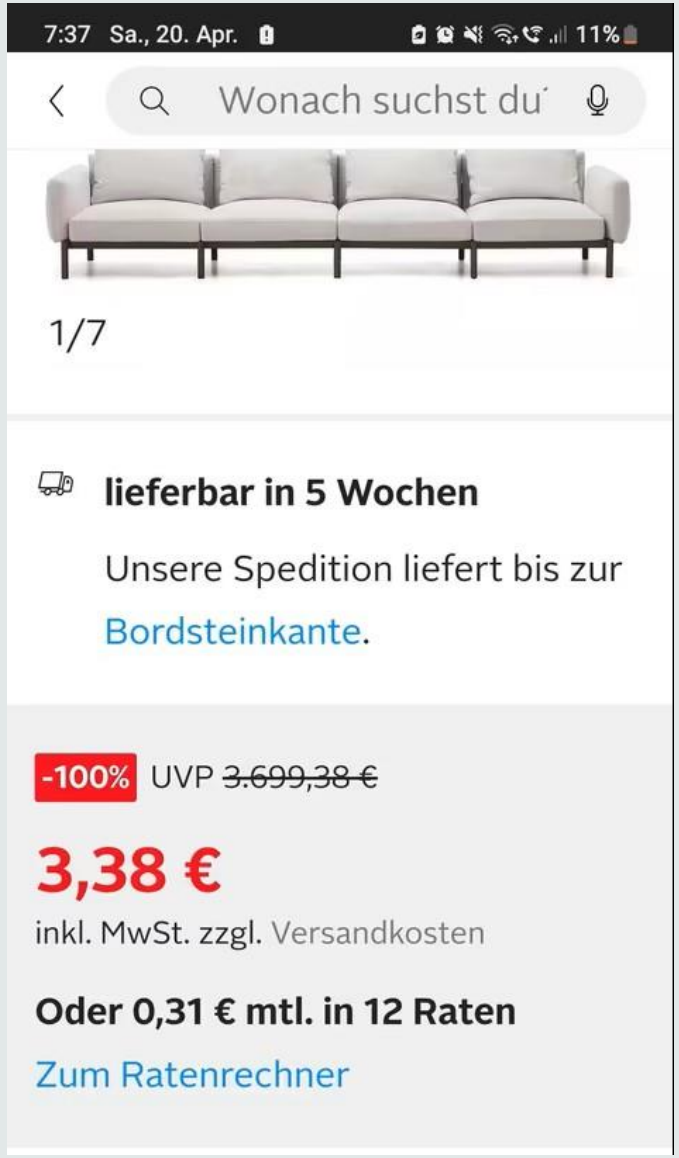
# How can we save money with testing?



# Invalid user input

3 days ago

3.700 € per order




https://www.mydealz.de/deals/preisfehler-bei-otto-sofa-von-natur24-fur-nur-338eur-2349338

# Full process has issues

8 days ago

$3,2 million

UNBERECHTIGTE AUSZAHLUNGEN

## Softwarefehler kostet Casinobetreiber Millionen

Mehrere Personen haben sich aufgrund eines Softwarefehlers zusammen 3,2 Millionen australische Dollar auszahlen lassen, die ihnen eigentlich nicht zustanden.

in Pocket speichern | merken

18. April 2024, 9:27 Uhr, Marc Stöckel


(Bild: pixabay.com / Bru-nO)

https://www.golem.de/news/unberechtigte-auszahlungen-softwarefehler-kostet-casinobetreiber-millionen-2404-184286.html

