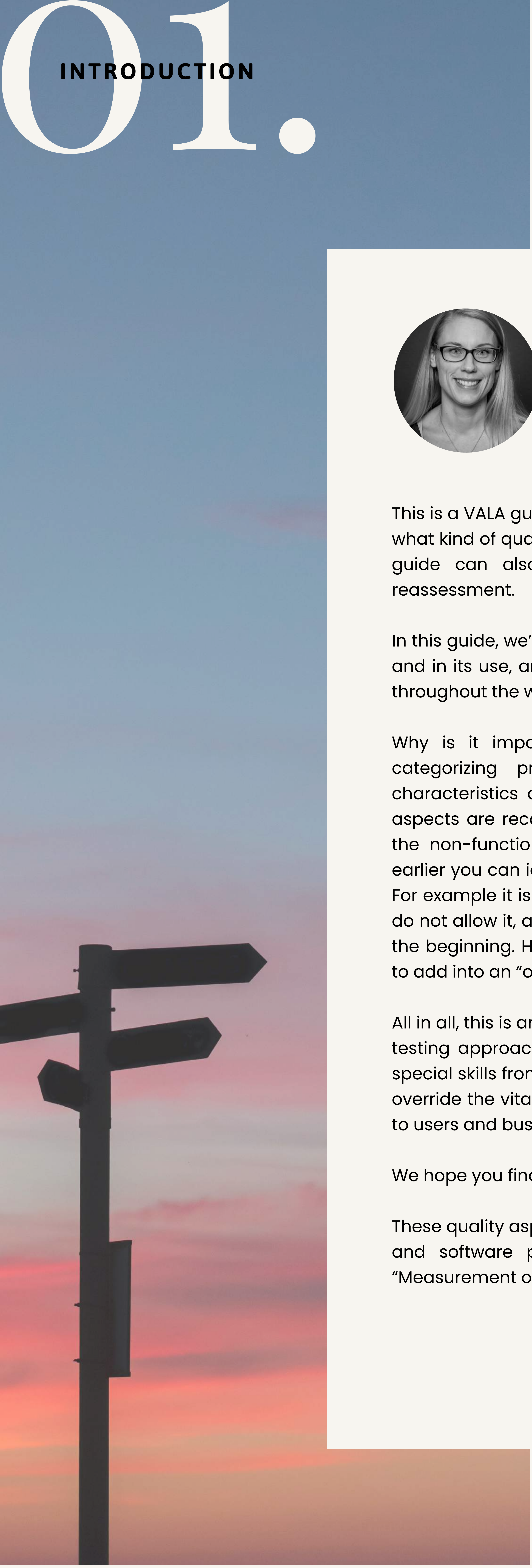
SOFTWARE QUALITY GUIDELINES

How to focus on what matters the most?

CONTENT

1. Introduction
2. Functional suitability
3. Reliability
4. Compatibility
5. Portability
6. Usability
7. Performance efficiency
8. Security
9. Maintainability
10. Conclusion

01. INTRODUCTION



Leena Saari

Co-Author

This is a VALA guide for quality leads and business representatives to determine what kind of quality aspects to focus on in a software development project. The guide can also be used as a basis for quality strategy planning or reassessment.

In this guide, we'll explain all the quality aspects there are in a software product and in its use, and how they can be taken into account not only in testing but throughout the whole software development lifecycle.

Why is it important to define these quality goals? ISO-standard based categorizing provides better confidence for covering various quality characteristics and for selecting the right ones to focus on. If these quality aspects are recognized and aligned with business needs from the beginning, the non-functional quality can be handled well by best professionals. The earlier you can identify a quality goal, the easier it is to build that in a software. For example it is difficult to reach high performance if the architectural choices do not allow it, and if development and testing haven't paid attention to it from the beginning. High performance, just like any other quality aspect is not easy to add into an "otherwise finished" software.

All in all, this is an excellent basis for creating quality strategy, selecting the right testing approach to fulfil these goals and preparing for testing that requires special skills from early on. Also the magnetism of building new features will not override the vital quality improvement work, if the quality goals and their value to users and business have been recognized.

We hope you find this guide useful in your quality work!

These quality aspects are based in SQuaRE-standards "Measurement of system and software product quality" (ISO/IEC 25023/2016) and also slightly in "Measurement of quality in use" (ISO/IEC 25022/2016).

Special thanks: *Special thanks to Esa Hakulinen, Johannes Heikkinen, Mari Hänninen, Samuli Paasimaa and Toni Roschier for proof-reading and adding their views to the text and all VALA Mhub test management community members for the bug examples!*

SOFTWARE QUALITY GUIDELINES

IN SHORT

FUNCTIONAL SUITABILITY

Functional suitability is a quality aspect which considers, whether the product suits "stated and implied" needs of the user, when used under specific circumstances.

CONSISTS OF

- Functional completeness
- Functional correctness
- Functional appropriateness

RELIABILITY

Reliability is a quality aspect, which tells if a system does what is specified and behaves in an expected manner in its environment.

CONSISTS OF

- Maturity
- Availability
- Fault tolerance
- Recoverability

COMPATIBILITY

From the user perspective compatibility appears as smooth usage, disregarding whether the usage flow runs on only one software or many.

CONSISTS OF

- Co-existence and
- Interoperability

PORTABILITY

Portable systems can be installed and used in different environments, or they can replace similar systems.

CONSISTS OF

- Adaptability
- Installability and
- Replaceability

USABILITY

When a user can use the software easily the way they want to and even feel satisfied while using it, the software can be considered usable.

CONSISTS OF

- Appropriateness and recognizability
- Learnability
- Operability and user error protection
- User interface aesthetics and
- Accessibility

PERFORMANCE EFFICIENCY

Performance efficiency measures software's Time-behavior and Resource utilisation.

CONSISTS OF

- Time-behavior and
- Resource utilisation

SECURITY

Secure software keeps all the data and information protected from unauthorized use.

CONSISTS OF

- Confidentiality
- Integrity and authenticity
- Non-repudiation, and
- Accountability.

MAINTAINABILITY

Maintainability is a quality aspect which explains how easy it is to update and change the system.

CONSISTS OF

- Modularity
- Reusability
- Analysability
- Modifiability
- Testability

1.1 High-quality software leads to a high-quality user experience

“The IEC and ISO encourage the development of products, systems and services that are safe, efficient and environmentally friendly.”

There are plenty of software quality definitions to pick from.

A few good examples of defining quality are:

1. A software that fits well into its purpose and to its user groups
2. Something that makes the users feel efficient, happy and productive
3. Optimized usage flows, avoiding unnecessary steps
4. Or something more precise, such as accessibility, where the software is designed, developed and tested in terms of making the software usable for people with disabilities.

These quality characteristics are explained in the following chapters based on Systems and software engineering - Systems and software quality requirements and evaluation (SQuaRE) - standards:

- Measurement of quality in use (ISO/IEC 25022/2016)
- Measurement of system and software product quality (ISO/IEC 25023/2016)

1.2 How to use Software Quality Aspects?

This guide concentrates on explaining product quality which leads to quality in use, and how product quality can be taken into account in quality work during the software development. Note, that this is not a template that needs to be filled, but a supporting document in your quality strategy work.

You can use this guide as a product owner in defining quality priorities, or as a quality manager aiming for a well justified quality strategy, or in any position where you need to plan and prioritize your work in software development. This guide is in best use when these quality aspects are considered in good co-operation with the business-oriented people, who may find it difficult to articulate their quality goals in such a detail that it can guideline the way of working. Additionally, when these quality goals are defined together, the importance of non-functional requirements can be appreciated more by stakeholders, and the time spent on for example performance testing is easily justified.

This quality goal definition is best to be done at the planning stage of a project, but it is also useful when planning the testing of a product and whenever there is a good spot to step up with quality.

Define software quality goals with the following steps:

IDENTIFY ON A HIGH LEVEL

what each product quality means in your system or product context.

PRIORITIZE

these quality aspects in running numbers together with the stakeholders of the system under development.

CHOOSE

from 3 to 5 aspects to improve in the next development phase in the current software development life-cycle.

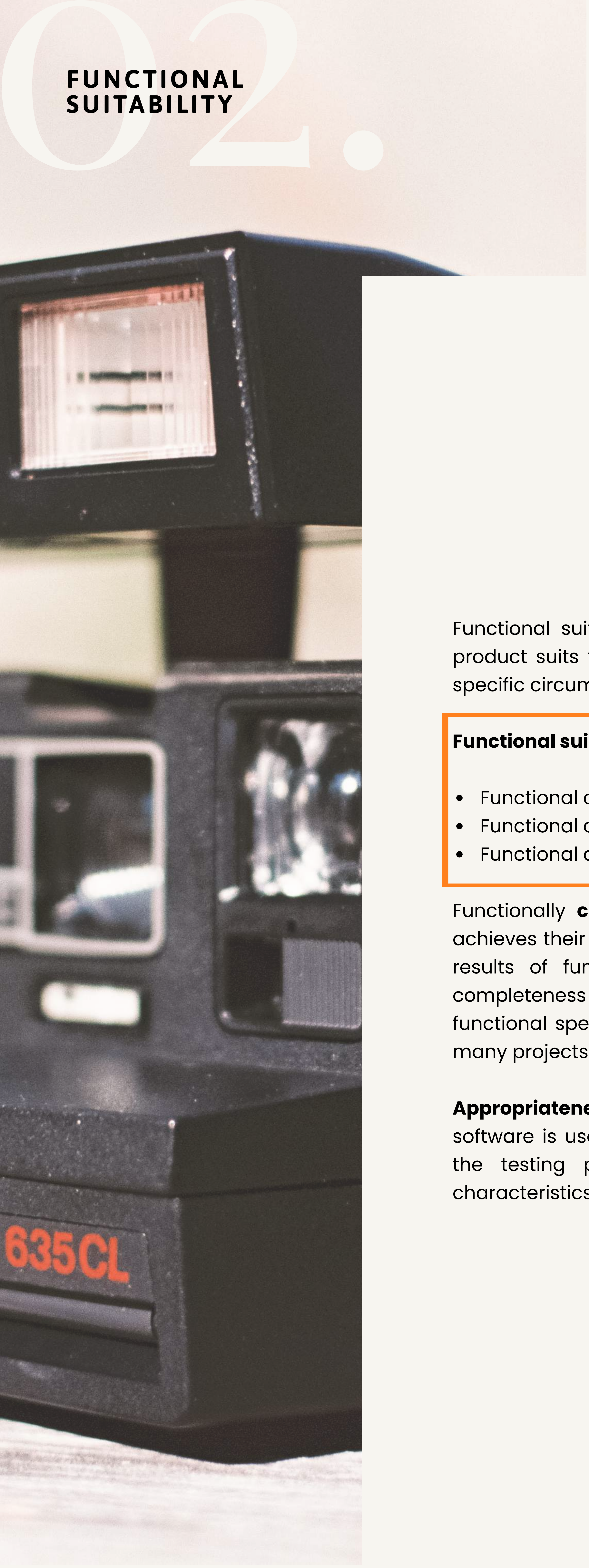
SPECIFY & EXPLAIN

how to improve the quality of the chosen aspects in your quality strategy document.

