Basics of Non-functional Testing

PA1417 Lecture Unit 6

Organizational

- The submission dates for resubmission attempts 1 and 2 of lab unit 1 and 2 are now available on Canvas
- Julian will be unavailable on Thursday and Friday this week (2nd & 3rd of May)
    - For questions about assignment 5 in lab 2, use the seminar slot as usual. The seminar will be given by Eriks this week.
    - For questions about the assignments 1-4 and technical aspects, please contact Julian via Mail or Discord until latest 1st of May.

Goals

Part I: Reiteration of test implementation principles

- Test case independence
- Exploratory testing

Part II: non-functional testing

- You will learn how to approach testing quality of the system
- How to use test design technique
- Determine good-enough threshold and interpret test results

Part I: Test Implementation Principles

Boxes like these contain **principles of test implementation**, which are important to consider when writing your own test code starting from assignment 2.

The principles of test case design

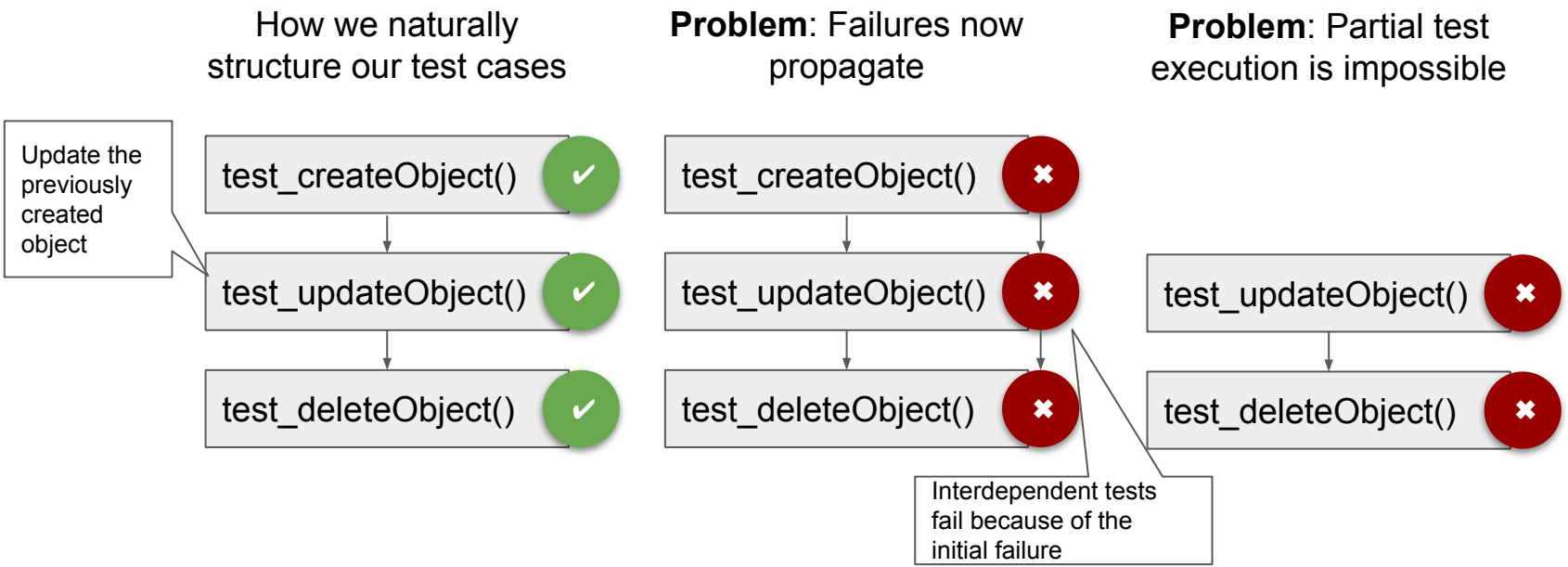
How can you be sure that your tests fail because they detect a failure in the system and not because the test code is wrong?

Sometimes, tests randomly fail because of some inadvertent dependency that is hard to discover. It is called *flaky tests* and should be minimized.