# Full process has issues

1 month ago

$40 million

BBC

Home News Sport Business Innovation Culture Travel Earth Video Live

## Commercial Bank of Ethiopia glitch lets customers withdraw millions

18 March 2024

Share

By Kalkidan Yibeltal, BBC News, Addis Ababa

https://www.bbc.com/news/world-68599027

# Leak of data



Forbes

FORBES > INNOVATION > CYBERSECURITY

## Warning As 26 Billion Records Leak: Dropbox, LinkedIn, Twitter Named

**Davey Winder** Senior Contributor ⓘ

*Veteran cybersecurity and tech analyst, journalist, hacker, author*

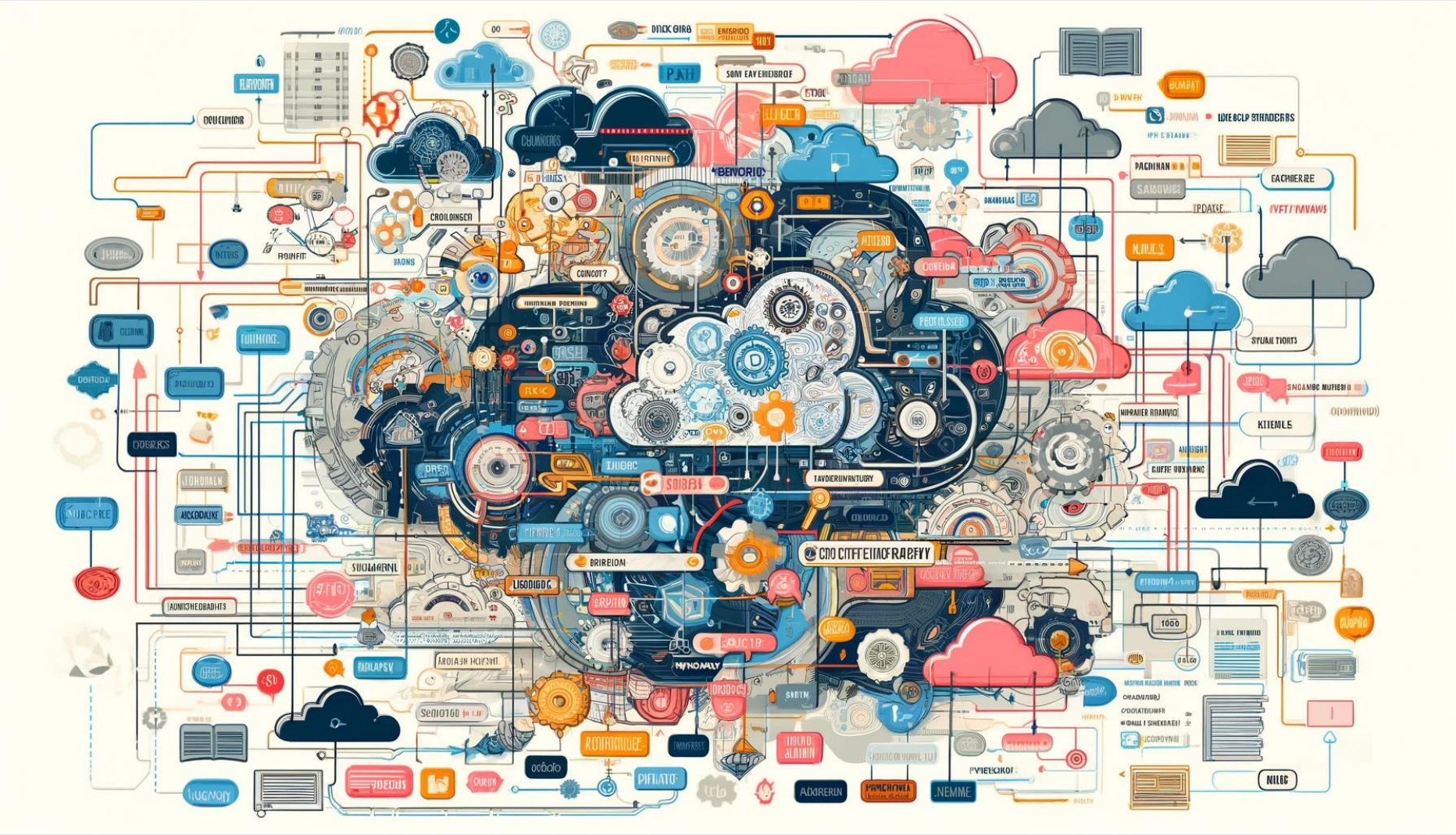
Follow

### Here's What You Need To Know

According to researchers from Security Discovery and CyberNews, the newly discovered database of leaked data runs to 12 terabytes in size and deserves the MOAB title.

https://www.bbc.com/news/world-68599027

# Not maintainable in 5 years





PASCAL PUCHTLER
Software Entwicklung + Beratung

# *Two ways to find the right kind of software tests*

**Risk analysis**



Quality attributes

# Risk & Damage

PASCAL PUCHTLER
Software Entwicklung + Beratung

| | | Risk | | | | |
|---|---|---|---|---|---|---|
| | | 1 Rare | 2 Unlikely | 3 Possible | 4 Likely | 5 Almost Certain |
| Damage | 5 Catastrophic | 5 | 10 | 15 | 20 | 25 |
| | 4 Major | 4 | 8 | 12 | 16 | 20 |
| | 3 Moderate | 3 | 6 | 9 | 12 | 15 |
| | 2 Minor | 2 | 2 | 6 | 8 | 10 |
| | 1 Negligible | 1 | 2 | 3 | 4 | 5 |

Risk = Low Moderate High Extreme

# Risk analysis

| Name | Risk | Damage | Risk * Damage |
|---|---|---|---|
| Invalid user input | 2 Unlikely | 5 Catastrophic | 10 |
| Full process has issues | 3 Possible | 5 Catastrophic | 15 |
| Leak of data | 3 Possible | 5 Catastrophic | 15 |
| Not maintainable in 5 years | 5 Almost Certain | 5 Catastrophic | 25 |

# *Two ways to find the right kind of software tests*

Risk analysis



**Quality attributes**

# Quality attributes - ISO 25010

| SOFTWARE PRODUCT QUALITY | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| FUNCTIONAL SUITABILITY | PERFORMANCE EFFICIENCY | COMPATIBILITY | INTERACTION CAPABILITY | RELIABILITY | SECURITY | MAINTAINABILITY | FLEXIBILITY | SAFETY |
| FUNCTIONAL COMPLETENESS<br>FUNCTIONAL CORRECTNESS<br>FUNCTIONAL APPROPRIATENESS<br>iso25000.com | TIME BEHAVIOUR<br>RESOURCE UTILIZATION<br>CAPACITY | CO-EXISTENCE<br>INTEROPERABILITY | APPROPRIATENESS RECOGNIZABILITY<br>LEARNABILITY<br>OPERABILITY<br>USER ERROR PROTECTION<br>USER ENGAGEMENT<br>INCLUSIVITY<br>USER ASSISTANCE<br>SELF-DESCRIPTIVENESS | FAULTLESSNESS<br>AVAILABILITY<br>FAULT TOLERANCE<br>RECOVERABILITY | CONFIDENTIALITY<br>INTEGRITY<br>NON-REPUDIATION<br>ACCOUNTABILITY<br>AUTHENTICITY<br>RESISTANCE | MODULARITY<br>REUSABILITY<br>ANALYSABILITY<br>MODIFIABILITY<br>TESTABILITY | ADAPTABILITY<br>SCALABILITY<br>INSTALLABILITY<br>REPLACEABILITY | OPERATIONAL CONSTRAINT<br>RISK IDENTIFICATION<br>FAIL SAFE<br>HAZARD WARNING<br>SAFE INTEGRATION |