If you harness these quality aspects in the software development work, you are more likely to be working on things that matter the most and your quality-related decisions will be well justified based on business needs!

02.

FUNCTIONAL SUITABILITY



Functional suitability is a quality aspect which considers, whether the product suits "stated and implied" needs of the user, when used under specific circumstances.

Functional suitability consists of (ISO_IEC_25023/2016)

- Functional completeness
- Functional correctness
- Functional appropriateness.

Functionally **complete** software covers all related tasks and the user achieves their goals. Functional **correctness** refers to correct and precise results of functionality. In practice it seems to be that functional completeness and correctness are the best considered quality areas, as functional specifications and functional testing is the main approach in many projects.

Appropriateness refers to software suiting well in situations where the software is used. Functional appropriateness is many times forgotten in the testing phase, even though it is as important as the two characteristics above; can the software really fulfill the needs of its users?

2.1

How to use it in quality management?

The designed functionality is often well covered in either feature testing or end-to-end test automation. However, functional appropriateness is best covered by exploratory testing done by a tester in the development team with an open mind and the understanding of the users, or by acceptance testers with strong context knowledge.

Some of the basic questions to consider regarding functional suitability are:

IS

the feature ready, so that the user can complete the specified tasks with the software?

DO

different input combinations produce correct results? Does the software have all the important features?

DOES

the software function the way it was designed? The designed functionality is often well covered in either feature testing or test automation.

BY

whom and in what situation is this software used? Does the software suit these different user groups in their varying usage flows? This is best covered by exploratory testing done by a tester with an open mind and understanding the mindset of the users, or acceptance testers with strong context knowledge.

2.2 In the eyes of the user: effective and suitable usage

When the software has good functional suitability, it makes the use of the software effective and well-suited to the situation it is used in.

This leads to the following the user experiences (ISO_IEC_25022/2016):

FEELING OF ACCURACY AND COMPLETENESS

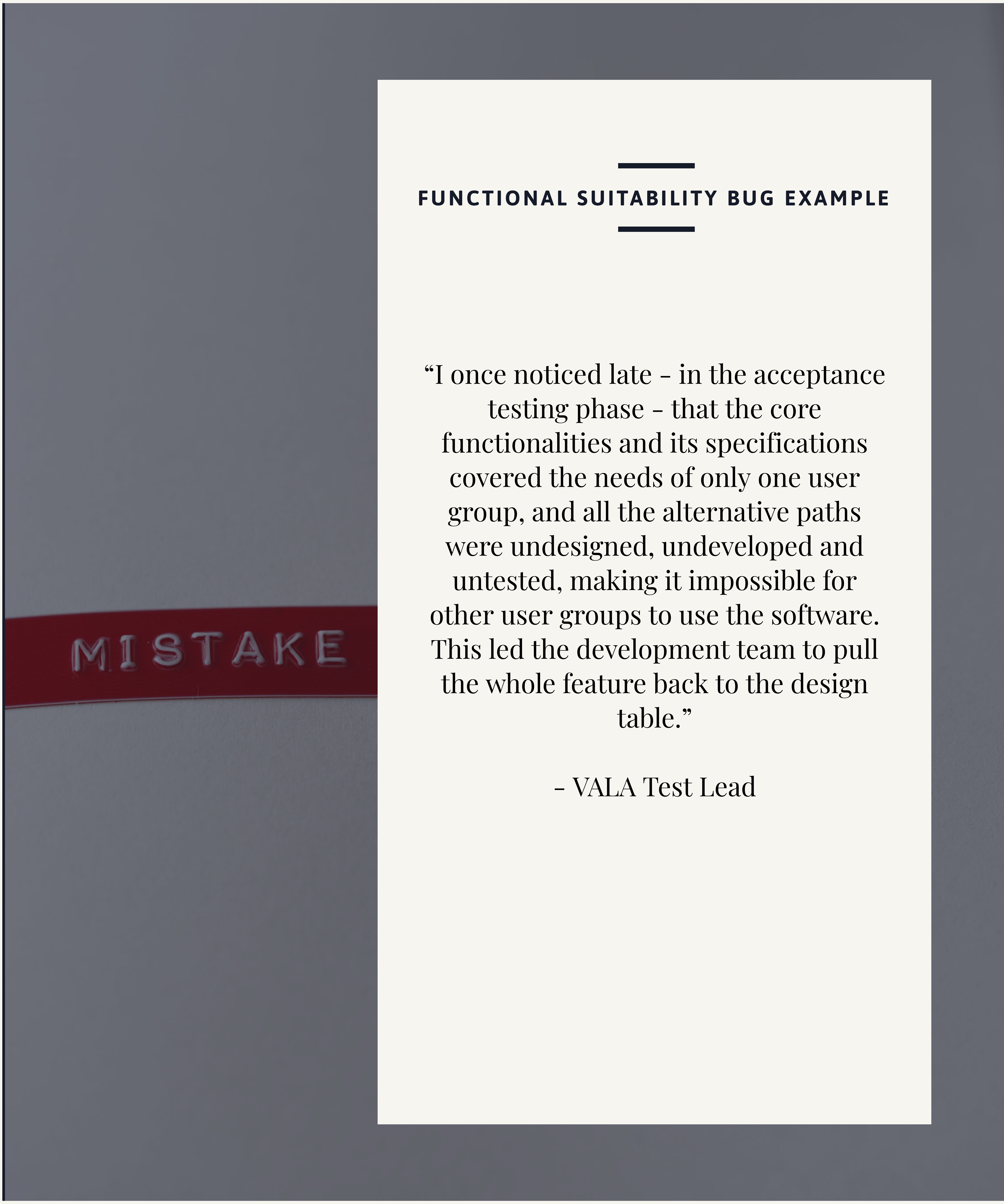
The user can achieve specific goals, in a way that feels right to the purposes of each user group.

CONTEXT COMPLETENESS

Feeling of getting things done. The user doesn’t get stuck before the goal is reached, and the use is effective, efficient, risk-free and satisfying.

FLEXIBILITY

The user can use the software in new situations, beyond the specified context.

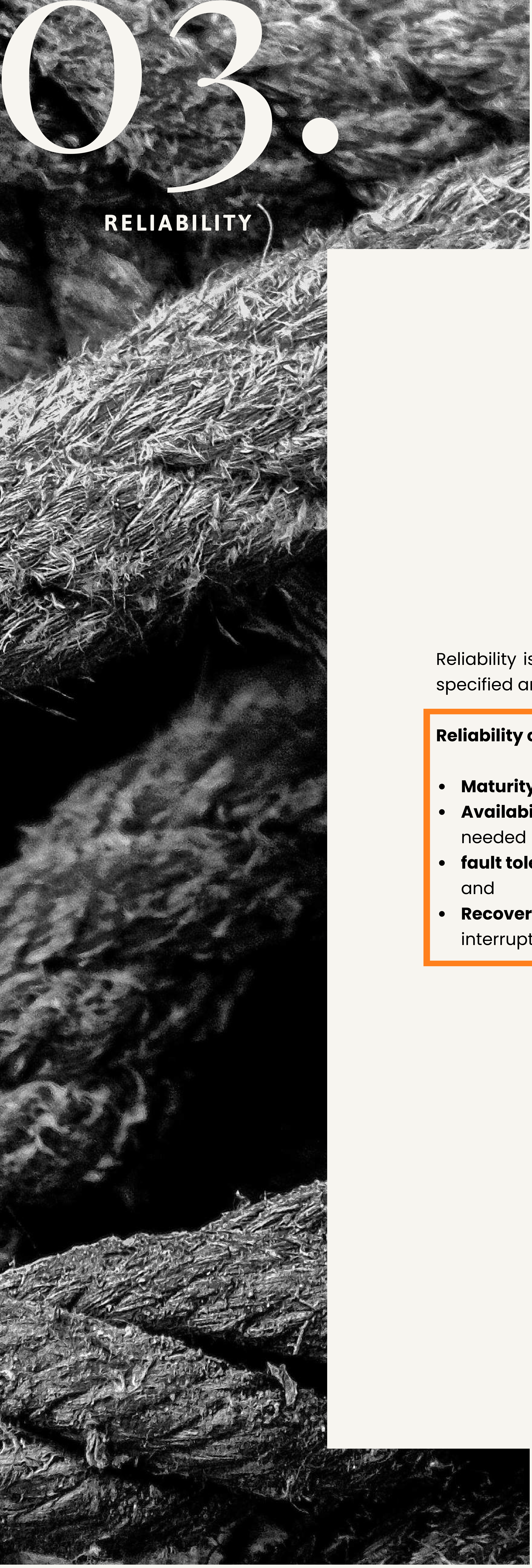


FUNCTIONAL SUITABILITY BUG EXAMPLE

"I once noticed late – in the acceptance testing phase – that the core functionalities and its specifications covered the needs of only one user group, and all the alternative paths were undesigned, undeveloped and untested, making it impossible for other user groups to use the software. This led the development team to pull the whole feature back to the design table."

– VALA Test Lead

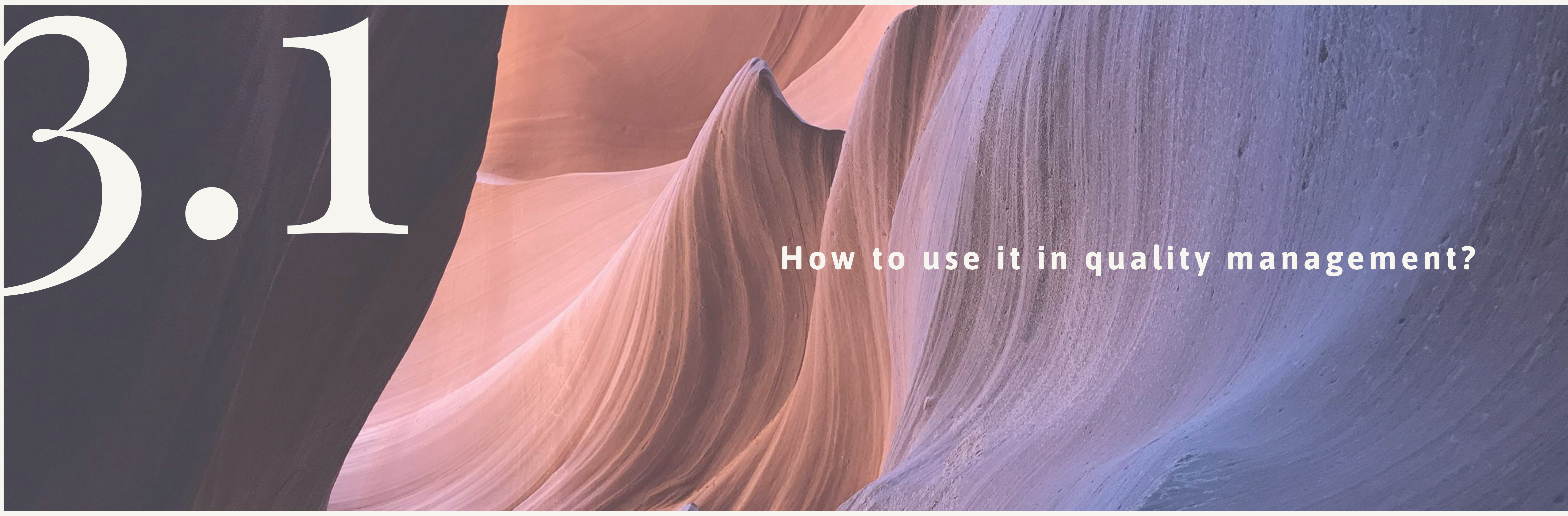
03. RELIABILITY



Reliability is a quality aspect, which tells if a system does what is specified and behaves in an expected manner in its environment.

