SUCHEN SIE DIE RICHTIGEN FEHLER?
WARUM ES SICH LOHNT, FEHLER ZU KATEGORISIEREN

Für einen effizienten Einsatz von Qualitätssicherungsmaßnahmen ist die Kenntnis über die häufigsten Fehler entscheidend. Dieser Artikel stellt eine neue Klassifikation von Fehlerursachen für objektorientierte Software vor und beschreibt, basierend auf einer Umfrage, die Häufigkeiten verschiedener Fehlerkategorien sowie die Korrelation dieser Fehlerhäufigkeiten mit Einflussfaktoren, wie z. B. der Programmiersprache und der Branche. Daraus werden praktische Hinweise zur Gestaltung des Testprozesses und zu den einzusetzenden Qualitätssicherungsmaßnahmen abgeleitet.

Tests, Reviews und Codeanalyse sind die am häufigsten eingesetzten Verfahren, um Softwarefehler in einem Produkt aufzudecken. Um einen effizienten und effektiven Einsatz dieser Verfahren zu erreichen, sollten Art und Wahrscheinlichkeit der zu erwartenden Fehler bekannt sein. Mit anderen Worten: Sie sollten beim Testen wissen, was Sie suchen, bevor Sie sich entscheiden, wie Sie suchen. Sind z. B. viele Fehler im Kontrollfluss zu erwarten, sollten Sie ein Testverfahren wie die Anweisungs- und Zweigüberdeckung anwenden. Treten Datenflussfehler sehr häufig auf, ist ein auf der Datenflussanalyse basierendes Verfahren das Mittel der Wahl (vgl. [Spi05]). Das Wissen über die Häufigkeit von Fehlerursachen hilft Ihnen auch, gezielt konstruktive Maßnahmen für die Vermeidung dieser Fehler ergreifen zu können.

Die vorhandene Literatur zum Thema Softwarefehler (z. B. [Bei90], [Bin00], [Fen97], [Mye04], [Rop94]) liefert allerdings meist nur unzureichende Daten über Fehlerhäufigkeiten. Darüber hinaus decken die darin untersuchten Systeme und Fehlerkategorien objektorientierte Software nicht ausreichend ab. Wir haben daher eine neue Klassifikation von Fehlerursachen in objektorientierten Systemen erarbeitet und stellen diese hier vor. Sie können diese Fehlerkategorisierung entweder unverändert oder in firmenspezifisch angepasster Form nutzen. Das versetzt Sie in die Lage, Fehlerursachen in der Fehlerdatenbank Ihres Unternehmens selber differenziert zu erheben, zu messen und zu bewerten, um so beim Testen gezielt die wichtigsten Fehlerkategorien anzugehen.

Außerdem finden Sie in diesem Artikel die Ergebnisse einer Online-Umfrage unter Testern und Entwicklern zur Häufigkeit von Softwarefehlern. Dabei wird deutlich, wie unterschiedlich Fehlerhäufigkeiten je nach Kontext (z. B. Branche und genutzte Programmiersprache) ausfallen können. Basierend auf den Ergebnissen der Studie geben wir Ihnen praktische Hinweise, wie Sie den Testprozess am Fehlerrisiko orientiert gestalten und Qualitätssicherungsmaßnahmen effektiv einsetzen können. Abschließend fassen wir den Nutzen der Fehlerkategorisierung zusammen.

Fehlerkategorien für objektorientierte Software

Eine Fehlerkategorisierung sollte umfassend sein, gleichzeitig aber auch übersichtlich und handhabbar. Wir haben daher eine hierarchische Fehlerkategorisierung entwickelt und uns dabei bewusst auf zwei Ebenen beschränkt. Die obere Hierarchieebene besteht aus sieben Hauptkategorien:

- Die Kategorie *Fehler durch falsche Interpretation der Anforderungen* umfasst alle Fehler, die aus einem falschen Verständnis der Anforderungen herrühren.
- *Fehler im Kontrollfluss* betreffen die Steuerung des dynamischen Ablaufs des Systems (oder von Teilen davon).

die autoren

Lars Borner (E-Mail: lars.borner@informatik.uni-heidelberg.de) ist wissenschaftlicher Mitarbeiter an der Universität Heidelberg und forscht auf dem Gebiet des Integrationstestens.

Lars Ebrecht (E-Mail: Lars.Ebrecht@dlr.de) ist wissenschaftlicher Mitarbeiter am Deutschen Zentrum für Luft- und Raumfahrt e.V. (DLR) und forscht im Bereich Validierung, Verifikation und Testautomatisierung von Hard- und Softwarekomponenten.

Dr. Stefan Jungmayr (E-Mail: jungmayr@testbarkeit.de) ist als Teilprojektleiter bei einem Automobilzulieferer für die Entwicklung von Steuergeräte-Diagnose-Software verantwortlich.

Dr. Matthias Hamburg (E-Mail: matthias.hamburg@sogeti.de) ist Managing Consultant bei der Sogeti Deutschland GmbH mit dem Schwerpunkt Test-Management und -Prozessberatung.

Prof. Dr. Mario Winter (E-Mail: mario.winter@fh-koeln.de) lehrt und forscht auf den Gebieten Softwareentwicklung und Projektmanagement mit dem Schwerpunkt Softwarequalität am Institut für Informatik der FH Köln.