PASCAL PUCHTLER
Software Entwicklung + Beratung

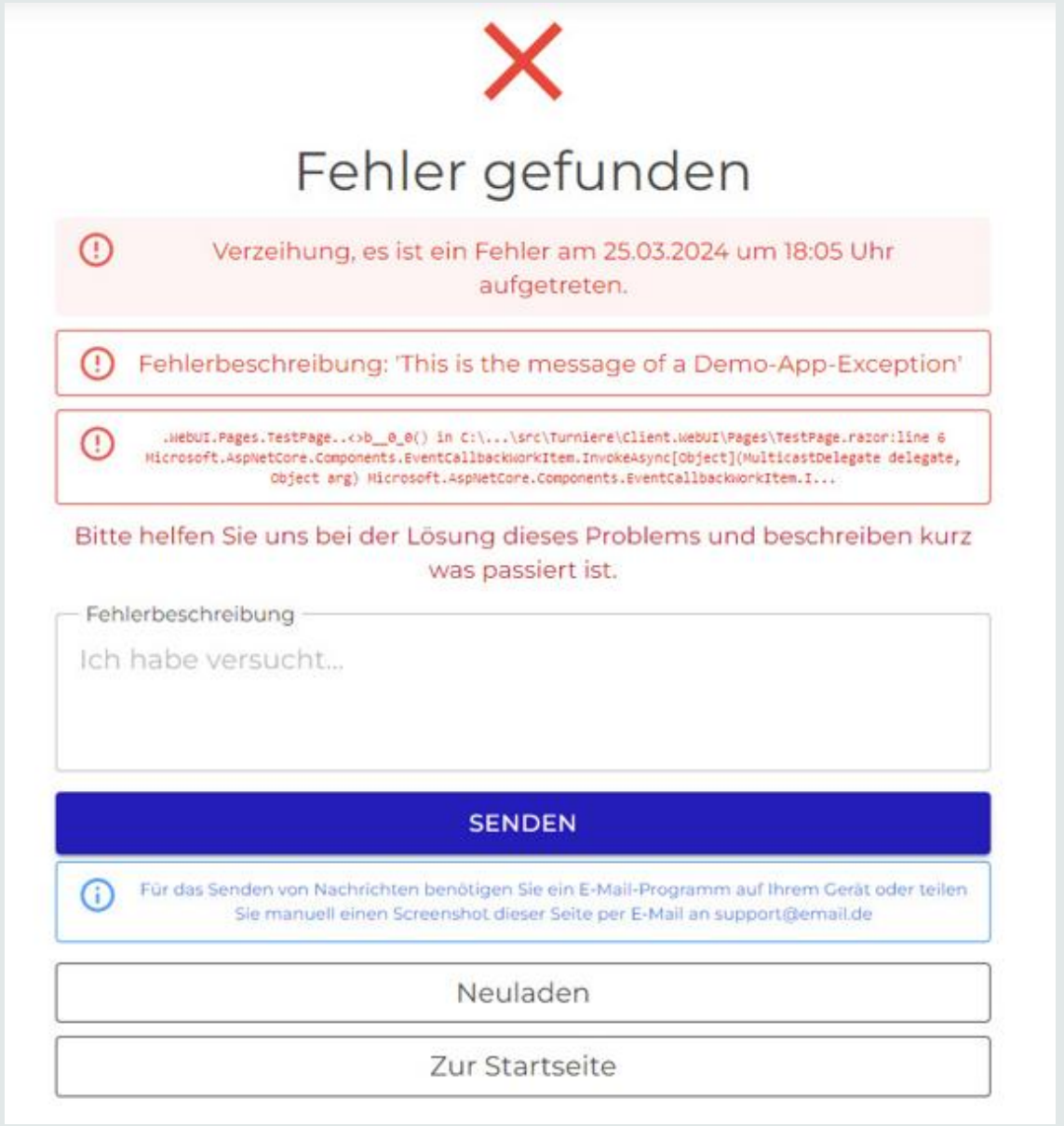
# Overview kind of software tests

| Code quality | Customer | Development process | Functional | Resources | Security | Stability | Data Since |
|---|---|---|---|---|---|---|---|
| Formatter | Test of documentation | Ticket review | Unit tests | Performance tests | Penetration test | Long time stability measurement | Training data |
| Linter | A/B testing | Code review | Mutation tests | Load test | Static security analysis | Hardware fault test | Training pipeline |
| Architectural analysis | Test environment for manual tests | Feature review | Integration tests | Stress test | | Compatibility test | Model validation |
| Typing | Crash reports | Test environments | System tests | Volume test | | Install/uninstall test | |
| | Color blindness test | | Full process test | | | Backup and recovery test | |
| | Click route analysis | | | | | Random data test | |
| | Test of slow/interrupted connection | | | | | Random click test | |
| | Feedback function | | | | | | |

# Customer as tester


Generated by https://chat.openai.com/

# Crash reports



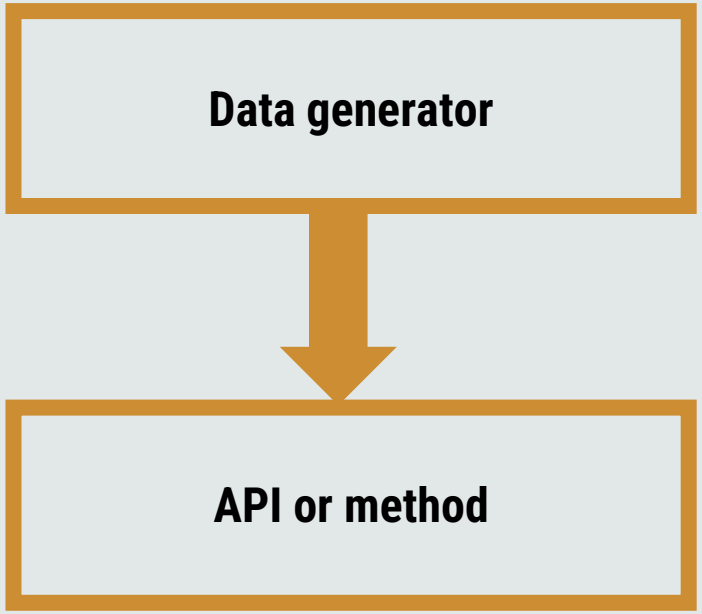
# *Test environment for manual tests*

**Developer environment**
Manual with on command

**Staging environment**
On commit (release branch)

**Customer test environment**
manual or on commit

# Random data test




```python
from faker import Faker

fake = Faker(["it_IT", "en_US", "ja_JP"])
for _ in range(10):
    print(fake.name())

# 鈴木 陽一
# Leslie Moreno
# Emma Williams
# 渡辺 裕美子
# Marcantonio Galuppi
# Martha Davis
# Kristen Turner
# 中津川 春香
# Ashley Castillo
# 山田 桃子
```

```
faker.providers.address
faker.providers.automotive
faker.providers.bank
faker.providers.barcode
faker.providers.color
faker.providers.company
faker.providers.credit_card
faker.providers.currency
faker.providers.date_time
faker.providers.emoji
faker.providers.file
faker.providers.geo
faker.providers.internet
faker.providers.isbn
faker.providers.job
faker.providers.lorem
```