- There sadly is no way to ensure this deterministically
- Rather, we must stick to heuristics of good test case design

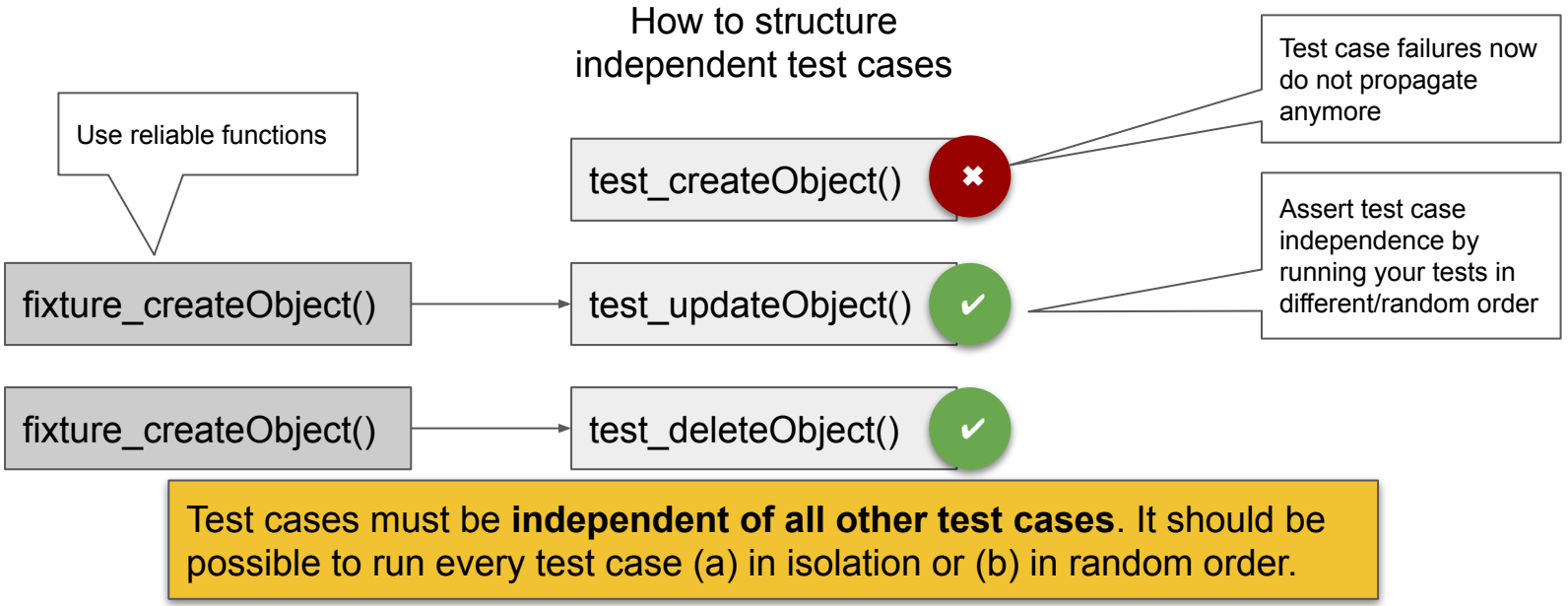
Independence of test cases

Test cases have to be independent of each other, i.e., the success/failure of one test case must be independent of the success/failure of other test cases.



Independence of test cases

In order to ensure, that each test case can be executed independently, implement each test case with a reliable setup and teardown.



Other types of dependencies

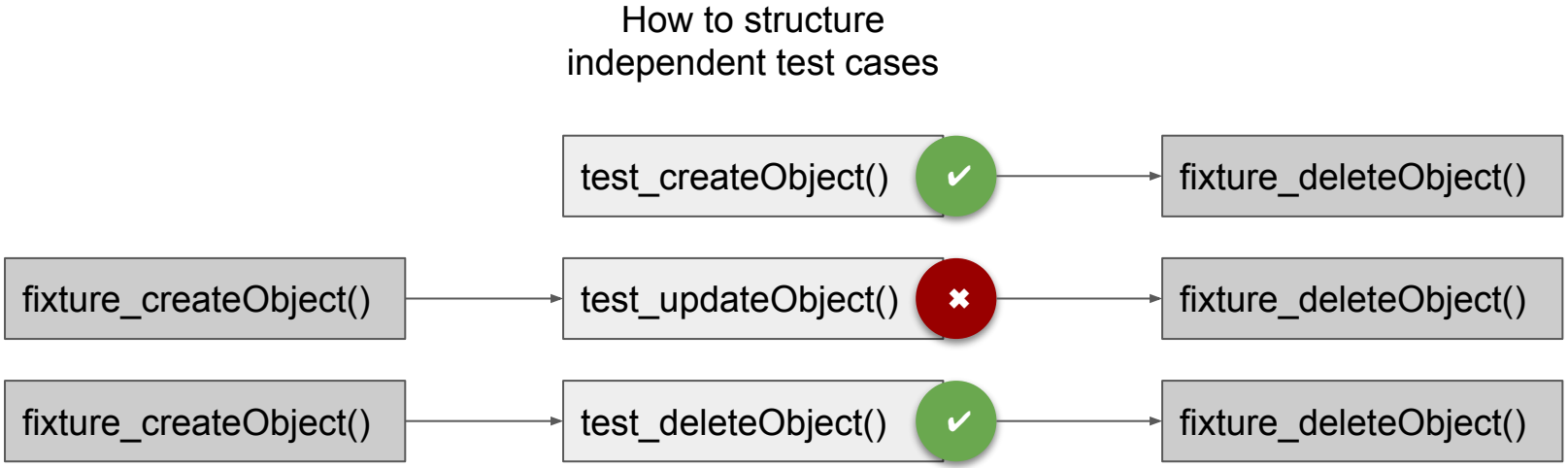
Interdependence is only one aspect of dependency that we need to control. Test cases further ought to be