1	**Fehlerhafte Interpretation der Anforderungen**	
	1.1	Fehlende Implementierung einer expliziten Anforderung
	1.2	Fehlende Implementierung einer impliziten Anforderung
	1.3	Implementierung ohne Vorliegen einer Anforderung
	1.4	Kundenanforderung falsch verstanden
	1.5	Fehler in Ausgabepräsentation
2	**Fehler im Kontrollfluss**	
	2.1	Fehler in Prädikat
	2.2	Fehler in Selektionskonstrukt
	2.3	Fehler in Schleifenkonstrukt
	2.4	Fehler in Zustandslogik
	2.5	Fehler in Behandlung von internem Ausnahmefall
	2.6	Methode auf falschem Objekt aufgerufen
	2.7	Falscher Methodenaufruf durch Polymorphismus bzw. Überladen
	2.8	Falsche Methode ausgewählt
	2.9	Fehler bei parallelen Prozessen
	2.10	Fehler in deklarativer Kontrollflusssteuerung
3	**Fehler in der Berechnung**	
	3.1	Falscher Algorithmus ausgewählt
	3.2	Fehler in Arithmetik
	3.3	Fehler in Zeigerarithmetik
	3.4	Fehler in String-Manipulation
	3.5	Fehler in Initialisierung
	3.6	Fehler in Ressourcen-Verwendung
4	**Fehler in Klassen oder Datentypen**	
	4.1	Fehlerhafte Deklaration eines Attributs bzw. einer Variable
	4.2	Fehler in Typtransformation
	4.3	Fehler durch Mehrfachvererbung
	4.4	Methode nicht implementiert/überschrieben
	4.5	Fehlerhafte Deklarierung einer Methode / eines Attributes als statisch bzw. dynamisch.
5	**Fehler im Datenfluss/Objektzugriff**	
	5.1	Datenflussanomalie
	5.2	Deadlock
	5.3	Falsche Zugriffsart
	5.4	Fehler durch Umgehung des Zugriffsschutzes
6	**Schnittstellenfehler**	
	6.1	Aufgerufene Klasse implementiert Schnittstelle nicht bzw. fehlerhaft
	6.2	Fehler in übergebenen Parametern
	6.3	Fehler in der Interpretation von Rückgabewerten
	6.4	Aufruf bzw. Aufrufreihenfolge der Methode falsch
	6.5	Nichtbeachtung von Kontrakten
7	**Konfigurationsfehler**	
	7.1	Aufgerufene Komponente fehlt oder liegt in falscher Version vor
	7.2	Fehler in Deployment-Deskriptoren
	7.3	Umgebungsvariablen falsch gesetzt oder ignoriert
	7.4	Testcode unbeabsichtigt im operativen System enthalten
	7.5	Eigenschaften der Laufzeitumgebung(en) nicht beachtet.

Tabelle 1: Fehlerkategorisierung.

- *Berechnungsfehler* beziehen sich auf die Berechnung und Manipulation von Variablen, Attributwerten und Ausdrücken.
- *Fehler in Klassen oder Datentypen* umfassen die fehlerhafte Deklaration/Transformation des Typs bzw. der statischen Struktur von Klassen/Datentypen.
- *Fehler im Datenfluss bzw. Objektzugriff* beinhalten die fehlerhafte Reihenfolge von Attribut-/Objektzugriffen, Fehler durch ungeeignete Objektzustände sowie Fehler durch dynamische objektübergreifende Abläufe.
- *Schnittstellenfehler* entstehen durch die falsche Verwendung einer Klassen- bzw. Objektschnittstelle.
- *Konfigurationsfehler* werden als eine falsche Konfiguration des Builds bzw. des laufenden Systems definiert.

Auf der nachfolgenden Ebene wird jede Hauptkategorie durch vier bis zehn Fehler-Unterkategorien verfeinert (siehe Tabelle 1). Die Fehlerkategorien orientieren sich an der einschlägigen Literatur sowie an unseren eigenen Erfahrungen aus Entwicklungsprojekten. Eine detaillierte Beschreibung der Fehlerkategorien finden Sie in [Bor06].

Der Fokus unserer Untersuchung lag vor allem auf Fehlern im Programm-Quelltext. Fehler in der Dokumentation und im Entwicklungsprozess sowie Fehler bezüglich nicht-funktionaler Qualitätskriterien wie Benutzbarkeit, Änderbarkeit oder Zuverlässigkeit wurden nicht in die Kategorisierung mit einbezogen. Das bedeutet natürlich nicht, dass diese Fehlerkategorien z. B. in Ihrem Firmenkontext zu vernachlässigen sind.

Häufigkeiten der Fehlerkategorien

Um eine Vorstellung der Häufigkeiten in der Praxis zu gewinnen, haben wir im Herbst 2005 eine Online-Umfrage durchgeführt, an der 1.219 Personen teilgenommen haben. Die Teilnehmer haben die Häufigkeit einer Fehlerkategorie anhand der Ordinal-Skala „nie", „selten", „oft" und „sehr oft" bewertet. Als Hilfe zur Bewertung wurden zu dieser Skala vergleichbare Intervalle relativer

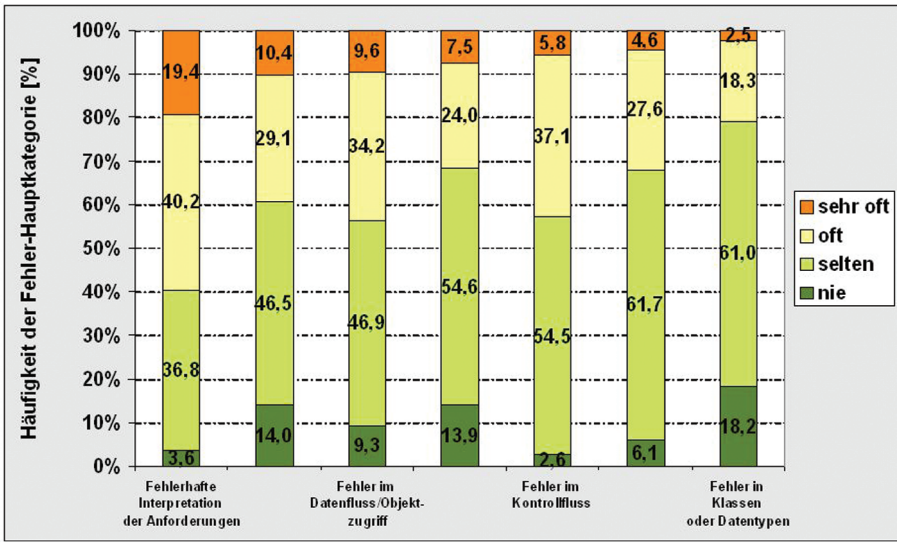

Abb. 1: Häufigkeit der Fehler-Hauptkategorien.