Reliability consists of (ISO_IEC_25023/2016):

- **Maturity**; how finalized the system is?
- **Availability;** system and its parts are available for use when needed
- **fault tolerance**; a system can be used even at failure situations and
- **Recoverability**; how well the system recovers from a fault or interruption.



3.1 How to use it in quality management?

System maturity can be followed in the development phase by measuring the test coverage, and the ratio of how many issues have been fixed out of all found issues.

Fault tolerance can be well revealed in exploratory testing where the system is pushed over the limits of normal use in order to find faulty situations and how can the system be used despite these errors.

Recoverability can be improved by ensuring there is a proper and up-to-date backup data for the production use, but also by analysing how long it takes for the system to recover from a failure.

The above mentioned metrics are too rarely measured from systems in the development phase, even though unstable development (including test) environments and unavailability of a system under test can slow the development speed significantly.

Reliability figures are often monitored and analysed in production. Significant deviations in production system availability, fault tolerance or reliability figures should cause an alarm, but trends in these figures provide useful input for the development team after each release.

Here are a few examples of reliability related service level objectives (SLO):

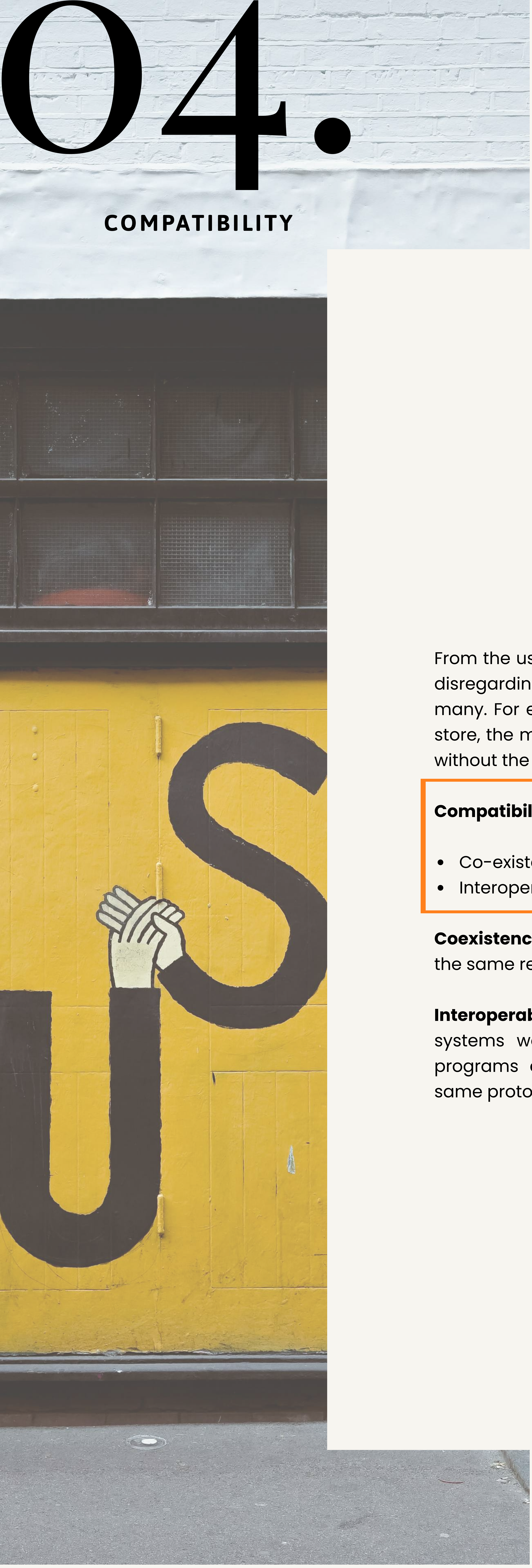
- How often can the system be unavailable?
- How long is the system down when a failure happens? Does it auto-recover?
- The mean time between failures during the system operation.

RELIABILITY BUG EXAMPLE

“The system didn't specify the max size of user input files, and if a user put a too large file in, the whole system went into failure and whole production was down until that was manually repaired.”

- Examples by VALA M-hub Community Members

04. COMPATIBILITY



From the user perspective compatibility looks like a smooth usage disregarding whether the usage flow runs on only one software or many. For example when a user is buying something from a web store, the money transactions are handled via third party systems without the user having to put in any payment details manually.

Compatibility consists of (ISO_IEC_25023/2016):

- Co-existence and
- Interoperability.

Coexistence means that two or more functions or systems share the same resources without mixing things up.

Interoperability means that the interfaces between different systems work well together. Interoperability refers to different programs exchanging information, sharing files and using the same protocols.

4.1 How to use it in quality management?

Compatibility needs to be specified properly; what interfaces do the components or systems have, and what kind of communication is supported between them? If APIs are well-specified, they are easy to be utilized in integration testing. The agreed API contracts should be verified with contract testing.

In feature testing it can be useful to mock out integrations, to ensure that we are testing only the software under test. For system integration testing, test environments need to provide the actual dependent systems to be utilized. If the dependent systems are too unstable or for other reasons can't be utilized, the dependent systems need to be mocked well to replicate the interaction with the system under test. However, if dependent systems are mocked out in all stages, it can lead to quality risks due to uncovered features in the mock or the mocks not being up-to-date.

To ensure good compatibility, it is important to do sufficient platform testing, component integration and system integration testing and/or API testing.

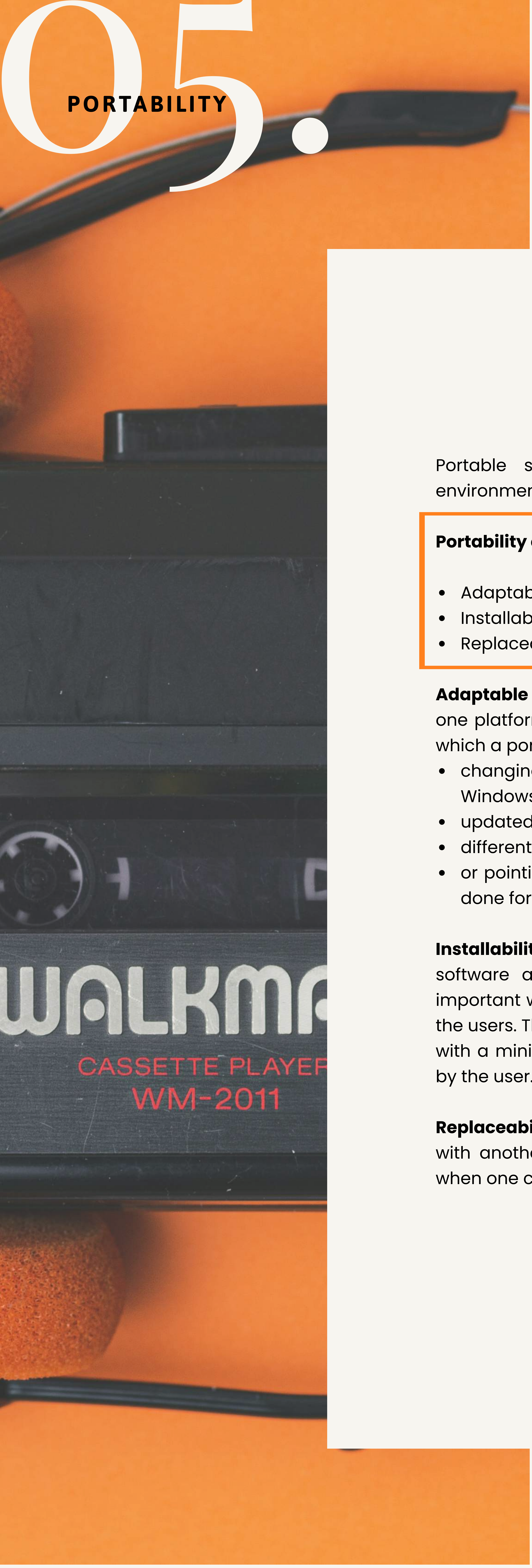
COMPATIBILITY BUG EXAMPLE

“A file was written in a UTF-8 format in a sending system. Reading system assumed the file was in ANSI format and read it that way. This caused special letters to be shown as a garbage.”

“Data submitted from device A worked just fine. Same kind of data submitted from device B messed up the system so badly that nothing could be submitted from any device after that until the table was cleared from corrupted data.”

- Examples by VALA m-hub community members

05. PORTABILITY



Portable systems can be installed and used in different environments, or they can replace similar systems.

Portability characteristics are (ISO_IEC_25023/2016):

- Adaptability
- Installability and
- Replaceability.

Adaptable systems can be transferred and installed easily from one platform to another. Here's a list of example target platforms which a portable system can easily switch between:

- changing operating systems (compatible with Mac and Windows, mobile app compatible with Android and iOS)
- updated operating system versions
- different browsers
- or pointing to a new interface used by the system (this can be done for example by standardized API structures).

Installability measures how effective and easy it is to install the software and software updates in different platforms. This is important when developing software that needs to be installed by the users. The installation should also be as automated as possible with a minimal amount of interference and configuration needed by the user.

Replaceability measures how easy it is to replace the software with another similar software. This needs to be considered also when one component is changed in a large system.

5.1

How to use it in quality management?

As testing portability is very repetitive, automation should be utilized to minimize toil, and to enable testing it continuously whenever changes have been made. To avoid huge costs in hardware, testing can utilize cloud-hosted test farms, where the tests can be run on multiple real-life devices.