- independent from the environment they are executed in
- independent from the system they are executed on
- …

Accommodating these types of independence may require more elaborate setups (like a dockerized test environment).

Other types of dependencies

In the scope of this course it is sufficient to ensure independence by making tests (1) not interdependent and (2) independent of the system state via setup and teardown.



Beyond the written oracle

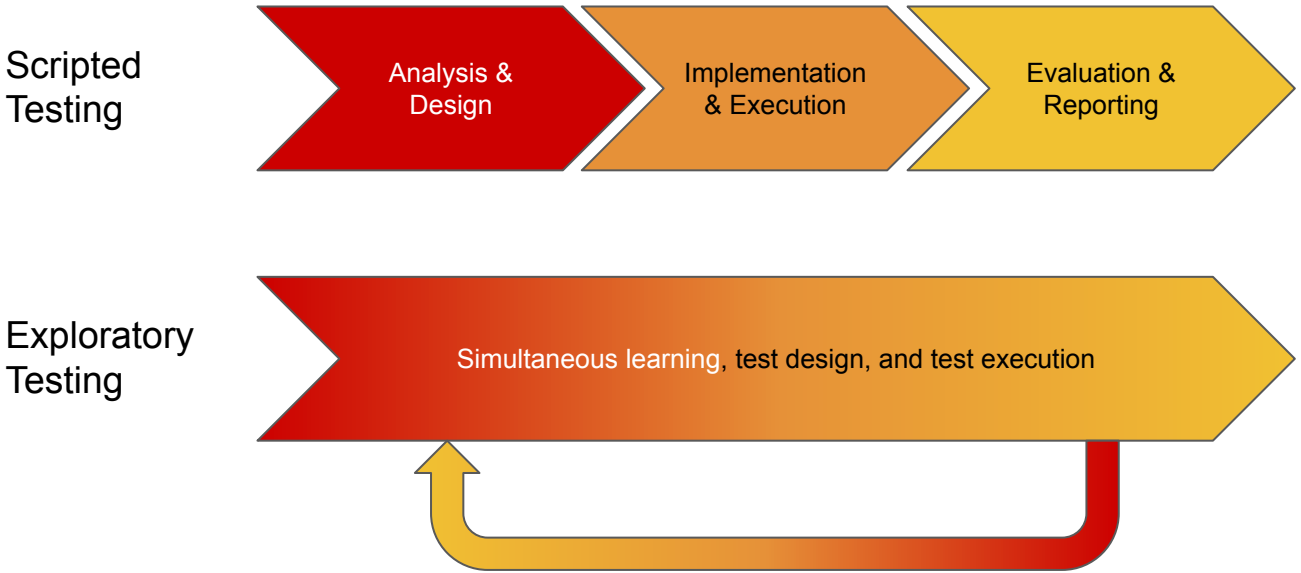
We are used to derive test cases rigorously from a written oracle (requirements specification, code documentation, …), but

- what if the specification is incomplete or does not exist?
- what if the specification is wrong?
- what if there is not enough time for rigorous test case development?

The other end of the spectrum - where **scripted testing** represents one extreme - is **exploratory testing**.

Exploratory testing

Exploratory testing is simultaneous learning, test design, and test execution.[1]



- create a test charter
- only limited documentation
- time-boxed sessions
- adapt to findings

[1]Ryber, T., & Meddings, P. (2007). *Essential software test design*. Fearless consulting., chapter 6.1

Exploratory testing

In exploratory testing, test cases are derived from experience, error-guessing, or checklists.

