Software Quality

What is 'good' software?

Software Quality

Desirable qualities from 3 points of view:

- The `user` requires that the software is **reliable**, **efficient** and **easy to use**.
- The `software company` requires that the software is **verifiable**, **maintainable**, **portable** and **extendible**
- The `project manager` requires that the process is **productive**, **predictable** and **easy to control**.

[.build-lists: true]

Internal and external qualities

We can divide the quality of the software into two categories: **internal** and **external**.

- External qualities are visible to the final users of the system.
- Internal qualities concern the developers.

Internal and external qualities

We can divide the quality of the software into two categories: **internal** and **external**.

- External qualities are visible to the final users of the system.
- Internal qualities concern the developers.

In general, software users are only interested in *external qualities*, but *internal qualities* (that are related with the software structure), allows to achieve external qualities.

For example, **verifiability** quality (an internal quality) is necessary to achieve **correctness** quality (an external quality).

^we can give more detailed look at some of these qualities

Correctness

Correctness

A **functional requirement** defines the correct relationship between inputs and outputs.

A program is functionally correct if it behaves according to what is established by the functional specifications.

In short, *if it does what it should do*.

e.g. the functional requirement is that the software sorts web pages based on the user's query

A software that does this is **functionally correct**

Correctness

Non-functional requirements also exist: for example, performance and scalability.

A software that sorts web pages based on the user's query, but that takes too much time, is still **functionally correct**,.

Its output is correct, but the time it takes is so high that the program is unusable.

Correctness

This definition of correctness assumes that:

- System specifications are available
- It is possible to determine without ambiguity whether a program satisfies them

Correctness

This definition of correctnessl assumes that **system specifications are available**

- sometimes they are **not** available
- there are cases in which the users' wishes can only be estimated (for example, by market analysis)
- specifications are often written in natural language, so they may contain ambiguity.

Correctness

This definition of correctnessl assumes that **it is possible to determine *without ambiguity* whether a program satisfies them**

Correctness

How can we **evaluate the correctness**?

- the specifications must be **available** and **not ambiguous**.

We have three approaches available

1) **formal verification** - it consists in a mathematical proof of correctness, as we do in mathematical theorems. - It can only be done for 'toy problems' - it is extremely complex - even if we were able to formally demonstrate the correctness of our algorithm, our software will run on a system that contains many other software that we probably can not prove to be correct.

^there will be unavoidable interactions between our software and other software. Even if our software is theoretically correct, unpredictable interactions with other software may generate errors.

2) **code inspections** - there are guidelines to perform the code inspection in a rational way. Guidelines indicate the timing and methods of code inspection.

Example:

- Code review rates should be between 200 and 400 lines of code per hour.
- Inspecting more than a few hundred lines of code per hour may be too fast to find errors
- checklists of common errors

^ **Examples of guidelines**

^ Kemerer,, C.F.; Paulk, M.C. (2009-04-17). "The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data". IEEE Transactions on Software Engineering. 35 (4): 534–550. doi:10.1109/TSE.2009.27. Archived from the original on 2015-10-09. Retrieved 9 October 2015.

^ "Code Review Metrics". Open Web Application Security Project. Open Web Application Security Project. Archived from the original on 2015-10-09. Retrieved 9 October 2015.

^ "Best Practices for Peer Code Review". Smart Bear. Smart Bear Software. Archived from the original on 2015-10-09. Retrieved 9 October 2015.

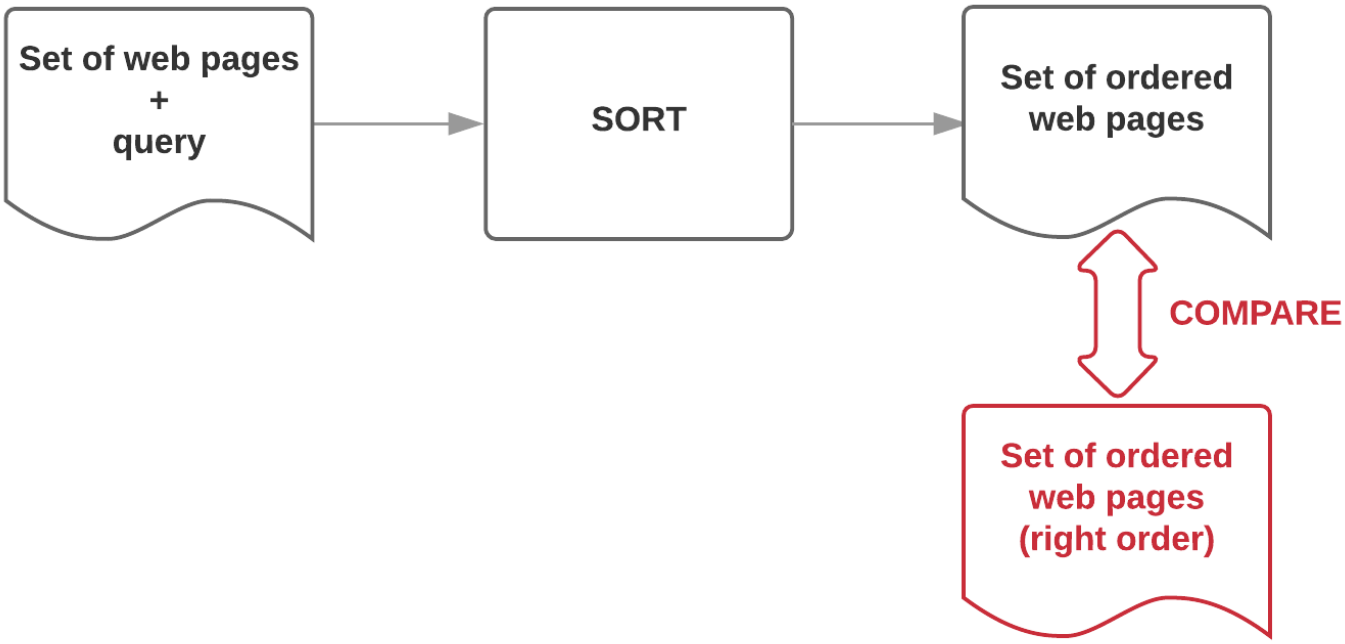
^ Bisant, David B. (October 1989). "A Two-Person Inspection Method to Improve Programming Productivity". IEEE Transactions on Software Engineering. 15 (10): 1294–1304. doi:10.1109/TSE.1989.559782. Retrieved 9 October 2015.

