Engineer

Defekt

ein Mangel, durch den ein Produkt Anforderungen verfehlt

Was es ist

Ein Defekt ist ein Mangel in einem Arbeitsergebnis, durch den es sich anders verhält als vorgesehen oder einen Teil seiner vorgesehenen Aufgabe nicht erfüllt. Die meisten Programmierer nennen ihn Bug; Tester und Normen verwenden auch die englischen Bezeichnungen fault oder flaw.

Das entscheidende Wort ist *Arbeitsergebnis*. Ein Defekt kann im Quellcode stecken, ebenso aber in einer Anforderung, die einer anderen widerspricht, einem Entwurf, der seiner vorgesehenen Last nicht standhält, einer Konfigurationsdatei, einem Test, der einen falschen Wert erwartet, oder einer Anleitung, die eine vor zwei Releases entfernte Schaltfläche des Produkts beschreibt. Wo auch immer er steckt: Ein Defekt gehört zur Sache selbst und ist vorhanden, ob ihn schon jemand bemerkt hat oder nicht.

Die Terminologie des Softwaretests hält drei benachbarte Begriffe auseinander, und diese Trennung ist nützlich. Ein Mensch begeht eine Fehlhandlung: eine falsch gelesene Spezifikation, eine falsche Annahme, ein Tippfehler. Diese Fehlhandlung hinterlässt einen Defekt in seinem Arbeitsergebnis. Wird der defekte Teil unter Bedingungen ausgeführt, die den Defekt offenlegen, verhält sich das Produkt falsch. Dieses sichtbare Fehlverhalten ist eine Fehlerwirkung. Die Kette verläuft in eine Richtung und kann an jedem Glied enden. Ein Defekt in einem nie ausgeführten Zweig oder einer, der eine nie gesendete Eingabe benötigt, kann jahrelang in einer Codebasis liegen, ohne eine einzige Fehlerwirkung auszulösen.

Was als falsch gilt, hängt von den Erwartungen ab, und Erwartungen reichen über die schriftliche Spezifikation hinaus. Das Produktqualitätsmodell von ISO/IEC 25010 beschreibt funktionale Eignung als den Grad, in dem ein Produkt „Funktionen bereitstellt, die bei Verwendung unter festgelegten Bedingungen

die ausdrücklich genannten und impliziten Bedürfnisse erfüllen“. Es unterteilt dies in Vollständigkeit (decken die Funktionen die festgelegten Aufgaben und die Ziele der vorgesehenen Nutzer ab), Korrektheit (sind die Ergebnisse genau) und Angemessenheit (helfen die Funktionen bei der Erledigung der Aufgabe). Eine falsche Summe ist ein Korrektheitsproblem; ein Bericht, den niemand exportieren kann, ist ein Vollständigkeitsproblem. Dasselbe Modell behandelt Zuverlässigkeit als eigenes Merkmal, zu dessen Bestandteilen Fehlerfreiheit, Verfügbarkeit, Fehlertoleranz und Wiederherstellbarkeit gehören. Ein Produkt, das richtig rechnet und dann zweimal täglich abstürzt, hat also weiterhin Defekte.

Eine Regression ist ein Defekt mit Vorgeschichte: Etwas hat funktioniert und wurde durch eine spätere Änderung beschädigt. Sie trägt einen eigenen Namen, weil sie einen Hinweis mitbringt, der einem ursprünglichen Defekt fehlt: die letzte als funktionierend bekannte Version.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist ein Defekt?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Ein Team führt eine Idee von der Skizze bis zu einem Produkt, bei dem Ihre Kunden sich anmelden können, statt sie zwischen drei Dienstleistern weiterzureichen.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird.
GIS-Kartenprodukt zu 10-fachem Umsatz geführt — 10x Umsatzwachstum durch die Leitung des Kartenprodukts