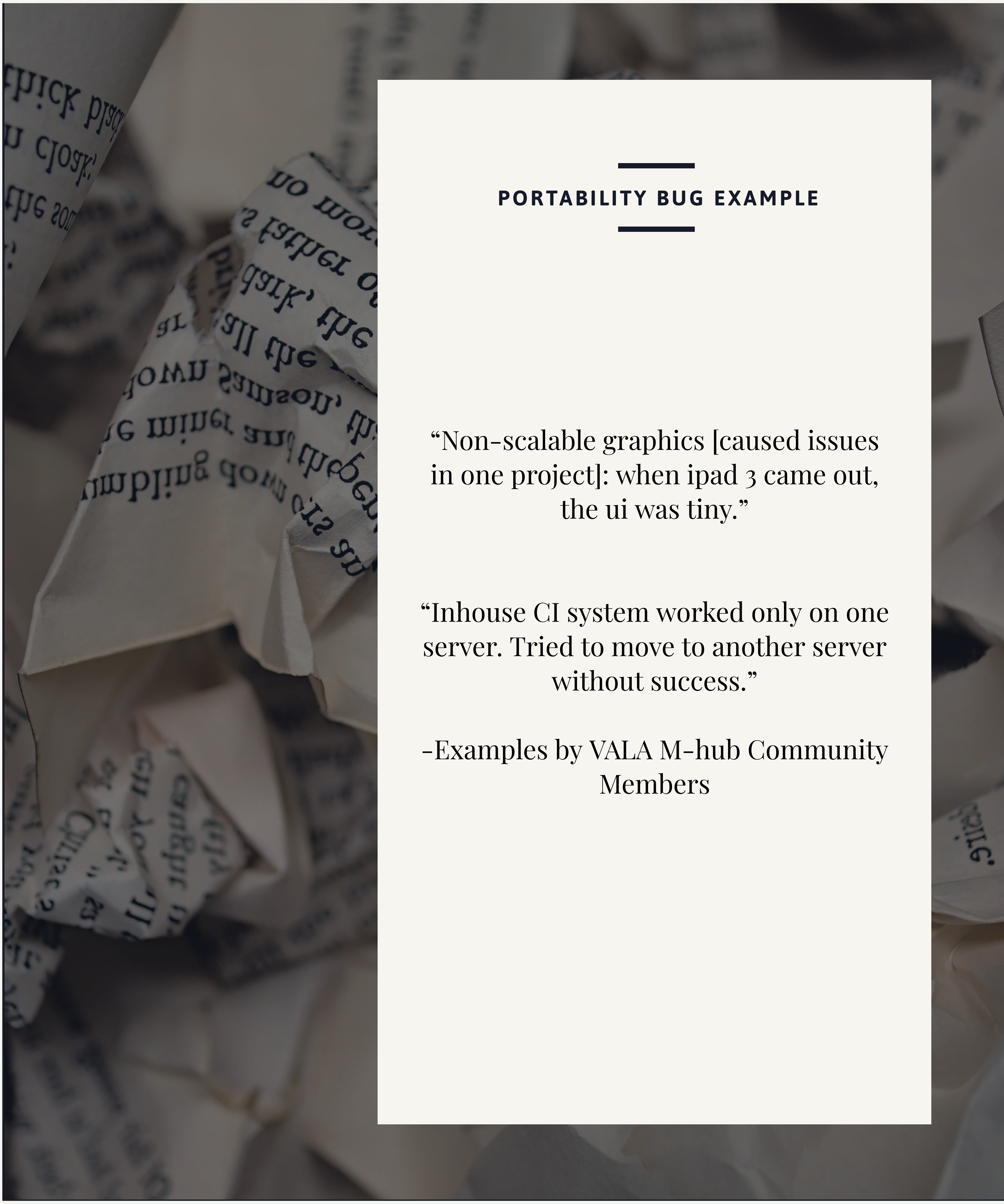
Replaceability can be evaluated with these questions:

HOW SIMILAR IS THE USAGE OF THE SOFTWARE? ARE THE POSSIBLE API'S SIMILAR?

CAN THE SOFTWARE FIT WELL IN THE OVERALL FUNCTIONALITY OF THE WHOLE SYSTEM? WOULD THE OVERALL USER EXPERIENCE OF THE SYSTEM STAY THE SAME?

ARE CHANGES IN THE DATA NEEDED?

CAN CONTRACT TESTING BE USED TO VERIFY REPLACEABILITY IN API'S?



PORTABILITY BUG EXAMPLE

“Non-scalable graphics [caused issues in one project]: when ipad 3 came out, the ui was tiny.”

“Inhouse CI system worked only on one server. Tried to move to another server without success.”

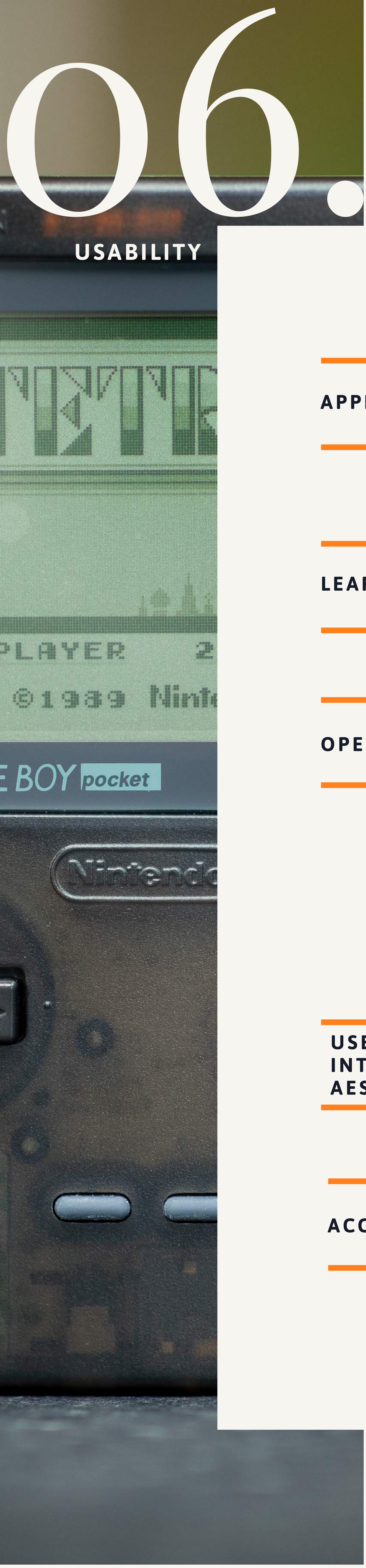
-Examples by VALA M-hub Community Members

06. USABILITY

When a user can use the software easily the way they want to and even feel satisfied using it, the software can be considered usable.

However, usability consists of many different areas (ISO_IEC_25023/2016):

- Appropriateness and recognizability,
- Learnability
- Operability and user error protection,
- User interface aesthetics and
- Accessibility.



06. USABILITY

APPROPRIATE

software meets the user's needs. It fulfills the tasks the user has to accomplish, and does it the way the user expects them to in this context. Recognizability relates to a new user recognizing if the software would fit their needs.

LEARNABLE

software is working in such an understandable way, that it provides an intuitive way to accomplish user goals without having to read loads of manuals.

OPERABLE

software is easy and consistent in use and enables the user to customize their own views and undo selections when wanted. A user can also control how to use it, for instance operate it with a keyboard, mouse or voice commands; moreover, there can be messages (such as popups and tooltips) to guide the user forward. For example if there is a summary of user selections, a user should be able to adjust the selections before proceeding.

USER INTERFACE AESTHETICS

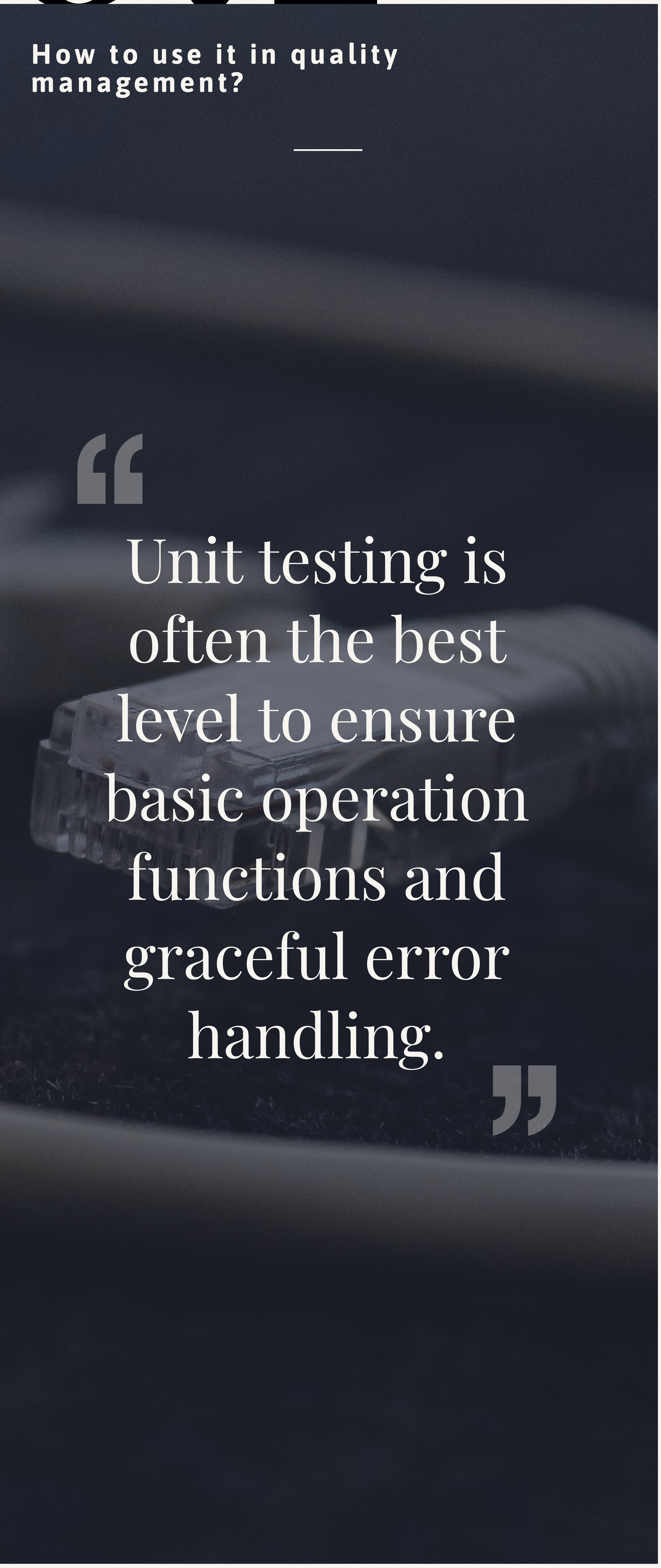
enables pleasing and satisfying interaction to the user, and is strongly related to the consumer product's brand and wanted visual look and feel.