https://pypi.org/project/Faker/

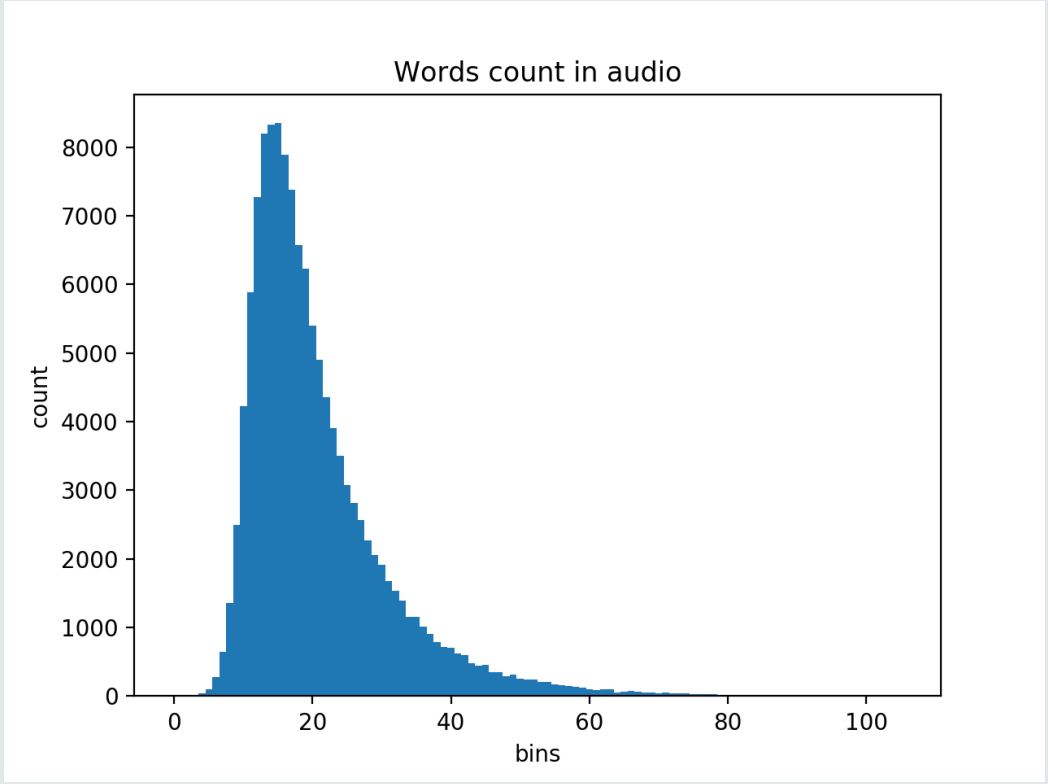
# Unit tests

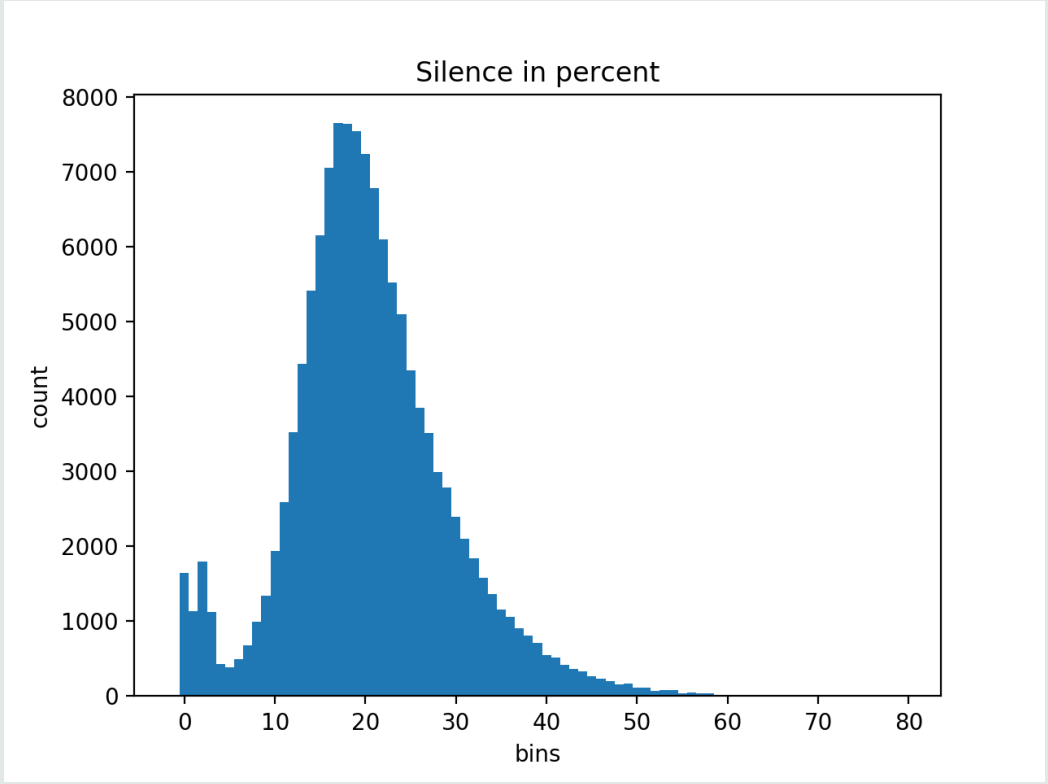
**Test positive and negative results**

**Functional orientated code is easier to test**

```python
from dataclasses import dataclass

@dataclass
class Human:
    age: int

    def is_adult(self) -> bool:
        return self.age >= 18

def test_human() -> None:
    assert Human(16).is_adult() is False
    assert Human(18).is_adult() is True
    assert Human(100).is_adult() is True
```

# Training data




**HUI-Audio-Corpus-German: https://opendata.iisys.de/**

# Summary

- Every project needs different kind of tests
- Evaluate needs with risk analysis or quality attributes

**Don't just test anything, test the right things.**

PASCAL PUCHTLER
Software Entwicklung + Beratung

# Pascal Puchtler

- Freelancer since 2023
- First programming experience 18 years ago
- Focused on testing with python
- 3 publications
- software architecture, databases, clean code, AI, Scrum, ..
- Python, JS/TS, C#, C++, …

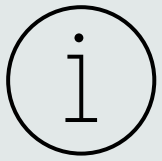
https://www.linkedin.com/in/pascal-puchtler/

puchtler.freiberufler@gmail.com

PASCAL PUCHTLER
Software Entwicklung + Beratung

# Appendix

# Code quality - Formatter



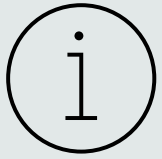
A tool that automatically adjusts the style and format of source code.

- **Consistency:** Ensures uniform formatting throughout the codebase.
- **Readability:** Clear structure and easy readability.
- **Time-saving:** Reduces manual effort.
- **Collaboration:** Facilitates teamwork.
- **Best Practices:** Supports adherence to standards.

**Tools**

- Black
- YAPF
- autopep8
- isort

# Code quality - Linter



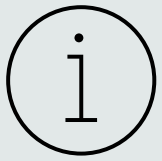
A tool that analyzes source code to flag programming errors, bugs, stylistic errors, and suspicious constructs.

- **Error Detection:** Identifies potential bugs and errors early in the development process.
- **Code Quality:** Promotes clean, maintainable, and bug-free code.
- **Consistency:** Enforces coding standards and best practices.
- **Learning:** Provides feedback to developers, helping them improve their coding skills.

**Tools**

- PyLint
- flake8
- pyflakes

# Code quality - Architectural analysis



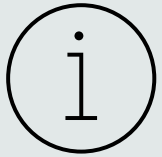
The process of examining the structure, design, and interactions of software components to evaluate its architectural qualities.