Fehlerhäufigkeit in konkreten Prozentwerten angegeben (vgl. [Bor06]). Zusätzlich zu den sieben vordefinierten Fehler-Hauptkategorien konnten die Teilnehmer eine eigene Fehler-Hauptkategorie und deren Häufigkeit angeben.

Die nachfolgende Liste sowie Abbildung 1 beschreiben die Verteilung der Antworten zu den Fehler-Hauptkategorien, sortiert in absteigender Anzahl der Antwort „sehr oft".

- *Fehlerhafte Interpretation der Anforderungen:* Diese Fehler-Hauptkategorie wurde um ein Vielfaches häufiger mit „sehr oft" bewertet als alle anderen Kategorien. Als häufigste Fehler-Unterkategorien wurden dabei die fehlende Implementierung von impliziten Anforderungen sowie falsch verstandene Kundenanforderungen genannt.
- *Konfigurationsfehler:* Bemerkenswert ist, dass Konfigurationsfehler bezüglich der Antwortkategorie „sehr oft" bereits an zweiter Stelle genannt wurden – noch vor den traditionellen Fehlerquellen, wie Fehler im Datenfluss bzw. Objektzugriff oder Schnittstellenfehler. Am häufigsten bewerteten die Teilnehmer die Fehler-Unterkategorien „Fehlen der aufgerufenen Komponente" sowie „Vorliegen einer falschen Version" mit „sehr oft".
- *Fehler im Datenfluss bzw. Objektzugriff:* Am häufigsten wurde die Fehler-Unterkategorie Datenflussanomalien mit „sehr oft" bewertet.
- *Schnittstellenfehler:* Am häufigsten wurde mit „sehr oft" die Nichtbeachtung von Kontrakten bewertet.
- *Fehler im Kontrollfluss:* Fehler im Kontrollfluss, die insbesondere in der prozeduralen Entwicklung die größte Aufmerksamkeit durch Testtechniken erfahren haben (siehe z. B. [Bei90]), lagen bezüglich ihrer Bedeutung nur im Mittelfeld. Von den Unterkategorien wurden Fehler in der Behandlung von internen Ausnahmefällen (*Exceptions*) und in parallelen Prozessen am häufigsten mit „sehr oft" bewertet.
- *Berechnungsfehler:* Die Berechnungsfehler, auch im Fokus traditioneller Testtechniken, lagen in der Bedeutung noch stärker zurück. Häufigste Fehler-Unterkategorie bezüglich der Nennung „sehr oft" waren Fehler in der Ressourcenverwendung.
- *Fehler in Klassen oder Datentypen:* Diese Fehler-Hauptkategorie wurde insgesamt am schwächsten bewertet. Die dabei noch am stärksten bewertete Fehler-Unterkategorie betrifft die Typtransformation.

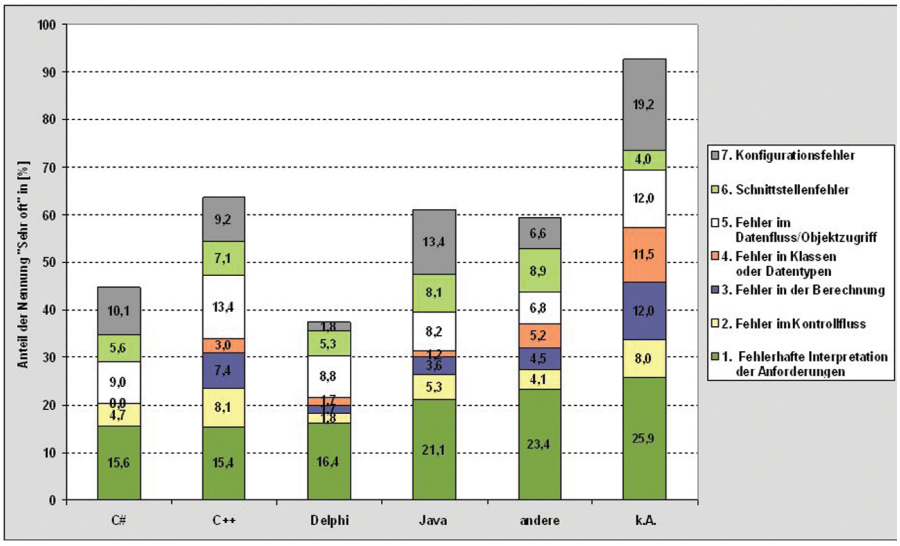

Abb. 2: Häufigkeit der Nennung „sehr oft" pro Fehler-Hauptkategorie und Programmiersprache.

Bei der Auswertung der durch die Teilnehmer selber ergänzten Fehler-Hauptkategorien wurden keine neuen nennenswerten Fehlerkategorien zur Korrektheit des Codes aufgedeckt. Häufig bezogen sich die zusätzlich angegebenen Fehlerkategorien auf Fehlerursachen, die explizit aus der Umfrage herausgehalten worden waren, da sie nicht im Fokus der Untersuchung standen, z. B. Fehler im Handbuch oder in der Dokumentation sowie fehlerhafte Projektplanung. Das deutet darauf hin, dass die hier vorgestellte Fehlerkategorisierung die potenziell in der Praxis auftretenden Fehler weitgehend abdeckt.

