Exploratives Testen mit KI-Unterstützung

Heiko Frey
Robert Bosch GmbH, Poststr. 70, D-71229 Leonberg, heiko.frey@de.bosch.com

Requirements-basiertes Testen (RBT) ist in der Softwareentwicklung ein etablierter Mindeststandard, um die spezifizierte Funktionalität eines Systems zu verifizieren. Es deckt die "bekannten Bekannten" (known knowns) ab und stellt sicher, dass das Produkt die dokumentierten Anforderungen erfüllt. Die allgemeine Erfahrung in komplexen Projekten zeigt jedoch, dass RBT allein nicht ausreicht, um eine hohe Produktqualität zu gewährleisten. Kritische Fehlerquellen liegen oft in den Bereichen, die nicht explizit spezifiziert wurden – den "unbekannten Unbekannten" (unknown unknowns).

Hier setzt das explorative Testen an. Es ist ein systematischer und erfahrungsbasierter Ansatz, bei dem Tests auf der Grundlage des Wissens und der Intuition der Testenden dynamisch entworfen und durchgeführt werden [2]. Ziel ist es, über die Grenzen des RBT hinauszugehen und Fehler aufzudecken, die durch rein formalisierte Testfälle unentdeckt bleiben würden. Obwohl dieser Ansatz sehr wertvoll ist, ist er als manueller Prozess kostenintensiv, schwer skalierbar und in seiner Effektivität stark von der Erfahrung der Testenden abhängig. Fünf Schwerpunktmethoden eignen sich besonders für das explorativen Testen:

1. Erfahrungsbasierter Test (einschl. Error guessing)
2. Grenzwertbetrachtung
3. User acceptance testing (UAT)
4. End user (usability) testing (EUT)
5. Performance testing

Eine moderne Teststrategie kombiniert daher RBT mit explorativem Testen. Während RBT eine Grundabdeckung sicherstellt, hilft exploratives Testen, unbekannte Risiken frühzeitig aufzudecken. Der Nutzen dieses gemischten Ansatzes lässt sich am besten am zeitlichen Verlauf der Fehlerentdeckung verdeutlichen. Bei einer reinen RBT-Strategie werden Fehler tendenziell erst in späteren Projektphasen entdeckt, wenn die formellen Testfälle ausgeführt werden. Die Kurve der Fehlerentdeckungen steigt also erst spät im Projektverlauf an. Werden explorative Tests jedoch frühzeitig in den Prozess integriert, verschiebt sich die Fehlerentdeckung signifikant nach vorne. Die Fehlerkurve steigt bereits in den frühen Phasen des Projekts stark an und flacht später ab.

Die frühzeitige Entdeckung unbekannter Risiken durch exploratives Testen reduziert also nicht nur die direkten Fehlerbehebungskosten, sondern minimiert auch das Gesamtrisiko des Projekts.

Der Einsatz von KI stößt im explorativen Testen an konzeptionelle Grenzen. Exploratives Testen lebt vom Kontextverständnis, implizitem Wissen und situativer Bewertung. Der Kenntnisstand einer KI über das System-under-Test (SUT) sowie dessen Nutzungskontext ist oft unvollständig oder nicht transparent. Entsprechend besteht die Gefahr, dass KI-generierte Testideen lediglich bekannte Heuristiken reproduzieren, ohne tatsächlich neue Erkenntnisse zu liefern. Eine Bewertung und Einordnung der Ergebnisse durch erfahrene Testende bleiben somit unerlässlich.
Es stellen sich daher die Fragen:

- Welche Unterstützung kann KI bei menschlichem explorativem Testen leisten?
- Inwieweit kann eine KI ein System-under-Test explorativ Testen und was ist das Ergebnis?

Welche Unterstützung kann KI bei menschlichem explorativem Testen leisten?

- **Test Charter Generierung**: KI kann auf Basis von Informationen zum SUT Test Charter generieren und Testideen vorschlagen.
- **Automatisierte Testfall-Erstellung**: KI kann auf Basis von Anforderungen oder Code automatisch sinnvolle Testfälle generieren.
- **Skriptloses Explorieren**: Insbesondere bei GUIs, allerdings ohne Semantik und UX-Bewertung
- **Fehlererkennung durch Musteranalyse**: KI erkennt ungewöhnliche Verhaltensmuster oder potenzielle Bugs, die manuell schwer auffindbar sind.
- **Self-Healing Mechanismen**: Bei UI-Tests kann KI automatisch auf Änderungen reagieren und Testskripte anpassen
- **Regressionstests**: Aufzeichnung und Reproduzierung von Testpfaden, „Bring mich wieder da hin“

- **Dokumentation & Reporting**: KI-gestützte Tools erfassen automatisch Testergebnisse, Screenshots und Kommentare.
- **Priorisierung von Tests**: KI kann basierend auf Risikoanalysen entscheiden, welche Tests zuerst ausgeführt werden sollten.

