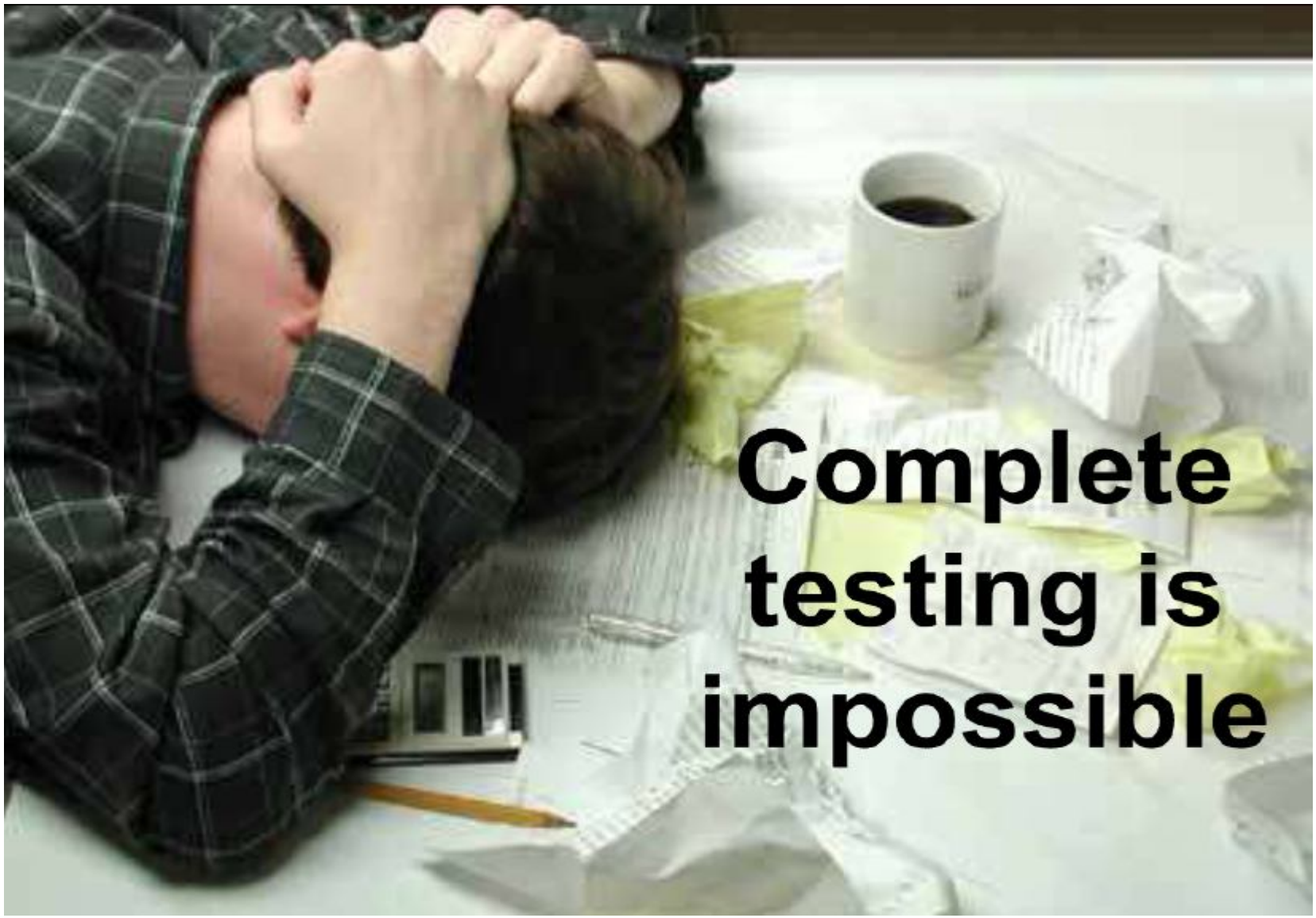
Software testing axioms

An Uncontested principle

An axiom is something that is impossible to prove to be true but could be disproved with a single experiment/example.