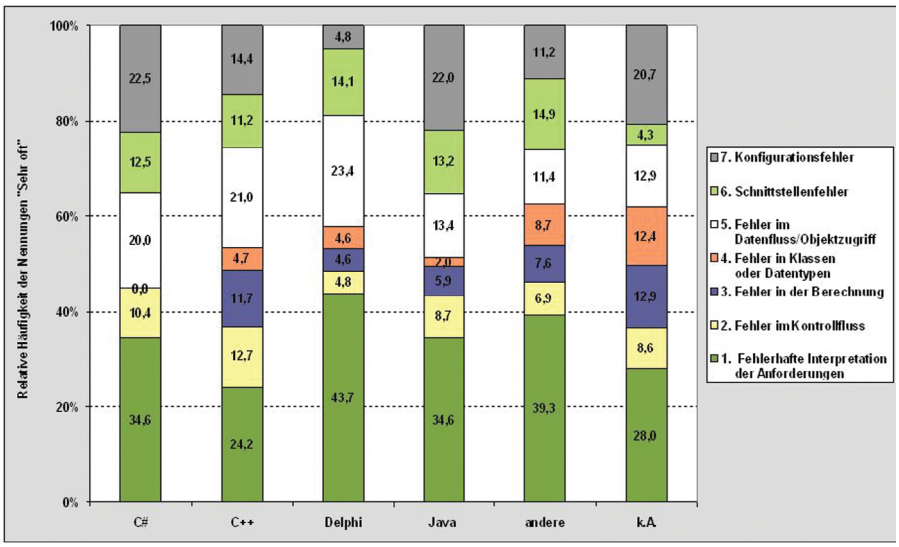

Abb. 3: *Relative Häufigkeit der Antwort „sehr gut" pro Fehler-Hauptkategorie und Programmiersprache.*

Eine detaillierte Beschreibung der Umfragedaten sowie der Auswertung mit Datentabellen und zusätzliche Grafiken finden Sie in [Bor06].

Fehlerkategorien und Kontextfaktoren

Für den effektiven Einsatz von Qualitätssicherungsmaßnahmen ist es sehr wichtig, diese auf die Häufigkeiten der erwarteten Fehler abzustimmen. Damit stellt sich die folgende Frage: Kann man eine Verteilung von Fehlerhäufigkeiten (wie sie z. B. im vorhergehenden Abschnitt vorgestellt wurde) für sein eigenes Projekt einfach „kopieren" oder variieren die Häufigkeiten der Fehlerkategorien je nach Projektkontext? Dieser Abschnitt beschreibt die in der Umfrage beobachteten Unterschiede der Häufigkeiten bezüglich der Kontextfaktoren „Programmiersprache" und „Branche" sowie „Arbeitsrolle der Teilnehmer".

Hinsichtlich der statistischen Auswertung der Ergebnisse unterscheiden wir zwischen signifikanten Korrelationen (die ein Indiz für ein Ursache-Wirkungs-Verhältnis sein können) sowie Tendenzen, d. h. Unterschieden in der Bewertung, die aufgrund der (für bestimmte Kombinationen aus Fehlerkategorien und Kontextfaktoren) teilweise zu kleinen Stichprobengröße statistisch nicht ausreichend signifikant sind.

Die Programmiersprache

Die in den Projekten der Teilnehmer am häufigsten eingesetzten Programmiersprachen sind Java (41%) und C++ (28%), gefolgt von C# (7%), Delphi (5%) und anderen (19%). **Abbildung 2** zeigt den jeweiligen Anteil der Antwort „sehr oft" pro Programmiersprache und Fehler-Hauptkategorie in Prozent[1]), gruppiert nach der Programmiersprache. Dabei fällt auf, dass die Summe der Nennung „sehr oft" für die Programmiersprachen deutlich ungleich ist.

Insgesamt – also über alle Fehler-Hauptkategorien hinweg – wurde die Antwort „sehr oft" bei C++ (63,6%) und Java (61,0%) etwa gleich häufig genannt, bei den Sprachen C# (44,9%) und Delphi (37,4%) weniger häufig. Da die bei Java häufiger genannten anforderungsbezogenen Fehler nicht aus spezifischen Eigenschaften der Programmiersprache resultieren (sondern vermutlich eher aus einer Korrelation mit der Branche), schneidet Java im Vergleich zu C++ bei den verbleibenden Fehlerkategorien besser ab.

Im Folgenden beschränken wir uns auf die explizit benannten Programmiersprachen:

Zwischen *anforderungsbezogenen Fehlern* und der genutzten objektorientierten Programmiersprache konnte keine signifikante Korrelation festgestellt werden. Nur bei Fehlern in der Ausgabepräsentation gibt es tendenziell Unterschiede.

Bei *Fehlern im Kontrollfluss* gibt es tendenzielle Unterschiede in der Bewertung mehrerer Fehler-Unterkategorien (bei „Fehler in Zustandslogik", „Methode auf falschem Objekt aufgerufen", „falscher Methodenaufruf durch Polymorphismus bzw. Überladen" und „Fehler bei parallelen Prozessen").

Die Bewertung von *Berechnungsfehlern* unterscheidet sich signifikant für unterschiedliche Programmiersprachen und tritt insgesamt überproportional bei C++ auf (7,4%). So wurden Fehler in der Ressourcenverwendung bei C++ zu 15,5% mit „sehr oft" bewertet, bei Java hingegen nur zu 5,7%. Fehler in der String-Manipulation wurden hingegen bei C# (8,3%) häufiger mit „sehr oft" bewertet als bei C++ (5,6%). Weitere, tendenzielle Unterschiede traten bei den Fehler-Unterkategorien „falscher Algorithmus ausgewählt" und „Fehler in Zeigerarithmetik" auf.

Bei *Fehlern in Klassen oder Datentypen* traten tendenziell unterschiedliche Bewertungen bei den Fehler-Unterkategorien „fehlerhafte Deklaration eines Attributs bzw. einer Variable", „Fehler in Typtransformation" und „Fehler durch Mehrfachvererbung" auf.

