Engineer

Defect

a flaw that keeps a product from meeting its requirements

What it is

A defect is something wrong inside a piece of work that makes the work behave differently from what it was meant to do, or leave out part of what it was meant to do. Most programmers call it a bug, and testers and standards also write fault or flaw.

The important word there is *work*. A defect can live in source code, and it can just as well live in a requirement that contradicts another requirement, a design that cannot carry the load it was drawn for, a configuration file, a test that asserts the wrong value, or a manual that describes a button the product dropped two releases ago. Wherever it lives, a defect belongs to the thing itself, and it is present whether or not anybody has noticed it.

Testing vocabulary keeps three neighbouring words apart, and the separation earns its keep. A person makes an error: a misread specification, a wrong assumption, a slip of the keyboard. The error leaves a defect behind in whatever that person produced. When the defective part runs under conditions that expose it, the product behaves wrongly, and that visible wrong behaviour is a failure. The chain runs in one direction and can stop at any link. A defect in a branch that never executes, or one that needs an input nobody sends, can sit in a codebase for years without causing a single failure.

What counts as wrong depends on what was expected, and expectations reach further than the written specification. The product quality model in ISO/IEC 25010 describes functional suitability as the degree to which a product “provides functions that meet stated and implied needs when used under specified conditions”. It splits that into completeness (do the functions cover the specified tasks and the intended users’ objectives), correctness (are the results accurate) and appropriateness (do the functions help get the task done). A wrong total is a

correctness problem; a report nobody can export is a completeness problem. The same model treats reliability as a separate characteristic, with faultlessness, availability, fault tolerance and recoverability among its parts, so a product that computes the right answer and then crashes twice a day still has defects.

A regression is a defect with a history: something that worked and was broken by a later change. It earns its own name because it brings a clue that an original defect lacks, a last version known to be good.

Use the concept in a decision

Put “What is a defect?” to work in a real decision. We can help relate the concept to the system you are building or buying. One team taking an idea from a sketch to something your customers can log into, instead of handing it between three suppliers.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Led a GIS map application to 10x revenue growth — 10x revenue growth, led from the map product

Led a GIS map application to 10x revenue growth Get in touch →

Why it matters

The cost of a defect depends mostly on where it is found. Caught by the person writing the code, it costs a minute and leaves no trace. Caught in review, it costs a conversation. Caught by a customer, it costs the failure itself, the support ticket, the investigation, the fix and some trust, and if it did real harm it has become part of an incident. Type systems, linters, tests and code review all work on the same line, trying to move the moment of discovery closer to the moment the defect was written.

Defects also come in families. The mistake behind one usually has relatives: the same misread interface called the same way in six places, the same unit confusion copied into every new module, the same unchecked assumption in each request handler. The productive question after a defect is less where the broken line is than what made that line easy to write, and where else the same thing happened.

And defects come back. Code changes constantly, and a repaired defect can return through a refactor, a merge or a dependency upgrade. A test written to fail on the defect and pass on the repair turns the repair into a lasting property of the build. When a regression appears anyway, the version history becomes a

search space: `git bisect` runs a binary search over the commits between a known good and a known bad version, so a range of a thousand commits narrows to the one that introduced the bug in about ten test runs.

Some defects are security problems as well. MITRE's Common Weakness Enumeration catalogues weaknesses, each of which it describes as "a condition in a software, firmware, hardware, or service component that, under certain circumstances, could contribute to the introduction of vulnerabilities". Plenty of entries on that list are ordinary defects that an attacker has a use for: a missing bounds check, user input passed unescaped into a query, an authorisation step skipped on one route.

What it looks like in practice

The defects that cost most on this site were the ones every automatic signal called fine. An audit turned up eleven accessibility defects that the automated rules had passed, because each was a property of how a page rendered rather than of what its markup said. Two of them sat in the checks themselves. Every window the layout check opened was 900 pixels tall, so a phone held sideways at 852 by 393, with seven pixels between two items, had never been looked at; and the rule engine had been running 70 of its 105 rules. Each defect went into a ledger together with the rule it produced, so the work closes a class of mistake and not just one occurrence. Once the layout check itself was repaired, it failed 128 times across three languages.

A second defect produced no symptom at all. The site has a dark theme by default and a light theme for readers whose system asks for one, and its full-page cover is built from three layers in the dark theme. A small structural check that counts the layers each theme paints found the light theme drawing only two on its very first run. The overlay had been absent since the light theme was written. Nothing threw an error and every page rendered, and a background with one layer fewer passes for a deliberate choice, so for months there was no reason to look. The repair was one rule; the check is what keeps it repaired.

Keeping repairs in place is the job of the repository's guard engine, where one rule ties every bug fix to evidence: a commit marked as a fix has to bring a test that fails without it. The guards also have 146 tests of their own, because a faulty guard can let through the changes it was meant to reject and appear to be working.

Where it goes wrong

The occurrence gets fixed and the class stays open. The reported line is corrected, the ticket closes, and three siblings of the same mistake keep waiting elsewhere in the code. After each fix, search for the pattern that produced it, and where a machine can recognise the pattern, turn the search into a check.

A repaired bug returns. A refactor half a year later restores the old behaviour, because nothing recorded what the repair was protecting. Write the test before the fix, watch it fail on the old code, and keep it in the suite for good.

The checker carries the defect. A rule set that quietly loads a fraction of its rules, a list of test windows that all share one height, an assertion that checks what the code does instead of what it should do: each reports success while the defect sits in plain view. Give checks tests of their own, and read from the tool's own output which rules actually ran.

Coverage stands in for correctness. A suite can execute every line and still pass a wrong result when no assertion looks at the outcome, and code that meets its written specification can still miss an implied need. Use coverage to find code that nothing tests, and judge a test by the defects it would catch.

"Works as designed" closes a real report. The behaviour matches the specification, the report is rejected, and the defective work product was the specification all along. When the requirement or the design is where the problem lives, file the defect against that, so the repair happens at the source.

Further reading

- Defect — the ISTQB definition used in software testing, with its synonyms and the related terms error and failure.
- ISO/IEC 25010 — an overview of the product quality model and each characteristic's sub-characteristics, functional suitability and reliability among them.
- git-bisect — Git's manual for the binary search that finds the commit that introduced a bug, including automated runs with a test script.
- About CWE — what the Common Weakness Enumeration is, how its list of software and hardware weaknesses is organised and who runs it.

Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is a defect?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/defect/