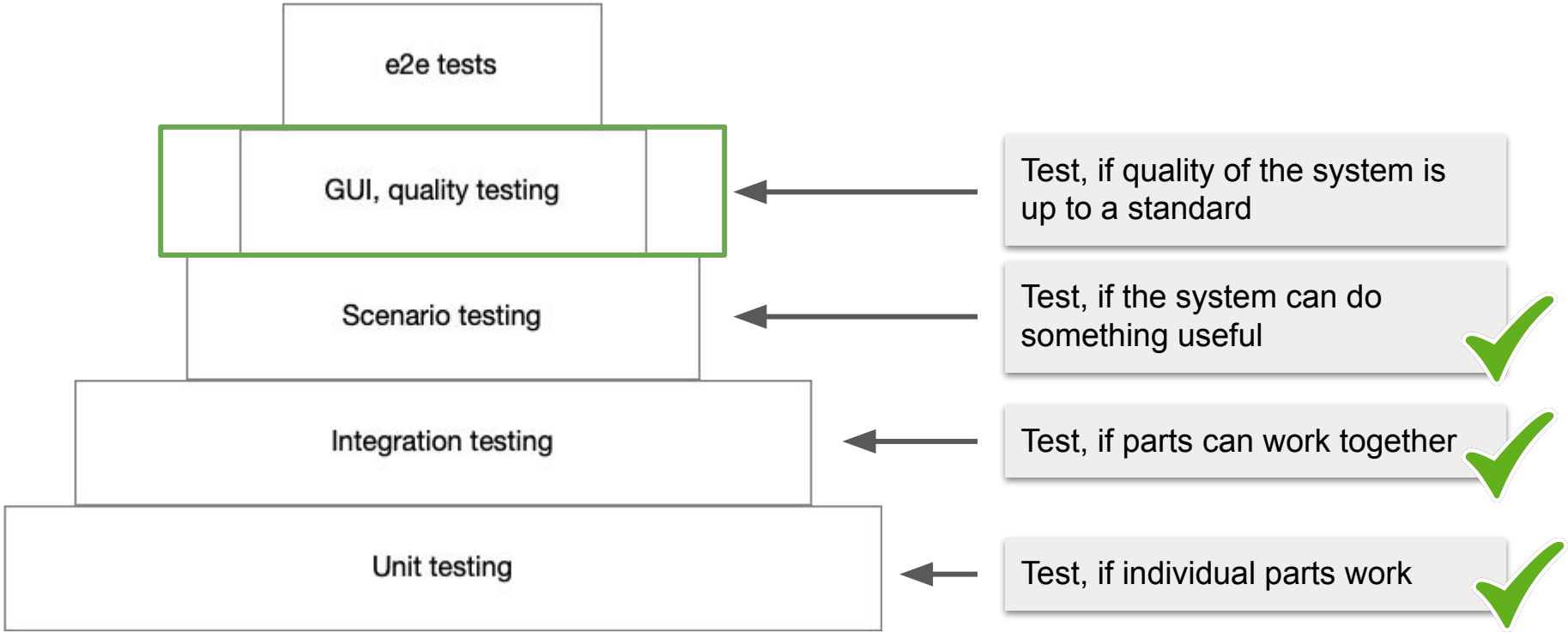
Quiz: Assume you need to test a method which checks for “valid” names of bands/songs/albums (e.g., for a music search engine), but the written oracle does not explain what “valid” means. Come up with values to test based on experience.

https://dustri.org/b/horrible-edge-cases-to-consider-when-dealing-with-music.html

Break

Part II: Non-functional testing

Test pyramid



Functional vs non-functional?

Functionality of the system is about **what** the system does, e.g.