- **Identifying Weaknesses:** Uncovers architectural flaws, such as tight coupling or excessive dependencies.
- **Scalability:** Assesses the system's ability to handle growth and changes in requirements.
- **Maintainability:** Evaluates the ease of maintaining and evolving the software over time.

**Tools**

- SonarQube
- CodeScene
- CAST Highlight
- Structure101

# Code quality - Typing



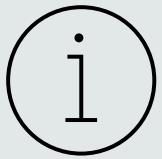
The process of annotating variables, function parameters, and return types with type hints to enable static type checking in Python.

- **Early Error Detection:** Identifies type-related errors at compile time rather than runtime.
- **Documentation:** Improves code readability and provides documentation on expected types.
- **IDE Support:** Enables IDEs to offer better code completion and type inference.
- **Code Quality:** Enhances code maintainability and reduces bugs related to type mismatches.

**Tools**

- Mypy
- Pyright

# Customer - Test of documentation



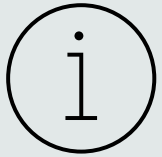
The process of evaluating the quality, accuracy, and usability of software documentation, such as user manuals, installation guides, and API references.

- **Clarity:** Ensures that user manuals are clear and easy to understand for users.
- **Completeness:** Verifies that all necessary information, including installation procedures and usage instructions, is included.
- **Accuracy:** Confirms that the information provided in the documentation is correct and up-to-date.
- **Usability:** Assesses the usability of documentation, including navigation and organization, to enhance user experience.
- **Compliance:** Ensures that documentation follows relevant standards and guidelines.

**Tools**

- Sphinx:
- MkDocs
- Read the Docs

# Customer - A/B testing



A statistical method used to compare two versions of a webpage or application to determine which one performs better in terms of a given metric.

**Data-Driven Decisions:** Allows for empirical validation of design choices and feature implementations.

**Improved User Experience:** Enables optimization of user experience based on real user behavior and preferences.

**Increased Conversions:** Identifies elements or features that lead to higher conversion rates, such as sign-ups or purchases.

**Reduced Risk:** Minimizes the risk associated with deploying changes by testing them on a smaller subset of users first.

**Continuous Improvement:** Facilitates iterative improvements by testing and iterating on various design and feature alternatives.

**Tools**

- Optimizely
- VWO (Visual Website Optimizer)

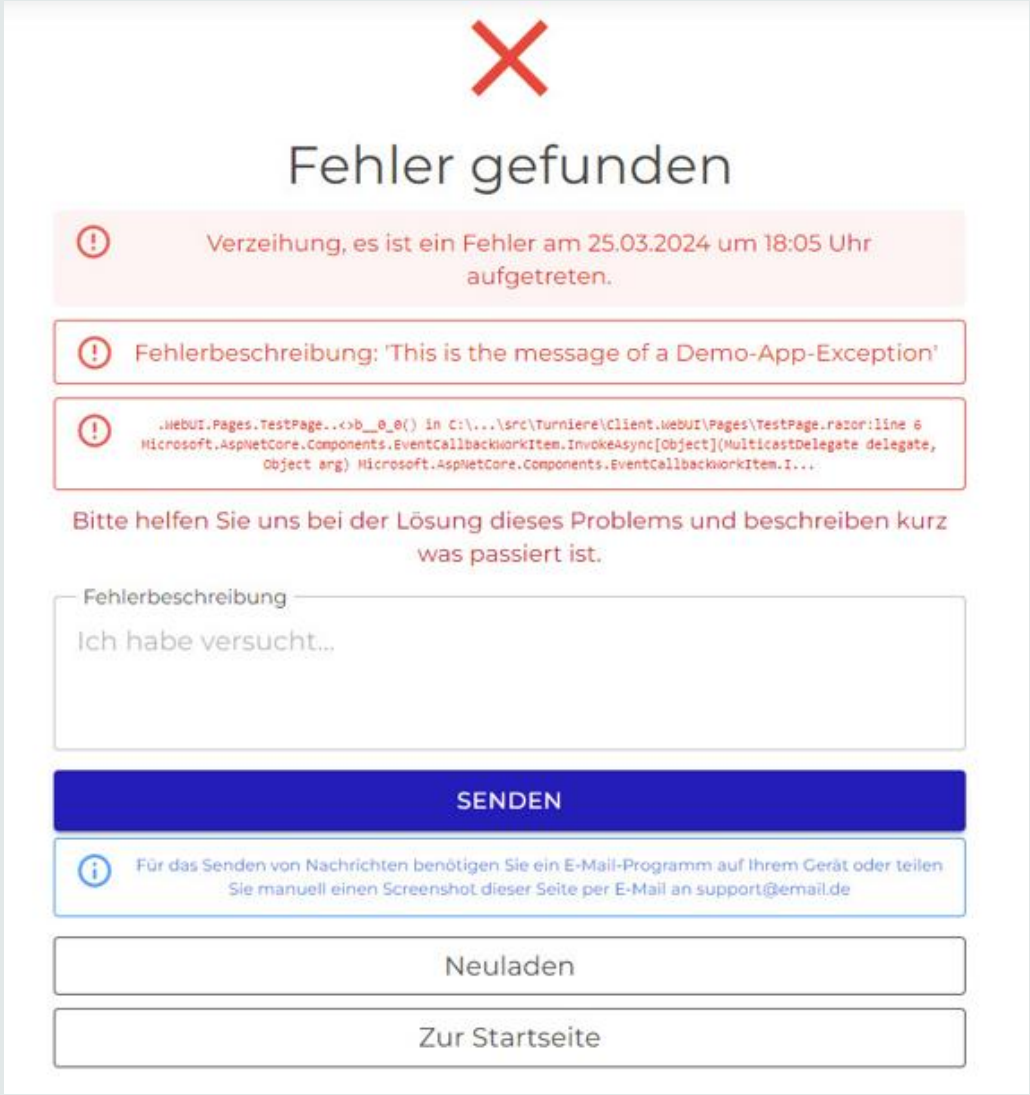
# Customer - Crash reports

Crash reports are generated when an application encounters a critical error or "crashes" during execution. These reports contain detailed information about the error, including the stack trace, device information, and application state at the time of the crash.

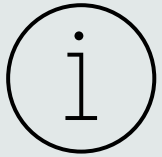
**Improved User Experience:** Allows customers to report issues easily and provides developers with valuable insights to quickly resolve them.

