Klassifikationsbaum-Methode

aus Wikipedia, der freien Enzyklopädie

Die **Klassifikationsbaum-Methode** (englisch *classification tree method*) ist eine im Bereich von eingebetteter Software verbreitete Methode[1] zur Ermittlung funktionaler Blackbox-Tests.[2] Sie wurde ursprünglich 1993 von Grimm und Grochtmann entwickelt.[3] Es handelt sich dabei *nicht* um Klassifikationsbäume im Sinne von Entscheidungsbäumen.

Die Klassifikationsbaum-Methode besteht im Wesentlichen aus zwei Schritten:[4][5]

1. Bestimmung der testrelevanten Aspekte (so genannte *Klassifikationen*) und möglicher Ausprägungen (genannt *Klassen*), so wie
2. Kombination von unterschiedlichen Klassen für alle Klassifikationen zu Testfällen.

Ausgangspunkt ist die funktionale Spezifikation (z. B. Requirements, Usecases, ...) des Testobjekts. Der zweite Schritt der Methode folgt den Prinzipien des kombinatorischen Testentwurfs.[4]

Ergebnis ist eine Menge von Testfallspezifikationen, die möglichst fehler-sensitive, jedoch keine redundanten Testfallspezifikationen enthalten soll. Durch die methodische Vorgehensweise soll sichergestellt werden, dass die resultierende Menge von Testfallspezifikationen alle relevanten Testfälle enthält.

Obwohl die Methode mit Stift und Papier anwendbar ist, wird üblicherweise der Klassifikationsbaum-Editor (engl. *Classification Tree Editor*) dafür verwendet.[6]

Inhaltsverzeichnis

Die Anwendung der Klassifikationsbaum-Methode

Voraussetzung für die Anwendung der Methode ist die Festlegung eines Testobjekts. Die Klassifikationsbaum-Methode ist eine Methode für den Black-Box-Test und unterstützt dabei ganz unterschiedliche Arten von Prüflingen, wie beispielsweise Hardware-Systeme, integrierte Hardware-Software-Systeme, reine Software-Systeme, insbesondere auch Eingebettete Systeme, Benutzer-Schnittstellen, Betriebssysteme, Parser, so wie auch Teilsysteme dieser.

Nach Auswahl des Prüflings besteht der erste Schritt nun in der Ermittlung der testrelevanten Aspekte.[4] Ein jedes System lässt sich beschreiben als eine Sammlung aus Klassifikationen für alle Eingabe- und Ausgabe-Parameter. (Wobei Eingabe-Parameter auch Umgebungsvariablen, Pre-Conditions und weitere, eher untypische Parameter enthalten können.)[1] Jede Klassifikation besteht aus einer beliebigen Menge von disjunkten Klassen, welche mögliche Ausprägungen der Parameter beschreiben. Die Wahl der Klassen folgt dabei typischerweise dem Prinzip der Äquivalenzklassen-Bildung für abstrakte Testfälle und der Grenzwert-Analyse für konkrete Testfälle.[5] Zusammen bilden alle Klassifikationen und Klassen den Klassifikationsbaum. Darüber hinaus lassen sich Klassifikationen auch semantisch in Kompositionen gruppieren.

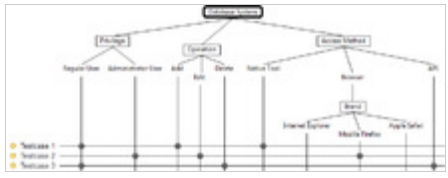
Diese Aufteilung des Problems in verschiedene Aspekte, die im Folgenden separat verfeinert werden können, reduziert die Komplexität des ursprünglichen Testproblems.

Eine Äquivalenzklasse kann man nach zusätzlichen Gesichtspunkten weiter klassifizieren. Durch die wiederholte Bildung von Unter-Klassifikationen mit den dazugehörigen Klassen ergibt sich schlussendlich der Klassifikationsbaum. Seine Wurzel bildet die Problemstellung; er wächst von oben nach unten; Klassifikationen sind durch Rechtecke eingerahmt; Klassen haben keinen Rahmen.