- A user can make a hotel reservation

Non-functional is about **how** the system does it and **how** it is built, e.g

- The system should be easy to use and accessible
- The system should be secure
- Updating the system should not cause interruption of services
- The system should be scalable to accommodate a growing number of users

Software quality attributes

- Wikipedia lists 80+ *common* quality attributes, many of them can be broken down further, e.g. usability has 100+ different sub-attributes
- No need to memorize all of them, some are used often, some are rare:
  - Usability, user experience, performance, reliability, availability, robustness are common
  - Others may be less common (but still important)
- Interpretation of a quality attribute depends on the system and its context of use, e.g.
  - Usability of a hotel booking app (minimizes time/number of steps to make a booking)
  - Usability of a game (minimizes the learning curve for GUI, rules, items, and game mechanics)
  - Usability of a climate control system in a car (minimizes the need for driver to adjust climate settings)

Quiz: Software quality attributes

Associate each of the following descriptions to the respective quality attributes:

1. The ability of a computer system to cope with errors during execution and cope with erroneous input.
2. The degree to which a software artifact (i.e. a software system, software module, requirements- or design document) supports testing in a given test context.
3. The characteristic of a product or system to work with other products or systems.
4. The database property which guarantees that transactions that have committed will survive permanently.
5. The property of a system to handle a growing amount of work by adding resources to the system.
6. The amount of useful work accomplished by a computer system.
7. The probability that an item will operate satisfactorily at any given point in time when used under stated conditions in an ideal support environment.

https://docs.google.com/forms/d/e/1FAIpQLSe-4ufp4HnRl5YD5GatqYmsNk_oJgtPkGK-lIJ2k-LX0f3A9w/viewform?usp=sf_link

Solution: Software quality attributes

Associate each of the following quality attributes to their description:

1. The ability of a computer system to cope with errors during execution and cope with erroneous input. (**reliability**)
2. The degree to which a software artifact (i.e. a software system, software module, requirements- or design document) supports testing in a given test context. (**testability**)
3. The characteristic of a product or system to work with other products or systems. (**interoperability**)
4. The database property which guarantees that transactions that have committed will survive permanently. (**durability**)
5. The property of a system to handle a growing amount of work by adding resources to the system. (**scalability**)
6. The amount of useful work accomplished by a computer system. (**performance**)
7. The probability that an item will operate satisfactorily at any given point in time when used under stated conditions in an ideal support environment. (**availability**)

Dynamic vs static qualities

- Testing dynamic qualities require to run the system
  - Testing performance of a system implies running the system
  - For example: Response time, resource utilization,

- Static qualities can be tested without executing any code of the SUT
  - Extensibility
  - Modularity
  - Code quality
  - Testability
  - DB normalization/data quality

Manual code analysis

A common practice is to ask someone else to review changes and provide comments before merging them to the main branch, i.e. PR reviews

- Ensures that the code follows certain standard
  (e.g. variables have descriptive names, something you cannot automate)
- Bonus: helps sharing knowledge and best practices in the team

A similar approach can be used to test the code for:

- **Extensibility** (how easy it is to add a new feature)
- **Understandability** (how easy is to understand the code)
- **Testability** (how easy it is to test the code)
- **Modularity**, complexity, fitness for particular purpose

makrei-p requested changes 25 days ago

View changes

makrei-p left a comment

Owner

Looks good in general, only a few remarks.

However, there are quite a few whitespace warnings by pylint (`pylint $(find . -type f -name "*.py" ! -path "**/.venv/**")`), you can fix most of them by using the reformat file feature in PyCharm !

`notebooks/configuration.json` Outdated

```
...  ...   @@ -0,0 +1,15 @@
        1  + {
        2  +   "paths": {
        3  +     "input_csv": "../../../../../../Documents/Promotion/Projects/[RED] Requirements Engineerir
```

makrei-p 25 days ago

Owner

Don't put sensitive data on git! Rather refer `"../data/data.csv"` and `"../data/codebook.csv"` respectively for the line below

JulianFrattini 24 days ago

Collaborator Author

True. I will change that.

Reply...

Code Review Guidelines

How to do a code review? Code reviews are often based on the reviewers experience in the domain and system.

To start, you can lean on established code review guidelines:

- Google: https://google.github.io/eng-practices/review/reviewer/
- Gitlab: https://docs.gitlab.com/ee/development/code_review.html
- and many more …

You can choose any of these code review guidelines (or none) as long as your code review fulfills its purpose, i.e., evaluates the code in respect to the chosen quality attribute(s).

Developing tests for non-functional features (qualities)

- Essential test activities and test case design technique still applies
  a. What are the requirements and what are you going to test?
  b. How selected quality attributes apply to your SUT?
  c. How much testing is needed?
  d. Test cases design
  e. Performing tests + use of exploratory testing + use of static test techniques
  f. Evaluating results
  g. Package and archive test artifacts

What are the requirements and test priorities?