Fehler im Datenfluss bzw. Objektzugriff wurden insgesamt am stärksten in C++ (13,4%) und am schwächsten in Java (8,2%) bewertet. Bei den Unterkategorien unterscheiden sich die Bewertungen von Datenfluss-Anomalien und *Deadlocks* signifikant für die verschiedenen Programmiersprachen. Datenfluss-Anomalien wurden bei C# (14,5%) und bei Java (14,2%) häufiger mit „sehr oft" bewertet als bei C++ (12,7%) und Delphi (11,1%). Bei *Deadlocks* ist die Spanne besonders groß: Bei C# wurde „sehr oft" kein einziges Mal genannt, bei C++ zu 4,4%[2]). Nur tendenzielle Unterschiede traten bei der Fehler-Unterkategorie „falsche Zugriffsart" auf.

Schnittstellenfehler treten insgesamt bei den in der Umfrage explizit abgefragten Programmiersprachen am häufigsten in Java (8,1%) und C++ (7,1%) „sehr oft" auf. Bei

[1]) Pro Fehlerkategorie kann die Antwort „sehr oft" maximal zu 100 % genannt werden. Bei sieben Fehlerkategorien kann die Summe aller Nennungen „sehr oft" in der gestapelten Darstellung des Diagramms daher maximal 700 % ausmachen.

[2]) Dies entspricht der Erwartung angesichts der unterschiedlichen Unterstützung der Deadlock-Vermeidung in den Programmiersprachen bis hin zur Semaphor-Klasse in C#.

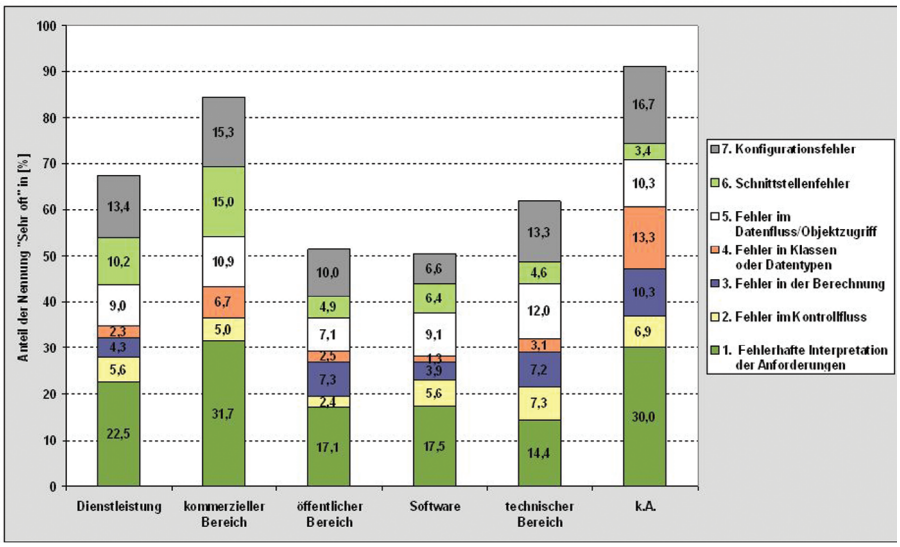

Abb. 4: Häufigkeit der Nennung „sehr oft" pro Fehler-Hauptkategorie und Branche.

den Unterkategorien wurden Fehler in übergebenen Parametern signifikant unterschiedlich bewertet: „sehr oft" kommt häufiger bei C# (5%) und Java (4,4%) vor, seltener bei Delphi (3,1%) und C++ (2,9%).

Konfigurationsfehler treten tendenziell je nach Programmiersprache unterschiedlich auf: Die Antwort „sehr oft" wurde am häufigsten bei Java genannte (13,4%), gefolgt von C# (10,%) und C++ (9,2%), hingegen selten bei Delphi (1,8 %). Auch bei einzelnen Unterkategorien gab es unterschiedliche Tendenzen: Fehler betreffend die Umgebungsvariablen wurden besonders häufig in Java mit „sehr oft" bewertet (10,3%), Fehler in Deployment-Deskriptoren besonders häufig in Java (8,7%) und C# (6,7%).

Aus Sicht der Autoren ist die Bewertung der Fehler-Hauptkategorie „fehlerhafte Interpretation der Anforderungen" nicht von der Programmiersprache abhängig, sondern eher von Faktoren wie der Branche. Für andere Kategorien, wie z. B. Konfigurationsfehler oder die Ressourcen-Verwendung, gibt es diese Abhängigkeit mutmaßlich schon. Ein Paradebeispiel ist dabei die Häufung von Zeigerarithmetik-Fehlern in C++ (siehe Abb. 3). Die Wahl der Programmiersprache hat also insgesamt

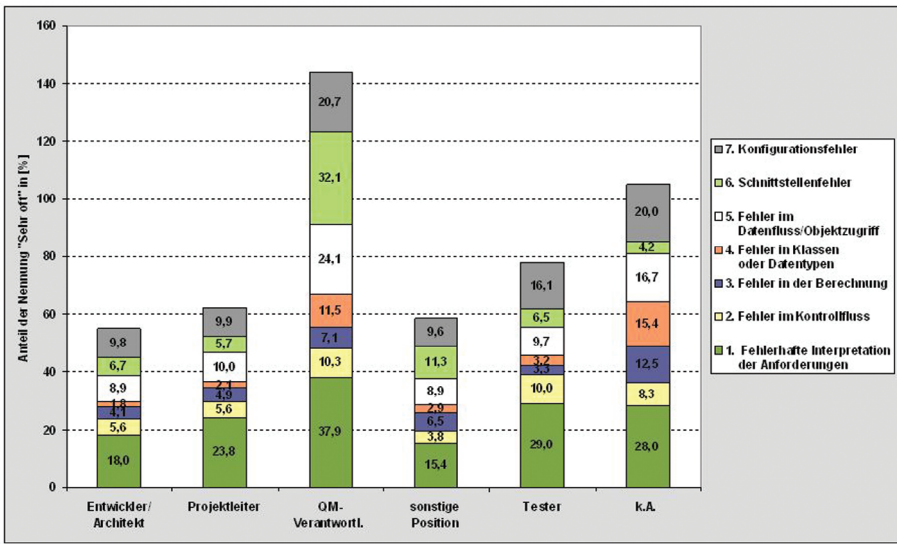

Abb. 5: Bewertung nach Arbeitsrolle.

eine nicht zu unterschätzende Auswirkung auf die zu erwartenden Fehlerkategorien.