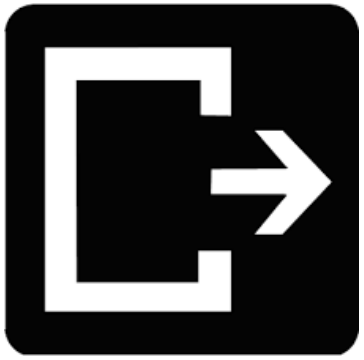
Software testing axioms

1. It is impossible to test a program completely.
2. Software testing is a risk-based exercise.
3. Testing cannot show the absence of bugs.
4. The more bugs you find, the more bugs there are.
5. Not all bugs found will be fixed.
6. It is difficult to say when a bug is indeed a bug.
7. Specifications are never final.
8. Software testers are not the most popular members of a project.
9. Software testing is a disciplined and technical profession.


Complete
testing is
impossible

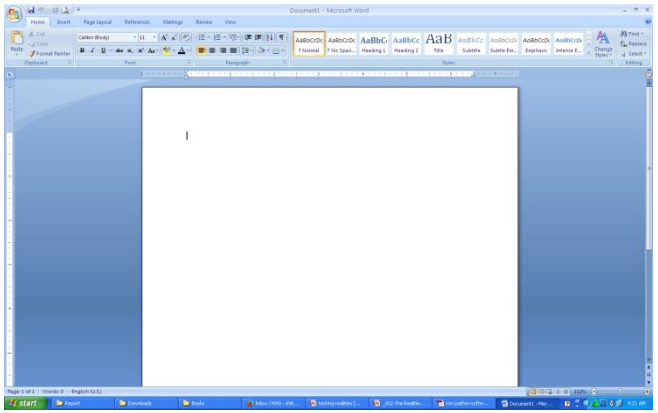
It's impossible to test a program completely

- There are the reasons below:
  - The number of possible inputs is very large.
  - The number of possible outputs is very large.
  - The number of paths through the software is very large.
  - The software specification is subjective.



- How many test cases do you need to exhaustively test:
  - Powerpoint
  - A calculator
  - MS Word
  - Any interesting software!

- The only way to be absolutely sure software works is to run it against all possible inputs and observe all of its outputs