GIS-Kartenprodukt zu 10-fachem Umsatz geführt Kontakt aufnehmen →

Warum es wichtig ist

Die Kosten eines Defekts hängen vor allem davon ab, wo er entdeckt wird. Entdeckt ihn die Person, die den Code schreibt, kostet er eine Minute und hinterlässt keine Spur. Im Review kostet er ein Gespräch. Entdeckt ihn ein Kunde, kostet er die Fehlerwirkung selbst, das Supportticket, die Untersuchung, die Korrektur und etwas Vertrauen. Hat er echten Schaden verursacht, ist er Teil eines Vorfalls geworden. Typsysteme, Linter, Tests und Code-Reviews arbeiten alle in dieselbe Richtung: Sie sollen den Entdeckungszeitpunkt näher an den Zeitpunkt bringen, an dem der Defekt entstand.

Defekte treten auch in Familien auf. Der Irrtum hinter einem hat meist Verwandte: dieselbe falsch verstandene Schnittstelle, die an sechs Stellen gleich aufgerufen wird, dieselbe Einheitenverwechslung, die in jedes neue Modul

kopiert wird, dieselbe ungeprüfte Annahme in jedem Request-Handler. Die produktive Frage nach einem Defekt lautet weniger, wo die kaputte Zeile steht, als vielmehr, wodurch sich diese Zeile so leicht schreiben ließ und wo dasselbe noch passiert ist.

Und Defekte kommen zurück. Code ändert sich ständig, und ein behobener Defekt kann durch ein Refactoring, einen Merge oder ein Abhängigkeitsupdate zurückkehren. Ein Test, der mit dem Defekt fehlschlägt und mit der Korrektur besteht, macht die Korrektur zu einer dauerhaften Eigenschaft des Builds. Tritt trotzdem eine Regression auf, wird die Versionshistorie zum Suchraum: `git bisect` führt eine binäre Suche über die Commits zwischen einer als funktionierend und einer als fehlerhaft bekannten Version durch. So lässt sich ein Bereich von tausend Commits in etwa zehn Testläufen auf den Commit eingrenzen, der den Bug eingeführt hat.

Manche Defekte sind zugleich Sicherheitsprobleme. MITREs Common Weakness Enumeration katalogisiert Schwachstellenarten, die sie jeweils als „einen Zustand in einer Software-, Firmware-, Hardware- oder Dienstkomponente, der unter bestimmten Umständen zur Entstehung von Sicherheitslücken beitragen könnte“ beschreibt. Viele Einträge dieser Liste sind gewöhnliche Defekte, die ein Angreifer nutzen kann: eine fehlende Bereichsprüfung, nicht maskierte Nutzereingaben in einer Abfrage, ein auf einer Route übersprungener Autorisierungsschritt.

Wie es in der Praxis aussieht

Die teuersten Defekte auf dieser Website waren jene, die alle automatischen Signale für unbedenklich hielten. Ein Audit deckte elf Barrierefreiheitsdefekte auf, die die automatisierten Regeln nicht beanstandet hatten, weil jeder davon eine Eigenschaft der gerenderten Seite statt ihres Markups war. Zwei steckten in den Prüfungen selbst. Jedes vom Layoutcheck geöffnete Fenster war 900 Pixel hoch. Ein quer gehaltenes Telefon mit 852 mal 393 Pixeln und sieben Pixeln Abstand zwischen zwei Elementen war deshalb nie geprüft worden. Zudem hatte die Regel-Engine nur 70 ihrer 105 Regeln ausgeführt. Jeder Defekt wurde zusammen mit der daraus entstandenen Regel in einem Register festgehalten. So schließt die Arbeit eine Fehlerklasse und korrigiert nicht nur ein Vorkommen. Nachdem der Layoutcheck selbst repariert war, schlug er über drei Sprachen hinweg 128-mal fehl.