ACCESSIBILITY

means that users with different characteristics (such as native language) or disabilities - permanent, non-permanent or situational - can use the system without difficulties. Accessibility is one of the primary outcomes of inclusive design, which focuses on making solutions usable for everyone, regardless of their conditions, environment or abilities.

6.1

How to use it in quality management?



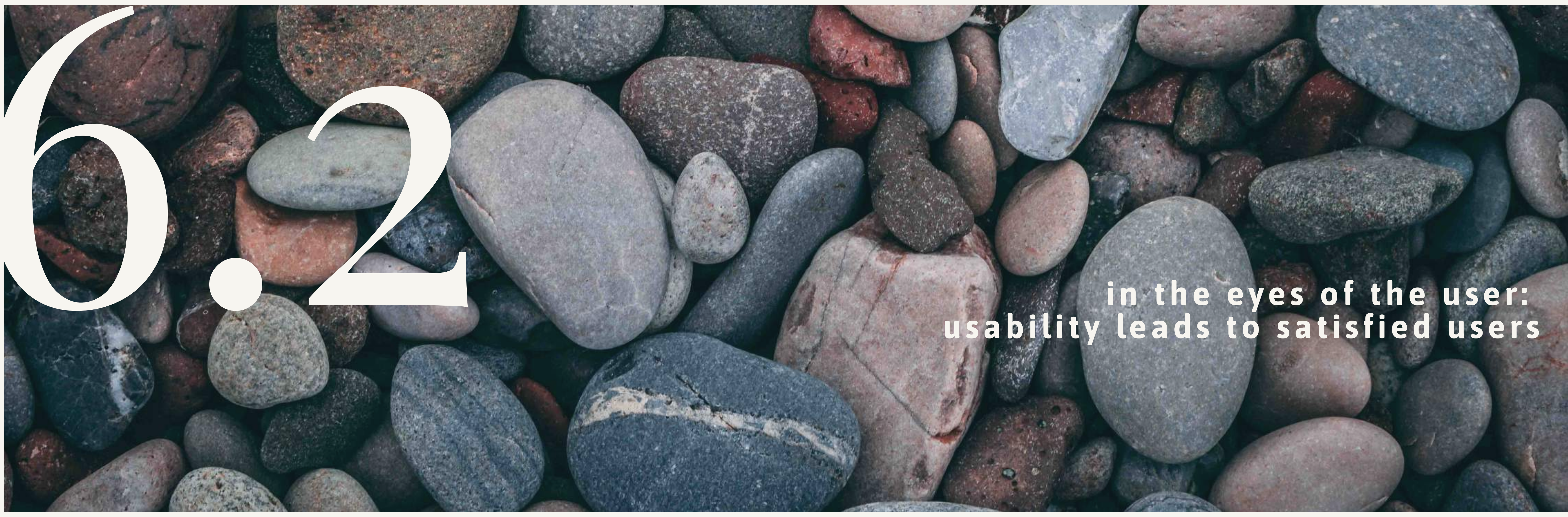
“Unit testing is often the best level to ensure basic operation functions and graceful error handling.”

To be able to predict in the development phase whether usability is sufficient, it is important to have a comprehensive understanding of the target users, their abilities and for what purpose they mainly use the system. This understanding can be gained by user interviews, by following their system usage, by collecting usage data from production or by discussing with someone from the business with deep customer understanding.

This understanding is then utilized in exploratory testing of a certain usability characteristics, for example this way:

- Thinking about what the user wants to accomplish and walking through the software in the users’ shoes.
- Using a tester without prior knowledge to test recognizability and learnability.
- Exploring the consistency of different use flows and that the software prevents a user from making mistakes.
- The look and feel of the user interface is pleasant and consistent with the design.
- The core features can actually be operated with for example a reading device.

UI test automation can be used to ensure operability and error handling, but also unit testing is often the best level to ensure basic operation functions and graceful error handling.



6.2 in the eyes of the user: usability leads to satisfied users

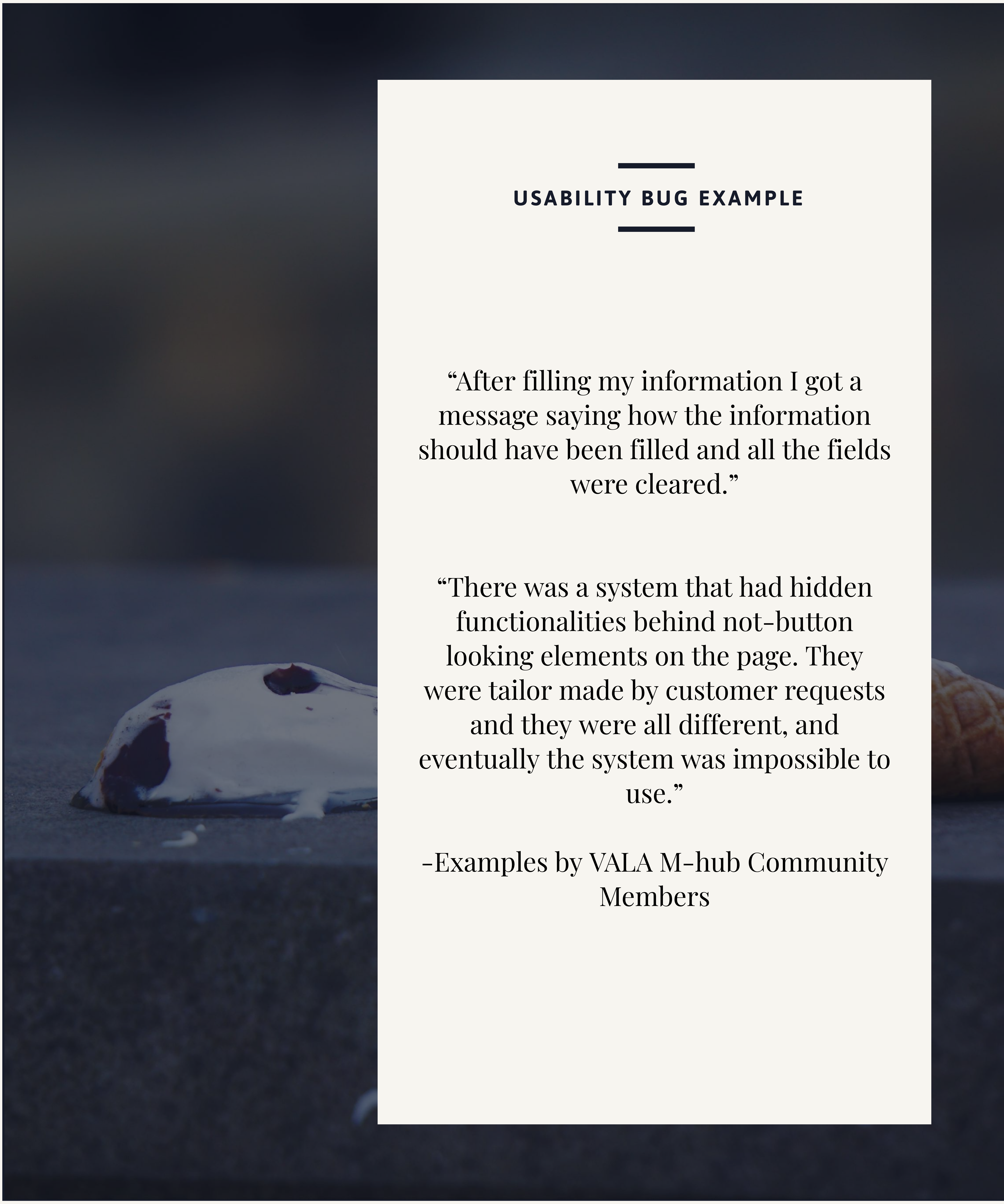
Usable software makes the usage of the software well fitting or simply stated it is usable.

Usability answers to the following questions (ISO_IEC_25022/2016):

- **Satisfaction:** is the user feeling good interacting with the system and having an overall positive attitude towards the system?
- **Usefulness**: does the user feel the goals are achieved, including the results and the consequences of using the system?
- **Trust:** how confident does the user feel that the system will behave as intended?
- **Pleasure:** how well are the user needs fulfilled, how pleased does the user feel using the system compared to other similar systems?
- **Comfort:** is the user feeling physically comfortable using the system?

In production user satisfaction can be clarified by interviewing users, following on one user or gathering feedback or by monitoring and gathering trends on for example these items:

- Amount of users per feature; the more popular, the better
- Amount of users leaving the process before finishing tells of poor usability
- Complaints per feature
- Users with complaints / all users
- Support requests / all users.



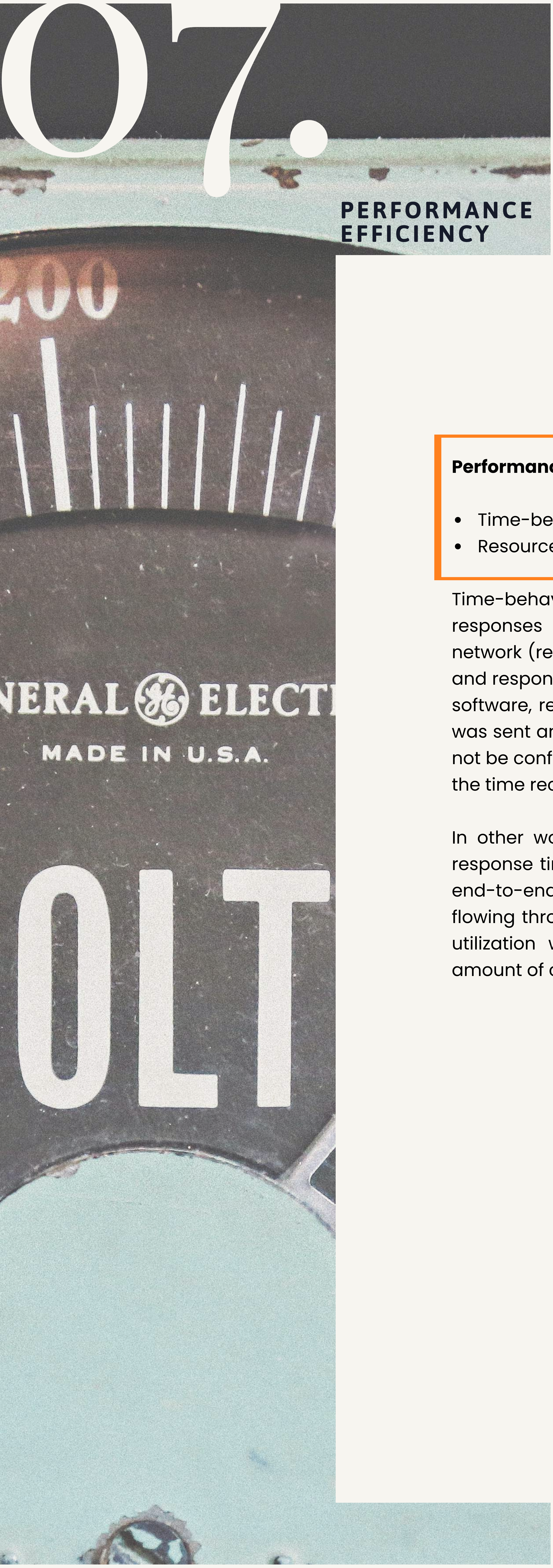
USABILITY BUG EXAMPLE

“After filling my information I got a message saying how the information should have been filled and all the fields were cleared.”