^ Ganssle, Jack (February 2010). "A Guide to Code Inspections" (PDF). The Ganssle Group. Retrieved 2010-10-05. http://www.ganssle.com/inspections.pdf

3) **testing**

The purpose of test techniques is to find errors in the software, and to verify that the software is fit for use.

It consists in submitting to the software a set of inputs and to compare the obtained output with the desired output.



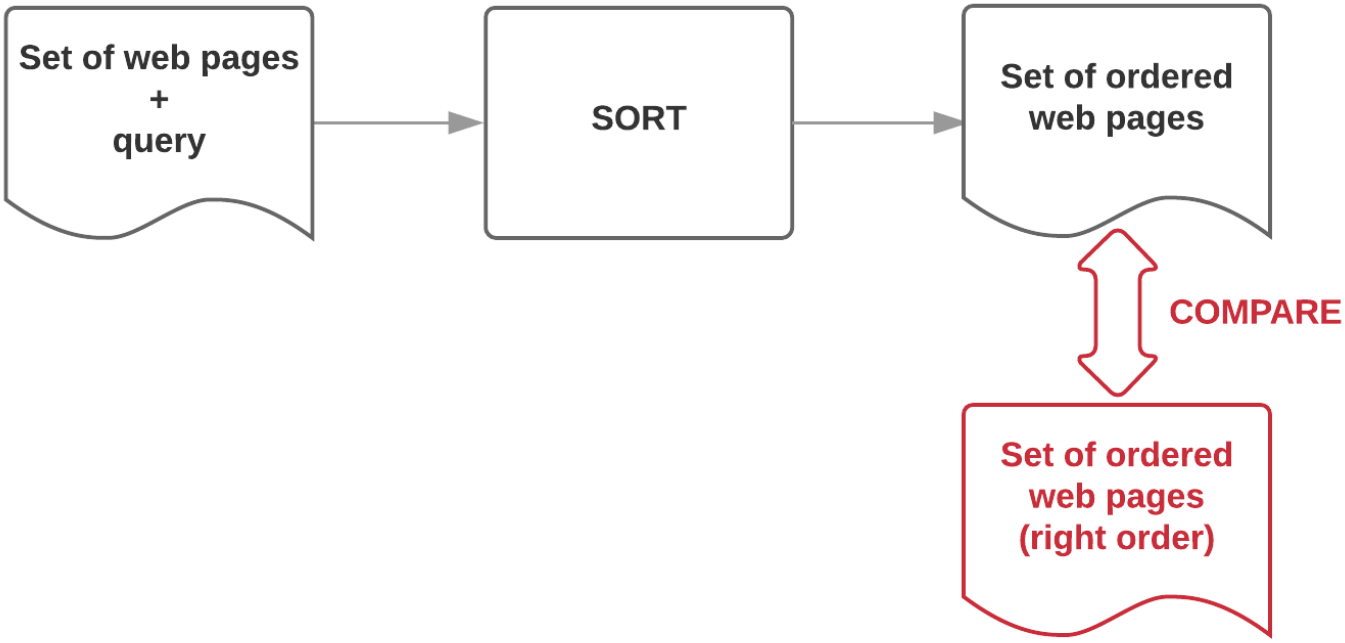
3) **testing**

The purpose of test techniques is to find errors in the software, and to verify that the software is fit for use.

It consists in submitting to the software a set of inputs and to compare the obtained output with the desired output.

Testing can be used to show the **presence of errors**, but never to show their absence!

Testing cannot guarantee that the system under test is error free.



[.build-lists: true]

Improve Correctness

How can we improve correctness ?

- **High-level programming languages** that are appropriate to the problem. Programming languages are merely 'tools'. We must choose those that best fit the problem.
- Well-known **standard algorithms**; standard module libraries (we do not have to reinvent the wheel...)
- **design patterns**: standard solutions to recurring problems
- using proven methodologies and processes, e.g. **agile methodologies**

Reliability

Reliability

Correctness is an absolute quality. If the output is equal to the expected output, the software is correct.

Reliability is relative. The software is reliable if the user can rely on its features.