Auswirkung der Branche

Bei der Branche konnten die Teilnehmer der Umfrage unter folgenden vorgegebenen Antwortmöglichkeiten wählen: „Dienstleistung", „kommerzieller Bereich", „öffentlicher Bereich", „Software", „technischer Bereich" und „keine Angabe". Die meisten Teilnehmer arbeiten für Softwarefirmen (44%), gefolgt von Dienstleistungsunternehmen (29%) und Unternehmen im technischen Bereich (16%).

Von den explizit angegebenen Branchen wurde – über alle Fehlerkategorien hinweg – am häufigsten im kommerziellen Bereich die Antwort „sehr oft" genannt (in Summe 84,5% von maximal 7x100%). Die anderen Branchen wiesen hingegen relativ dazu bis zu 40 % weniger „sehr oft"-Nennungen auf (siehe Abb. 4):

Anforderungsbezogene Fehler sind über die Branchen signifikant ungleich verteilt. Insgesamt wurden sie am geringsten im technischen Bereich bewertet (14,4% „sehr oft"), am höchsten im kommerziellen Bereich (31,7% „sehr oft"). Insbesondere die fehlende Implementierung einer impliziten Anforderung wurde im kommerziellen Bereich zu 28,3% und im Dienstleistungsbereich zu 23,5% mit „sehr oft" bewertet, im technischen Bereich hingegen nur zu 15,5%, und im öffentlichen Bereich sogar nur zu 3,3%.

Fehler im Kontrollfluss wurden erwartungsgemäß am häufigsten im technischen Bereich mit „sehr oft" bewertet (7,3%). Von den Unterkategorien wurden Fehler in der Behandlung von internen Ausnahmefällen signifikant unterschiedlich bewertet: in den Branchen Dienstleistung 20,4% der Nennungen „sehr oft", im kommerziellen Bereich 17,1%, im öffentlichen Bereich hingegen nur 6,5%. Tendenziell unterschiedlich bewertet wurden Fehler in der Zustandslogik.

Bei *Fehlern in der Berechnung* gab es tendenziell verschiedene Bewertungen, je nach Branche bei den Fehler-Unterkategorien „Falscher Algorithmus ausgewählt", „Fehler in Zeigerarithmetik" und „Fehler in Initialisierung".

Fehler in Klassen oder Datentypen werden über die Branchen hinweg signifikant unterschiedlich bewertet. So werden beispielsweise Fehler in der Typtransformation im technischen Bereich am stärksten

bewertet (9,3% nannten „sehr oft"). Tendenziell werden Fehler in Klassen oder Datentypen am höchsten im kommerziellen und am niedrigsten im Softwarebereich bewertet. Tendenziell unterschiedlich bewertet wurden die Fehler-Unterkategorien „Fehlerhafte Deklaration eines Attributs bzw. einer Variable", „Fehler durch Mehrfachvererbung", „Methode nicht implementiert/überschrieben" sowie „fehlerhafte Deklarierung einer Methode / eines Attributes als statisch bzw. dynamisch".

Bei *Fehlern im Datenfluss/Objektzugriff* wurden die Fehler-Unterkategorien „Falsche Zugriffsart" sowie „Fehler durch Umgehung des Zugriffsschutzes" tendenziell unterschiedlich bewertet.

Schnittstellenfehler treten tendenziell am häufigsten im kommerziellen Bereich auf, am seltensten im technischen Bereich.

Zwischen *Konfigurationsfehlern* und der Branche besteht eine signifikante Korrelation, wobei die Konfigurationsfehler im kommerziellen Bereich am höchsten eingeschätzt werden (15,3% „sehr oft"). Der Softwarebereich ist mit 6,6% hingegen die einzige Branche, bei der weniger als 10% „sehr oft" genannt haben.

Deployment-Deskriptoren werden im kommerziellen Bereich zu 13,3% mit „sehr oft" bewertet, im technischen Bereich nur zu 1%. Dieser Effekt dürfte wesentlich auf die in den Branchen tendenziell ungleich stark eingesetzten Programmiersprachen zurückzuführen sein.

Die Ursache der unterschiedlichen Bewertungen je nach Branche kann aus der Umfrage nicht abgeleitet werden, dürfte aber durch folgende Punkte beeinflusst werden:

- unterschiedlich stark eingesetzte Technologien
- unterschiedliche Anforderungen der Auftraggeber in der Branche
- unterschiedliche Erfordernisse der Anwendungsdomäne, die sich aus der Kritikalität und technischen Komplexität ergeben

Auswirkung der Arbeitsrolle

Die überwiegende Mehrzahl (72%) der Teilnehmer arbeitet als Entwickler oder Architekt, gefolgt von Projektleitern (12%) sowie einer geringeren Anzahl von Testern (2,6%) und Verantwortlichen für das Qualitätsmanagement (QM-Verantwortlichen) (2,4%). Der Rest entfällt auf sonstige Arbeitsrollen (8,7%) und fehlende Angaben (2,1%).

Aus **Abbildung 5** ist ersichtlich, das QM-Verantwortliche insgesamt etwa doppelt so häufig wie Entwickler, Projektleiter oder Tester Fehlerkategorien mit „sehr oft" bewertet haben. Insbesondere der Unterschied in der Bewertung zu Testern ist überraschend, da sowohl QM-Verantwortliche als auch Tester eine Qualitätsperspektive haben.