**Faster Resolutions:** Enables developers to reproduce and diagnose issues more efficiently, leading to faster resolutions and fewer disruptions for customers.

**Feedback Loop:** Establishes a feedback loop between customers and developers, fostering a collaborative approach to software improvement.



# Customer - Crash reports



Color blindness simulation in software testing involves evaluating the accessibility and usability of digital interfaces for individuals with color vision deficiencies. This can be achieved by using tools or settings to simulate how the interface appears to users with various types of color blindness.

**Accessibility Testing:** Ensures that digital interfaces are usable and understandable by individuals with color vision deficiencies.

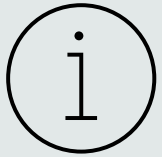
**Compliance:** Helps software developers comply with accessibility standards and guidelines, such as the Web Content Accessibility Guidelines (WCAG).

**User Experience:** Improves the overall user experience by addressing potential issues related to color perception and contrast.

**Tools**

- Browser

# Customer - Click route analysis



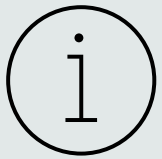
Click route analysis is a method used to track and analyze the sequence of user interactions, such as clicks, taps, or swipes, within a digital interface. It provides insights into how users navigate through the interface and interact with its various elements.

**User Behavior Understanding:** Gain insights into how users navigate through the interface and identify common user paths.

**Usability Evaluation**: Evaluate the effectiveness and efficiency of the user interface design based on user interaction patterns.

**Optimization Opportunities:** Identify areas for improvement, such as optimizing navigation flows or enhancing user engagement with specific features.

# Customer - Slow/interrupted connection



Testing for slow or interrupted connections involves simulating network conditions with low bandwidth, high latency, or intermittent connectivity to assess the performance and resilience of software applications under adverse network conditions.

**Real-World Simulation:** Replicates real-world scenarios where users may experience slow or unreliable network connections, such as in rural areas or on mobile networks.

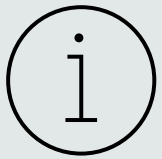
**Performance Evaluation:** Evaluates how well the application performs under adverse network conditions, including loading times and responsiveness.

**Resilience Testing:** Tests the application's ability to gracefully handle network interruptions, such as retries, error handling, and offline support.

**Tools**

- Poorconn (pip package)
- Charles Proxy
- Google Chrome DevTools

# Customer – Feedback function



A feedback button is a user interface element typically placed within an application or on a website that allows users to provide feedback, suggestions, or report issues directly to the developers or support team.

**User Engagement:** Encourages users to actively participate in improving the product by providing their input and suggestions.

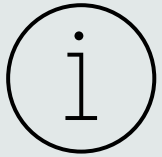
**Issue Reporting:** Provides a convenient way for users to report bugs, errors, or usability issues encountered during their use of the application.

**Continuous Improvement:** Facilitates a feedback loop between users and developers, enabling ongoing refinement and enhancement of the product based on user feedback.

**Tools**

- Email
- Request

# Development process - Ticket review



Ticket review is a process where software development teams examine and prioritize tickets or issues reported by users, stakeholders, or automated systems. It involves analyzing the details of each ticket, assessing its severity and impact, and determining the appropriate course of action.

**Issue Triage:** Prioritize tickets based on their severity, impact on users, and urgency for resolution.

**Resource Allocation:** Allocate resources effectively by assigning tickets to appropriate team members or teams based on their expertise and availability.

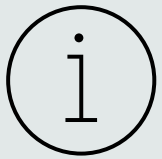
**Quality Assurance:** Ensure that reported issues are thoroughly reviewed and addressed to maintain product quality and user satisfaction.

**Communication:** Facilitate communication between stakeholders, users,

**Tools**

- Jira
- Trello
- GitHub Issues

# Development process - Code review



Code review is a systematic examination of code by one or more developers to identify and address issues, improve code quality, and ensure adherence to coding standards and best practices.

**Quality Assurance:** Identify bugs, logic errors, and potential security vulnerabilities before they impact users.

**Knowledge Sharing:** Facilitate knowledge sharing and learning among team members by providing feedback and suggestions for improvement.

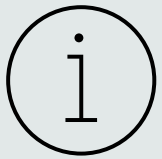
**Code Consistency:** Ensure that code is consistent with established coding standards and follows best practices.

**Continuous Improvement:** Promote a culture of continuous improvement by identifying areas for optimization and enhancement.

**Tools**

- GitHub Pull Requests
- Bitbucket
- GitLab Merge Requests

# Development process - Feature review



Feature review is a process of examining newly developed or modified features in a software application to ensure they meet requirements, adhere to design specifications, and provide value to users.

**Quality Assurance:** Verify that new features function as intended and meet quality standards before release.

**User Satisfaction:** Ensure that features align with user needs and expectations, enhancing overall user experience.

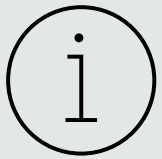
**Feedback Collection:** Gather feedback from stakeholders and users to identify areas for improvement and prioritize future development efforts.

**Risk Management:** Mitigate the risk of introducing bugs or regressions by thoroughly reviewing new features prior to deployment.

**Tools**

- Intern review
- Customer review
- UserTesting

# Development process - Test environments



Deployment in different environments involves the process of releasing software to various stages or environments e.g. development, testing, staging or customer environments.

**Developer Efficiency:** Streamlined testing processes enable developers to iterate quickly and confidently.

**Customer Satisfaction:** Identifying and fixing issues before deployment improves the user experience.

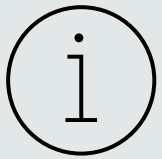
**Risk Mitigation:** Testing in realistic environments reduces the risk of bugs and failures in production.

**Agile Development:** Enables continuous integration and delivery practices for rapid software delivery

**Tools**

- Docker
- Jenkins
- GitHub actions

# Functional - Unit tests



Unit tests are automated tests designed to validate the functionality of individual units or components of a software application in isolation. These tests verify that each unit of code behaves as expected under various input conditions.

**Verification:** Ensure that each unit of code functions correctly and produces the expected output.

**Regression Prevention:** Detect regressions or unintended side effects resulting from code changes.

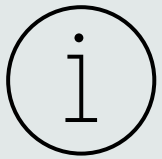
**Documentation:** Serve as executable documentation, demonstrating how units of code should be used and behave.

**Design Facilitation:** Encourage modular and loosely-coupled code design by promoting the development of testable units

**Tools**

- Pytest
- Unittest
- Mockito

# Functional - Mutation tests



Mutation testing is a technique used to evaluate the effectiveness of a test suite by introducing small changes or mutations into the source code and observing if the tests can detect these changes.