Im zweiten wesentlichen Schritt der Methode werden Testfällen dann dadurch gebildet, dass für jede Klassifikation im Baum genau eine Klasse gewählt wird. Die Auswahl der Testfälle war ursprünglich[3] eine manuelle Tätigkeit des Testers.

Die Blätter des Klassifikationsbaums (d.h. Klassen ohne weitere Klassifikation) bilden den Kopf der Kombinationstabelle. Eine Zeile in der Kombinationstabelle spezifiziert einen Testfall, indem die Blattklassen des Baums markiert werden, aus denen ein Wert für diesen Testfall verwendet wird.

Beispiel



Klassifikationsbaum für eine Datenbank-System

Für ein Datenbanksystem soll ein Testentwurf durchgeführt werden. Bei der Anwendung der Klassifikationsbaum-Methode ergibt im ersten Festlegung der testrelevanten Aspekte im ersten Schritt die Klassifikationen: *Benutzerrolle*, *Datenbankanfrage* und *Zugriffsart*. Für die *Benutzerrolle*n wurden zwei Klassen identifiziert: *Normaler Benutzer* und Administrator. Es gibt drei Arten von *Datenbankanfrage*n: *Hinzufügen*, *Bearbeiten* und *Löschen*. Für *Zugriffsart* wurden drei Klassen gefunden: *Natives Tool*, *Webbrowser*, *API*. Die *Webbrowser*-Klasse ist weiter verfeinert mit dem Testaspekt *Marke*, drei mögliche Klassen gibt es hier: *Internet Explorer*, *Mozilla Firefox*, and *Apple Safari*.

Der erste Schritt der Methode ist nun abgeschlossen. Natürlich gibt es weitere mögliche Testaspekte, wie zum Beispiel die Geschwindigkeit der Netzanbindung oder die Zahl der vorhandenen Datenbankeinträge. Durch die graphische Darstellung der Testaspekte im Baum lassen sich die gewählten Aspekte und die vorgesehenen Werte schnell einsehen.

Im zweiten Schritt wurden drei Testfälle manuell gewählt:

1. Ein normaler Benutzer fügt einen neuen Datensatz mittels eines nativen Tools ein.
2. Ein Administrator bearbeiten einen bestehenden Datensatz mit dem Firefox Browser.
3. Ein normaler Benutzer löscht einen Datensatz über die API.

Noch ein wenig Statistik: Es gibt 30 mögliche Testfälle insgesamt (2 Benutzerrollen * 3 Datenbankanfragen * 5 Zugriffsarten). Für die Minimalabdeckung reichen 5 Testfälle aus, da es 5 Zugriffsarten gibt (und Zugriffsart die Klassifikation mit der größten Anzahl an Klassen ist).

Erweiterungen

Ursprung

Die Klassifikationsbaum-Methode wurde im Software-Forschungslabor von Daimler-Benz in Berlin entwickelt und ist eine Weiterentwicklung der Category-Partition Method[7] (CPM) von Ostrand und Balcer. Gegenüber der CPM bietet sie folgende Vorteile:[1]

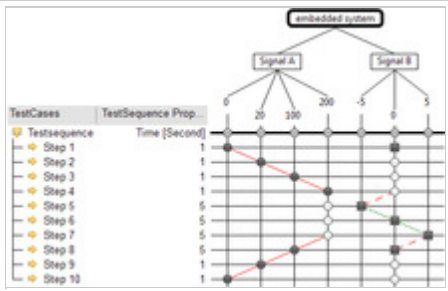
- **Notation**: Bei der CPM gab es nur texuelle Beschreibung im Gegensatz zur graphischen Baum-Darstellung.
- **Verfeinerungen** erlauben die Spezifikation von bedingten Testaspekten.

  CPM bot nur Restriktionen. Die Klassifikationsbaum-Methode bietet hierarchische Verfeinerungen direkt im Baum für *implizite Abhängigkeiten*.

- **Werkzeugunterstützung**: Das Werkzeug von Ostrand und Balcer unterstützte nur Testfallgenerierung ohne Bestimmung der Testaspekte.