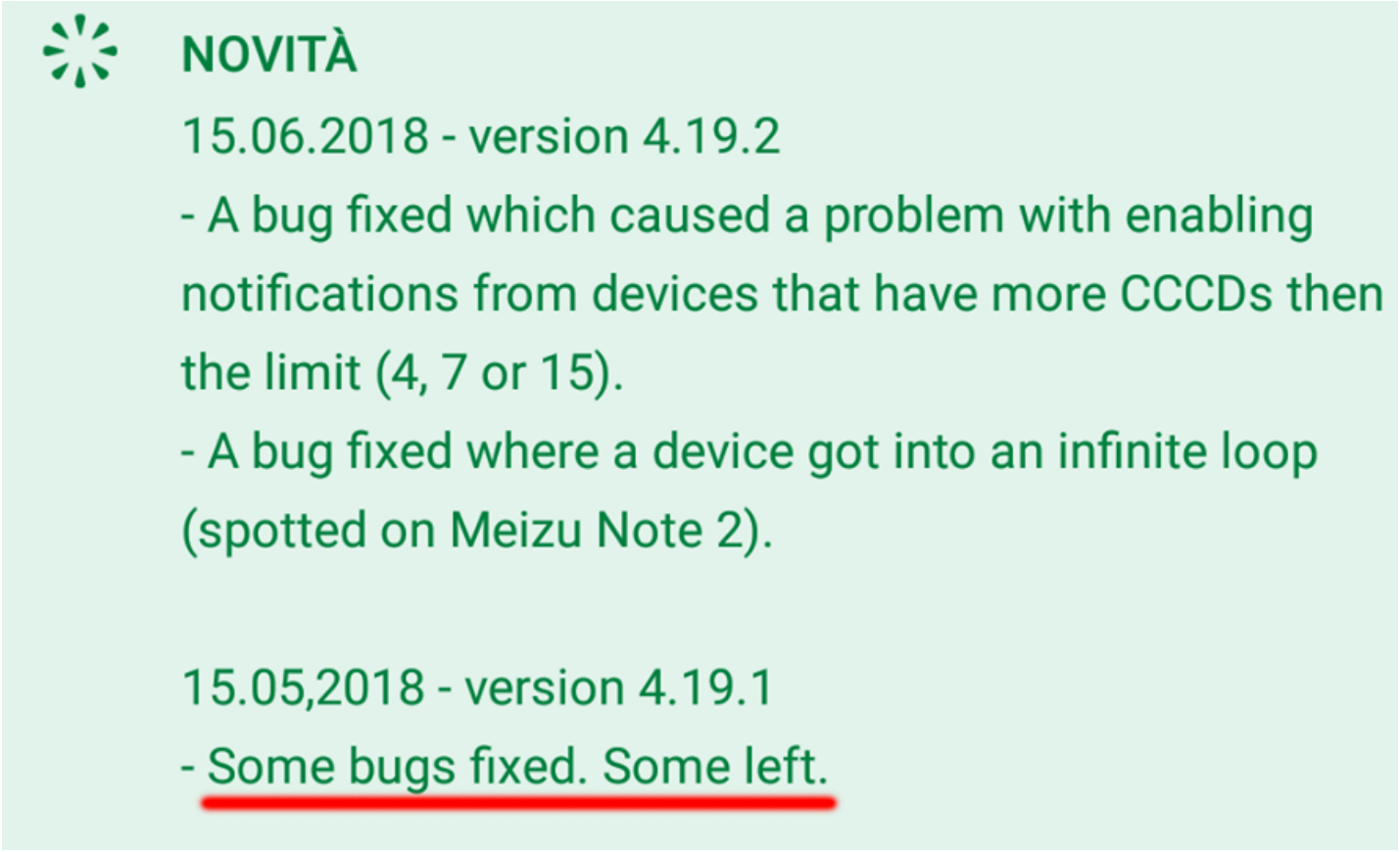
If the software makes minor errors, it can still be considered reliable.

^ you can consider a bathroom scale. if it makes mistakes on the order of a few grams, it is not formally correct, but it can be reliable, you can trust the outcome for all practical uses. ^ if you consider a word processor, it could sometimes produce tables that are not perfectly aligned. It is a minor error, You can trust your software, despite this minor error. ^The same error would not be acceptable in a professional publishing software

Software contains errors!

Despite our efforts, the software will contain errors! Developers are well aware of this.

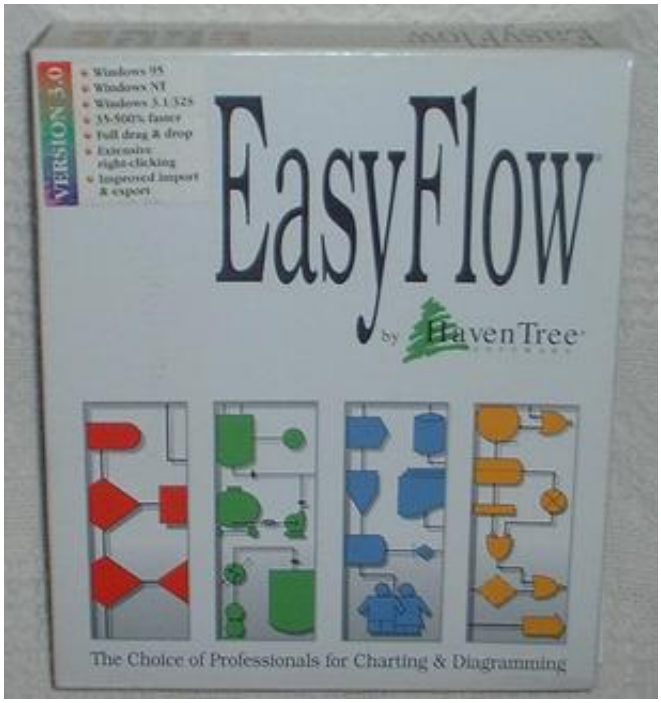
So developer protect themselves against legal actions using the disclaimers.