**Test Suite Evaluation:** Assess the quality and effectiveness of the test suite by measuring its ability to detect changes in the code.

**Fault Detection:** Identify weaknesses in the test suite's ability to catch potential bugs or errors in the code.

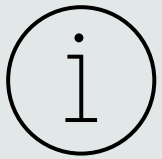
**Code Quality Improvement:** Encourage developers to write more robust and comprehensive tests that cover edge cases and corner cases.

**Benchmarking:** Provide a quantitative measure of the test suite's adequacy and reliability compared to industry standards or best practices.

**Tools**

- mutmut
- Stryker (not for python)

# Functional - Integration tests



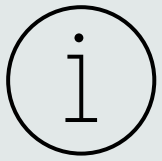
Integration tests are automated tests designed to verify the interactions between different components or modules of a software application. These tests validate that integrated units of code work together as expected and produce the desired output.

**Integration Verification:** Ensure that individual components or modules integrate correctly and function together as intended.

**Tools**

- Pytest
- Unittest
- Mockito

# Functional - System tests



System tests are end-to-end tests designed to validate the entire software system as a whole, including its functionality, performance, and behavior in real-world scenarios. These tests verify that the system meets its specified requirements and behaves correctly from the user's perspective.

**Functional Verification:** Validate that the software system functions correctly and meets its intended requirements and specifications.

**User Experience Testing:** Evaluate the system's usability, accessibility, and overall user experience in a realistic environment.

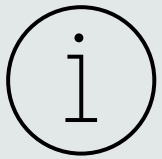
**Performance Evaluation:** Assess the system's performance, scalability, and reliability under normal and peak load conditions.

**Stakeholder Validation:** Demonstrate that the system meets the expectations and requirements of stakeholders, including end-users, customers, and business owners.

**Tools**

- Pytest
- Unittest
- Robot Framework
- Selenium

# Functional - System tests



Full process testing is a comprehensive testing approach that validates the entire end-to-end process of a customer journey across interconnected software systems. It involves simulating real-world scenarios where users interact with multiple systems sequentially, ensuring seamless integration and functionality.

**End-to-End Validation:** Verify the entire customer journey from start to finish, including interactions with all connected systems and services.

**Workflow Testing:** Test the complete end-to-end workflow, spanning multiple systems and components, to ensure a smooth and cohesive user experience.

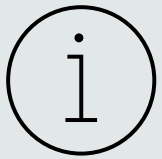
**Interoperability:** Validate the interoperability and integration between different software systems, ensuring they work together seamlessly.

**Business Process Validation:** Ensure that business processes and transactions are executed correctly across all interconnected systems, meeting business requirements and objectives.

**Tools**

- Pytest
- Unittest
- Robot Framework
- Selenium

# Resources - Performance tests



Response time measures the duration between sending a request to a system and receiving a response. In performance testing, it is a critical metric for assessing the speed and responsiveness of a software application.

**User Experience:** Ensure that critical operations or transactions respond within acceptable time limits to meet user expectations.

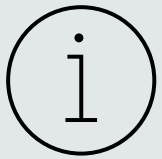
**Performance Benchmarking:** Establish performance baselines and benchmarks for critical system functions or features.

**Performance Optimization:** Identify performance bottlenecks and areas for improvement to enhance overall system performance and responsiveness.

**Tools**

- Gatling
- Locust

# Resources - Load test



Load testing is a type of testing that evaluates the behaviour of a system under expected and peak load conditions. It assesses how the system performs in terms of speed, throughput, and resource usage when subjected to a high volume of concurrent users or transactions.

**Throughput Evaluation:** Measure the system's ability to process transactions within a specified time frame, typically expressed as transactions per second (TPS).

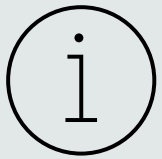
**Scalability Assessment:** Determine how the system handles increasing transaction volumes and concurrent user interactions.

**Performance Optimization:** Optimize system resources and configurations to ensure optimal throughput and response times under heavy load conditions.

**Tools**

- Gatling
- Locust

# Resources - Stress test



Stress tests are a type of testing that evaluates the behaviour of a system under extreme load conditions beyond its normal capacity. These tests assess how the system behaves when subjected to high levels of stress, such as increased transaction volumes, concurrent users, or resource utilization.

**Failure Point Identification:** Determine the system's breaking point or failure mode under extreme load conditions, such as peak transaction volumes or user concurrency.

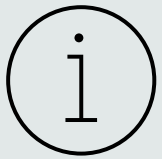
**Resilience Assessment:** Evaluate the system's resilience and stability under stress, ensuring it can recover gracefully from failure and continue to operate.

**Performance Degradation Analysis:** Identify performance degradation patterns and bottlenecks that occur under stress, providing insights into system limitations and weaknesses.

**Tools**

- Gatling
- Locust

# Resources - Stress test



Stress tests are a type of testing that evaluates the behaviour of a system under extreme load conditions beyond its normal capacity. These tests assess how the system behaves when subjected to high levels of stress, such as increased transaction volumes, concurrent users, or resource utilization.

**Failure Point Identification:** Determine the system's breaking point or failure mode under extreme load conditions, such as peak transaction volumes or user concurrency.

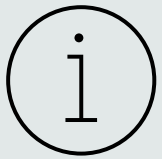
**Resilience Assessment:** Evaluate the system's resilience and stability under stress, ensuring it can recover gracefully from failure and continue to operate.

**Performance Degradation Analysis:** Identify performance degradation patterns and bottlenecks that occur under stress, providing insights into system limitations and weaknesses.

**Tools**

- Gatling
- Locust

# Resources - Volume test



Volume tests are a type of performance testing that evaluates how a system handles large amounts of data or transactions over an extended period. These tests assess the system's scalability, resource utilization, and performance when subjected to high volumes of data or transactions.

**Scalability Assessment:** Determine how the system scales with increasing data volumes or transaction loads, ensuring it can handle future growth.

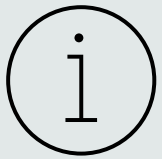
**Resource Utilization:** Evaluate the system's resource consumption, such as memory, CPU, and storage, under high-volume conditions.

**Data Management:** Assess the system's ability to manage and process large datasets efficiently without performance degradation or data loss.

**Tools**

- Gatling
- Locust

# Security - Penetration test



Penetration tests, also known as pen tests, are a type of security testing that evaluates the security of a system by simulating real-world attacks. These tests identify vulnerabilities, weaknesses, and potential security risks that could be exploited by malicious actors.