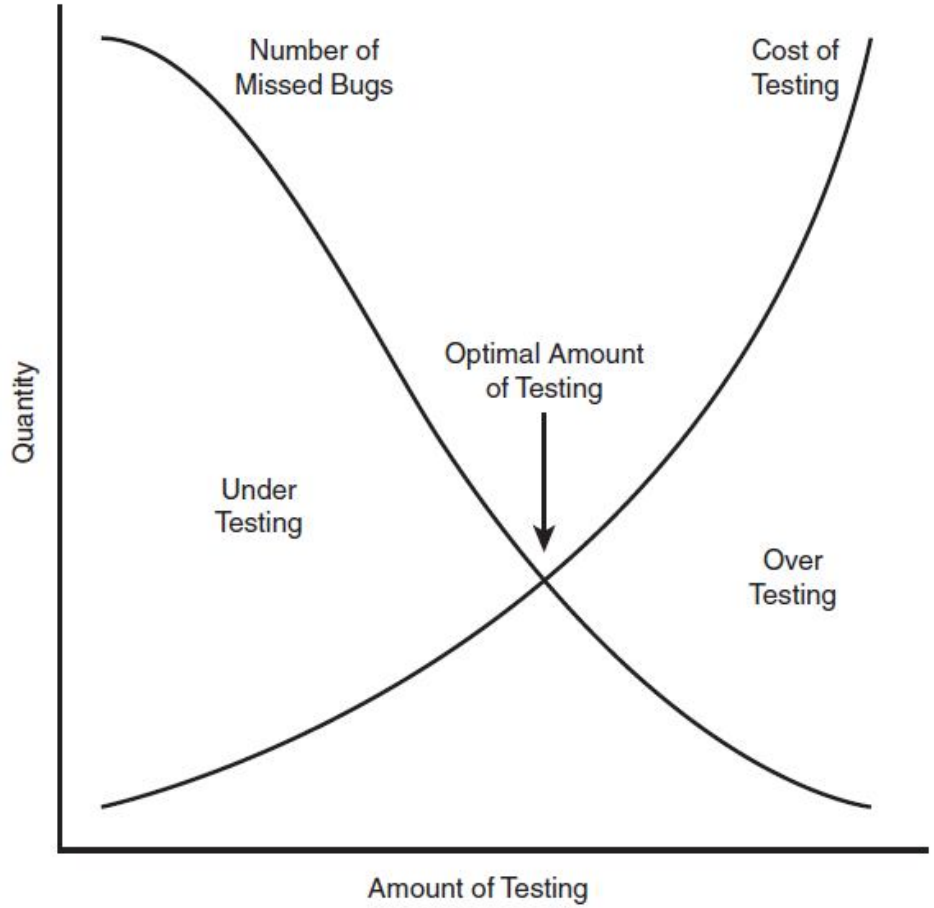



Software testing
is a risk-based
exercise

Software testing is a risk-bases exercise

- If you do not test the software for all inputs (a wise choice) you take a risk.
- Hopefully you will skip a lot of inputs that work correctly.
- What if you skip inputs that cause a fault?
  - Risk: financial loss, security, loss of money, loss of life!
  - That is a lot of pressure for a tester!
- If you try to test too much, the development cost becomes prohibitive.
- If you test too little, the probability of software failure increases
- software failures can cost us big time!

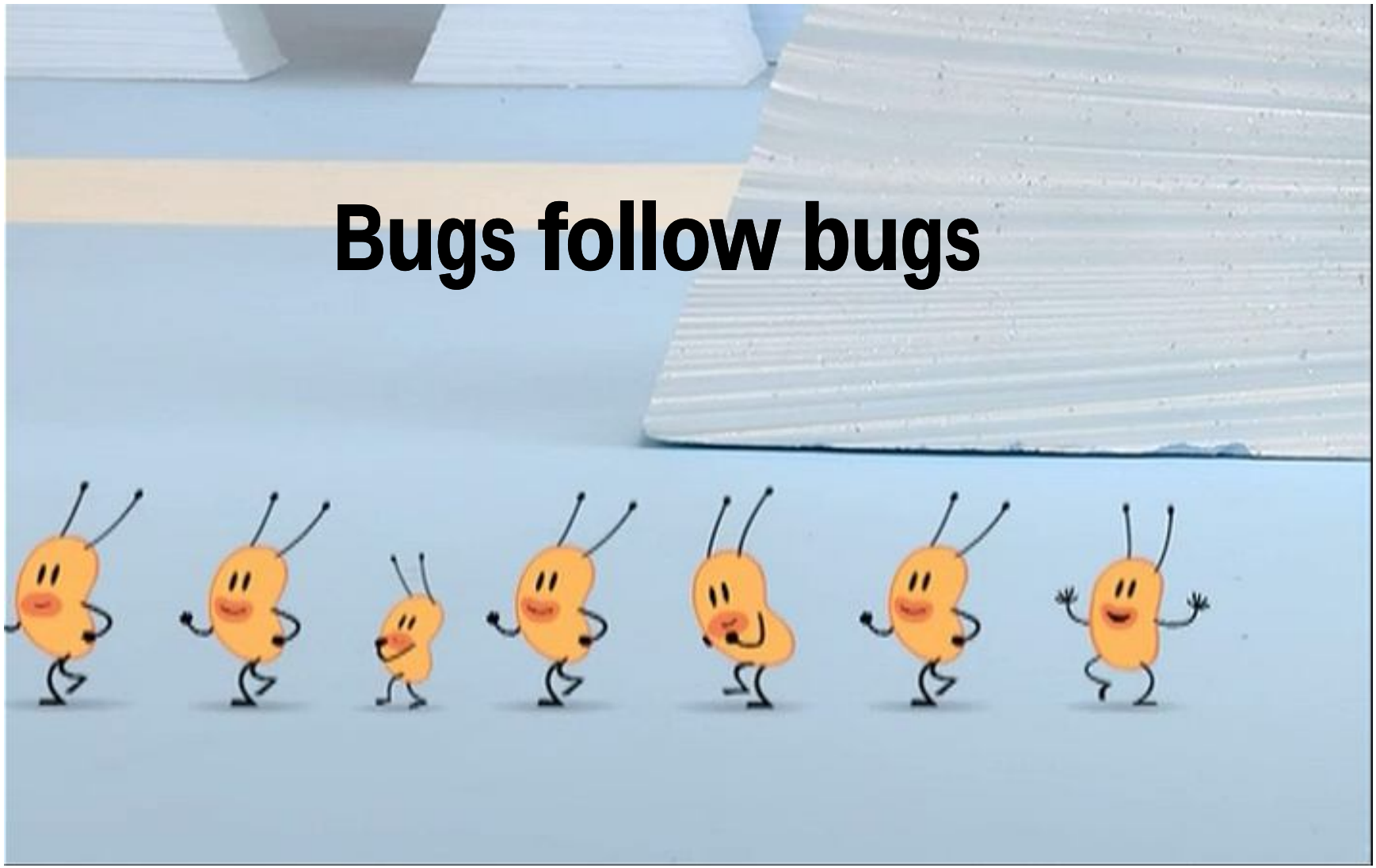
Software testing is a risk-bases exercise