Die Arbeitsrolle als Kontextfaktor weist sehr häufig eine Korrelation mit der Anzahl der genannten Fehlerhäufigkeit auf, allerdings (durch die ungleiche Verteilung der Arbeitsrollen und die damit teilweise geringen Häufigkeiten pro Kombination) allein für die Fehler-Hauptkategorie „Fehlerhafte Interpretation der Anforderungen" in signifikantem Ausmaß. Dabei haben Projektleiter und Tester im Verhältnis zu den anderen Fehlerkategorien anforderungsbezogene Fehler überdurchschnittlich bewertet (38% bzw. 37% aller „sehr oft"-Nennungen dieser Arbeitsrolle), QM-Verantwortliche hingegen unterdurchschnittlich (26%). Nur tendenzielle Unterschiede gibt es bei den Schnittstellenfehlern, die von den QM-Verantwortlichen überproportional, von Testern hingegen unterproportional bewertet werden.

Diese Ergebnisse zeigen, dass eine objektive Erfassung der Häufigkeiten unterschiedlicher Fehlerkategorien wichtig ist, um den Fokus der Testaktivitäten richtig zu wählen.

Die ordinale Skala der Antworten erlaubt aus Sicht der Messtheorie nicht die Nutzung von Mittelwerten zur Beschreibung der Daten. Aufgrund der kleinen Anzahl der möglichen Antworten (vier) lassen sich andererseits Unterschiede in den Daten anhand der Mediane bzw. Quartilen nicht gut darstellen. Daher beschreiben wir die Daten im Text meist anhand der Häufigkeit der Antwort „sehr oft". Diese Einschränkung betrifft aber nicht die im Abschnitt „Fehlerkategorien und Kontextfaktoren" erläuterten Korrelationen, d. h. die Angaben zu den Korrelationen basieren sehr wohl auf allen vier möglichen Antworten von „nie" bis „sehr oft".

Die Fehler-Unterkategorien wurden unabhängig von den Fehler-Hauptkategorien bewertet. Die Häufigkeit einer Fehler-Unterkategorie berechnet sich aus dem Verhältnis der Anzahl der Antworten „sehr oft" dieser Unterkategorie zu der Anzahl aller Antworten zu dieser Unterkategorie. Daher ist die Summe der Prozentsätze einer Antwort zu den Fehler-Unterkategorien einer Fehler-Hauptkategorie in der Regel nicht identisch mit dem Prozentsatz dieser Antwort der Fehler-Hauptkategorie.

Die Teilnehmer waren schwerpunktmäßig Entwickler, wobei dies nur einen relativ geringen Einfluss auf die Verteilung der Fehlerkategorien hat. Eine stärkere Einschränkung der Gültigkeit der Ergebnisse ergibt sich aus dem Umstand, dass die Mehrheit der Teilnehmer mutmaßlich nur ihren persönlichen subjektiven Eindruck der Fehlerhäufigkeiten wiedergeben konnte, weil Fehlerdatenbanken mit gleichzeitiger Fehlerkategorisierung in der betrieblichen Praxis selten anzutreffen sind.

Kasten 1: *Anmerkungen zur statistischen Auswertung der Daten und Gültigkeit der Ergebnisse.*

Hinweise für einen effektiven Test

Aus den Ergebnissen der Umfrage ergeben sich folgende Schlussfolgerungen und Hinweise für einen effizienten sowie effektiven Testablauf:

- Kategorisieren Sie die Fehler in der Fehlerdatenbank. Subjektive Einschätzungen über notwendige Testschwerpunkte entsprechen nicht immer der Realität Ihrer Software.
- Vor dem eigentlichen Test sollen die Anforderungen möglichst frühzeitig geprüft werden, da diese die kritischste Fehlerkategorie darstellen. Hier kön-

nen verschiedene Review-Verfahren helfen, in die der Kunde auf jeden Fall miteinbezogen werden sollte. Dem Einsatz von Techniken zur Aufdeckung von impliziten Anforderungen soll eine große Bedeutung zukommen, insbesondere im kommerziellen Bereich und der Branche Dienstleistung. Auch konstruktive Maßnahmen, wie eine Standardisierung der Anforderungsdokumente, tragen zu expliziten, unmissverständlichen Anforderungen bei.

- Die Abdeckung deklarativer Codeanteile (die z. B. in Konfigurationsdateien stecken) soll in den meisten Branchen explizit als Testziel behandelt werden. Speziell darauf ausgerichtete Testmethoden, wie z. B. die minimale Abdeckung eines mehrdimensionalen Parameterraums durch „Covering Arrays" (vgl. [Yil06]), sind allerdings noch sehr selten. Daher sollten konstruktive Möglichkeiten gesucht werden, um Konfigurationsfehler möglichst zu vermeiden (Entwurf für Testbarkeit).

- In der Umfrage wurde die Fehler-Unterkategorie „Datenflussanomalie" nach den anforderungsbezogenen Fehler-Unterkategorien am häufigsten mit „sehr oft" bewertet. Die Anwendung von datenflussorientierten Testmethoden lohnt sich daher, sofern die entsprechenden Werkzeuge vorhanden sind.

- Kontrollflusstests sollten den Fokus auf spezifische Fehler-Unterkategorien legen, wie z. B. interne Ausnahmefälle und Fehler bei parallelen Prozessen. Hier können auch moderne Werkzeuge zur statischen Codeanalyse effizient eingesetzt werden.