- Determined by the system type, purpose, users, stakeholders.
- SRS (incl. user stories) are usually bad at specifying quality attributes in a testable way.
  - Qualities may be difficult to quantify
  - Perception of quality can be subjective (usability, attractiveness)
  - Difficult to determine a pass-fail threshold
  - Many quality attributes are interlinked
- Communication with stakeholders and experience in the domain help a lot to determine what's important

Performing tests

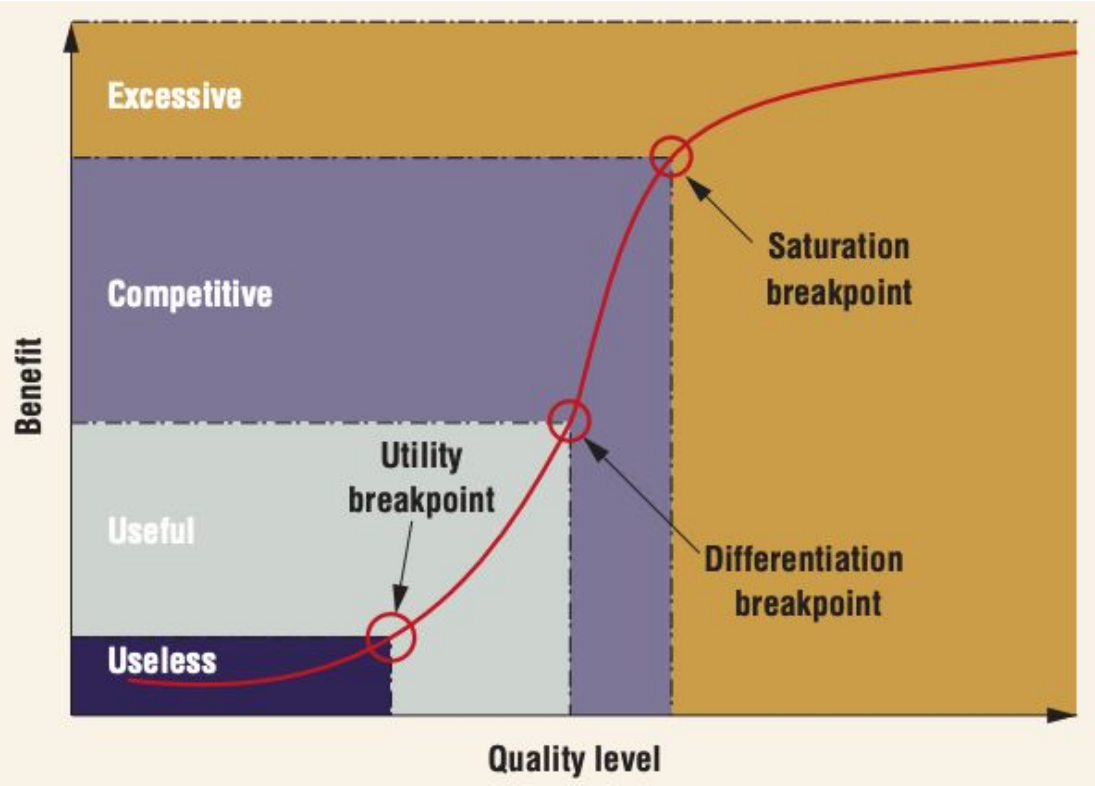
	Tool-based	Manual
Dynamic tests	- `pytest-benchmark` - toolkits for penetration testing - tools for load/performance/stability testing - Accessibility checkers -	- Exploratory testing - User testing
Static tests	- Code quality (SonarCube) - Code linters -	Reviews/analysis of software and its artifacts - Source code - Requirements - GUI mockups - Architecture/process designs - Log files

Evaluating results

- How do you know if the quality is up to a standard?
  - Ask the oracle
  - Apply expert judgement
  - Listen to user feedback
  - Establish a baseline for further reference, compare to an earlier baseline
  - Identify quality breakpoints

Consider including examples of some quality attributes, interpretations, testing strategies, quantification

Using breakpoints to determine “good-enough” quality



Regnell, B., Svensson, R. B., & Olsson, T. (2008). Supporting Roadmapping of Quality Requirements. IEEE Software, 25(2), 42–47. doi:10.1109/ms.2008.48

Wrap up

- Test design technique and test steps apply to non-functional tests as well
- **Interpret quality attributes for your system**
- Quality can be subjective, apply a mix of tools and manual techniques
- Establish a baseline first
- Involve stakeholders in deciding what a “good-enough” quality is
- It is easier to work with breakpoints than exact values

Questions?