“There was a system that had hidden functionalities behind not-button looking elements on the page. They were tailor made by customer requests and they were all different, and eventually the system was impossible to use.”

-Examples by VALA M-hub Community Members

07. PERFORMANCE EFFICIENCY



Performance efficiency measures software's (ISO_IEC_25023/2016):

- Time-behavior
- Resource utilisation.

Time-behavior evaluates if the software provides appropriate responses to requests within a reasonable time frame over the network (response time) and if it can handle enough of such requests and responses (throughput rate). When measuring time-behavior of a software, response time is the difference between time when request was sent and time when response has been fully received. This should not be confused with the term latency, which doesn't take into account the time receiving the response content.

In other words, if the system under development was a high-way, response time would measure the time it takes for one car to drive it end-to-end, throughput rate would measure the amount of cars flowing through the high-way in a reasonable time, and the resource utilization would measure the sufficient amount of lanes for the amount of cars using the road.

7.1 How to use in quality management?

Performance efficiency improvements need to focus especially on the most important core functionalities, but more rarely used functions don't usually need so much optimising. In production, resource utilization may cause significant expenses if you have too little resources (leading to unstable software for example due to running out of memory) or too much resources (leading to high resource expenses) in production use.

The acceptable level of performance efficiency needs to be ensured at the very beginning, in requirement definition and architectural design phases.

Some of the considerations could be:

- How many simultaneous users does the system need to support? What is the median of usage load?
- Are there going to be some occasional or seasonal load peaks? Do we need to be able to scale the capacity up rapidly, or is the user volume more static?
- Which functionalities need to be optimized and which can be slower?
- Is it okay to cause a lot of internet traffic or should it be optimized in the software?
- How much hardware resources are required for the software to run smoothly?
- Are some functions too resource intensive?

Performance testing is a special skill that may require an additional performance testing expert to design and implement performance test cases and guide the team with implementing them. However this should not be a one time task, as performance should be automatically and continuously tested against baselines throughout the project.

A baseline can be a median response time for a type of request (e.g. a certain API call). Continuous testing is needed to ensure that changes do not suddenly cause degraded performance or sharp increases in resource consumption. When testing the software continuously against the baselines, it can be detected if changes had an effect on performance. Measuring performance just before the release leaves no time for adjusting a performance critical system.

When the limits of the system are identified, the software needs to ensure that too capacity intensive actions are prevented, for example by preventing users from loading too large files into the system. More on software reliability in Chapter 3, Reliability.

7.2

In the eyes of the user: Adequate performance leads to efficiency of use

"The acceptable level of performance efficiency needs to be ensured at the very start, in requirement definition and architectural design phases."

When enough resources are expended to match the accuracy and completeness with which users achieve goals, it leads to (ISO_IEC_25022/2016):

- A correct way of building the product to match the usage loads and
- Optimized core workflows make it easier to reach the most important user goals.

When using a software with adequate performance, the user doesn't need to noticeably wait for the system to process the wanted functions or the system doesn't end up in failure if the user does a heavy search or inputs a large file for system handling.

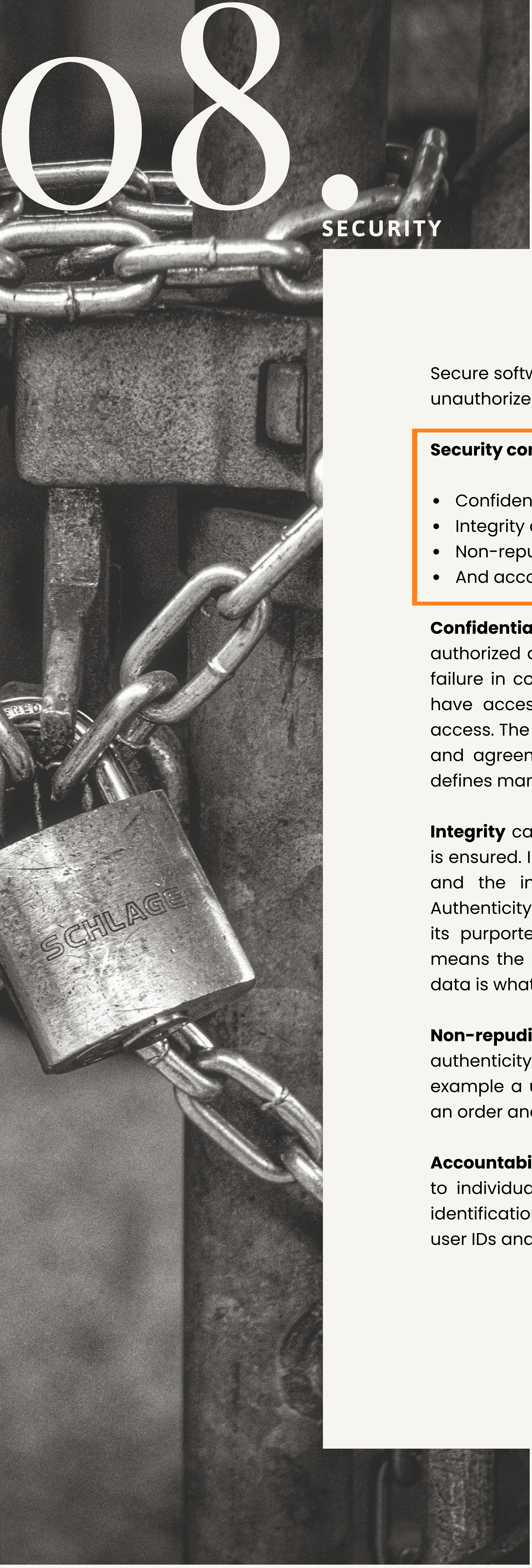
PERFORMANCE BUG EXAMPLE

“After submitting a form, the back end processed the info so long that the user was logged out from the website due to inactivity and the process needed to start from the beginning.”

“Opening a table with a large amount of data took so long, that it timed out in the frontend. The heavy users of this software couldn’t access their vital information in production.”

“The phone answering machine should have been able to collect at least hundred simultaneous phone call attempts. Instead it succeeded to serve only three. Not yet ready for production...”

-Examples by VALA M-hub Community Members



08. SECURITY

Secure software keeps all the data and information protected from unauthorized use.

Security consists of (ISO_IEC_25023/2016):

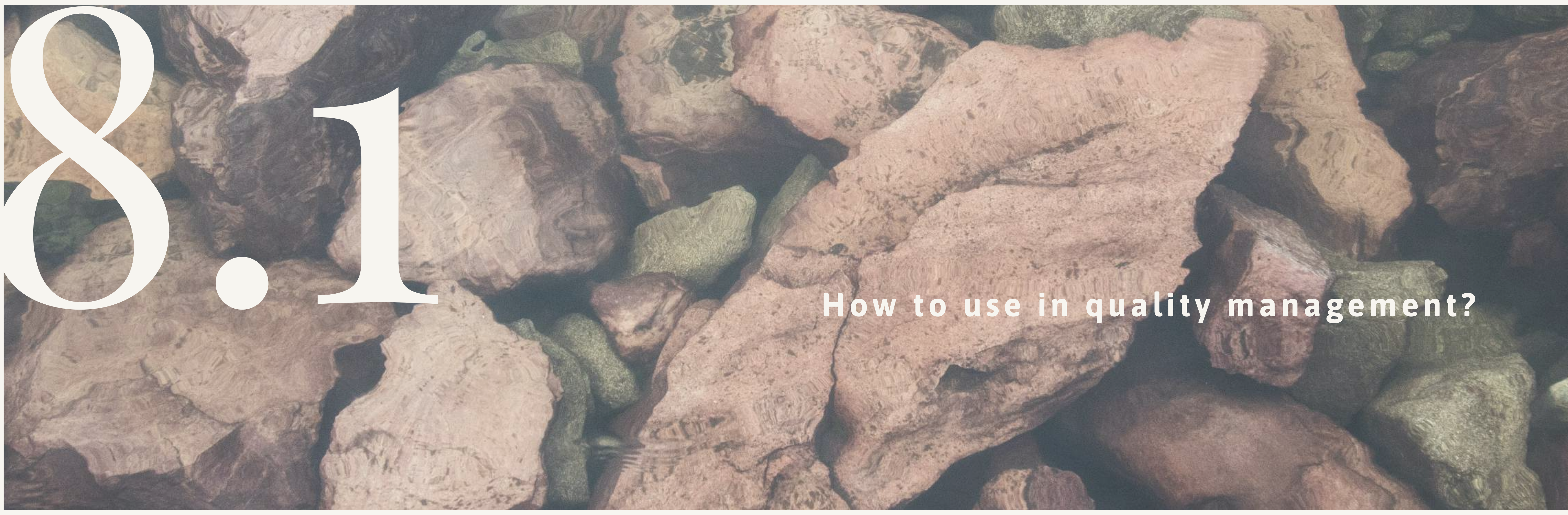
- Confidentiality
- Integrity and authenticity
- Non-repudiation
- And accountability.

Confidentiality means that the data or information is available for authorized and unavailable for unauthorized persons or groups. A failure in confidentiality happens when someone who should not have access to certain information or data, manages to gain access. The confidentiality rules may come from company policies and agreements but in certain domains regulatory compliance defines many confidentiality rules by law.

Integrity can be explained by how the authenticity of information is ensured. In other words, the source of the information is genuine and the information is not altered, corrupted or destroyed. Authenticity guarantees that data is genuine and originated from its purported source. It is a special case of integrity. Integrity means the data is untouched and complete, authenticity means data is what it claims to be.

Non-repudiation refers to a situation, where a party can't deny the authenticity of their signature on information or data. So for example a user can't later deny he/she was the one who placed an order and therefore needs to pay for it.

Accountability means that transactions can be traced back later to individual users. E.g. accountability is met when unique user identification and authentication are used. The usage of shared user IDs and passwords destroys accountability.



8.1 How to use in quality management?

It is beneficial to consult security experts on security related issues and how they should be handled by the software. It is not very effective to only run a security scan before a release, as many things need to be implemented early on in the product life-cycle to make the software under development secure enough in the given domain. Adding abuser stories to user stories could be one way of adding security characteristics in the software.