Disclaimer

"**This software is provided "as is"** and any expressed or implied **warranties**, including, but not limited to, [...] fitness for a particular purpose **are disclaimed**.

In no event shall we be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to ...) arising in any way out of the use of this software, even if advised of the possibility of such damage."



Disclaimer

"We don't claim **Interactive EasyFlow** is good for anything -- if you think it is, great, but it's up to you to decide. If Interactive EasyFlow doesn't work... **If you lose a million because Interactive EasyFlow messes up, it's you that's out the million, not us**. If you don't like this disclaimer... **We reserve the right to do the absolute minimum provided by law, up to and including nothing**.

Disclaimer

"We don't claim **Interactive EasyFlow** is good for anything -- if you think it is, great, but it's up to you to decide. If Interactive EasyFlow doesn't work... **If you lose a million because Interactive EasyFlow messes up, it's you that's out the million, not us**. If you don't like this disclaimer... **We reserve the right to do the absolute minimum provided by law, up to and including nothing**.

This is basically the same disclaimer that comes with all software packages, but ours is in plain English and theirs is in legalese. We didn't really want to include any disclaimer at all, but our lawyers insisted. We tried to ignore them but they threatened us with the attack shark at which point we relented."

Usability

Usability

"extent to which a system, product or service can be used by **specified users** to achieve **specified goals** with *effectiveness*, *efficiency* and *satisfaction* in a specified **context of use**"

[ISO 9241-210:2010, 2.13]

https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en

It is a general definition, applicable to any artifact, even the simplest one, and allows us to **define measures**.

How usable is a product?

[.build-lists: true]

Usability

"extent to which a system, product or service can be used by **specified users** to achieve **specified goals** with *effectiveness*, *efficiency* and *satisfaction* in a specified **context of use**"

- I can see the display of my cell phone very well.
- If I am old and have problems seeing a small screen…
- If I'm chatting while standing up in a bus that runs along a dirt road
- ...

This '*multidimensional*' definition breaks down usability on three essentially independent variables: **effectiveness**, **efficiency** and **user satisfaction**.

These values can be measured.

Effectiveness: accuracy and completeness with which users achieve specific objectives.

It considers the "level of precision" with which the user can achieve his goals, measured numerically.

Question: DOES IT WORK?

Efficiency: amount of resources spent by the user to achieve his goal.

These resources can be quantified (time spent on the task, number of keys to be pressed, number of operations to be performed, etc.)

Question: HOW MUCH DOES IT COST?

^for example: the time taken to obtain a certain result, the number of keys to be pressed to perform a certain function, the number of operations of a certain type to be performed, etc.

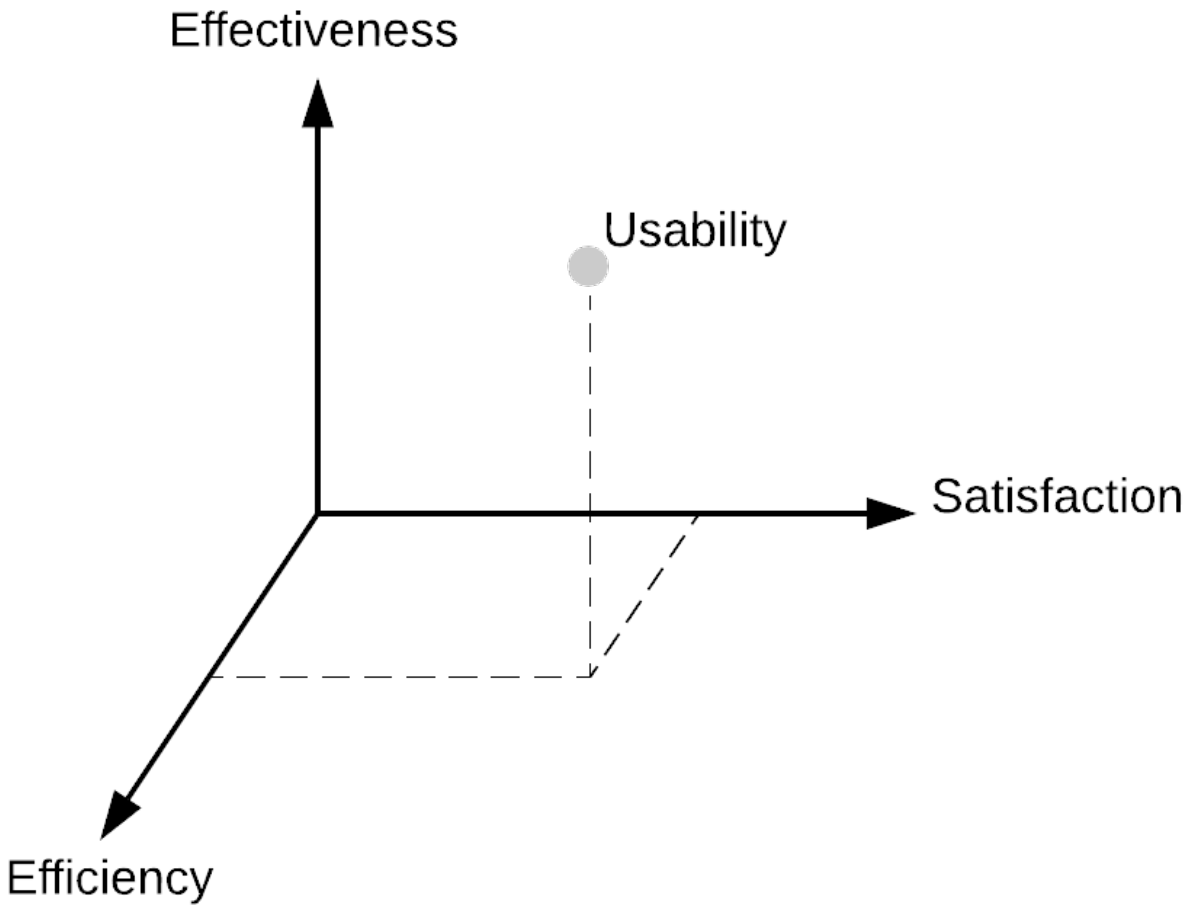
Satisfaction: Freedom from discomfort and positive attitude towards the use of the product.