  Der *Classification Tree Editor* (CTE) von Grochtmann und Wegener hingegen unterstützt sowohl Partitionierung als auch Testfallgenerierung.[6]

Klassifikationsbaum-Methode für eingebette Systeme



Classification Tree for Embedded System Example containing concrete values, concrete timing, (different) transitions and distinguish between States and Actions

Die Klassifikationsbaum-Methode war ursprünglich für Entwurf und Spezifikation abstrakter Testfälle vorgesehen. Mit der Klassifikationsbaum-Methode für eingebette Systeme[8] lassen sich auch Tests implementieren. Dazu wurden einige Zusatzinformationen vorgesehen:

1. Zusätzlich zu atomaren Testfällen lassen sich Testsequenzen mit mehreren einzelnen Testschritten definieren.
2. Konkretes Zeitverhalten (z. B. in Sekunden, Minuten ...) kann für jeden einzelnen Testschritt festgelegt werden.
3. Signalverläufe (z. B. Linear, Spline, Sinus ...) können zwischen gewählten Klassen aufeinander folgender Testschritte angegeben werden.
4. Eine Unterscheidung zwischen Aktion und Zustand lässt sich modellieren. Sie wird durch unterschiedliche graphische Markierungen im Test dargestellt.

Das Modultest-Werkzeug Tessy nutzt ebenfalls einige dieser Erweiterung.

Konkrete Testdaten und zeitliche Aspekte

Ansätze aus Klassifikationsbäumen Testdaten zu generieren wurden ebenfalls verfolgt.[9][10]

Die Methode Time Partition Testing (TPT) ist eine Erweiterung der Klassifikationsbaum-Methode um zeitliche Aspekte, die beim Test von Steuerungs- und Regelungssystemen eine große Rolle spielen. E. Lehmann beschreibt in seiner Dissertation,[11] wie Klassifikationen auf Zustände und Zustandsübergänge in den TPT-Automaten abgebildet werden können, und so auch zeitliche Aspekte abgebildet werden können.

Abhängigkeitsregeln und Automatische Testfallgenerierung

Der Verfeinerungsmechanismus in der Klassifikationsbaum-Methode lässt sich für die Modellierung von Abhängigkeiten nutzen. Allerdings lassen sich so keine Abhängigkeiten zwischen Klassen unterschiedlicher Klassifikationen modellieren. Lehmann und Wegener haben Abhängigkeitsregeln basierend auf Boolescher Algebra für den CTE eingeführt.[12] Außerdem ermöglichten sie die automatische Generierung von Testfallmengen basierend auf kombinatorischem Testdesign (z. B. der Pairwise-Methode).

Priorisierende Testfallgenerierung

Jüngere Erweiterungen zu Klassifikationbaum-Methode betreffen die priorisierende Testfallgenerierung: Mit ihnen ist es möglich, Elementen des Klassifikationsbaums Gewichte in Form von Eintrittswahrscheinlichkeit, Fehlerwahrscheinlichkeit und Risiko zuzuordnen. Diese Gewichte lassen sich dann in der Testfallgenerierung verwenden, um Testfälle in Reihenfolge ihrer Priorität (der abgedeckten Gewichte) zu erzeugen.[13] Statistisches Testen (z. B. für den Test von Materialermüdung und Verschleiß) ist ebenfalls möglich, indem die zugeordneten Gewichte als diskrete Wahrscheinlichkeitsverteilung interpretiert werden.

Testsequenz-Generation

Durch das Hinzufügen von gültigen Übergängen zwischen den einzelnen Klassen einer Klassifikation lassen sich diese Klassifikationen als State-Machine interpretieren. Die Gesamtheit aller Klassifikationen im Baum wird zum Statechart. Dieses Vorgehen definiert eine erlaubte Abfolge von Klassenverwendungen in Testschritten und damit auch die automatische Generierung von gültigen Testsequenzen.[14] Unterschiedliche Abdeckungsraten sind verfügbar, wie Zustandsüberdeckung, Transitionsüberdeckung (Pfadüberdeckung) und Abdeckung von Zustandspaaren und Transitionspaaren.