Ein zweiter Defekt verursachte überhaupt kein Symptom. Die Website hat standardmäßig ein dunkles Design und ein helles für Leser, deren System es anfordert. Ihr seitenfüllender Hintergrund besteht im dunklen Design aus drei Schichten. Eine kleine Strukturprüfung zählt die von jedem Design gezeichneten Schichten und fand beim ersten Lauf im hellen Design nur zwei. Das Overlay hatte seit der Einführung des hellen Designs gefehlt. Nichts löste einen Fehler aus, und jede Seite wurde gerendert. Ein Hintergrund mit einer Schicht weniger wirkt wie eine bewusste Entscheidung, weshalb monatelang kein Anlass zum Nachsehen bestand. Die Korrektur war eine einzige Regel; die Prüfung sorgt dafür, dass sie Bestand hat.

Korrekturen dauerhaft zu sichern ist die Aufgabe der Guard-Engine des Repositorys. Eine ihrer Regeln verknüpft jede Fehlerkorrektur mit einem Nachweis: Ein als Fix gekennzeichneter Commit muss einen Test mitbringen, der ohne die Korrektur fehlschlägt. Die Guards haben zudem 146 eigene Tests. Denn ein fehlerhafter Guard kann Änderungen durchlassen, die er ablehnen sollte, und dabei den Anschein erwecken, dass er funktioniert.

Wo es schiefgeht

Das Vorkommen wird behoben, die Fehlerklasse bleibt offen. Die gemeldete Zeile wird korrigiert, das Ticket geschlossen, und drei verwandte Fehler warten anderswo im Code weiter. Suchen Sie nach jeder Korrektur nach dem zugrunde liegenden Muster. Wo eine Maschine es erkennen kann, machen Sie aus der Suche eine Prüfung.

Ein behobener Bug kehrt zurück. Ein Refactoring stellt ein halbes Jahr später das alte Verhalten wieder her, weil nirgends festgehalten wurde, was die Korrektur schützen sollte. Schreiben Sie den Test vor der Korrektur, vergewissern Sie sich, dass er mit dem alten Code fehlschlägt, und behalten Sie ihn dauerhaft in der Testsuite.

Das Prüfwerkzeug trägt den Defekt. Ein Regelsatz, der stillschweigend nur einen Teil seiner Regeln lädt, eine Liste von Testfenstern mit durchgehend gleicher Höhe, eine Zusicherung, die das tatsächliche statt des vorgesehenen Codeverhaltens prüft: Alle melden Erfolg, während der Defekt offen sichtbar bleibt. Geben Sie Prüfungen eigene Tests und lesen Sie in der werkzeugeigenen Ausgabe nach, welche Regeln tatsächlich liefen.

Abdeckung ersetzt Korrektheit. Eine Testsuite kann jede Zeile ausführen und ein falsches Ergebnis dennoch durchlassen, wenn keine Zusicherung das Ergebnis prüft. Und Code, der seine schriftliche Spezifikation erfüllt, kann trotzdem ein implizites Bedürfnis verfehlen. Nutzen Sie Abdeckung, um ungetesteten Code zu finden, und beurteilen Sie einen Test danach, welche Defekte er erkennen würde.

„Funktioniert wie vorgesehen“ erledigt eine berechtigte Meldung. Das Verhalten entspricht der Spezifikation, die Meldung wird zurückgewiesen, und das defekte Arbeitsergebnis war von Anfang an die Spezifikation. Liegt das Problem in der Anforderung oder im Entwurf, erfassen Sie den Defekt dort, damit die Korrektur an der Quelle erfolgt.

Weiterführende Lektüre

- Defekt — die im Softwaretest verwendete ISTQB-Definition samt Synonymen und den verwandten Begriffen Fehlhandlung und Fehlerwirkung.
- ISO/IEC 25010 — ein Überblick über das Produktqualitätsmodell und die Teilmerkmale jedes Merkmals, darunter funktionale Eignung und Zuverlässigkeit.
- git-bisect — das Git-Handbuch zur binären Suche nach dem Commit, der einen Bug eingeführt hat, einschließlich automatisierter Läufe mit einem Testskript.
- Über CWE — was die Common Weakness Enumeration ist, wie ihre Liste von Software- und Hardwareschwachstellenarten aufgebaut ist und wer sie betreibt.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist ein Defekt?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/defect/