We can measure it by means of questionnaires.

Question: DO I SUFFER MORE THAN NECESSARY?

This definition allows us to 'measure' usability using the values of **effectiveness**, **efficiency** and **satisfaction**.

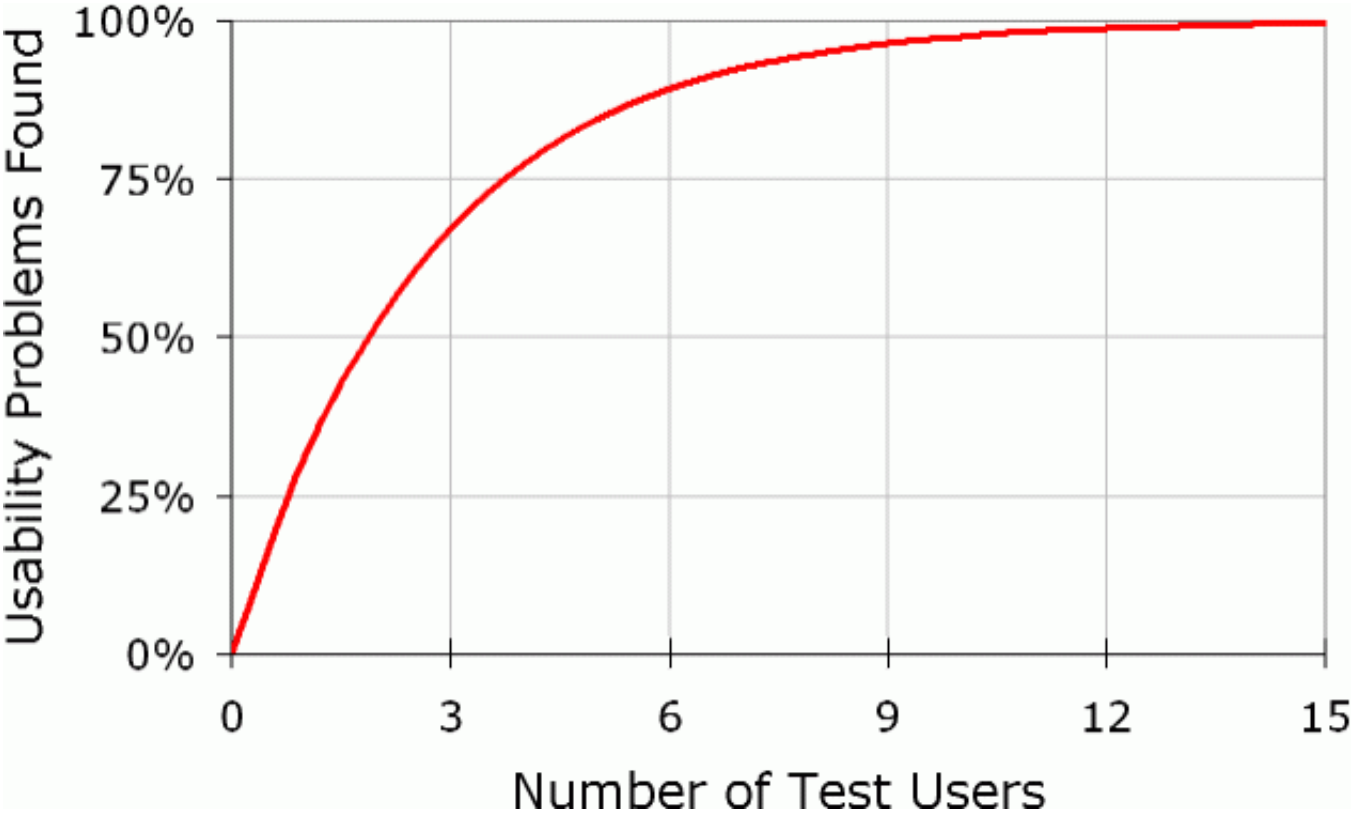
These quantities must be defined on a case-by-case basis, depending on the nature of the specific system.



KEY POINT: The measure of usability is related to **subjective factors** but **IT IS NOT ARBITRARY**.

How many users are needed to test the program in order to obtain reliable usability measures and limit subjective factors? One might think that a lot of users are needed to get reliable measurements. This idea is a disincentive to perform usability tests

This is false: major problems emerge even with few users!



J.Nielsen, T.K.Landauer, A Mathematical Model of the Finding of Usability Problems", in Proceedings of the ACM INTERCHI 93 conf., 1993.