**Vulnerability Identification:** Identify security vulnerabilities and weaknesses in the system, including software flaws, misconfigurations, and insecure practices.

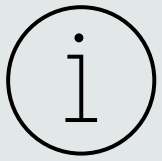
**Risk Assessment:** Assess the potential impact and likelihood of security breaches, data breaches, and unauthorized access to sensitive information.

**Security Assurance:** Provide assurance to stakeholders that the system is secure and resilient against potential cyber threats and attacks.

**Tools**

- Metasploit
- Burp Suite

# Security - Static security analysis



Static security analysis, also known as static code analysis or static application security testing (SAST), is a type of security testing that analyzes source code or application binaries without executing them. It identifies security vulnerabilities, coding errors, and potential weaknesses by analyzing the code structure, syntax, and patterns.

**Vulnerability Detection:** Identify security vulnerabilities and weaknesses in the codebase, such as injection flaws, buffer overflows, and insecure coding practices.

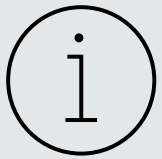
**Code Quality Improvement:** Improve code quality and reliability by identifying coding errors, syntax issues, and potential bugs early in the development process.

**Security Assurance:** Provide assurance to stakeholders that the application is developed with security best practices and does not contain known security vulnerabilities.

**Tools**

- Bandit
- Pylint
- SonarQube

# Stability - Long time stability measurement



Long-time stability measurement is a type of esting that evaluates the stability and reliability of a system over an extended period. It assesses how the system behaves under sustained load conditions, ensuring consistent performance and availability over time.

**Stability Assessment:** Evaluate the system's stability and reliability over an extended duration to identify potential issues such as memory leaks, resource exhaustion, and performance degradation.

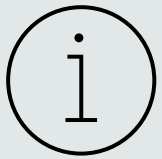
**Endurance Testing:** Measure the system's ability to maintain performance and availability under sustained load conditions, simulating real-world usage scenarios.

**Regression Detection:** Detect and address performance regressions or degradation that may occur over time due to software updates, configuration changes, or increased usage.

**Tools**

- Gatling
- Locust

# Stability - Hardware fault test



Software resilience to hardware failures is a type of testing that evaluates how software systems respond and recover from hardware failures, such as virtual machine loss in a microservices architecture or power-off scenarios. It assesses the system's ability to maintain functionality, availability, and data integrity despite hardware failures.

**Failure Recovery:** Evaluate how software systems handle hardware failures, such as the loss of virtual machines or power-off scenarios and recover gracefully to maintain continuity of service.

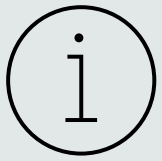
**Fault Tolerance:** Assess the system's ability to tolerate hardware failures without service disruption or data loss, ensuring uninterrupted operation in the event of hardware failures.

**Data Integrity:** Ensure that critical data remains intact and consistent despite hardware failures or malfunctions, minimizing the risk of data corruption or loss.

**Tools**

- The Netflix Simian Army
- Faulty Disk

# Stability - Compatibility test



Compatibility tests are a type of testing that evaluates how software performs across different environments, platforms, devices, and configurations. These tests ensure that the software is compatible with various systems and components, providing a consistent user experience across different contexts.

**Cross-Platform Compatibility:** Assess the software's compatibility with different operating systems, web browsers, and devices to ensure consistent performance and functionality.

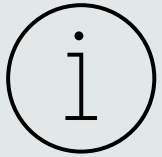
**Interoperability Testing:** Verify that the software can interact seamlessly with other systems, applications, and services, ensuring compatibility with third-party components and APIs.

**Configuration Testing:** Test the software under various configurations, settings, and environments to identify compatibility issues and ensure optimal performance and usability.

**Tools**

- BrowserStack
- Selenium
- Appium

# *Stability - Install/uninstall test*

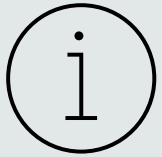


Install/uninstall tests assess software installation and uninstallation processes, ensuring smooth, error-free installations and complete removal without leaving traces or causing system instability.

**Uninstallation Verification:** Ensure that the software uninstallation process removes all components, files, and registry entries associated with the application, leaving the system clean and stable.

**User Experience Enhancement:** Improve the overall user experience by providing seamless installation and uninstallation processes that meet user expectations and requirements.

# Stability - Backup and recovery test



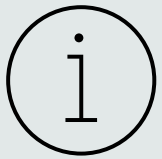
Backup and recovery tests are a type of testing that evaluates the effectiveness of backup and recovery processes for software applications. These tests ensure that data can be reliably backed up and restored in the event of data loss, corruption, or system failure, minimizing downtime and data loss.

**Data Protection:** Ensure the integrity and availability of critical data by testing backup and recovery processes to mitigate the risk of data loss due to hardware failures, human errors, or malicious attacks.

**Business Continuity:** Verify that backup and recovery procedures are in place to facilitate rapid recovery and restore operations in the event of system downtime or disaster scenarios.

**Compliance Requirements:** Ensure compliance with regulatory requirements and industry standards for data protection, retention, and recovery, reducing the risk of legal and financial consequences.

# Stability - Random data test



Random data tests are a type of testing that evaluates how software handles random or unexpected input data. These tests assess the robustness and reliability of the software under unpredictable conditions, ensuring that it can handle a wide range of input scenarios without errors or crashes.

**Error Handling:** Assess the software's ability to handle unexpected or invalid input data gracefully, preventing crashes, errors, or security vulnerabilities.

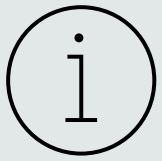
**Robustness Testing:** Verify that the software can withstand random or malicious input data without compromising system stability or security.

**Boundary Testing:** Test the software's boundaries and edge cases by providing random or extreme input data to uncover potential vulnerabilities or weaknesses.

**Tools**

- Faker
- Hypothesis
- mock-data-generator

# Stability - Random click test



Random click tests are a type of testing that evaluates the responsiveness and robustness of user interfaces by simulating random or unexpected user interactions, such as clicks, taps, or gestures. These tests help identify potential usability issues, interface glitches, or functional inconsistencies in software applications.

**Usability Evaluation:** Assess the responsiveness and usability of user interfaces by simulating random or unexpected user interactions and observing the software's response.

**Error Detection:** Identify interface glitches, functional inconsistencies, or usability issues that may arise from random or unexpected user interactions.

**Robustness Testing: Test** the software's ability to handle user input gracefully and recover from unexpected interactions without crashing or freezing.

**Tools**

- Selenium
- Appium