Testing confidentiality is a part of normal blackbox testing, where it is ensured that only authorized user groups have access to the given features, and the user-defined password is required to be strong enough. Integrity is suitable to be tested in whitebox testing; how well is the unauthorized access and editing prevented in the component or system integration level?

It needs to be evaluated, if the data is strongly encrypted or is the data lying around in natural language, easily read by anyone with access? Also, using well-established digital signature helps in having non-repudiable service, if applicable.

Accountability can be tested by making an action and seeing if it can be found in the system logs, revealing the correct user in action. If data history needs to be saved, the earlier versions of data need to be saved in the database as obsolete versions (as audit trail entries), and this can be tested by changing the data and seeing the old versions in the database correctly.

If security protection is vital, penetration testing can be used to simulate an attack with an aim in unauthorized access in a more forceful way.

8.2

In the eyes of the user: Secure software leads to safe software usage

"When a software is secure, the user doesn't have to take economic risks using it."

When security factors are well taken care of in the software, it provides secure usage of the software to users.

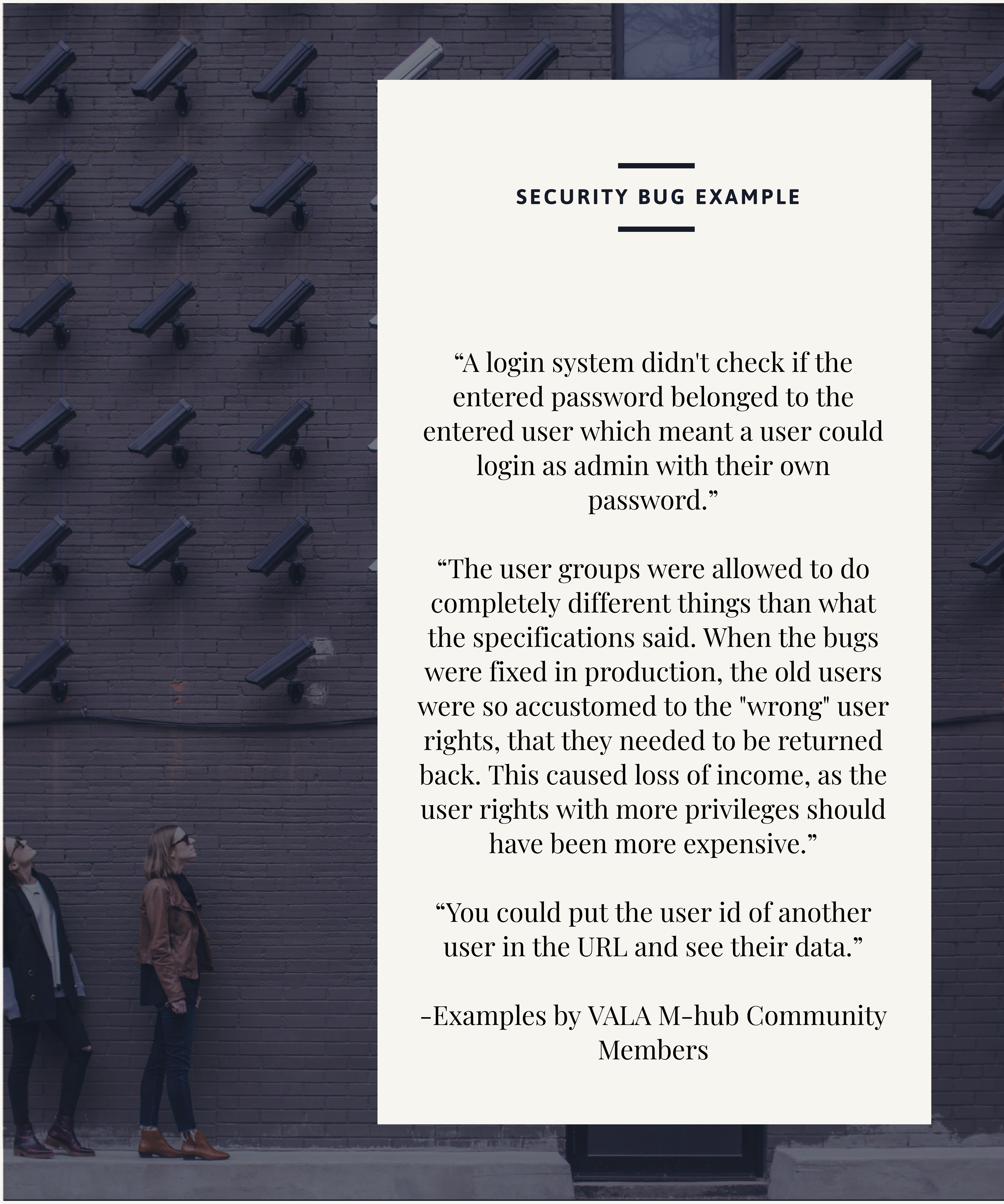
Security in usage consists of (ISO_IEC_25022/2016):

- Avoiding economic damage risk
- Avoiding health and safety risk
- Avoiding environmental harm risk

So when software is secure the user doesn't have to take economic risks using it (as in unsecure payments or leaking financial information for criminal use). Also the company's commercial property and reputation are better protected, which is important as issues in economics not only cause direct economical effects, but can lead to severe reputation turnbacks leading to massive customer getaway for a long period of time.

Secure software usage doesn't lead to health or safety risks to anyone. So for example a social networking application should prevent harassing others behind disguised identities and a software in car dashboard should want the user if some of the safety systems malfunction.

Environmentally secure software does not for example enable the user to select to leak toxic waste into nature, or for example doesn't require alarming amounts of energy to operate, as current blockchain technology -based solutions such as cryptocurrencies do.



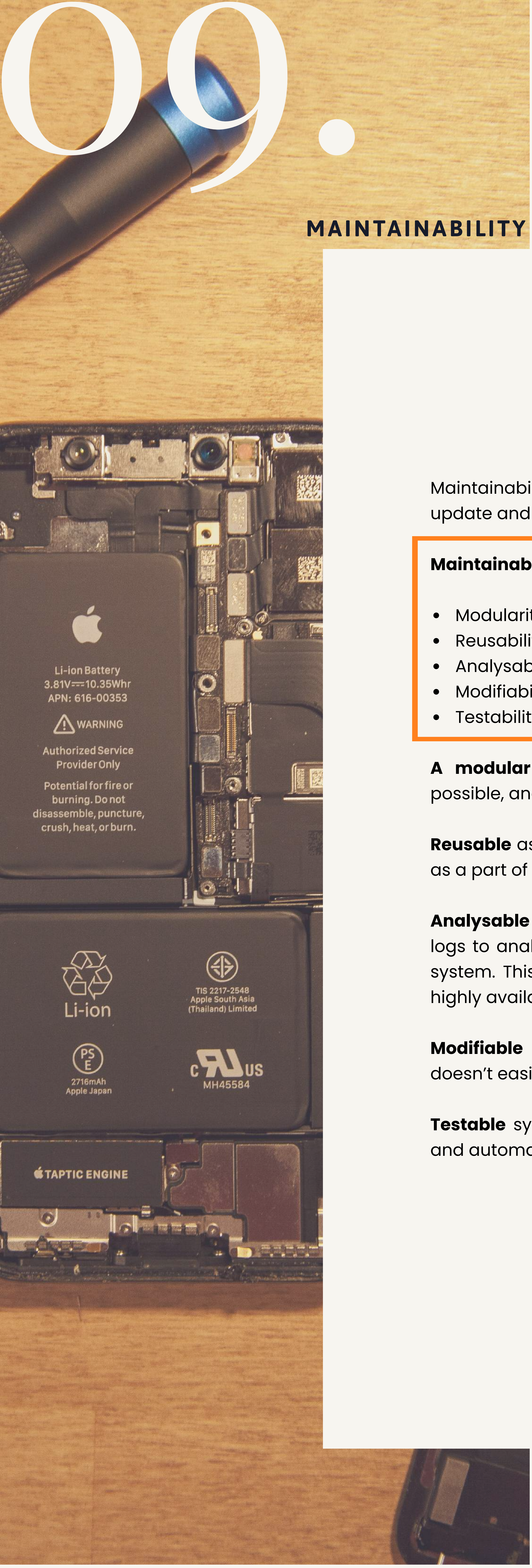
SECURITY BUG EXAMPLE

“A login system didn't check if the entered password belonged to the entered user which meant a user could login as admin with their own password.”

“The user groups were allowed to do completely different things than what the specifications said. When the bugs were fixed in production, the old users were so accustomed to the "wrong" user rights, that they needed to be returned back. This caused loss of income, as the user rights with more privileges should have been more expensive.”

“You could put the user id of another user in the URL and see their data.”

-Examples by VALA M-hub Community Members


Li-ion Battery
3.81V⎓10.35Whr
APN: 616-00353
WARNING
Authorized Service Provider Only
Potential for fire or burning. Do not disassemble, puncture, crush, heat, or burn.
Li-ion
TIS 2217-2548 Apple South Asia (Thailand) Limited
PS E
2716mAh Apple Japan
c US
MH45584
TAPTIC ENGINE

09. MAINTAINABILITY

Maintainability is a quality aspect which explains how easy it is to update and change the system.

Maintainability consists of (ISO_IEC_25023/2016):

- Modularity
- Reusability
- Analysability
- Modifiability
- Testability

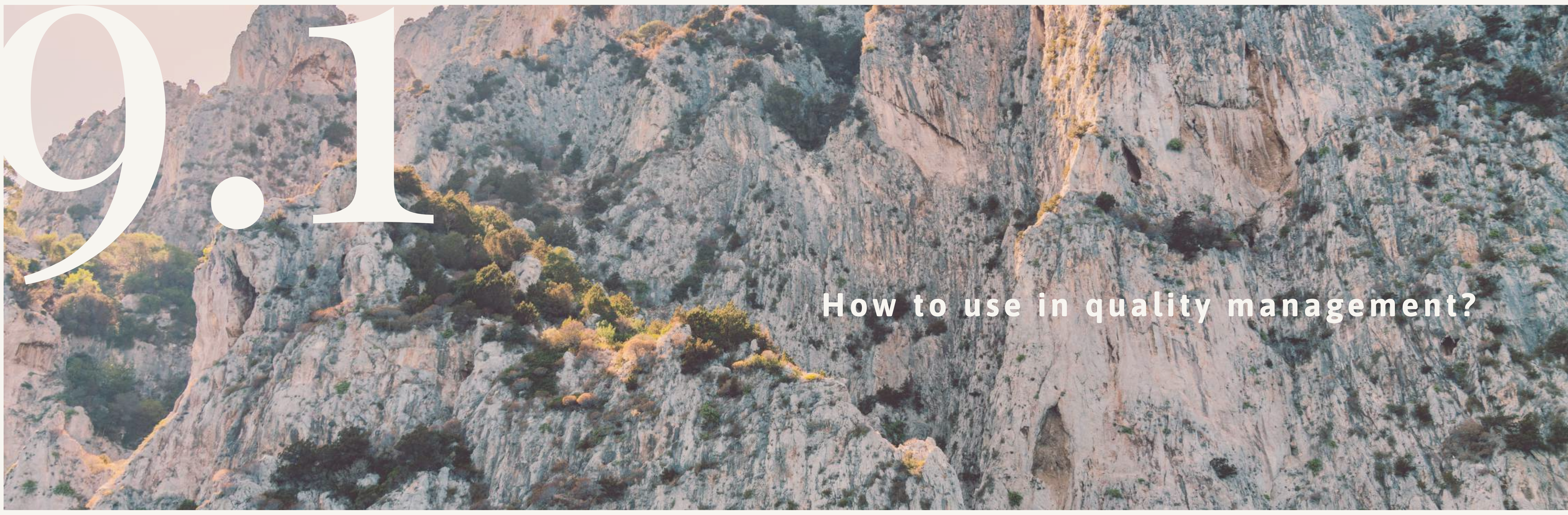
A modular system consists of as independent modules as possible, and each module is not very complex.