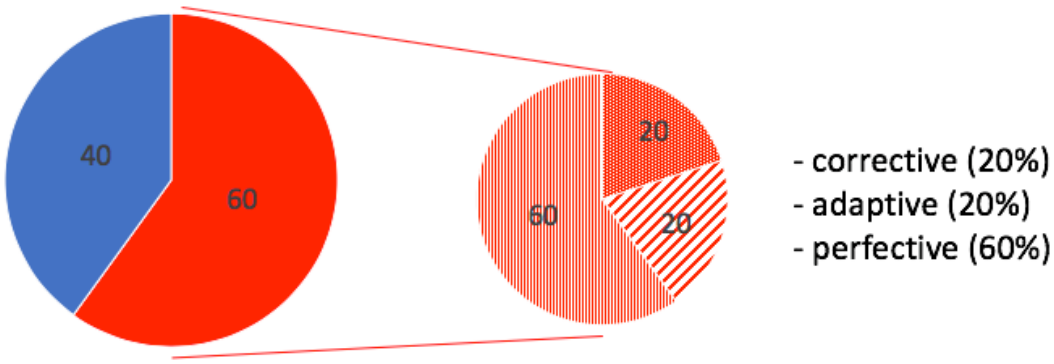
J.Nielsen, Why You Only Need to Test With 5 Users (2000) https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

Maintainability

Maintainability

Software maintenance regards changes made to a software **after its initial release**.

Maintenance costs exceed 60% of total costs of the software.



Corrective maintenance (20%): fix errors/bugs present when you release the software

Adaptive maintenance (20%) involves changes to the application in response to **changes in the environment**

- a new version of the hardware
- a new version of operating system
- a new DBMS
- ...

Perfective maintenance (60%) involves changes in software to **improve certain features or functionality**.

It is often necessary to

- modify the functions offered by the application,
- add new ones,
- improve performance,
- make the software easier to use,
- ...

Those who do not evolve, die [1]

Technopteryx,

Paolo Perelli, 2015

http://www.paoloperelli.com

Reusability

Reusability

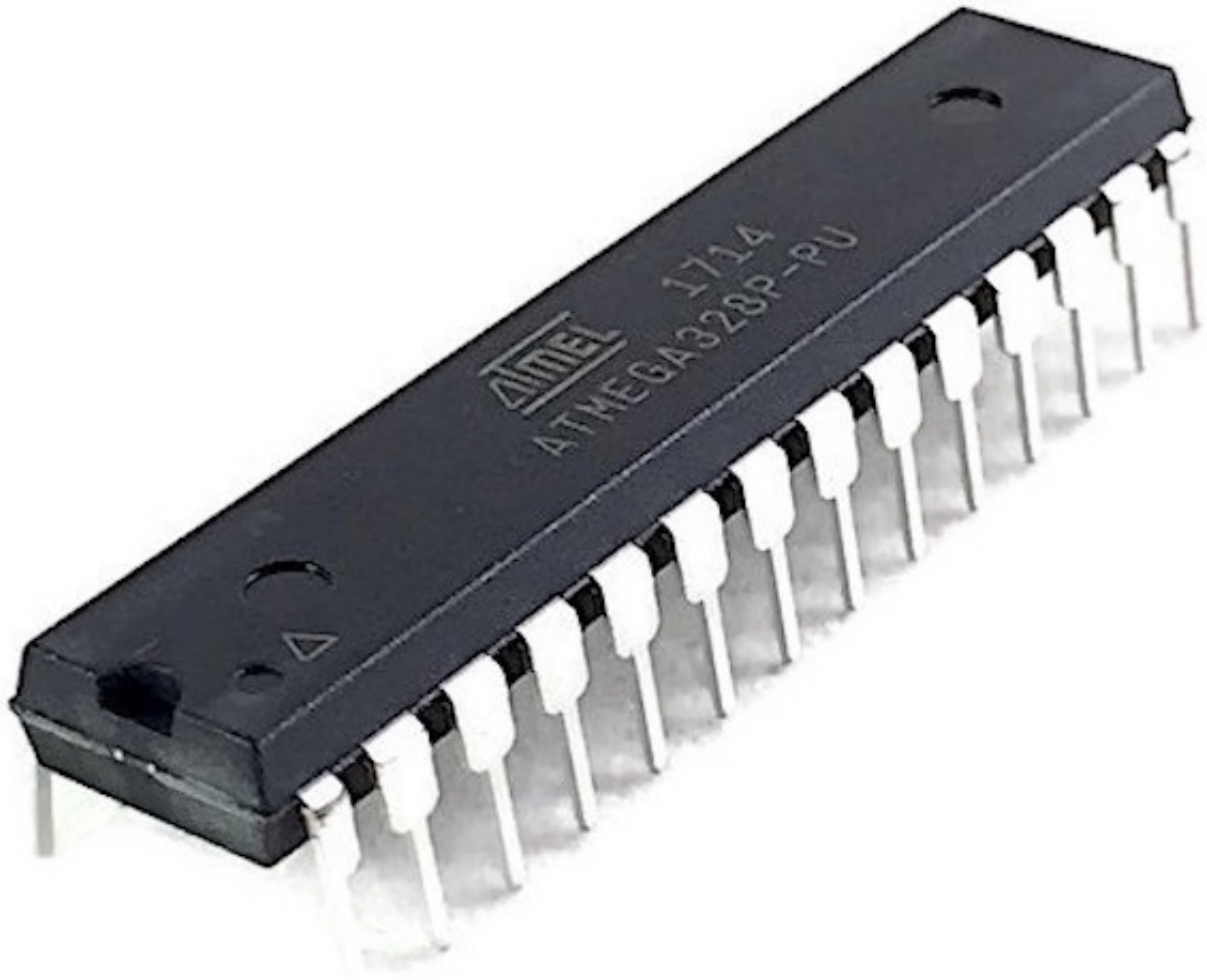
> ‘Mr. Winston Churchill, sir, to what do you attribute your success in life?’
>
> ‘*Economy of effort. Never stand up when you can sit down, and never sit down when you can lie down.*’

Reusability is the **use of existing parts** in some form within the software product development process - code - software components - test suites - designs - documentation.