Every software project has an optimal test effort

Testing can't show that bugs don't exist

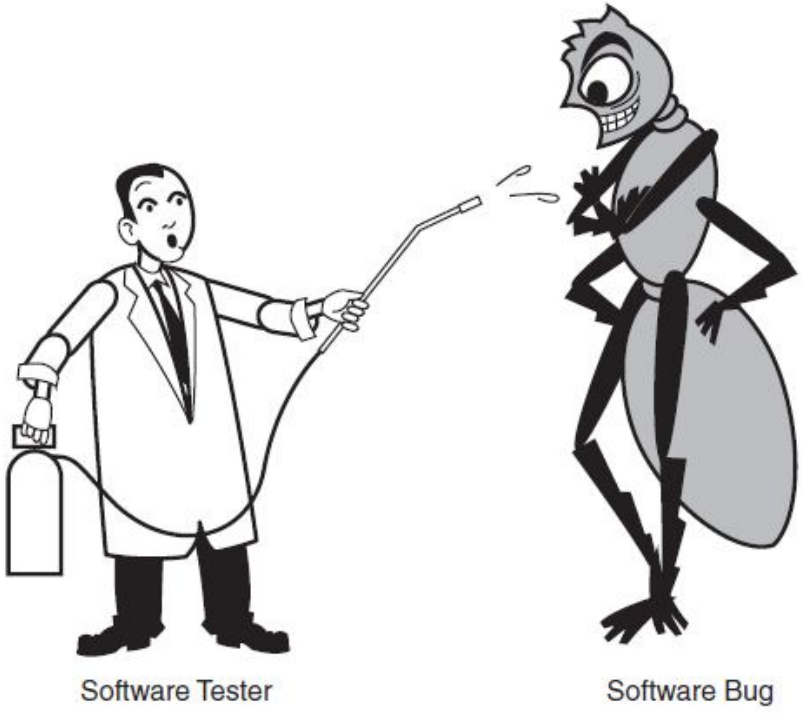
- Are there bugs in our software?
- Are there bugs in other software?
- Can you guarantee that there are no bugs to find?