- Durch den Paradigmenwechsel von der strukturierten zur objektorientierten Entwicklung hat sich – durch tendenziell kleinere Methoden und die Nutzung von Polymorphismus – eine Verschiebung von Berechnungsfehlern hin zu Interaktions- und Datenflussfehlern ergeben. Daher muss auch hier ein konstruktives Ziel die Vermeidung unnötiger Komplexität sein. Bei Berechnungsfehlern sollte sich die Qualitätssicherung auf die spezifischen Schwachstellen der verwendeten Programmiersprache (z. B. bei C++) konzentrieren.

- Bei der Wahl der Programmiersprache sollte auch die Fehlerhäufigkeit berücksichtigt werden, da hier Unterschiede von bis zu 25% auftreten können.

Nutzen der Fehlerkategorisierung

Testtechniken sind Medizin gegen Fehler. Ihre Effektivität hängt vom Typ des Krankheitserregers (der Fehlerkategorie) ab – was die eine Fehlerart wirksam bekämpft, kann gegen eine andere wirkungslos sein. Die Techniken, die Sie einsetzen, müssen den bei Ihnen vorkommenden Fehlerkategorien entsprechen. Allerdings unterscheidet sich die subjektive Wahrnehmung der Häufigkeit der Fehlerkategorien zwischen den Verantwortlichen in der Softwareentwicklung teilweise stark. Der Einsatz einer Fehlerkategorisierung hilft hier, die Schwerpunkte in der Qualitätssicherung aus einer gemeinsamen, einheitlichen Perspektive heraus zu setzen.

Eine Fehlerkategorisierung ist die Voraussetzung für die Sammlung von Fehlerstatistiken. Führen Sie eine Klassifizierung in Ihrer Fehlerdatenbank ein (die hier vorgestellte oder eine eigene Variante), um die geeigneten Qualitätssicherungsmaßnahmen auszuwählen und keine wesentlichen Bereiche der Fehlerrisiken zu vernachlässigen. Als fortgeschrittene Technik können Sie die Fehlerstrom-Analyse (vgl. [Stu07]) nutzen, d. h. die Untersuchung, welche Fehlerart in welcher Entwicklungsphase wie häufig entsteht und in welcher sie gefunden wird. Das erlaubt Ihnen eine weitere Fokussierung und damit einen effektiveren Einsatz von Qualitätssicherungstechniken.

Eine Fehlerkategorisierung erfordert einen kleinen Zusatzaufwand bei der Fehlererfassung, wobei es durchaus legitim ist, durch eine Beschränkung der Kategorisierung auf besonders relevante Fehler diesen Zusatzaufwand zu minimieren. Falls Sie noch nicht über eine eigene Fehlerdatenbank mit Fehlerkategorisierung verfügen, bieten die hier präsentierten Ergebnisse Ihnen einen ersten branchenbezogenen Anhaltspunkt über die Verteilung der Fehlerkategorien.

Die Fehlerkategorisierung hilft Entwicklern, aus ihren Fehlern zu lernen und Programmiersprachen, Werkzeuge und Prozesse verbessert bzw. adäquat einzusetzen. Dadurch verschieben sich in der Regel die Komplexität und die Fehlerhäufigkeit bei der Softwareentwicklung. Wie in der Medizin sollten Sie daher den Gesundheitszustand Ihrer Softwareentwicklung ständig beobachten und Testtechniken bzw.

Literatur & Links

[Bei90] B. Beizer, Software Testing Techniques, 2nd Ed. Thomson Computer Press, 1990

[Bin00] R.V. Binder, Testing Object-Oriented Systems. Addison-Wesley, 2000

[Bor06] L. Borner, F. Fraikin, M. Hamburg, S. Jungmayr, A. Schönknecht, Fehlerhäufigkeiten in objektorientierten Systemen: Basisauswertung einer Online-Umfrage, Technical Report SWEHD-TR-2006-01, Uni Heidelberg, 2/06, siehe: www-swe.informatik.uniheidelberg.de/research/publications/reports.htm

[Fen97] N. E. Fenton, S. Lawrence Pfleeger, Software Metrics 2nd. Ed. International Thomson Computer Press, 1997

[Mye04] G. Myers, The Art of Software Testing, 2nd. Ed. John Wiley & Sons, 2004

[Rop94] M. Roper, Software Testing, McGraw-Hill, 1994

[Spi05] A. Spillner, T. Linz, Basiswissen Softwaretest, 3. überarbeitete Auflage, dpunkt, 2005

[Stu07] M. Stupperich, Automatisierte Erfassung von Änderungs- und Fehlermetriken zur Beurteilung von Prozessen, 2007, siehe: www.itq.ch/pdf/sqm/07/sqm_stupperich_pr.pdf

[Yil06] C. Yilmaz et al., Covering Arrays for Efficient Fault Characterization in Complex Configuration Spaces, in: IEEE Transactions on Software Engineering, Vol. 32, No. 1, 1/06

[TOO] Arbeitskreis TOOP (Testen objektorientierter Programme) der GI, Homepage, siehe: toop.gi-ev.de

Testschwerpunkte an veränderte Fehlerhäufigkeiten anpassen.

Einladung und Dank

Wenn Sie die Ergebnisse der Umfrage mit Mitarbeitern des Arbeitskreises „Testen objektorientierter Programme" (TOOP) diskutieren bzw. eigene Erfahrungen, Kommentare oder Zusatzinformationen einbringen wollen, so laden wir Sie herzlich ein, über die Homepage des Arbeitskreises (vgl. [TOO]) Kontakt mit den Autoren aufzunehmen.

Wir danken Michael Averstegge, Falk Fraikin, Kay Krämer, Roger Müller und Andreas Schönknecht, die an der Entwicklung der Umfrage beteiligt waren. Ebenso gilt unser Dank der Zeitschrift OBJEKTspektrum, der Computer-Zeitung, dem Heise Zeitschriften Verlag, dem Verlag dpunkt, der Arbeitsgruppe Software Systeme der Universität Heidelberg, Markus Niehammer sowie weiteren, in [Bor06] genannten Unterstützern der Umfrage.