Reusable assets can be used many times; in different functions or as a part of building other modules.

Analysable system consists of readable and complete system logs to analyse production usage and to diagnose failures in the system. This is vital in error recovery in systems that need to be highly available.

Modifiable system is easy to change; the work is efficient and doesn't easily lead to errors.

Testable system is easy to test manually using the system itself and automatically with an external test automation software.



9.1 How to use in quality management?

All maintainability issues can be avoided with proper technical design and modern software development methods such as agile and DevOps, where testability and automation is a natural part of planning on any feature to be developed.

Modularity, reusability and modifiability can be assessed in the code review before any code can be put in the main branch. To improve reusability, all code needs to be written according to agreed coding rules. Analysability can be checked and improved in development including testing, so there'll be no problems in analysing urgent production issues when they arise.

Testability of the whole system under development is also a factor that needs to be assessed before the actual testing begins.

A quality manager needs to consider for example these questions:

- How complete user flows can the testing cover?
- How independent is the testing from external systems, or can they be mocked out?
- How should the tests be automated in the most simple way? Are the UI elements and their properties available for the testing tool? What about APIs?
- And last, how easy is it to stop testing and have a fresh restart of the testing when needed?

MAINTAINABILITY BUG EXAMPLE

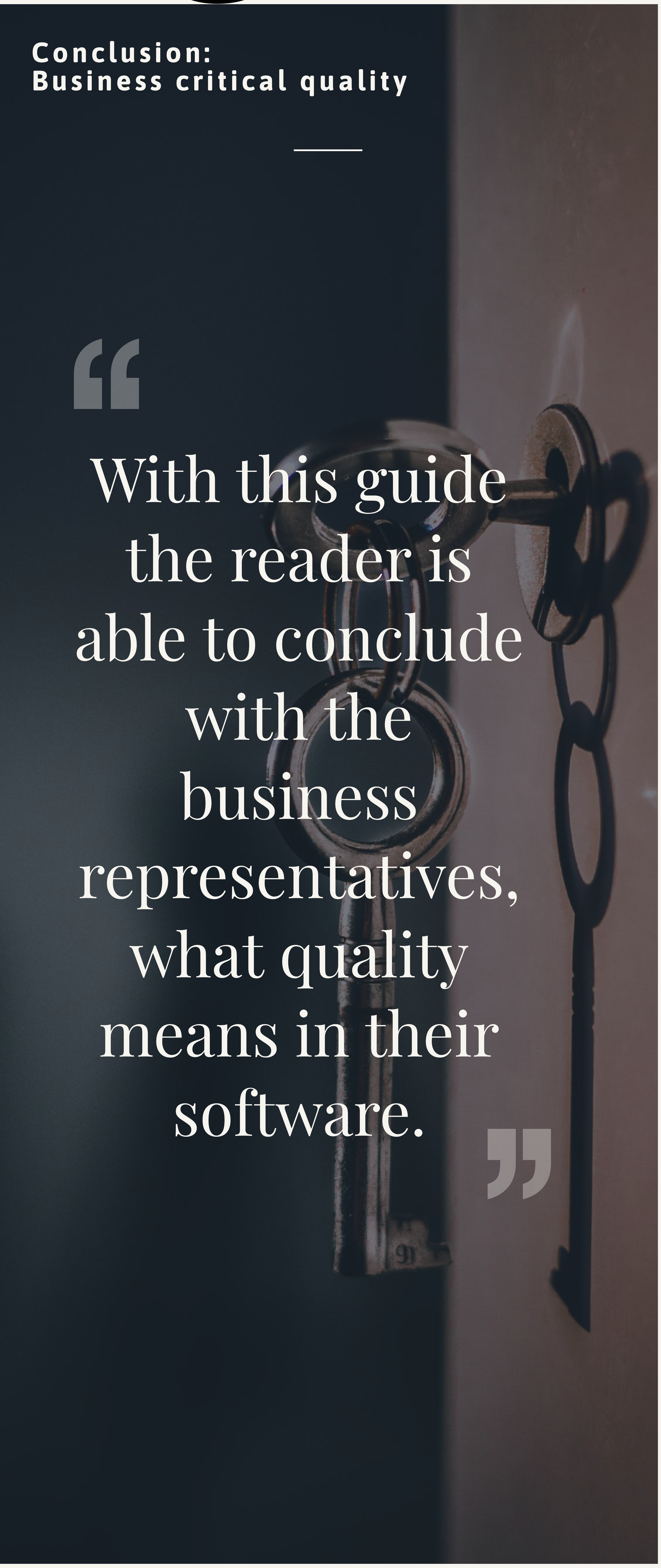
"Our oldest test automation code was done so that all the test cases, part of the functionalities and test data were all put in the same spaghetti code. We threw it away, it was not easily fixable."

"The system was built so that all form elements (with their related HTML code) were stored in a database. I spent 4 hours trying to add a new form field on a web page which should be a one minute thing."

-Examples by VALA M-hub Community Members

10

Conclusion: Business critical quality



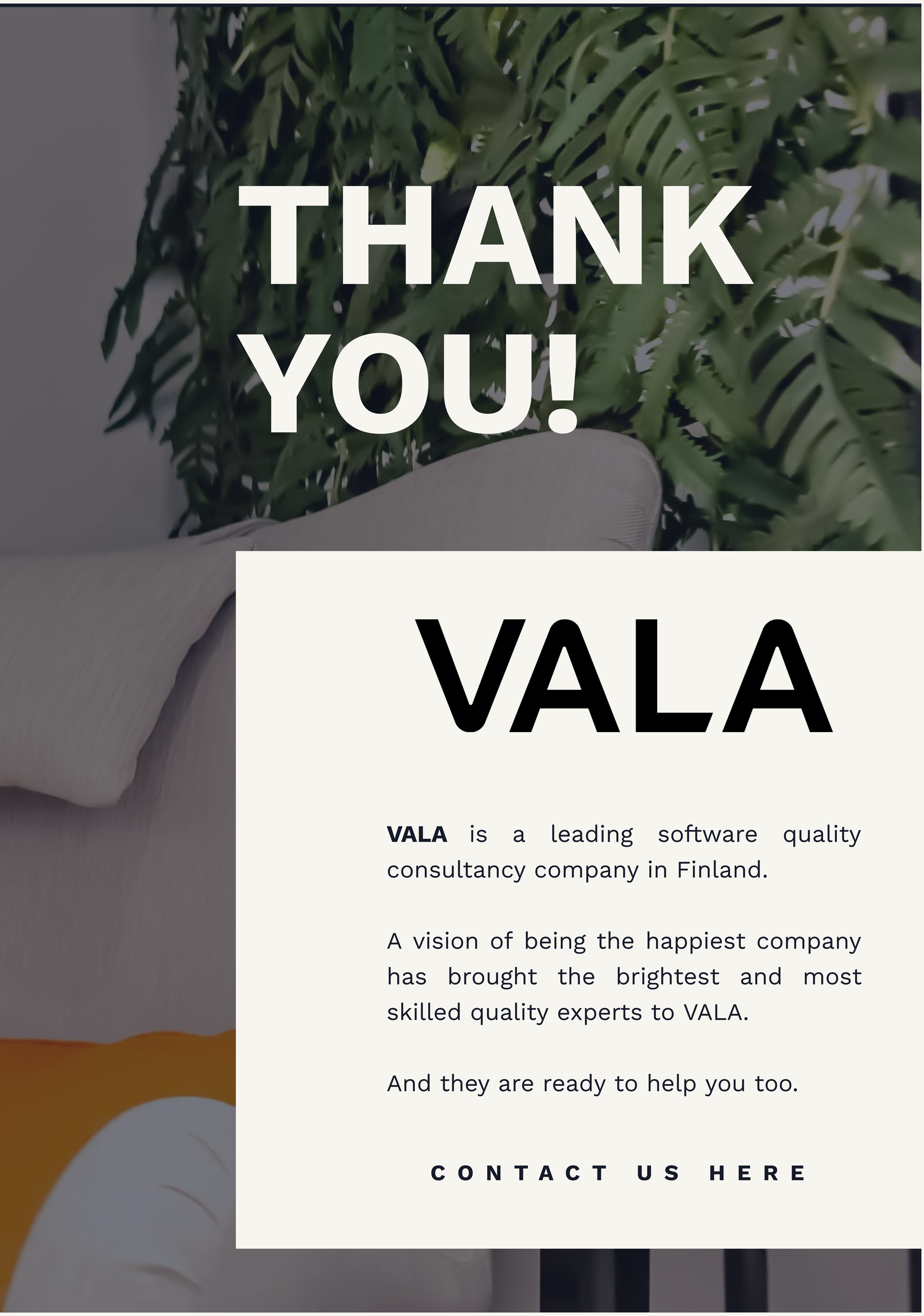
"With this guide the reader is able to conclude with the business representatives, what quality means in their software."

With this guide the reader is able to conclude with the business representatives, what quality means in their software. Moreover, they are more aware of what is critical and what is not at a given stage of the product life-cycle. For example, they should be able to choose the top three priorities.

We should trust the development team - including testing - and enable them to find the best ways to reach the business critical quality goals. When these goals and their priorities are thought over and agreed upon in advance, the quality strategy and the development efforts are more likely to lead into wanted results.

Having shared goals is the key to successful software development, and right goals are critical to successful business.

Best of luck in quality improvement endeavours!



THANK YOU!

VALA

VALA is a leading software quality consultancy company in Finland.

A vision of being the happiest company has brought the brightest and most skilled quality experts to VALA.

And they are ready to help you too.

CONTACT US HERE

WWW.VALAGROUP.COM