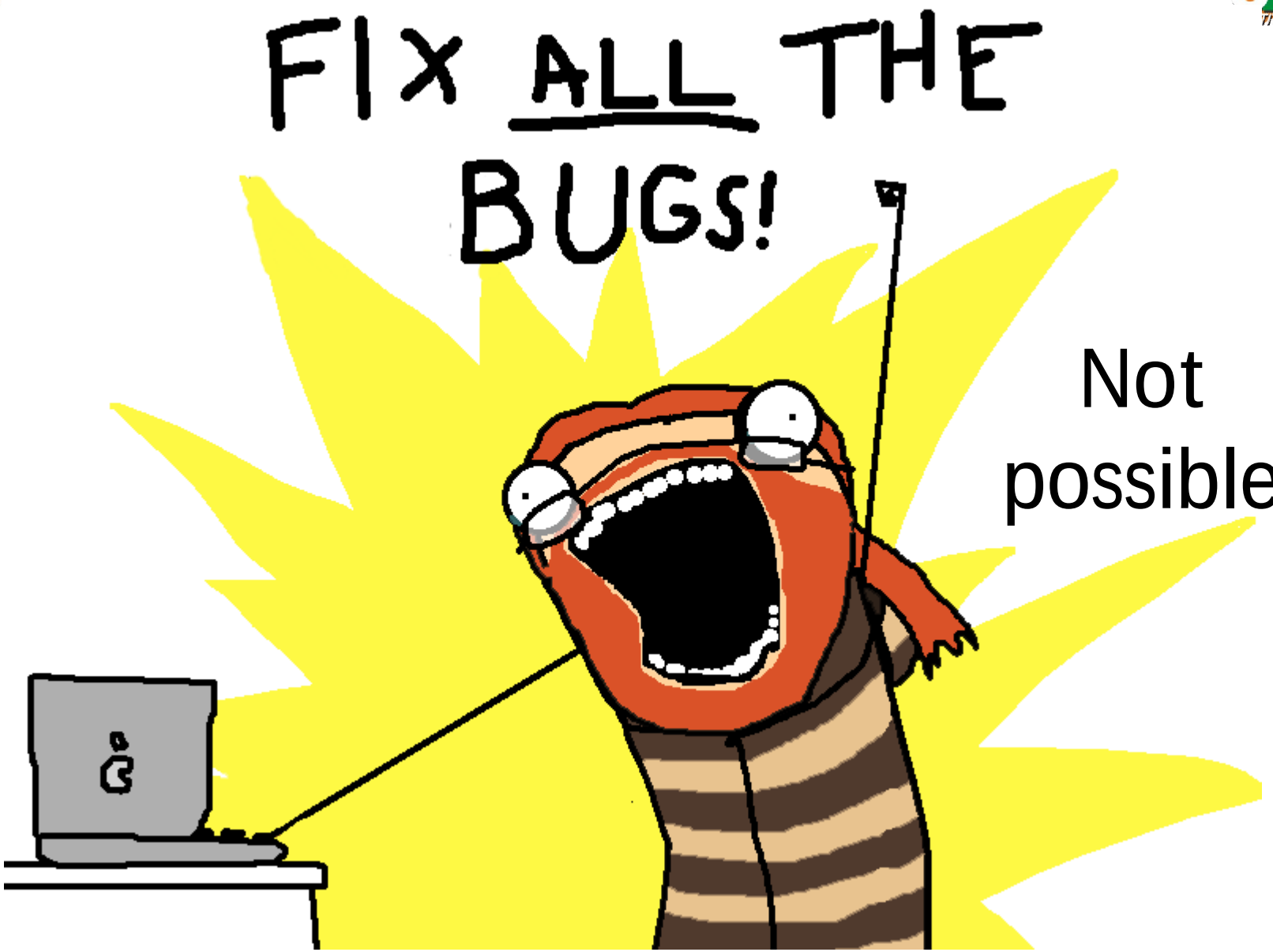

Bugs follow bugs

Bugs follow bugs

- Frequently: find one bug, then find other and more, Why?
  - Programmers have bad days
  - Programmers often make the same mistake
  - Some bugs are really just the tip of the iceberg

- **Boris Beizer coined the term pesticide paradox to describe the phenomenon that the more you test software the more immune it becomes to your test cases.**
  - **Remedy: continually write new and different tests to exercise different parts of the software**





Not possible

Not all bugs you find will be fixed

Why? .....There are several reasons :

- **There are not enough time.**
- **It is really not a bug.**
- **It is too risk to fix.**
- **It is just not worth it.**

There are not enough time

- In every project there are many software features:
  - Too few people to code and test.
  - Not enough room to left them
  - Must have software in time.

It is really not bug

- It’s common for misunderstandings
- Test errors
- Specification change to result

It is too risk to fix

- Software is fragile.
- Sometime is like spaghetti. (inter-twined)
- Under the pressure to release a product under tight schedule.