Subroutines or functions are the simplest form of **reuse**.

Reusability

The reusability of standard parts characterizes the maturity of an industrial sector.



An integrated circuit can be used as a black box in different contexts (we are still far from this!)

The numerical libraries (initially in FORTRAN and now in C, C++) were the first examples of reusable components. We can use sotware components in the development of new software without having to reinvent or re-code well-known algorithms.

Reusability & OOP

One of the purposes of the OOP is to achieve better reusability.

Has OOP achieved its purpose?

All I need is to copy the Class from another project. But... This Class is related to other Classes and Objects... I need to copy all the hierarchy...

Not everyone agrees on the usefulness of the OOP to obtain reusable software components.

^ *"All I gotta do is simply grab that Class from the other project and use it. Not just that Class. We're gonna need the parent Class. We're going to need ALL of the parents. Now it won't compile. Why?? Oh, I see…

This object contains this other object. I need the object's parent and its parent's parent and so on and so on with every contained object and ALL the parents of what those contain along with their parent's, parent's, parent's... "

Goodbye, Object Oriented Programming (?)

"The problem with object-oriented languages is they've got all this implicit environment that they carry around with them. You wanted a banana but what you got was a gorilla holding the banana and the entire jungle." [2]

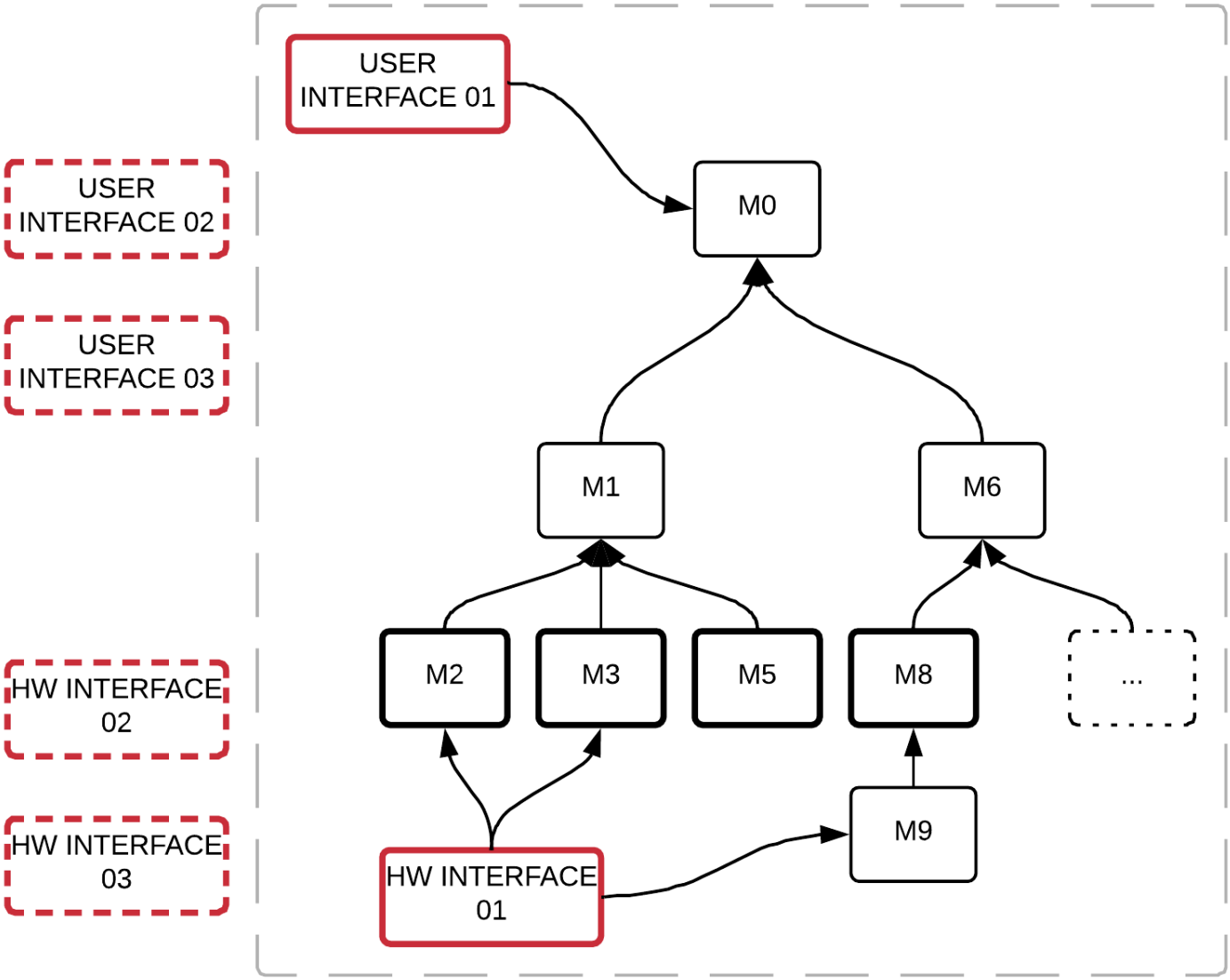
Portability

Portability

The software is portable if it can be executed in different environments (S.O, DBMS, user interface support system, etc.)