Numerische Abhängigkeiten

Zusätzlich zu Boolschen Abhängigkeiten, die Aussagen über das gemeinsame Auftreten von Klassen in einem Testfall machen, lassen sich auch numerische Abhängigkeiten definieren, die eine Berechnungsvorschrift vorgeben, bei denen Klassifikationen als Variablen interpretiert werden und deren Belegung sich aus den im Testfall genutzten Klassen ergibt.[15]

Classification Tree Editor CTE

Der Klassifikationsbaumeditor CTE unterstützt den Entwurf des Klassifikationsbaums sowie die Spezifikation der Testfälle. Der CTE besitzt einen Graphikeditor, der speziell zum Zeichnen von Klassifikationsbäumen gedacht ist. Beispielsweise berücksichtigt der CTE die vorgeschriebene Abfolge von Klassifikation bzw. Komposition und Klasse und schlägt beim Zeichnen das jeweils passende Baumelement vor. Die Testfallspezifikationen werden in der Kombinationstabelle angelegt und durch den CTE verwaltet. Sie können in Testsequenzen zusammengefasst werden, die zur Beschreibung von dynamischen Abläufen benötigt werden. Zur Dokumentation können von CTE Reports in Excel, Word oder als Textdatei erzeugt werden. Zum Modultest-Werkzeug Tessy können die erstellten Testfälle direkt exportiert werden.

Vorteile

Visualisierung der Testideen

Ein wesentlicher Vorteil der Klassifikationsbaum-Methode ist die Visualisierung der Testideen, die somit leicht vermittelbar sind, beispielsweise bei Reviews. Reviews sind auch ein probates Mittel, wenn es darum geht, sich zu versichern, dass keine Testfallspezifikationen fehlen.

Schafft Vertrauen

Hat man einen sorgfältig erarbeiteten Baum und wohlüberlegte Testfallspezifikationen erstellt, gibt es eine große Wahrscheinlichkeit dafür, dass man die meisten oder vielleicht sogar fast alle Fehler finden wird

Reduziert die Komplexität

Beim ersten Nachdenken über die notwendigen Testfälle für ein bestimmtes Problem hat man zumeist zahlreiche Einfälle und Ideen. Die Herausforderung besteht eher darin, die Einfälle zu strukturieren und redundante Testfälle zu erkennen. Die Klassifikationsbaum-Methode reduziert die in dieser Aufgabe liegende Komplexität, denn sie erlaubt, einzelne Aspekte (Klassifikationen) isoliert zu betrachten und sich gezielt Gedanken über die nötigen (und unnötigen!) Äquivalenzklassen zu machen.

Überprüft die Spezifikation

Der erste Schritt der Klassifikationsbaum-Methode ist die Analyse der Problemspezifikation. Dies erzwingt automatisch die Überprüfung der Spezifikation auf Auslassungen, Widersprüche und Unklarheiten.

Abschätzung des Testaufwands

Schon aus dem Klassifikationsbaum kann man gewisse Maße ermitteln, z. B. die minimale Anzahl von Testfällen für einen gegebenen Baum. Diese und andere Maße werden durch das Werkzeug CTE ermittelt.

Nachteile und Kritik

Das Erstellen von guten Klassifikationsbäumen wird durch die Vermischung unterschiedlicher Aspekte erschwert. Während bei der Äquivalenzklassenmethode und der Grenzwertanalyse zunächst jede Größe für die Zerlegung in relevante Bereiche allein betrachtet wird und erst in einem zweiten Schritt Abhängigkeiten zwischen Größen betrachtet werden, müssen beide Schritte im Sinn der Klassifikationsbaum-Methode zusammen erfolgen. Dies ist notwendig, da die Baumstruktur wesentlich von Abhängigkeiten zwischen Größen beeinflusst wird.

Klassifikationsbäume sind übersichtlich bei einer begrenzten Anzahl von Klassifikationen, Äquivalenzklassen und Testfällen. Bei großen Klassifikationsbäumen ist es schwierig die Übersicht zu behalten.