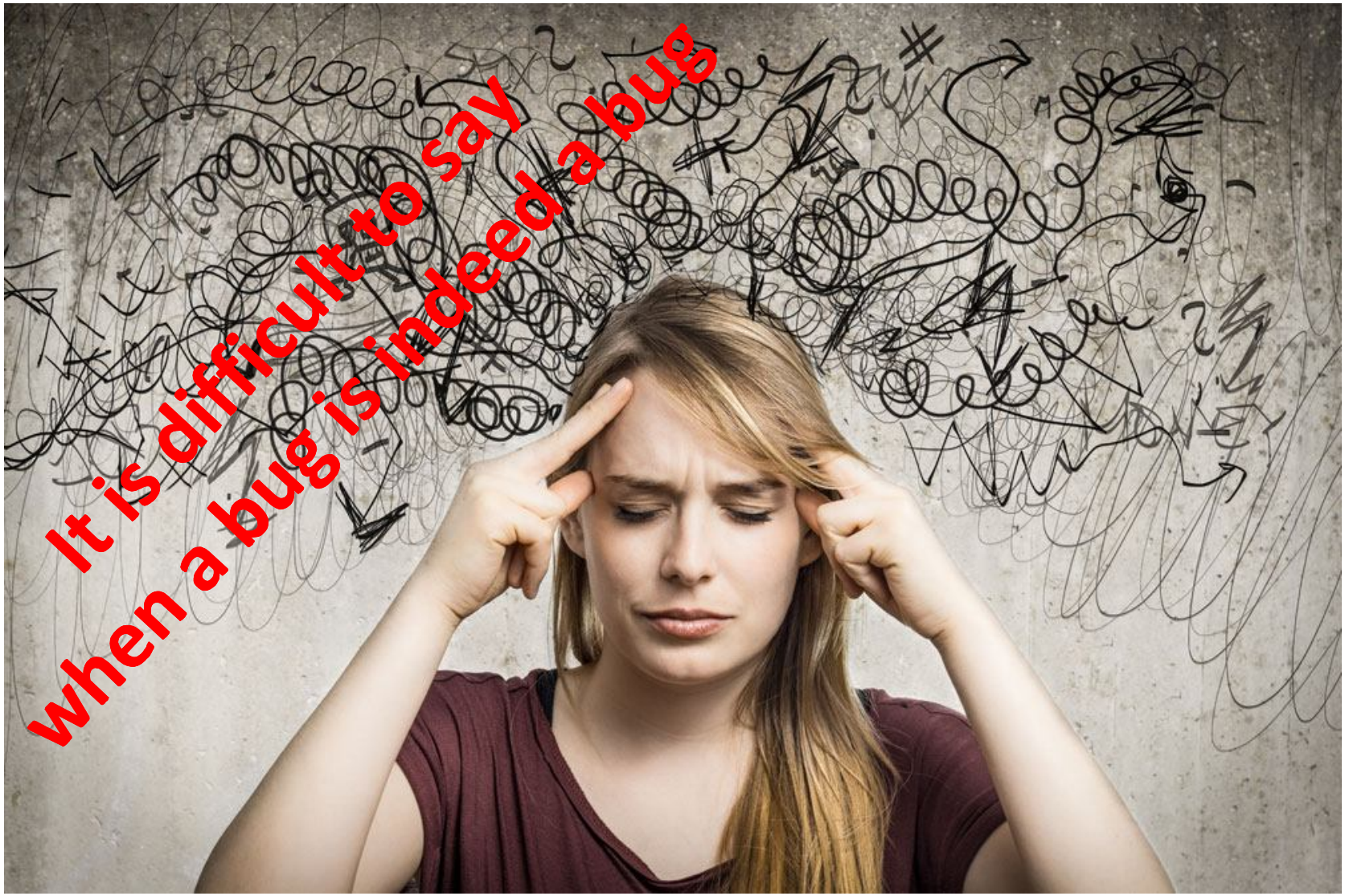
It may be better to leave in the know bug to avoid the risk of creating than unknown ones

It’s just not worth it.

- Bugs that would occur infrequently.
- Bugs that appear in little-used features.
- A user can prevent or avoid the bug.

What happens when you make the wrong decision?

The Intel Pentium test engineers **found bug before the chip was released**, but the product team decided that it was such small, rare bug that it **wasn't worth** fixing. They were under a tight schedule and decided to meet their current deadline and fix the bug in later releases of the chip. Unfortunately, the bug was **discovered and rest.**


It is difficult to say
when a bug is indeed a bug

It is difficult to say when a bug is indeed a bug

- If there is a problem in the software but no one ever discovers it ... is it a bug?
  - Parody of “if a tree falls in the forest ... does it really make a noise?”
- What is your opinion? Does a bug have to be observable in order for it to me a bug?
- Bugs that are undiscovered are called latent bugs.

Specifications are never final

Specifications are never final

- **Building a product based on a "moving target" specification is fairly unique to software development.**
  - **Competition is fierce**
  - **Very rapid release cycles**
  - **Software is "easy" to change**
- **Not true in other engineering domains**
  - **E.g., the Brooklyn Bridge could not be adjusted to allow train traffic to cross it once its construction started.**

Software testers are not the most popular members of a project

- **Goal of a software tester:**
  - Find bugs
  - Find bugs early
  - Make sure bugs get fixed

- **Tips to avoid becoming unpopular:**
  - Find bugs early
  - Temper your enthusiasm ... act in a professional manner
  - Don’t report just the bad news

Software testing is a disciplined and technical profession

- When software was simpler and more manageable software testers were often untrained and testing was not done methodically.
  - Contrary to the urban legend, if you hire a million moneys and have them test for a million years ... you will not find all of the bugs in your software.
- It is now too costly to build buggy software. As a result testing has matured as a discipline.
  - Sophisticated techniques
  - Tool support
  - Rewarding careers

Thank you