Portability can be achieved by **modularizing** the software.

Dependencies from the environment are isolated in a few modules, which can be modified when the software is transported in a different environment.



[.build-lists: true]

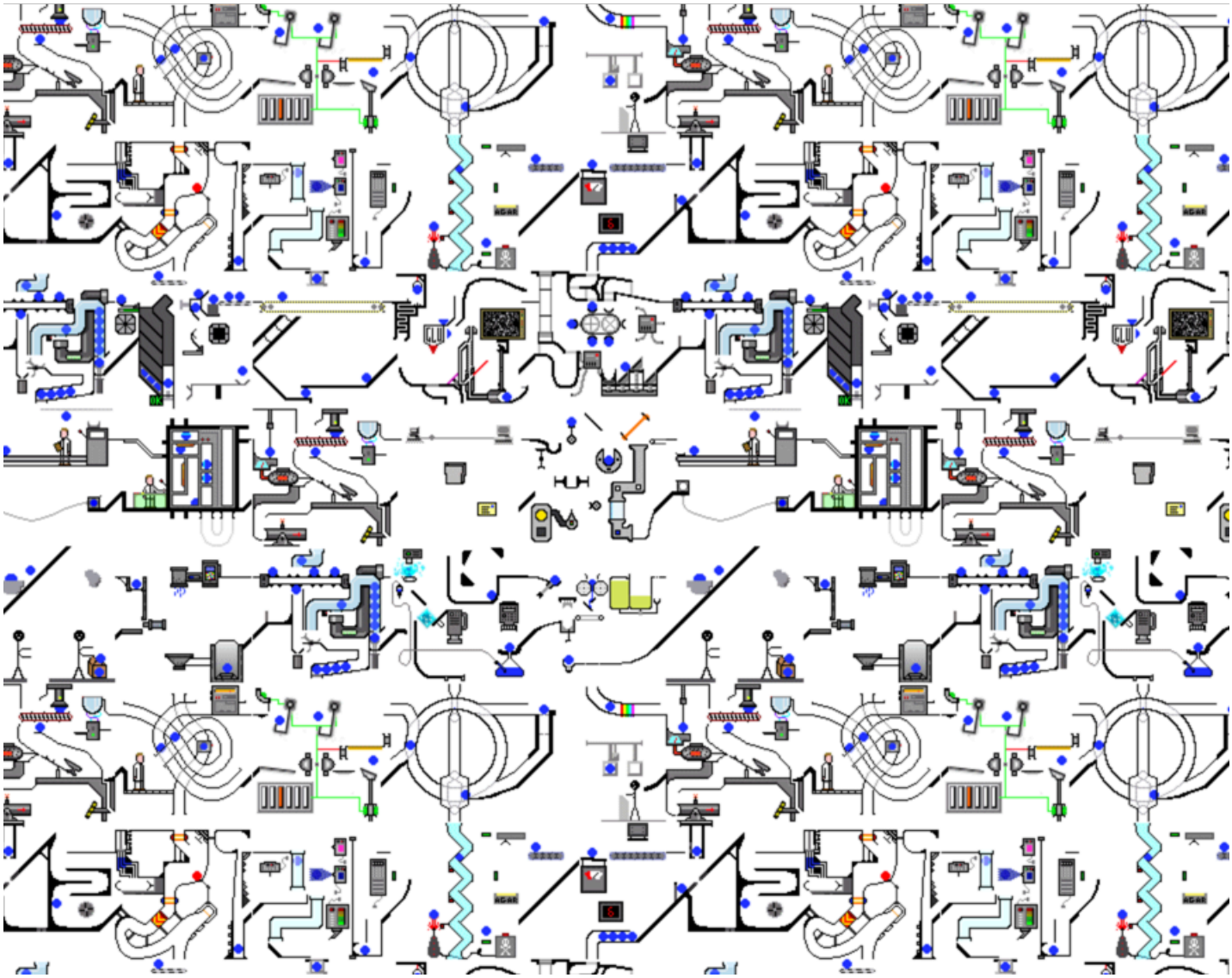
Comprehensibility

Some software systems are easier to understand than others. The comprehensibility of a software system depends on - **how the software is designed** - the problem faced by the software: *some problems are inherently more complex than others*.

Understanding of software is critical for its maintenance.

Modularity improve the comprehensibility of a system.

- Try to change a small feature or to fix a bug in a software that **you** wrote a month ago :scream:
- Try to change a small feature or to fix a bug in a software that **others** wrote a month ago :scream: :scream: :scream:



https://www.csee.umbc.edu/~squire/blueball.htm

Efficiency

Efficiency is related to the amount of resource (time, memory space) that the software needs to perform a task.

- Space complexity is a measure of the amount of **working storage** an algorithm needs
- Time complexity is the computational complexity that describes the **amount of time** it takes to run an algorithm

---

1. The Author provided a completely different interpretation of his work ↩

2. https://medium.com/@cscalfani/goodbye-object-oriented-programming-a59cda4c0e53 ↩