Siehe auch

- CART (Algorithmus)
- Entscheidungsbaum
- TPT – Eine Spezialisierung der Klassifikationsbaum-Methode für den Test von Steuerungs- und Regelsystemen

Einzelnachweise

1. Anne Mette Jonassen Hass: *Guide to advanced software testing*. Artech House, Boston 2008, ISBN 1596932864, S. 179–186.
2. Graham Bath, Judy McKay: *The software test engineer's handbook : a study guide for the ISTQB test analyst and technical test analyst advanced level certificates*, 1st, Rocky Nook, Santa Barbara, CA 2008, ISBN 9781933952246.
3. Matthias Grochtmann, Klaus Grimm: *Classification Trees for Partition Testing*. In: *Software Testing, Verification & Reliability*. 3, Nr. 2, 1993, S. 63–82. doi:10.1002/stvr.4370030203 (http://dx.doi.org/10.1002%2Fstvr.4370030203).
4. D. Richard Kuhn, Raghu N. Kacker, Yu Lei: *Introduction to combinatorial testing*. Crc Pr Inc, 2013, ISBN 1466552298, S. 76–81.
5. Pierre Henry: *The testing network an integral approach to test activities in large software projects*. Springer, Berlin 2008, ISBN 978-3-540-78504-0.
6. Matthias Grochtmann, Wegener, Joachim: *Test Case Design Using Classification Trees and the Classification-Tree Editor CTE*. (http://www.systematic-testing.com/documents/qualityweek1995_1.pdf) In: *Proceedings of the 8th International Software Quality Week(QW '95), San Francisco, USA*. 1995.
7. T. J. Ostrand, M. J. Balcer: *The category-partition method for specifying and generating functional tests*. In: *Communications of the ACM*. 31, Nr. 6, 1988, S. 676–686. doi:10.1145/62959.62964 (http://dx.doi.org/10.1145%2F62959.62964).
8. Mirko Conrad, Alexander Krupp: *An Extension of the Classification-Tree Method for Embedded Systems for the Description of Events*. In: *Electronic Notes in Theoretical Computer Science*. 164, Nr. 4, 1. Oktober 2006, S. 3–11. doi:10.1016/j.entcs.2006.09.002 (http://dx.doi.org/10.1016%2Fj.entcs.2006.09.002).
9. Stefan Lützkendorf, Klaus Bothe: *Attributierte Klassifikationsbäume zur Testdatenbestimmung*. (http://pi.informatik.uni-siegen.de/stt/23_1/01_Fachgruppenberichte/FG217/05_Bothe1.ps) In: *Softwaretechnik-Trends*. 23, Nr. 1, 2003.
10. Zhen Ru Dai, Peter H. Deussen, Maik Busch, Laurette Pianta Lacmene, Titus Ngwangwen, Jens Herrmann, Michael Schmidt: *Automatic Test Data Generation for TTCN-3 using CTE*. (http://webmail.testingtech.de/download/publications/ICSSEA-2005-CTE.pdf) In: *Proceedings of the 18th International Conference Software & Systems Engineering and their Applications (ICSSEA)*. 2005.
11. Eckard Lehmann: *Time Partition Testing – Systematischer Test des kontinuierlichen Verhaltens von eingebetteten Systemen (http://www.piketec.com/downloads/papers/ELehmannDissertation.pdf)* 2003.
12. Eckard Lehmann, Joachim Wegener: *Test Case Design by Means of the CTE XL*. (http://www.systematic-testing.com/documents/eurostar2000.pdf) In: *Proceedings of the 8th European International Conference on Software Testing, Analysis & Review (EuroSTAR 2000)*. 2000.
13. Peter M. Kruse, Magdalena Luniak: *Automated Test Case Generation Using Classification Trees*. In: *Software Quality Professional*. 13, Nr. 1, Dezember 2010, S. 4–12.
14. Peter M. Kruse, Joachim Wegener: *Test Sequence Generation from Classification Trees*. In: *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. April 2012, S. 539–548. doi:10.1109/ICST.2012.139 (http://dx.doi.org/10.1109%2FICST.2012.139).
15. Peter M. Kruse, Jürgen Bauer, Joachim Wegener: *Numerical Constraints for Combinatorial Interaction Testing*. In: *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. April 2012, S. 758–763. doi:10.1109/ICST.2012.170 (http://dx.doi.org/10.1109%2FICST.2012.170).

Kategorie: Testen (Software)

- Diese Seite wurde zuletzt am 15. Juli 2014 um 20:29 Uhr geändert.
- Abrufstatistik