Als Fallbeispiel wurde eine Webseite gewählt, die einen Parkkostenrechner simuliert [3]. KI-basiert wurde ein Testcharter für eine 30-minütige Testsession generiert und Testziele auf Basis der fünf Schwerpunktmethoden formuliert. Hier wurden konkrete Aspekte für den manuellen explorativen Tests genannt, wie z.B. Berechnungsfehler, ungewöhnliche Eingaben, Usability und Grenzfälle etwa bei sehr langen Zeiträumen.

Somit konnte die Nützlichkeit einer KI-Unterstützung in diesem Fall bestätigt werden. Allerdings sei erwähnt, dass die KI (Microsoft Copilot) nach eigenen Angaben das Fallbeispiel bereits kannte; Zitat „wird oft für Tests von Datum- und Zeitberechnung genutzt“.

Inwieweit kann eine KI ein System explorativ Testen und was ist das Ergebnis?

Mit demselben Fallbeispiel wurden KI-basiert 20 Testfälle und eine Testsuite dafür generiert. Die automatisierte Durchführung der Testfälle zeigte Fehler auf, die beispielsweise fehlerhafte Parkzeitberechnungen betrafen. Die Testlogs konnten ebenfalls KI-gestützt automatisiert ausgewertet werden. Auf Basis dieser Auswertung konnten verfeinerte Testfälle für eine weitere Iteration der Testdurchführung generiert werden.

Nach Durchführung dieser beiden Beispiele kann für die KI-Unterstützung beim explorativen Testen im Hinblick auf die 5 Schwerpunkttestmethoden gesagt werden:

1. Erfahrungsbasierter Test einschl. Error guessing: Sehr gute Unterstützung
2. Grenzwertbetrachtung: Sehr gute Unterstützung
3. User acceptance testing (UAT): Eingeschränkte Unterstützung wegen unklarer Bewertung von Usability und nicht-formulierter Erwartungen. Der Nutzen bleibt in der Abdeckung funktionaler Aspekte.
4. End user (usability) testing (EUT): Keine Unterstützung wegen unklarer Bewertung von Usability und Nicht-Testbarkeit von nicht-formulierter Erwartungen.
5. Performance Testing: Eingeschränkte Unterstützung wegen Abhängigkeit zu einer performanten Testumgebung.

Des Weiteren erwähnt werden sollen hier noch skriptlose explorative Testwerkzeuge [5]. Diese Tools interagieren direkt mit einer grafischen Benutzeroberfläche (GUI), indem sie zur Laufzeit alle interaktiven Elemente wie Buttons und Menüs identifizieren und autonom Aktionen ausführen.

Dieser Ansatz, der auch als "intelligentes Monkey-Testing" bezeichnet wird, simuliert die Erkundung durch einen Benutzer, um die Anwendung auf Robustheit zu prüfen und Abstürze oder nicht reagierende Zustände aufzudecken – ohne Verwendung eines Testskriptes [6]. Während diese Werkzeuge die Stabilität einer Anwendung effizient validieren, fehlt ihnen jedoch das tiefere semantische Verständnis, um die Korrektheit komplexer Geschäftslogik zu bewerten. Sie stellen fest, dass die Anwendung abstürzt, aber nicht, ob ein berechnetes Ergebnis inhaltlich falsch ist.

Fazit

KI kann beim explorativen Testen bei der Testvorbereitung unterstützen, z.B. beim Erstellen von Testchartern oder bei der Formulierung von Testideen. KI-generierte Testfälle müssen weiterhin auf Sinnhaftigkeit und Qualität überprüft werden; der Kenntnisstand der KI über das SUT ist oft unklar. Testergebnisse können jedoch ebenfalls durch KI bewertet werden und zur weiteren Testfallverfeinerung genutzt werden.

Die KI-Unterstützung findet ihre Grenzen bei der Bewertung nicht-formulierter Kunden- und Endnutzererwartungen und bei Benutzbarkeits-kategorien [7], wie Erlernbarkeit, Bedienbarkeit, Schutz vor Fehbedienungen (verständliche Warnungen), Ästhetik, Zugänglichkeit, Barrierefreiheit, Verständlichkeit.

Quellen

[1] https://www.5oi.org/the-five-orders-of-ignorance, abgerufen 14.02.2026

[2] https://istqb.org/wp-content/uploads/2024/11/ISTQB_CTFL_Syllabus_v4.0.1.pdf, abgerufen 14.02.2026

[3] https://www.shino.de/parkcalc, abgerufen 14.02.2026

[4] A Survey on Web Testing: On the Rise of AI and Applications in Industry, I. Kertusha et al., Preprint submitted to Journal of Systems and Software, August 2025

[5] Scripted and scriptless GUI testing for web applications: An industrial case, Axel Bons et al., Information and Software Technology 158 (2023) 107172

[6] TESTAR – scriptless testing through graphical user interface, Tanja E.J. Vos et al., Softw. Test. Verif. Reliab. 2021;31:e1771

[7] ISO25010-2011, Systems and software engineering, Systems and software Quality Requirements and Evaluation (SQuaRE), System and software quality models